arXiv:2212.13881v1 [cs.LG] 28 Dec 2022

Feature learning in neural networks and kernel machines that recursively learn features

Adityanarayanan Radhakrishnan[*,1], Daniel Beaglehole[*,2], Parthe Pandit[3], Mikhail Belkin[2,3]

December 29, 2022

Abstract

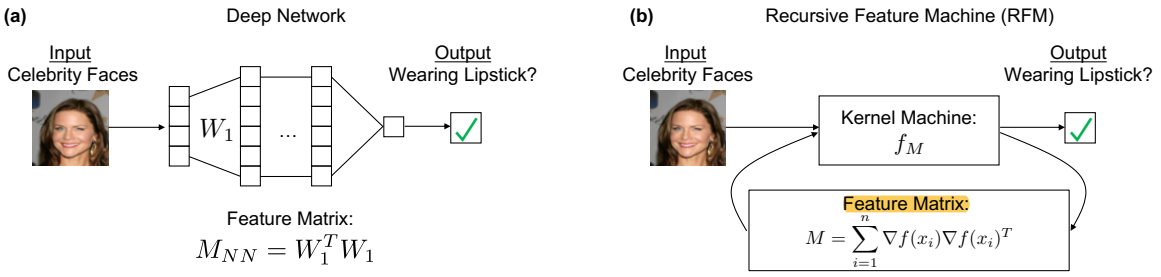
Neural networks have achieved impressive results on many technological and scientific tasks. Yet, their empirical successes have outpaced our fundamental understanding of their structure and function. By identifying mechanisms driving the successes of neural networks, we can provide principled approaches for improving neural network performance and develop simple and effective alternatives. In this work, we isolate the key mechanism driving feature learning in fully connected neural networks by connecting neural feature learning to the average gradient outer product. We subsequently leverage this mechanism to design *Recursive Feature Machines* (RFMs), which are kernel machines that learn features. We show that RFMs (1) accurately capture features learned by deep fully connected neural networks, (2) close the gap between kernel machines and fully connected networks, and (3) surpass a broad spectrum of models including neural networks on tabular data. Furthermore, we demonstrate that RFMs shed light on recently observed deep learning phenomena such as grokking, lottery tickets, simplicity biases, and spurious features. We provide a Python implementation to make our method broadly accessible [GitHub].

1 Introduction

In the last few years, modern neural networks have helped achieve major progress on a variety of applications including image generation [52], protein folding [69], and language understanding and generation [10]. Indeed, the impressive empirical successes of these complex models have recently far outpaced our fundamental understanding of their structure and function. Identifying key mechanisms driving the success of neural networks would lead to principled approaches for improving their performance and developing effective alternatives.

A mechanism of particular interest is the one by which neural networks learn features. This aspect of neural networks is thought to be a central contributor to their superior performance, e.g., [60, 75]. Yet, despite significant research effort in this area, identifying the component of neural networks associated with feature learning has been a challenge. Different lines of investigation connect feature learning to various aspects of neural network methodology such as model capacity (e.g., network width/depth) [73], the initialization scheme [75], or the optimization method [38]. Specifically, given that very wide neural networks are equivalent to kernel machines under certain assumptions [30], several works argue that finite width analyses through perturbative methods [24, 53] or through higher order approximations [6, 27] are necessary for capturing feature learning. In contrast, the work [75] argues that model capacity is not the main issue and instead, initializing networks near zero is key to learning features. Furthermore, there is a line of work demonstrating that tuning the optimization method (in particular, increasing the learning rate) can improve the features learned by neural networks and ultimately improve their performance [37, 38, 80]. Recently, several theoretical works attempt to characterize how neural networks can learn features from data by studying the evolution of network layers during gradient descent [16, 60]. Still, no conclusive unifying framework for feature learning has yet emerged.

[*]Equal contribution.
[1]MIT, Broad Institute of MIT and Harvard.
[2]Computer Science and Engineering, UC San Diego.
[3]Halıcıoğlu Data Science Institute, UC San Diego.



(c) Top Eigenvector of Feature Matrices on CelebA Prediction Tasks

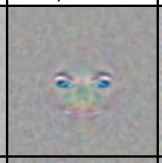
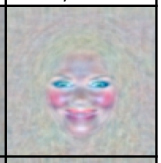
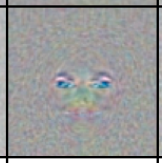
	Lipstick	Eyebrows	5 o'clock shadow	Necktie	Smiling	Rosy Cheeks
Deep Network Feature Matrix: $W_1^T W_1$						
RFM Feature Matrix: $\sum_{i=1}^{n} \nabla f(x_i) \nabla f(x_i)^T$						
Correlation	0.999	0.999	0.999	0.999	0.999	0.999
Deep Network Test Acc.	90.53%	75.71%	85.88%	88.77%	89.83%	87.22%
RFM-T Test Acc.	91.62%	78.11%	88.18%	90.39%	91.24%	88.72%

Figure 1: An overview of our main results. (a) We identify the feature matrix M_{NN}, which is given by the product of the transpose of the first layer weight matrix, W_1, with itself. The feature matrix captures the features used by the deep network for prediction. (b) By connecting the feature matrix from neural networks to the average gradient outer product, we construct a class of kernel machines that recursively learn features (RFMs). Training RFMs involves iteratively estimating a predictor with a kernel machine and then estimating features with the averaged gradient outer product. (c) RFMs accurately capture features learned by deep fully connected neural networks. The top eigenvector of feature matrices learned by such networks and RFMs are highly correlated (Pearson correlation of 0.999) and appear visually indistinguishable. Moreover, training a kernel machine after thresholding top features identified by RFMs (denoted RFM-T) leads to higher test accuracy over deep networks for these tasks.

In this work, we isolate a key mechanism of feature learning in deep fully connected neural networks by connecting feature learning with a statistical estimator for feature selection known in the literature as the *expected gradient outer product* [25, 68]. Furthermore, we posit that this is the primary mechanism behind feature learning in such neural networks. We subsequently leverage this connection to develop a class of kernel machines, *Recursive Feature Machines* (RFMs), that learn features. We demonstrate that RFMs (1) accurately capture features learned by deep networks, (2) close the gap between kernel machines and neural networks, and (3) achieve state-of-the-art performance on tabular data, in particular outperforming a spectrum of machine learning models including modern neural networks.

We also provide theoretical evidence for the connection between feature learning in fully connected networks and the expected gradient outer product. In particular, under certain simplifying conditions on network initialization and training scheme, we show that the features learned in linear and nonlinear fully connected networks are equivalent to those given by expected gradient outer product.

In addition, we show that RFMs shed light on several striking phenomena recently identified in deep learning literature including grokking [50], lottery tickets [20], simplicity biases [59], and spurious feature identification [62]. Thus, in addition to their practical value, we envision that RFMs will provide an analytically tractable framework for analyzing these phenomena.

The mechanism driving feature learning in neural networks. In this work, we isolate a simple mathematical object that is key to capturing the features learned by the neural network through training, which we refer to as the *feature matrix* (See Fig. 1a). Letting $W_1 \in \mathbb{R}^{k\times d}$ denote the first layer weights in a fully connected network, this feature matrix is given by the Gram matrix $W_1^T W_1$. We connect the feature matrix learned by neural networks to the expected gradient outer product [68], which is a statistical estimator used for feature selection. We refer to this connection as the Neural Feature Ansatz. We provide extensive experimental evidence that (1) neural networks indeed learn features that are similar to those given by the expected gradient outer product and (2) prediction using these features closely matches performance of neural networks. We also provide theoretical evidence showing that, under certain conditions, the feature matrix at a given step of network training is equivalent to the expected gradient outer product of the neural network at that step.

Developing kernel machines that learn features. Kernel machines are classical machine learning models that are conceptually simple and are easy to implement computationally [57]. There has been renewed interest in such models since recent work [30] proved under certain conditions on network initialization that infinitely wide neural networks implement kernel machines using a Neural Tangent Kernel (NTK) and, indeed, it is the case for a quite broad class of neural networks [79]. The NTK has provided a simple and effective alternative to neural networks on a variety of tasks from image classification [2, 36] to virtual drug screening [51]. Yet, a fundamental limitation of the NTK and many other typically used methods is that they use kernel functions that are not data adaptive. Thus, on certain tasks, kernel machines significantly underperform neural networks [16, 56, 75]. Recent works have analyzed so-called *after kernels* [41] extracted from neural networks *after* training, in contrast to usual NTKs, which are obtained at initialization, *before* training. The kernel machines trained using these kernels match the performance of neural networks, thereby suggesting that data-adaptive kernels can be as powerful as neural nets. Still, using after kernels is clearly impractical as their construction requires training a full neural network, thus obviating the need for another predictor.

In this work, we develop RFMs, which are a class of kernel machines that can learn features and are thus data-adaptive. RFMs contain a feature matrix that is iteratively refined via the average gradient outer product, thereby enabling feature learning (See Fig. 1b). RFMs consist of iterating a two step procedure where the first step involves estimating a predictor via a kernel machine and the second step involves using the average gradient outer product to update the feature matrix. Unlike neural networks, which generally have non-convex optimization landscapes, training RFMs involves iteratively solving a small number of convex optimization problems. The solution to each of these problems can be written in closed form. In addition to their simplicity, we show through extensive experimentation that the features learned by RFMs and deep networks are closely related. As an example, in Fig. 1c, we show that the features learned by RFMs and deep networks for prediction tasks from the CelebA dataset [40] are highly correlated (Pearson correlation of 0.999 for the top eigenvector of the feature matrix). Moreover, as shown in Fig. 1, training RFMs on these learned features leads to improved predictive performance over deep neural networks.

2 Neural Feature Learning and Recursive Feature Machines

We provide our main insight connecting feature learning in neural networks to the average gradient outer product. This insight leads naturally to the development of recursive feature machines, which are a class of kernel machines that learn features. We begin by referencing the definition of average gradient outer product from [68] below.[4]

Average Gradient Outer Product. Given a function $g : \mathbb{R}^d \to \mathbb{R}$ and samples $\{x_i\}_{i=1}^n \subset \mathbb{R}^d$, the *average gradient outer product* is defined as

$$\mathcal{G}(g) = \frac{1}{n}\sum_{i=1}^{n} \nabla g(x_i)\nabla g(x_i)^T \in \mathbb{R}^{d\times d} \quad \text{with} \quad \nabla g(x_i) := \nabla g(x)\big|_{x=x_i}.$$

[4]For simplicity, we consider the gradient outer product of real-valued functions, but the gradient outer product naturally extends to functions that have multi-dimensional output by considering the product of the Jacobian and its transpose.

The Feature Matrix. Let $W_1 \in \mathbb{R}^{k \times d}$ denote the first layer weights of a deep fully connected network. We define the *feature matrix* of the neural network f as $M_f \in \mathbb{R}^{d \times d}$, which has the form

$$M_f = {W_1}^T W_1.$$

Let $f^{(t)}$ denote the predictor learned by the network after t steps of gradient descent. Our main insight, which we call the Neural Feature Ansatz, is that

$$M_{f^{(t)}} \text{ and } \mathcal{G}(f^{(t)}) \text{ capture similar sets of features.} \qquad \text{(Neural Feature Ansatz)}$$

Before describing the evidence for this claim, both theoretical and empirical, we first discuss key implications. Assuming that $f^{(t)}$ generalizes well, i.e., $f^{(t)}$ is close to the optimal predictor f^*, the Neural Feature Ansatz implies that $M_{f^{(t)}}$ is close to $\mathcal{G}(f^*)$. This, in-turn, suggests that features learned by estimating $\mathcal{G}(f^*)$ should closely match those learned by deep networks. This observation implies that the neural network learns features and a predictor simultaneously within the training process.

While the joint learning of features and the predictor from these features is a remarkable aspect of neural networks, there is no reason to believe that such simultaneous learning performed by neural networks is optimal. A more direct approach is to use an iterative two-step strategy that alternates between estimating the feature matrix and the predictor. Namely, given a candidate estimator $\widehat{f}$, we can directly compute the corresponding feature matrix $\mathcal{G}(\widehat{f})$ and then refine the estimator based on the newly learned features, iteratively. When this strategy is used in conjunction with kernel machines, we will refer to it as a recursive feature machine (RFM). We note this approach is closely related to the methodology developed in [68], which uses nearest neighbor methods for estimating predictors and features.

We now outline evidence for our main claim connecting the matrices $M_{f^{(t)}}$ and $\mathcal{G}(f^{(t)})$ for neural networks. Our two main lines of evidence are theoretical and empirical and are summarized below.

Theoretical Evidence. We provide a series of settings in Appendix A under which we prove that $M_{f^{(t)}} = \mathcal{G}(f^{(t)})$. In particular, we establish this result for one hidden layer fully connected networks where only the first layer is trained and for deep nonlinear networks after one step of gradient descent. We further establish this connection for two steps of gradient descent in one-hidden layer linear networks where both layers are trained.[5] Our analysis proceeds by taking infinite width limits of networks after finitely many steps of gradient descent. Moreover, our analysis holds for networks initialized near zero for which features naturally emerge from learning.

Empirical Evidence. Our main line of empirical evidence is that the features learned by RFMs are highly correlated with those captured by the feature matrix of trained neural networks. Additionally, the top eigenvectors of these matrices are essentially identical (Pearson correlation greater than 0.99). Furthermore, we show that predictors built by using our RFM features match or surpass performance of trained neural networks. Since the neural network feature matrix depends only on the first layer, this suggests that only the first layer of fully connected neural networks learns features relevant for prediction.

RFMs: a class of kernel machines that learn features. To alternate between estimating a predictor and learning features, we turn to a simple, effective, and well-studied class of models known as kernel machines [57]. Intuitively, training a kernel machine involves simply solving linear regression after applying a feature transformation on the data. To avoid working with explicit feature transformations, kernel machines are implemented using a kernel function, which is a positive semi-definite, symmetric function of the form $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ [1]. Given a kernel function and a dataset $(X, y) \in \mathbb{R}^{d \times n} \times \mathbb{R}^{1 \times n}$, training a kernel machine gives a predictor, $\widehat{f}$, of the form[6]

$$\widehat{f}(x) = \widehat{\alpha} K(X, x) \qquad ; \qquad \widehat{\alpha} = y K(X, X)^{-1} \; ;$$

[5] We note the difficulty in a more general analysis is the non-independence of layers and nesting of non-linear derivatives after multiple steps of training.

[6] When $K(X, X)$ is not invertible, we use the Moore-Penrose pseudoinverse, $K(X, X)^\dagger$, instead of $K(X, X)^{-1}$.

where $K(X,x) \in \mathbb{R}^n$ with $(K(X,x))_j = K(x_j,x)$ and $K(X,X) \in \mathbb{R}^{n\times n}$ with $(K(X,X))_{ij} = K(x_i,x_j)$. Further background on kernels including training kernel machines with ridge regularization is provided in Appendix B.

To develop kernel machines that learn features, we incorporate a positive semi-definite, symmetric matrix, M, as a learnable parameter into the kernel function. Namely, for kernel functions that depend on the distance between points,[7] e.g., $K(x,z) = \phi(\|x-z\|_2)$ for $\phi : \mathbb{R} \to \mathbb{R}$ and $x, z \in \mathbb{R}^d$, we incorporate the feature matrix by using the Mahanolobis distance[8]

$$\|x-z\|_M^2 := (x-z)^T M(x-z).$$

Thus, we consider kernels of the form

$$K_M(x,z) := \phi(\|x-z\|_M).$$

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 model trained using this procedure as an RFM and present the general algorithm in Algorithm 1.

Algorithm 1 RFM

Input: X, y, K, T ▷ Training data, X, training features y, kernel function, K, and number of iterations, T
Output: α, M ▷ Solution to kernel regression, α, and feature matrix, M

```
M = I_{d×d}                                  ▷ Initialize M to be the identity matrix
for t ∈ T do
    K_train = K_M(X, X)
    α = y K_train^{-1}
    M = (1/n) Σ_{x∈X} ∇f(x) ∇f(x)^T           ▷ Average gradient outer product of f(x) = α K_M(X, x)
end for
```

Note that this algorithm allows for a number of variations. One variation of RFMs that we found useful is thresholded RFMs (RFM-T), which involve re-training RFMs after thresholding to the top diagonal entries of M. We point out that training RFMs involves solving a small number of successive convex optimization problems whose solutions, furthermore, can be written in closed form. This is in contrast to the complexity of training deep neural networks, which involves non-convex optimization.

While RFMs can be implemented with any kernel function, we mainly utilize the Laplace kernel due to its simplicity and empirical effectiveness.[9] The Laplace kernel with bandwidth parameter L is defined as

$$K_M(x,z) = \exp\left(-\frac{\|x-z\|_M}{L}\right),$$

and its gradient[10] has the form

$$\nabla_z K_M(x,z) = \frac{Mx - Mz}{L\|x-z\|_M} K_M(x,z).$$

We provide a memory efficient, vectorized implementation of gradient computation for the Laplace kernel in our code.

[7] We note that a similar modification can be made for general kernels by considering kernels $K_M(x,z) = K(M^{\frac{1}{2}}x, M^{\frac{1}{2}}z)$.

[8] We note that in statistical literature this distance is given by $d_M(x,z) = \sqrt{(x-z)^T M^{-1}(x-z)}$ [44], but here, we make use of the notation from metric learning literature [9], which omits the inverse.

[9] We note also recent connection between Laplace kernels and the NTK [11, 21].

[10] While the Laplace kernel is not differentiable, when the inputs to the kernel function are identical, we replace the gradient with zero at such points. This is equivalent to smoothing the kernel at this point and is analogous to how deep network practitioners handle the non-differentiability of the rectified linear unit (ReLU) activation.

(a) Top Eigenvector of Feature Matrices for CelebA Classification Tasks

Task	Glasses	Mustache	Goatee	Hat	Blonde	Male
Deep Network Feature Matrix: $W_1^T W_1$						
RFM Feature Matrix: $\sum_{i=1}^{n} \nabla f(x_i) \nabla f(x_i)^T$						
Correlation	0.999	0.999	0.999	0.995	0.997	0.999

(b) Diagonals of Feature Matrices for CelebA Classification Tasks

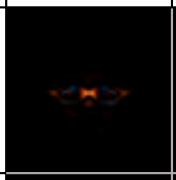
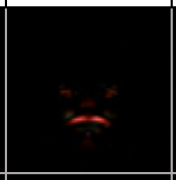
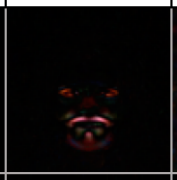
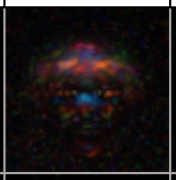
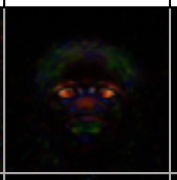
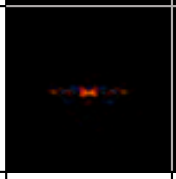
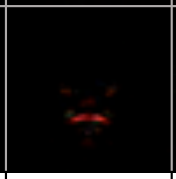
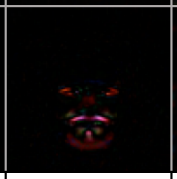
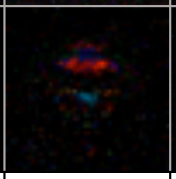
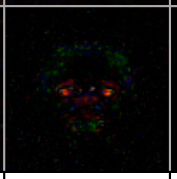
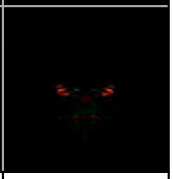
Task	Glasses	Mustache	Goatee	Hat	Blonde	Male
Deep Network Feature Matrix: $W_1^T W_1$						
RFM Feature Matrix: $\sum_{i=1}^{n} \nabla f(x_i) \nabla f(x_i)^T$						
Correlation	0.947	0.982	0.949	0.817	0.815	0.965

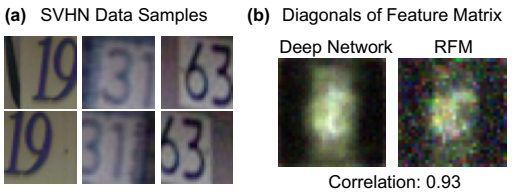
Figure 2: Comparing the feature matrices from RFMs and deep fully connected neural networks (denoted Deep Network) on CelebA prediction tasks. (a) The top eigenvector of the feature matrices from RFMs and deep networks visually highlights features related to the predictive task. These top eigenvectors are highly correlated (Pearson correlation greater than 0.99). (b) The diagonals of feature matrices from RFMs and deep networks highlight a sparse subset of features related to the predictive task. These diagonals are highly correlated (Pearson correlation greater than 0.8). Note that such high correlation is not due to sparsity since for example, the correlation between the diagonals for glasses and mustache is only 0.110.

3 Empirical Results

We now present empirical evidence demonstrating that

1. Features learned by deep fully connected networks are accurately captured by RFMs (Section 3.1).
2. On real-world and synthetic datasets where feature learning is important, RFMs close the gap between kernel machines and neural networks (Section 3.2).
3. RFMs provide state-of-the-art results on large tabular datasets and in particular, outperform a spectrum of models including modern neural networks such as transformers and ResNets on a broad set of classification and regression tasks (Section 3.3).

All datasets, models, and training procedures are detailed in Appendix C.



(c) Performance Comparison

Model	Test Accuracy	Test MSE
Laplace Kernel	74.70%	0.046
NTK	79.39%	0.049
Deep Network	84.39%	0.039
RFM (NTK)	**84.68%**	**0.038**

Figure 3: RFMs close the gap between kernel machines and deep fully connected networks on SVHN. (a) Samples from the SVHN dataset in which the goal is to identify the center digit from a view of potentitally many digits. (b) Upon visualizing the diagonals of the feature matrices of RFMs and deep networks, we observe that these models learn to select the center digit. (c) 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. Using the NTK of a 2-hidden layer network on features learned by RFMs (denote RFM (NTK)) leads to improved test accuracy over deep neural networks.

3.1 RFMs accurately capture features learned by fully connected networks.

We present a striking series of examples illustrating the ability of RFMs to capture features from neural networks by comparing the features learned when using these models to predict attributes of celebrities from the CelebA dataset [40]. This dataset, commonly used for computer vision tasks, contains over 150,000 images of celebrity faces with each image labelled with up to 40 different attributes such as whether the individual is wearing glasses, smiling, has a mustache, etc. A description of the dataset and pre-processing methodology used in our experiments is provided in Appendix C. To avoid computational bottlenecks in comparing feature matrices of full resolution CelebA images, we downsample the images to have 96×96 resolution for our experiments.

In Fig. 2a and b, we visualize and compare the features matrices learned by fully connected networks and RFMs on a subset of six of predictive tasks in CelebA. In particular, in Fig. 2a, we visualize the top eigenvector of the feature matrix from deep neural networks and RFMs, and in Fig. 2b, we visualize the diagonals of these two feature matrices. Overall, we observe striking similarity between these learned features, which is corroborated by a high Pearson correlation (greater than 0.8) between the two sets of features. We note that the high correlation between the diagonals of the feature matrices is not due to the sparsity of the features. For example, the correlation between the diagonals of the feature matrices for glasses and mustache classification is 0.110. Top three eigenvectors and corresponding eigenvalues are presented in Appendix Fig. 10. In general, we observe that there is a large gap between the top eigenvalue and the second largest eigenvalue for both deep networks and feature matrices. This large gap and the similarity in the top eigenvector for deep network and RFM feature matrices further indicates that RFMs are accurately capturing features learned by deep networks.

3.2 RFMs close the gap between kernels and fully connected networks.

In addition to accurately capturing features learned by fully connected networks, RFMs surpass previous kernel methods in predictive performance and match or outperform neural networks in predictive performance. As an initial illustrative example, we consider the task of classifying street view house numbers (SVHN) [46]. As the SVHN classification task involves predicting the center digit in an image containing possibly multiple digits (see Fig. 3a), this task is naturally conducive to feature learning. Indeed, both deep networks and RFMs learn to subset to the center columns of the image as is shown by visualizing the diagonals of the feature matrix in Fig. 3b. Moreover, by learning to select this central region, RFMs and deep networks far outperform previous kernel methods with a gap of over 10% between these models and the classical Laplace kernel (see Fig. 2d). In addition to the real-world example of SVHN classification, RFMs additionally match or outperform neural networks on other illustrative tasks, including low rank polynomial regression identified by previous works [16, 73], where feature learning is important (see Appendix Fig. 11). In such low-rank polynomial settings, we also prove that transforming the data with the expected gradient outer product obtains an improvement in sample complexity (see Appendix E).

(a) Performance across all 121 classification datasets from Fernández-Delgado et al. 2014

Classifier	Friedman Rank	Average Accuracy	P90	P95	PMA
RFM	**19.36**	**85.21%**	**94.21%**	**80.99%**	**97.20% ± 3.77%**
Laplace Ridge Regression	28.48	83.76%	90.08%	74.38%	95.95% ± 5.41%
Random Forest*	33.52	81.96%	83.47%	68.60%	93.48% ± 12.10%
NTK Ridge Regression	33.55	82.70%	85.95%	68.60%	94.84% ± 8.17%
Gaussian SVM	37.50	81.81%	82.35%	69.75%	93.21% ± 11.37%

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

(b) Performance across the subset of 90 "small" classification datasets from Fernández-Delgado et al. 2014

Classifier	Friedman Rank	Average Accuracy	P90	P95	PMA
RFM	**18.43**	**82.83%**	**92.22%**	**80.00%**	**96.85% ± 3.96%**
H-γ-exp. SVM [Geifman et al. 2020]	26.26	82.25%	**92.22%**	73.33%	96.07% ± 4.83%
NTK SVM [Arora et al. 2019]	28.34	81.95%	88.89%	72.22%	95.72% ± 5.17%
Laplace SVM [Geifman et al. 2020]	32.98	81.80%	86.67%	65.56%	94.88% ± 6.85%
Random Forest [Arora et al. 2019]	33.51	81.56%	85.56%	67.78%	95.25% ± 5.30%
NN [Arora et al. 2019]	38.06	81.02%	85.56%	60.00%	94.55% ± 5.17%

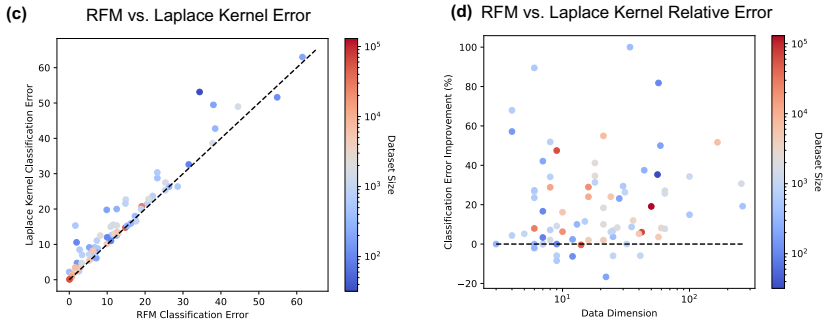


Figure 4: RFMs surpass a broad spectrum of models including NTKs, random forests, and fully connected neural networks on tabular datasets from [17] under several standard performance metrics. All metrics, models, and training details are outlined in Appendices C and D. (a) RFMs surpass NTKs, Laplace kernels, and 179 models on the 121 datasetst from [17]. (b) RFMs outperform NTKs [3] and variants of the Laplace kernel (H-γ-exp. SVM) on a subset of 90 classification tasks from [17] containing at most 5000 training examples. (c) We compare the error rate (100% - accuracy) between RFMs and Laplace kernels, which are equivalent to RFMs when the feature matrix is the identity matrix. We observe that Laplace kernels generally result in high error than RFMs. (d) We report the relative decrease in error between RFMs and Laplace kernels, i.e., $100 \cdot \frac{\text{Error Laplace} - \text{Error RFM}}{\text{Error Laplace}}$, as a function of data dimension and dataset size. RFMs provide an improvement in error for large datasets with high data dimension.

3.3 RFMs outperform neural networks and NTK on tabular data benchmarks.

We now demonstrate the remarkable effectiveness of RFMs in comparison with neural networks, NTK, and over 180 other classification methods on two tabular data benchmarks [17, 23]. Overall, we observe that RFMs achieve the best performance on these benchmarks across a variety of commonly used performance metrics.

Tabular Benchmark 1 [17]. The first benchmark we consider is from [17], which compares the performance of 179 different machine learning methods on 121 tabular classification tasks. Recent works [3, 21] demonstrated that NTKs and variants of Laplace kernels (H-γ-exponential kernels) outperformed on a subset of 90 small datasets from the 121 that contained at most 5000 training examples. We will now demonstrate that RFMs outperform NTKs, variants of Laplace kernels, and all 179 other methods from [17] on the subset of 90 small datasets and the full 121 datasets from the original benchmark. In our analysis, we compare performance according to the following metrics used in [17] and in [3]:

- Friedman rank: The average rank of the classifier across all datasets.

(a) Accuracy across classification datasets from Grinsztajin et al. 2022

Dataset	FT Transformer	Gradient Boosting Tree	Resnet	SAINT	XG Boost	RFM
MiniBooNE	79.74%	79.47%	78.59%	79.77%	79.56%	**79.83%**
HIGGS	**73.13%**	72.55%	72.39%	72.75%	72.87%	72.19%
Covertype	94.56%	94.14%	94.61%	94.32%	94.47%	**94.72%**
Jannis	90.69%	89.76%	89.39%	89.59%	89.74%	**93.78%**
ADTM	0.739	0.292	0.256	0.478	0.539	**0.750**

(b) R^2 across regression datasets from Grinsztajin et al. 2022

Dataset	FT Transformer	Gradient Boosting Tree	Resnet	SAINT	XG Boost	RFM
Diamonds	0.945	0.947	0.941	0.945	0.948	**0.949**
NYC Taxi Green Dec. 2016	0.120	0.624	0.247	0.534	**0.629**	0.569
Year	0.117	0.307	0.119	0.289	0.307	**0.334**
ADTM	0.182	0.898	0.086	0.687	0.926	**0.961**

Figure 5: RFMs generally outperform modern neural networks (transformers and ResNets) and modern tree-based models (gradient boosted trees and XGBoost) on large tabular tasks from [23]. Results from all models other than RFMs are reported from the tables provided by [23]. All metrics and training details are outlined in Appendices C and D. Tasks have 50000 training examples except for Jannis (40306 examples) and Diamonds (37758 examples). RFMs achieve the highest average distance to minimum (ADTM) on (a) large classification tasks and (b) large regression tasks.

- 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.

In Fig. 4a and b, we observe that RFMs outperform all previous methods across all metrics on both the subset of 90 small datasets and all 121 tasks. Moreover, we importantly note that while some of the datasets of contain up to 130000 training examples, RFMs are computationally fast to train through the use of fast linear system solvers such as EigenPro [42, 43].

In Fig. 4c, we compare the difference in error (100% - accuracy) between RFMs and the classical Laplace kernel, which is equivalent to an RFM where the feature matrix M is the identity matrix. In particular, we observe that the Laplace kernel generally results in high error than the RFM for larger datasets. In Fig. 4d, we report the relative decrease in error between RFMs and the Laplace kernel. We observe sizable improvement in relative error for larger datasets and those with higher dimensional data.

Tabular Benchmark 2 [23]. The second tabular benchmark we consider is from [23], which compares the performance of modern neural networks such as transformers [55, 65] against recent advances in tree-based models such as gradient boosted trees [12]. While this work has fewer "large" tabular datasets (those with greater than 10000 training samples), we again observe that RFMs generally outperform these modern tree-based models and neural networks.

In Fig. 5a and b, we compare RFMs to two transformer models [55, 65], ResNet [26], and two gradient boosting tree models [12, 49] across all "large" classification and regression tasks considered in [23]. Since there are only seven total large tasks in this benchmark, we report the accuracy on each task individually. Following [23], we additionally report the average distance to the minimum (ADTM). This metric accounts for varying dataset difficulty by normalizing the test accuracy for each task between 0 and 1 via min-max scaling across classifier performance. We observe that RFMs achieve the highest ADTM on these large classification and regression tasks, again demonstrating their effectiveness on large tabular datasets.

In Appendix Tables 2 and 3, we note that RFMs outperform modern neural networks and are competitive with tree-based models on "medium" size tabular datasets (fewer than 10000 training examples) considered in [23].[11] We note that this is in accordance with intuition since we would expect feature learning to be particularly beneficial on larger datasets. In Tables 4 and 5, we present results for models on datasets that have categorically encoded data. Here, we again observe that RFMs generally provide an improvement over neural networks but are slightly outperformed by tree-based models on regression tasks.

[11] In these tables, missing entries are denoted with a $-$ and indicate that the result is not found in the logs provided by [23].

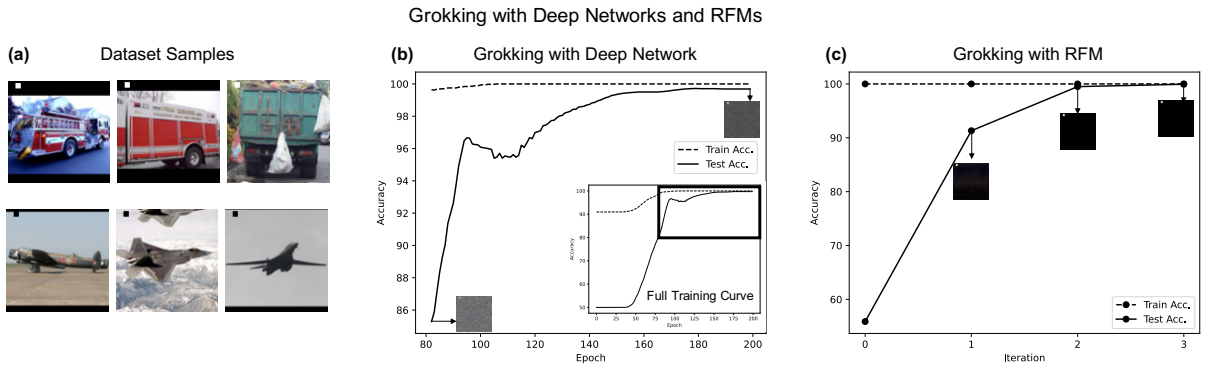


Figure 6: Grokking in fully connected networks and RFMs. (a) Corrupted samples from a subset of STL-10 in which a small 5×5 square in the upper left indicates whether the image is a truck or an airplane. (b) A deep fully connected network with width larger than number of training samples quickly reaches near 100% training accuracy. Yet, as training continues past this point, the test accuracy rises drastically from 85% to 99.68%. Corresponding feature matrices (shown as inserts) indicate that test accuracy improved since the network has learned the 5×5 pixels that give away the class label. (c) While training accuracy is always 100% at all iterations of the RFM, iteration leads to a drastic rise in test accuracy from 55.8% to 99.94%. Again, feature matrices (shown as inserts) indicate that test accuracy improved since the RFM has learned the 5×5 pixels that give away the class label.

4 Deep Learning Phenomena through the Prism of RFM

Empirical studies of deep neural networks have brought to light a spectrum of phenomena, which are absent in analytically tractable kernel methods. In this section, we show that such phenomena are reproducible in RFMs, and we envision that RFMs will provide a framework for rigorous analysis of these phenomena. In particular, we discuss four specific phenomena below.

- Grokking, Section 4.1.
- The Lottery Ticket Hypothesis, Section 4.2.
- Inductive Bias of Deep Networks, Section 4.3.
- Spurious Feature Selection in Deep Networks, Section 4.4.

4.1 Grokking in RFMs and deep networks.

Introduced in recent work [50], grokking refers to the phenomena in which deep networks can begin to exhibit a dramatic increase in test accuracy when training past the point where training accuracy is 100%. While the work of [50] demonstrated this phenomenon in transformers trained on small algorithmic datasets, we demonstrate that both RFMs and deep fully connected networks can exhibit behavior similar to grokking.

We showcase this phenomenon by training neural networks and RFMs to classify between a subset of 96×96 resolution images of airplanes and trucks from the STL-10 dataset [13] (training details for RFMs and deep networks are presented in Appendix C). 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 5×5 square of pixels in the upper left corner of each image that is colored white or black based on the class label (see Fig. 6a). The test set has no such class imbalance with 800 examples of each class.

Given the small size of the dataset, training only the last layer of a finite width networks with greater than 553 hidden units is sufficient for achieving 100% training accuracy. Indeed, as shown in Fig. 6b, we observe that the network rapidly approaches 100% training accuracy. Yet, when continuing to train the network, we observe that the test accuracy rises drastically from 85% to 99.68% in the last 120 epochs of

Lottery Tickets and RFMs

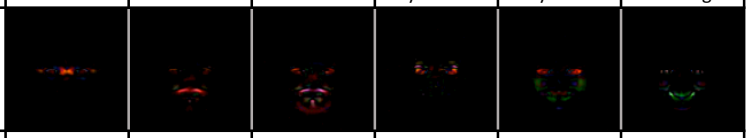
Task	Glasses	Mustache	Goatee	Eyebrows	Rosy Cheeks	Smiling
Thresholded RFM Features						
Percent of Pixels Remaining	1.42%	1.66%	3.40%	1.44%	2.43%	1.76%
RFM Accuracy	90.92%	88.15%	89.40%	74.36%	86.66%	89.62%
RFM-T Accuracy	**94.06%**	**91.32%**	**91.19%**	**78.11%**	**88.72%**	**91.24%**

Figure 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, by re-training RFMs upon thresholding to less than 3.5% of total pixels in CelebA tasks, performance consistently improves for these tasks.

training. This is due to the fact that the network learns the feature corresponding to the 5×5 pixel during this stage of training (see inserts in Fig. 6b).

On the other hand, Fig. 6c demonstrates that RFMs also exhibit this phenomenon. Indeed, we find that kernel regression with the Laplace kernel (RFM at iteration 0) achieves 100% training accuracy but only 55.8% test accuracy since the feature matrix is the identity matrix. Yet, upon training the RFM for more iterations, we find that training accuracy remains at 100% but test accuracy rises to 99.94%. This is again since the RFM subsets the 5×5 square of pixels through more iterations (see inserts Fig. 6c).

4.2 Lottery tickets in RFMs and deep networks.

Introduced in [20], 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. 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. 2b.

In line with the lottery ticket hypothesis, we demonstrate that retraining an RFM after thresholding coordinates of the data corresponding to these sparse regions in the feature matrix leads to a consistent increase in performance in many settings. In Fig. 7, we prune over 95% of pixels in CelebA images according to the features identified by RFMs and indeed, observe a consistent increase in predictive performance upon retraining a kernel on the thresholded features. We note that such thresholding drastically reduces

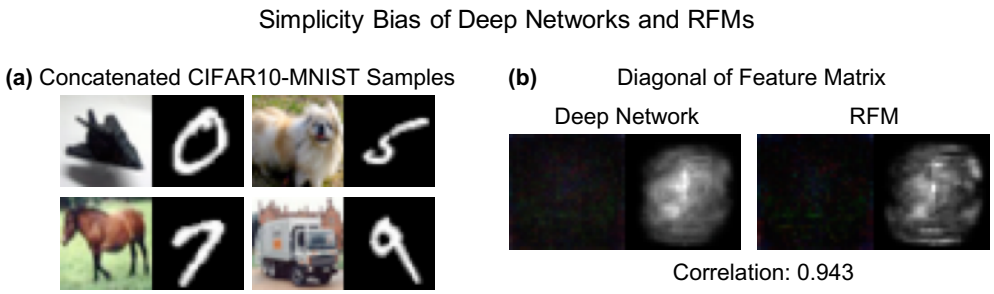


Figure 8: RFMs accurately capture inductive biases and in particular, simplicity biases of deep fully connected networks. (a) We train RFMs and neural networks on a dataset similar to that from [59] in which we concatenate images of CIFAR-10 objects with unique digits from MNIST. (b) Upon visualizing the diagonals of the feature matrices of trained deep networks and RFMs, we observe that both models learn to mask the CIFAR-10 image and focus on the MNIST digit for prediction.

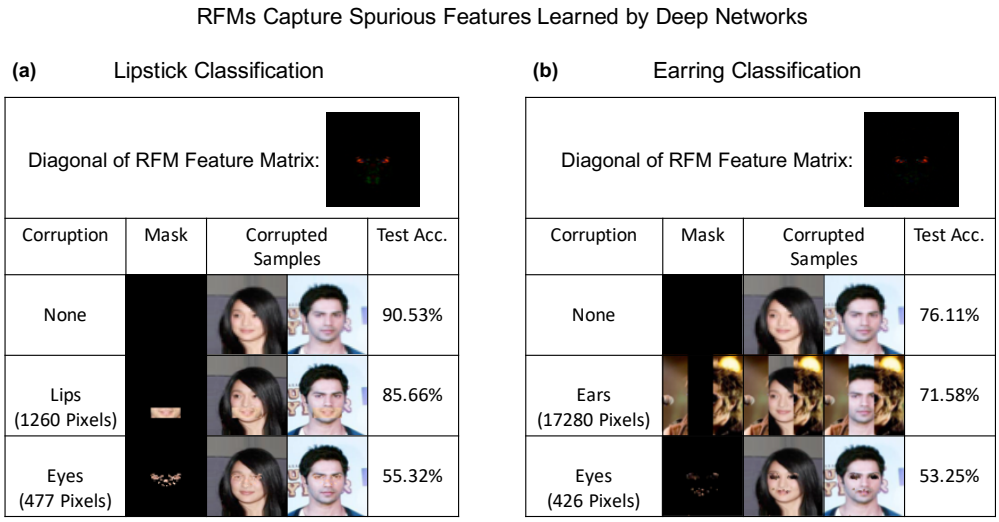


Figure 9: Feature matrices of RFMs capture spurious features learned by deep networks. (a, b) The diagonal of the feature matrix for an RFM trained on lipstick (or earring) classification unusually indicates that eyes are used as a key feature. We thus construct a mask based on the top RFM features and replace the eyes of all test samples with those of a single individual. A neural network trained on lipstick (or earring) classification does 35% (or 23%) worse on these corrupted samples. On the other hand, replacing the lips (or ears) of all test samples with those from the same individual leads to only a minor, 5%, decrease in accuracy.

computational and memory expenditure for the predictor both at test and run-time. Details regarding pruning thresholds are provided in Appendix C.

4.3 Inductive biases of RFMs and deep networks.

A recent trend in deep learning is to use neural networks that are over-parameterized, i.e., capable of perfectly fitting training data. Despite their over-parameterization, such models yield improved performance [77]. To understand why such over-parameterized models generalize, recent works analyzed their inductive biases, i.e., properties of the predictor learned through training [4, 32, 47, 71]. Recent works [28, 59] have identified a particular form of inductive bias in deep networks, referred to as simplicity bias, where deep networks rely on simplest available features for classification.

As an example, consider the experimental setup similar to that from [59] where we train a deep fully connected network on concatenated images of CIFAR-10 [34] and MNIST [35] with each CIFAR-10 class paired with a unique MNIST digit (see Fig. 8a). The work [59] showed through perturbative analyses that that neural networks trained on such a dataset relied only on the MNIST half of the concatenated image to make predictions, as, presumably, MNIST digits are "simpler" than CIFAR images. We show that the feature matrices of neural networks and RFMs both accurately capture such simplicity biases. In Fig. 8b, we visualize the diagonals of the feature matrices for the deep network and RFM. Both feature matrices indicate that these models are indeed learning to mask coordinates related to CIFAR-10 images and instead, focus on the MNIST digit for classification.

4.4 RFMs capture spurious features and biases in deep networks.

Recent works [29, 62] demonstrate that deep networks can leverage spurious features for prediction, which can make these models unreliable in mission-critical domains. In image classification, such features are those correlated with but not part of objects of interest. Examples include food co-occurring with plates or fingers co-occurring with band-aids [62]. It has also been argued [29] that such features are responsible for so-called "adversarial examples" [66], where small, often visually imperceptible alterations of images lead to dramatic loss of accuracy.

We now demonstrate that RFMs can serve as a useful tool for identifying spurious features used by deep networks during prediction. In particular, we showcase how such feature matrices can be used to identify spurious features leveraged by deep networks in CelebA lipstick and earring prediction tasks. In Fig. 9a, we observe that the diagonal of the RFM feature matrix indicates that eyes are unusually identified as a key feature for lipstick classification. Since eyes appear to be a spurious feature in RFMs and RFMs accurately capture neural network features, applying a small perturbation on these identified eye coordinates should lead to a large decrease in a network trained to predict lipstick. Indeed, we demonstrate this drop in accuracy in Fig. 9a. By replacing the eyes of all test images with those of a specific male, we see that the lipstick classification accuracy of a deep network drops from 90.53% to 55.23%. On the other hand, replacing the lips of all test images with those of the same male leads to only a small 4.87% drop in accuracy. We demonstrate a similar phenomenon for earring classification in Fig. 9b in which RFMs also identify eyes as a key feature for earring classification.

5 Summary, Discussion, and Outlook

Summary of results. In this work, we isolated a key mechanism of feature learning in deep fully connect neural networks. In particular, we posited the Neural Feature Ansatz – that the first layer of neural networks is responsible for feature learning and the learned features are closely related to those given by the average gradient outer product. We provided evidence, both empirical and theoretical, for this claim. We subsequently leveraged the connection with the average gradient outer product to develop a simple class of kernel machines, Recursive Feature Machines (RFMs), that learn features. We demonstrated that RFMs are able to (1) accurately capture features learned by deep fully connected neural networks, (2) close the gap between kernel methods and fully connected networks, and (3) yield state-of-the-art performance on large tabular datasets. Finally, we demonstrated that many intriguing phenomena of deep learning can be understood through the prism of our algorithm, as manifestations of alternate feature selection and predictor construction.

Comments, connections and consequences. We now discuss connections between our results and machine learning literature as well as some implications of our results.

- *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 [30, 39]. 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 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, RFMs are able to circumvent such modelling complexity without sacrificing performance.

- *The role of depth and network architecture in feature learning.* In this work, we connected feature learning in the first layer of deep fully connected networks with the average gradient outer product. Moreover, given that kernels trained on the RFM features match or surpass the performance of trained networks, we posited that feature learning happens largely in the first layer of deep networks; and that the remaining layers act as a kernel predictor over first layer features with deeper networks intuitively corresponding to "sharper", more localized kernels. Whether feature learning in higher levels is significant for prediction (as argued in a range of works, e.g., [76]), remains an open question.

- *Interpretability.* Interpretability has become a key area of focus in deep learning given the broad impact and ubiquity of deep networks. There is a rich literature on gradient-based methods for understanding features leveraged by deep networks for image classification tasks [58, 61, 76]. These methods utilize gradients of trained networks to identify patterns in a given datum that are used by deep networks for classification. Rather than consider features important for any given sample, we show that the RFM

feature matrix indicates the common subset of features used by fully connected networks on all samples. To highlight the impact of this finding, we identified that eyes were spurious features used by deep networks for classification of lipstick and earrings in the CelebA dataset (Fig. 9). We envision that the transparency provided by RFMs can serve as a key tool for increasing interpretability of machine learning models more generally.

- *Kernel learning.* Since the feature matrix and thus, the kernel function is updated during each iteration of training, RFMs can be viewed as a simple method for learning a data-dependent kernel. We note that there is a rich literature on learning kernels for a given task. For example, to improve kernel selection for supervised learning, a series of works consider selecting a combination of kernels to maximize alignment with the *target kernel matrix*, yy^T [14, 15, 63]. Recently, a line of works have studied the connection between kernel learning and neural networks through the time-dependent evolution of the NTK [19]. For example [5] showed that as a neural network is trained, the empirical NTK increases alignment with the target kernel matrix. An insightful work [41] showed that training kernel machines with the empirical NTK at the end of training, i.e., the *after kernel*, match the performance of the original network. 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.

- *Role of data in feature learning.* While we demonstrated that feature learning, both through RFMs and neural networks, can lead to large performance improvements on a variety of tasks, it need not always lead to such improvements. In fact, previous work [3, 51] showed that non-feature-learning kernel methods outperformed corresponding neural networks on a variety of tasks and in particular, those involving small datasets. Moreover, on the 121 tabular datasets from [17], we observed that the standard Laplace kernel (the non-feature learning, iteration 0 of RFMs) was chosen as the best performing model during cross-validation on 45 of the datasets. It is, of course, possible that the specific data adaptation method used in RFM may not be optimal, and other methods would lead to improvements over the baseline on these datasets. Thus, an important direction of research is understanding the properties of data under which feature learning provides an advantage.

- *Connections to other statistical and machine learning methods.* Our approach has connections to a number of classical methods from statistics and machine learning. Below we briefly touch upon some of these connections.

  — *Metric and manifold learning.* Updating the feature matrix in an RFM can also be viewed as learning a data-dependent Mahalonobis 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 [9]. We believe that RFMs can be extended and improved by incorporating ideas from the unsupervised and semi-supervised manifold learning and nonlinear dimensionality reduction literature [8, 54].

  — *FisherFaces and EigenFaces.* We further note the strong similarity between the eigenvectors of feature matrices (e.g., Figure 1) analyzed in this work and those given by EigenFace [64, 70], and FisherFace [7] algorithms. While EigenFaces are obtained in a purely unsupervised fashion, the FisherFace algorithm uses labeled images of faces and Fisher's Linear Discriminant [18] to learn a linear subspace for dimensionality reduction. RFMs also learn linear subspaces based on labeled data but in a recursive way, using nonlinear classifiers.

  — *Debiasing.* Debiasing [78] is a statistical procedure which has recently been popular in literature. Simply, given a high-dimensional problem with a hidden low-dimensional structure, a step of variable selection is performed by using methods such as Lasso [67] or sparse PCA [31]. A low-dimensional model is then fitted to the selected coordinates alone. We note that this procedure is parallel to a single step of RFM. Indeed, RFM 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 [45] 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 can be a productive research 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.

Looking forward. Neural networks trained by gradient-based optimization methods have remarkable properties, which are still not fully understood. One of their particularly interesting aspects is the ability to learn a predictor and features at the same time through training. Yet, there is no reason to believe that such simultaneous learning is optimal. Indeed, in this work, by decoupling feature learning and predictor learning, we constructed effective algorithms that outperform neural networks on a variety of generalization tasks and are also far easier to train.

In our view, separating predictor and feature learning components of neural networks provides a modular path forward leading both to algorithms that are easier to design, implement, and analyze theoretically, as well as to improvements of neural networks.

Acknowledgements

A.R. is supported by the Eric and Wendy Schmidt Center at the Broad Institute. A.R. thanks Caroline Uhler for support on this work, and A.R., M.B. thank her for many insightful discussions over the years. We are grateful for support from the National Science Foundation (NSF) and the Simons Foundation for the Collaboration on the Theoretical Foundations of Deep Learning[12] through awards DMS-2031883 and #814639 as well as the TILOS institute (NSF CCF-2112665). This work used the programs (1) XSEDE (Extreme science and engineering discovery environment) which is supported by NSF grant numbers ACI-1548562, and (2) ACCESS (Advanced cyberinfrastructure coordination ecosystem: services & support) which is supported by NSF grants numbers #2138259, #2138286, #2138307, #2137603, and #2138296. Specifically, we used the resources from SDSC Expanse GPU compute nodes, and NCSA Delta system, via allocations TG-CIS220009.

References

[1] N. Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950. 4

[2] 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. 3

[3] 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. 8, 14

[4] D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, et al. A closer look at memorization in deep networks. In *International Conference on Machine Learning*, 2017. 12

[5] A. Atanasov, B. Bordelon, and C. Pehlevan. Neural networks as kernel learners: The silent alignment effect. In *International Conference on Learning Representations*, 2022. 14

[6] 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. 1

[7] 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. 14

[8] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003. 14

[12] https://deepfoundations.ai/

[9] A. Bellet, A. Habrard, and M. Sebban. Metric learning. *Synthesis lectures on artificial intelligence and machine learning*, 9(1):1–151, 2015. 5, 14

[10] 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. 1

[11] L. Chen and S. Xu. Deep Neural Tangent Kernel and Laplace kernel have the same RKHS. In *International Conference on Learning Representations*, 2021. 5

[12] 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. 9

[13] 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. 10

[14] 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. 14

[15] N. Cristianini, J. Shawe-Taylor, A. Elisseeff, and J. Kandola. On kernel-target alignment. *Advances in Neural Information Processing Systems*, 14, 2001. 14

[16] 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. 1, 3, 7, 26, 31

[17] 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. 8, 14, 27, 28

[18] R. A. Fisher. The statistical utilization of multiple measurements. *Annals of eugenics*, 8(4):376–386, 1938. 14

[19] 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. 14

[20] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. 2, 11

[21] A. Geifman, A. Yadav, Y. Kasten, M. Galun, D. Jacobs, and B. Ronen. On the similarity between the laplace and neural tangent kernels. In *Advances in Neural Information Processing Systems*, 2020. 5, 8

[22] 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. 28, 29

[23] 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. 8, 9, 27

[24] B. Hanin and M. Nica. Finite depth and width corrections to the Neural Tangent Kernel. In *International Conference on Learning Representations*, 2020. 1

[25] 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. 2

[26] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, 2016. 9

[27] 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. 1

[28] 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. 12

[29] 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. 12

[30] A. Jacot, F. Gabriel, and C. Hongler. Neural Tangent Kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018. 1, 3, 13

[31] 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. 14

[32] 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. 12

[33] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. 27

[34] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009. 12

[35] 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. 12

[36] J. Lee, S. Schoenholz, J. Pennington, B. Adlam, L. Xiao, R. Novak, and J. Shol-Dickstein. Finite versus infinite neural networks: an empirical study. In *Advances in Neural Information Processing Systems*, 2020. 3

[37] 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. 1

[38] 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. 1

[39] C. Liu, L. Zhu, and M. Belkin. On the linearity of large non-linear models: when and why the tangent kernel is constant. *Advances in Neural Information Processing Systems*, 33:15954–15964, 2020. 13

[40] 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. 3, 7

[41] P. M. Long. Properties of the after kernel. *arXiv preprint arXiv:2105.10585*, 2021. 3, 14

[42] 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. 9

[43] S. Ma and M. Belkin. Kernel machines that adapt to GPUs for effective large batch training. In *Conference on Machine Learning and Systems*, 2019. 9, 26, 27

[44] P. C. Mahalanobis. On the generalized distance in statistics. In *Proceedings of the National Institute of Science of India*, 1936. 5

[45] T. K. Moon. The expectation-maximization algorithm. *IEEE Signal processing magazine*, 13(6):47–60, 1996. 14

[46] 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. 7

[47] B. Neyshabur, R. Tomioka, and N. Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. In *International Conference on Learning Representations Workshop Contribution*, 2015. 12

[48] 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 *Advances in Neural Information Processing Systems*, 2019. 26

[49] 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 (JMLR)*, 12:2825–2830, 2011. 9

[50] 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. 2, 10

[51] 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. 3, 14

[52] 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. 1

[53] 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. 1

[54] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000. 14

[55] I. Rubachev, A. Alekberov, Y. Gorishniy, and A. Babenko. Revisiting pretraining objectives for tabular deep learning. *arXiv preprint arXiv:2207.03208*, 2022. 9

[56] M. Sahraee-Ardakan, M. Emami, P. Pandit, S. Rangan, and A. K. Fletcher. Kernel methods and multi-layer perceptrons learn linear models in high dimensions. *arXiv preprint arXiv:2201.08082*, 2022. 3

[57] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002. 3, 4, 25

[58] 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. 13

[59] 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. 2, 11, 12

[60] 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. 1

[61] A. Shrikumar, P. Greenside, and A. Kundaje. Learning important features through propagating activation differences. In *International Conference on Machine Learning*, 2017. 13

[62] S. Singla and S. Feizi. Salient ImageNet: How to discover spurious features in deep learning? In *International Conference on Learning Representations*, 2022. 2, 12

[63] A. Sinha and J. C. Duchi. Learning kernels with random features. *Advances in Neural Information Processing Systems*, 29, 2016. 14

[64] L. Sirovich and M. Kirby. Low-dimensional procedure for the characterization of human faces. *JOSA A*, 4(3):519–524, 1987. 14

[65] G. Somepalli, M. Goldblum, A. Schwarzschild, C. B. Bruss, and T. Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021. 9

[66] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014. 12

[67] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 14

[68] S. Trivedi, J. Wang, S. Kpotufe, and G. Shakhnarovich. A consistent estimator of the expected gradient outerproduct. In *UAI*, pages 819–828, 2014. 2, 3, 4

[69] K. Tunyasuvunakool, J. Adler, Z. Wu, T. Green, M. Zielinski, A. Žídek, 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. 1

[70] 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. 14

[71] G. Valle-Perez, C. Q. Camargo, and A. A. Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. In *International Conference on Learning Representations*, 2019. 12

[72] 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. 26

[73] N. Vyas, Y. Bansal, and P. Nakkiran. Limitations of the ntk for understanding generalization in deep learning. *arXiv preprint arXiv:2206.10012*, 2022. 1, 7, 26, 31

[74] G. Wahba. *Spline models for observational data*. SIAM, 1990. 26

[75] G. Yang and E. J. Hu. Tensor Programs IV: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, 2021. 1, 3

[76] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014. 13

[77] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017. 12

[78] 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. 14

[79] L. Zhu, C. Liu, and M. Belkin. Transition to linearity of general neural networks with directed acyclic graph architecture. In *Advances in Neural Information Processing Systems*, 2022. 3

[80] L. Zhu, C. Liu, A. Radhakrishnan, and M. Belkin. Quadratic models for understanding neural network dynamics. *arXiv preprint arXiv:2205.11787*, 2022. 1

A Theoretical evidence connecting feature learning to expected gradient outer product

We now provide theoretical evidence connecting feature learning in neural networks to the expected gradient outer product. We consider several settings for neural networks for which the Gram matrix of the first layer is equal to the expected gradient outer product after gradient descent. An outline of our theoretical results is provided in Table 1.

Setting	Activation	Training	Steps	Depth	Outer layers	Initialization
Theorem 1	Linear	Standard	Any	2	i.i.d, Fixed	Zero
Proposition 1	Linear	Standard	2 steps	2	i.i.d, Trainable	Zero
Theorem 2	ReLU	Standard	Any	2	Constant, Fixed	i.i.d.
Theorem 3	ReLU	Standard	1 step	Any	i.i.d., Re-sampled	Zero
Theorem 4	ReLU	Zero output	Any	Any	i.i.d., Re-sampled	Any

Table 1: Settings for which we theoretically connect the expected gradient outer product and neural network feature matrix. *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.

We begin with the following theorem analyzing feature learning in two layer linear neural networks where only the first layer is trained.

Theorem 1 (Linear, Full GD)**.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ denote a two layer neural network of the form*

$$f(x) = A\frac{1}{\sqrt{k}}Bx;$$

where $A \in \mathbb{R}^{1\times k}, B \in \mathbb{R}^{k\times d}$. Consider the case where 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 $B^{(0)} = \mathbf{0}$,

$$\lim_{k\to\infty} {B^{(t)}}^T B^{(t)} = \lim_{k\to\infty} \nabla f^{(t)} \nabla {f^{(t)}}^T \ ;$$

where $\nabla f^{(t)}$ is the gradient of $f^{(t)}$.[13]

Proof of Theorem 1. The gradient descent updates proceed as follows:

$$B^{(t+1)} = B^{(t)} + \frac{\eta}{\sqrt{k}} {A^{(0)}}^T \left(y - A^{(0)}\frac{1}{\sqrt{k}}B^{(t)}X\right) X^T \ .$$

We provide a proof by induction. We begin with the base case with $t = 1$. The base case follows from the fact that $\lim_{k\to\infty} \frac{1}{k} A^{(0)} {A^{(0)}}^T = 1$ and $B^{(1)} = \frac{\eta}{\sqrt{k}} {A^{(0)}}^T yX^T$ and thus,

$$\begin{aligned}
\lim_{k\to\infty} {B^{(1)}}^T B^{(1)} &= \lim_{k\to\infty} \frac{\eta^2}{k} Xy^T A^{(0)} {A^{(0)}}^T yX^T = \eta^2 Xy^T yX^T \ ; \\
\lim_{k\to\infty} \nabla f_1 \nabla f_1^T &= \lim_{k\to\infty} {B^{(1)}}^T {A^{(0)}}^T \frac{1}{k} A^{(0)} B^{(1)} \\
&= \lim_{k\to\infty} \eta^2 Xy^T \left(A^{(0)}\frac{1}{k}{A^{(0)}}^T\right)\left(A^{(0)}\frac{1}{k}{A^{(0)}}^T\right) yX^T = \eta^2 Xy^T yX^T \ .
\end{aligned}$$

[13] Note that since $f^{(t)}$ is linear, the gradient is constant.

Thus, we now assume the inductive hypothesis that

$$\lim_{k\to\infty} {B^{(t)}}^T B^{(t)} = \lim_{k\to\infty} \nabla f^{(t)} \nabla {f^{(t)}}^T$$

and analyze the case for timestep $t+1$. We first have:

$$\begin{aligned}
{B^{(t+1)}}^T B^{(t+1)} &= \left[B^{(t)} + \frac{\eta}{\sqrt{k}} {A^{(0)}}^T \left(y - A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} X\right) X^T\right]^T \left[B^{(t)} + \frac{\eta}{\sqrt{k}} {A^{(0)}}^T \left(y - A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} X\right) X^T\right] \\
&= {B^{(t)}}^T B^{(t)} + {B^{(t)}}^T \frac{\eta}{\sqrt{k}} {A^{(0)}}^T y X^T - {B^{(t)}}^T \frac{\eta}{k} {A^{(0)}}^T A^{(0)} B^{(t)} X X^T \\
&\quad + \frac{\eta}{\sqrt{k}} X y^T A^{(0)} B^{(t)} + \frac{\eta^2}{k} X y^T A^{(0)} {A^{(0)}}^T y X^T - \frac{\eta^2}{k} X y^T A^{(0)} {A^{(0)}}^T A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} X X^T \\
&\quad - \frac{\eta}{k} X X^T {B^{(t)}}^T {A^{(0)}}^T A^{(0)} B^{(t)} - \frac{\eta^2}{k} X X^T {B^{(t)}}^T \frac{1}{\sqrt{k}} {A^{(0)}}^T A^{(0)} {A^{(0)}}^T y X^T \\
&\quad + \frac{\eta^2}{k^2} X X^T {B^{(t)}}^T {A^{(0)}}^T A^{(0)} {A^{(0)}}^T A^{(0)} B^{(t)} X X^T \ .
\end{aligned}$$

To simplify notation, we let

$$Z = \lim_{k\to\infty} A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} \qquad ; \qquad M = \lim_{k\to\infty} {B^{(t)}}^T B^{(t)},$$

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 $k \to \infty$, applying the inductive hypothesis and the fact that $\lim_{k\to\infty} \frac{1}{k} A^{(0)} {A^{(0)}}^T = 1$, we reduce the above to

$$\begin{aligned}
\lim_{k\to\infty} {B^{(t+1)}}^T B^{(t+1)} &= M + \eta Z^T y X^T - \eta M X X^T \\
&\quad + \eta X y^T Z + \eta^2 X y^T y X^T - \eta^2 X y^T Z X X^T \\
&\quad - \eta X X^T M - \eta^2 X X^T Z^T y X^T + \eta^2 X X^T M X X^T \ .
\end{aligned}$$

We will now show that $\lim_{k\to\infty} \nabla f^{(t+1)} \nabla {f^{(t+1)}}^T$ is of the same form. Namely, we have

$$\begin{aligned}
\nabla f^{(t+1)} \nabla {f^{(t+1)}}^T &= {B^{(t+1)}}^T {A^{(0)}}^T \frac{1}{k} A^{(0)} B^{(t+1)} \\
&= {B^{(t)}}^T {A^{(0)}}^T \frac{1}{k} A^{(0)} B^{(t)} + {B^{(t)}}^T {A^{(0)}}^T \frac{1}{k} A^{(0)} \frac{\eta}{\sqrt{k}} {A^{(0)}}^T y X^T \\
&\quad - {B^{(t)}}^T {A^{(0)}}^T \frac{1}{k} A^{(0)} \frac{\eta}{k} {A^{(0)}}^T A^{(0)} B^{(t)} X X^T \\
&\quad + \frac{\eta}{\sqrt{k}} X y^T A^{(0)} {A^{(0)}}^T \frac{1}{k} A^{(0)} B^{(t)} + \frac{\eta^2}{k} X y^T A^{(0)} {A^{(0)}}^T \frac{1}{k} A^{(0)} {A^{(0)}}^T y X^T \\
&\quad - \frac{\eta^2}{k} X y^T A^{(0)} {A^{(0)}}^T \frac{1}{k} A^{(0)} {A^{(0)}}^T A^{(0)} \frac{1}{\sqrt{k}} B^{(t)} X X^T \\
&\quad - \frac{\eta}{\sqrt{k}} X X^T {B^{(t)}}^T {A^{(0)}}^T A^{(0)} {A^{(0)}}^T \frac{1}{k} A^{(0)} B^{(t)} \\
&\quad - \frac{\eta^2}{k} X X^T {B^{(t)}}^T \frac{1}{\sqrt{k}} {A^{(0)}}^T A^{(0)} {A^{(0)}}^T \frac{1}{k} A^{(0)} {A^{(0)}}^T y X^T \\
&\quad + \frac{\eta^2}{k^2} X X^T {B^{(t)}}^T {A^{(0)}}^T A^{(0)} {A^{(0)}}^T \frac{1}{k} A^{(0)} {A^{(0)}}^T A^{(0)} B^{(t)} X X^T \ .
\end{aligned}$$

Now taking the limit as $k \to \infty$, we reduce the above to

$$\begin{aligned}
\lim_{k\to\infty} \nabla f^{(t+1)} \nabla {f^{(t+1)}}^T &= M + \eta Z^T y X^T - \eta M X X^T \\
&\quad + \eta X y^T Z + \eta^2 X y^T y X^T - \eta^2 X y^T Z X X^T \\
&\quad - \eta X X^T M - \eta^2 X X^T Z^T y X^T + \eta^2 X X^T M X X^T \ .
\end{aligned}$$

Hence, we conclude

$$\lim_{k\to\infty} {B^{(t+1)}}^T B^{(t+1)} = \lim_{k\to\infty} \nabla f^{(t+1)} \nabla {f^{(t+1)}}^T,$$

which completes the proof by induction. □

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 1. *Let $f : \mathbb{R}^d \to \mathbb{R}$ denote a two layer neural network of the form*

$$f(x) = A\frac{1}{\sqrt{k}}Bx;$$

where $A \in \mathbb{R}^{1\times k}, B \in \mathbb{R}^{k\times d}$. 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 $B^{(0)} = \mathbf{0}$,

$$\lim_{k\to\infty} {B^{(2)}}^T B^{(2)} = \lim_{k\to\infty} \nabla f^{(2)} \nabla {f^{(2)}}^T \ ;$$

where $\nabla f^{(2)}$ is the gradient of $f^{(2)}$.

Proof. We prove the statement directly. The gradient descent updates proceed as follows:

$$A^{(t+1)} = A^{(t)} + \frac{\eta}{\sqrt{k}}\left(y - A^{(t)}\frac{1}{\sqrt{k}}B^{(t)}X\right)X^T{B^{(t)}}^T \ ,$$
$$B^{(t+1)} = B^{(t)} + \frac{\eta}{\sqrt{k}}{A^{(t)}}^T\left(y - A^{(t)}\frac{1}{\sqrt{k}}B^{(t)}X\right)X^T \ .$$

Thus, after 1 step of gradient descent, we have

$$A^{(1)} = A^{(0)} \qquad ; \qquad B^{(1)} = \frac{\eta}{\sqrt{k}}{A^{(0)}}^T yX^T.$$

From the proof of Theorem 1, we have that

$$\lim_{k\to\infty} {B^{(1)}}^T B^{(1)} = \lim_{k\to\infty} {B^{(1)}}^T \frac{1}{k}{A^{(0)}}^T A^{(0)} B^{(1)} \ ,$$

and so, we define the matrix M to be:

$$M := \lim_{k\to\infty} {B^{(1)}}^T B^{(1)}.$$

Next, after 2 steps of gradient descent, we have:

$$\begin{aligned} A^{(2)} &= A^{(1)} + \frac{\eta}{\sqrt{k}}\left(y - A^{(1)}\frac{1}{\sqrt{k}}B^{(1)}X\right)X^T{B^{(1)}}^T \\ &= A^{(0)} + \frac{\eta}{\sqrt{k}}yX^T{B^{(1)}}^T - \frac{\eta}{k}A^{(0)}B^{(1)}XX^T{B^{(1)}}^T \\ &= A^{(0)} + \frac{\eta^2}{k}yX^TXy^TA^{(0)} - \frac{\eta^3}{k^2}A^{(0)}{A^{(0)}}^T yX^TXX^TXy^TA^{(0)}; \end{aligned}$$

and

$$\begin{aligned} B^{(2)} &= B^{(1)} + \frac{\eta}{\sqrt{k}}{A^{(1)}}^T\left(y - A^{(1)}\frac{1}{\sqrt{k}}B^{(1)}X\right)X^T \\ &= \frac{2\eta}{\sqrt{k}}{A^{(0)}}^T yX^T - \frac{\eta}{k}{A^{(0)}}^T A^{(0)}B^{(1)}XX^T \\ &= \frac{2\eta}{\sqrt{k}}B^{(1)} - \frac{\eta}{k}{A^{(0)}}^T A^{(0)}B^{(1)}XX^T \end{aligned}$$

Thus, we simplify ${B^{(2)}}^T B^{(2)}$ as follows:

$$
\begin{aligned}
{B^{(2)}}^T B^{(2)} = \frac{4\eta^2}{k} {B^{(1)}}^T B^{(1)} - \frac{2\eta^2}{k\sqrt{k}} {B^{(1)}}^T {A^{(0)}}^T A^{(0)} B^{(1)} X X^T \\
- \frac{2\eta^2}{k\sqrt{k}} X X^T {B^{(1)}}^T {A^{(0)}}^T A^{(0)} B^{(1)} + \frac{\eta^2}{k^2} X X^T {B^{(1)}}^T {A^{(0)}}^T A^{(0)} {A^{(0)}}^T A^{(0)} B^{(1)} X X^T \ .
\end{aligned}
$$

Taking the limit as $k \to \infty$, we simplify the above expression to

$$
\lim_{k\to\infty} {B^{(2)}}^T B^{(2)} = 4\eta^2 M + \eta^2 X X^T M X X^T \ .
$$

A key observation is that as $k \to \infty$, 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 $\lim_{k\to\infty} \nabla f^{(2)} \nabla {f^{(2)}}^T$ 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 $\lim_{k\to\infty} \nabla f^{(2)} \nabla {f^{(2)}}^T$. Using this observation, we have:

$$
\lim_{k\to\infty} \nabla f^{(2)} \nabla {f^{(2)}}^T = \lim_{k\to\infty} {B^{(2)}}^T \frac{1}{k} {A^{(2)}}^T A^{(2)} B^{(2)} = \lim_{k\to\infty} {B^{(2)}}^T \frac{1}{k} {A^{(0)}}^T A^{(0)} B^{(2)},
$$

which by the expansion of ${B^{(2)}}^T B^{(2)}$ and the proof of Theorem 1, is equivalent to $4\eta^2 M + \eta^2 X X^T M X X^T$. □

We now analyze the case of nonlinear two layer neural networks in which only the first layer is trained under a constant initialization on the last layer weights.

Theorem 2 (Non-linear, full GD)**.** *Let $f(x) = A\phi(Bx)$ where $A \in \mathbb{R}^{1\times k}$, $B \in \mathbb{R}^{k\times d}$, and ϕ is the ReLU activation. Let $f^{(t)}$ denote f after t steps of gradient descent on the dataset $(X, y) \subset \mathbb{R}^{d\times n} \times \mathbb{R}^{1\times n}$. Assume $A^{(0)} = \mathbf{1}$ and is fixed through training. Let $B_{i,j}^{(0)} \overset{i.i.d}{\sim} \mathcal{N}(0, 1)$. Let $g(x) = \phi(w^T x)$ and let $g^{(t)}$ denote g after t steps of steps of gradient descent on the dataset (X, y) with $w^{(0)} \sim \mathcal{N}(\mathbf{0}, I_d)$. For $z \sim \mathcal{N}(\mathbf{0}, I_d)$,*

$$
\lim_{k\to\infty} \frac{1}{k} {B^{(t)}}^T B^{(t)} = 2\mathbb{E}_{z, w^{(0)}} \left[\left(\nabla g^{(t)}(z) \right) \left(\nabla g^{(t)}(z) \right)^T \right];
$$

where $\nabla g^{(t)}(z) = \nabla g^{(t)}(x)\big|_{x=z}$.

Proof of Theorem 2. Let $f_i(x) := A_i \phi(B_{i,:} x)$. After t steps of gradient descent

$$
\nabla f_i^{(t)}(z) = A_i \phi'(B_{i,:}^{(t)} z) {B_{i,:}^{(t)}}^T = \phi'(B_{i,:}^{(t)} z) {B_{i,:}^{(t)}}^T \ .
$$

Now,

$$
\nabla f_i^{(t)}(z) \nabla f_i^{(t)}(z)^T = \phi'(B_{i,:}^{(t)} z)^2 {B_{i,:}^{(t)}}^T B_{i,:}^{(t)} \implies \mathbb{E}_z \left[\nabla f_i^{(t)}(z) \nabla f_i^{(t)}(z)^T \right] = \frac{1}{2} {B_{i,:}^{(t)}}^T B_{i,:}^{(t)} \ .
$$

Thus, we conclude

$$
\begin{aligned}
&\frac{1}{k} {B^{(t)}}^T B^{(t)} = \sum_{i=1}^{k} {B_{i,:}^{(t)}}^T B_{i,:}^{(t)} = \frac{2}{k} \sum_{i=1}^{k} \mathbb{E}_z \left[\nabla f_i^{(t)}(z) \nabla f_i^{(t)}(z)^T \right] \\
&\implies \lim_{k\to\infty} \frac{1}{k} {B^{(t)}}^T B^{(t)} = 2\mathbb{E}_{z, w^{(0)}} \left[\left(\nabla_x g^{(t)}(z) \right) \left(\nabla_x g^{(t)}(z) \right)^T \right] \ .
\end{aligned}
$$

□

Next, we build on the previous analyses to demonstrate the connection between expected gradient outer product and the neural network feature matrix for deep nonlinear neural networks.

Deep Network Notation. For $\ell, t \in \mathbb{Z}_{>0}$,

$$\phi_{(\ell,t)}(x) \triangleq \begin{cases} \phi\left(\frac{1}{\sqrt{k}} B_{\ell-1}^{(t)} \phi_{(\ell-1,t)}(x)\right) & \ell > 0 \\ x & \ell = 0 \ . \end{cases}$$

We also consider "slices" of this recursive function. For $\ell' < \ell$,

$$\phi_{(\ell:\ell',t)}(x) \triangleq \phi\Big(\frac{1}{\sqrt{k}} B_{\ell}^{(t)} \phi\Big(\frac{1}{\sqrt{k}} B_{\ell-1}^{(t)} \Big(\cdots \phi\Big(\frac{1}{\sqrt{k}} B_{\ell'}^{(t)} x\Big)\Big)\Big)\Big) \ .$$

To write the gradients of a deep network we introduce additional notation. In particular, for the k'-th row of the weights at depth $\ell' \geq 1$ and input z,

$$\widetilde{B}_{\ell',k'}(z) \triangleq B_{\ell',k'} \odot \phi'(B_{\ell'-1}\phi_{\ell'-2,t}(z)) \in \mathbb{R}^{k \times 1}, \qquad \widetilde{a}^{(t)} \triangleq a^{(t)} \odot \phi'(B_\ell \phi_{\ell-1,t}(z)) \in \mathbb{R}^{1 \times k}.$$

We also define a deep network and an auxiliary deep network as follows,

$$g^{(t)}(x) \triangleq (a_\ell^{(t)})^\top \phi_{(\ell,t)}(x), \tag{1}$$

$$\widetilde{g}^{(t)}(x) \triangleq (a_\ell^{(t)})^\top \phi_{(\ell:1,t)} \left(\frac{1}{\sqrt{k}} \phi\left(B_0^{(t)} - B_0^{(t-1)}\right) x \right). \tag{2}$$

We drop the subscript t if it is irrelevant (and fixed) in an expression. Then, for the i^{th} row of the first layer weight matrix, and where $\widetilde{B}_{1,[:,i]} \in \mathbb{R}^{k \times 1}$ is the i^{th} column of $\widetilde{B}_1$,

$$\nabla_{B_{0,i}} g(x) = \widetilde{a}^{(t)} \widetilde{B}_\ell \cdots \widetilde{B}_2 \widetilde{B}_{1,[:,i]} x \in \mathbb{R}^d \ .$$

We write the gradient update for the inner-most weights $B_0^{(t)}$ (with the output of the model taken to be 0 at all steps). Consider the i^{th} row of this matrix, $B_{0,i}^{(t)}$. Then,

$$B_{0,i}^{(t+1)} \leftarrow B_{0,i}^{(t)} + \frac{\eta}{k^{\ell/2}} \sum_{p=1}^n y_p \widetilde{a}^{(t)} \widetilde{B}_\ell \cdots \widetilde{B}_2 \widetilde{B}_{1,[:,i]} x_p = B_{0,i}^{(t)} + \frac{\eta}{k^{\ell/2}} \sum_{p=1}^n \widetilde{a}^{(t)} \widetilde{B}_\ell \cdots \widetilde{B}_2 B_{1,[:,i]} \phi'((B_{0,i}^{(t)})^\top x_p) y_p x_p$$

Recall the notation of a deep fully connected network above in eq. (1).

Theorem 3 (Deep, 1-step)**.** *Consider a depth L fully connected network $g = \phi_\ell(\cdot)$. Let $\{x_i, y_i\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$ denote the training data, with $a_k^{(t)}$ i.i.d., $\mathbb{E}\left[(a_k^{(t)})^2\right] = 1$, and $\mathbb{E}\left[a_k^{(t)}\right] = 0$ for all k and $t \in \{0,1\}$. Suppose we train the first layer weights B_0 by gradient descent with step-size η with initialization $B_0^{(0)} = 0$,*

$$\lim_{k\to\infty} \frac{1}{k} \sum_{i=1}^k B_{0,i}^{(1)} B_{0,i}^{(1)\top} = 2\mathbb{E}_z \left[\lim_{k\to\infty} \left(\nabla_z g^{(1)}(z)\right) \left(\nabla_z g^{(1)}(z)\right)^\top \right], \qquad \textit{where} \quad z \sim \mathcal{N}(0, I_d).$$

The above result is a special case of the next theorem, and we provide a detailed proof of the following. Recall the notation of the auxiliary deep network above in eq. (2).

Theorem 4 (Deep, arbitrary steps)**.** *Consider a fully connected network g with L hidden layers. Let $\{x_i, y_i\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$ denote the training data and consider learning the weights B in the following manner:*

- *update B with stepsize η, initialized at* 0,
- *For training steps $t = 1, ..., T$, sample $a_k^{(t)}$ i.i.d. manner so that $\mathbb{E}\left[(a_k^{(t)})^2\right] = 1$, and $\mathbb{E}\left[a_k^{(t)}\right] = 0$, and $B_{\ell,i}^{(t)} \sim \mathcal{N}(0, I_k)$, for $\ell \geq 1$.*
- *modify the gradient updates so that the model's output is set to* 0, i.e.*, change the update,*

$$\textit{from} \quad B_0^{(t+1)} \leftarrow B_0^{(t)} - \eta \sum_{p=1}^n (y_p - g^{(t)}(x_p)) \nabla g^{(t)}(x_p) \qquad \textit{to} \quad B_0^{(t+1)} \leftarrow B_0^{(t)} - \eta \sum_{p=1}^n y_p \nabla g(x_p).$$

Then we have the following relation,

$$\lim_{k\to\infty}\frac{1}{k}\sum_{i=1}^{k}(B_{0,i}^{(t+1)}-B_{0,i}^{(t)})(B_{0,i}^{(t+1)}-B_{0,i}^{(t)})^\top = 2^L\mathbb{E}_z\left[\lim_{k\to\infty}\nabla_z\widetilde{g}^{(t+1)}(z)\nabla_z\widetilde{g}^{(t+1)}(z)^\top\right] \quad \textit{where} \quad z\sim\mathcal{N}(0,I_d).$$

Proof of Theorem 4. We write $L=\ell+1$. Then, the left hand side is given by:

$$\begin{aligned}(B_{0,i}^{(t+1)}-B_{0,i}^{(t)})(B_{0,i}^{(t+1)}-B_{0,i}^{(t)})^\top &= \frac{\eta^2}{k^{\ell+1}}\left(\sum_{p=1}^{n}\widetilde{a}^{(t)}\widetilde{B}_\ell\cdots\widetilde{B}_2B_{1,[:,i]}\phi'((B_{0,i}^{(t)})^\top x_p)y_px_p\right)\\ &\quad\times\left(\sum_{p=1}^{n}\left(\widetilde{a}^{(t)}\widetilde{B}_\ell\cdots\widetilde{B}_2B_{1,[:,i]}x_p\right)^\top\cdot y_p\phi'((B_{0,i}^{(t)})^\top x_p)\right)\\ &= \frac{\eta^2}{k^{\ell+1}}\left[\sum_{p,q=1}^{n}y_py_q\phi'((B_{0,i}^{(t)})^\top x_p)\phi'((B_{0,i}^{(t)})^\top x_q)\left(\widetilde{a}^{(t)}\widetilde{B}_\ell\cdots\widetilde{B}_2B_{1,[:,i]}x_p\right)\left(\widetilde{a}^{(t)}\widetilde{B}_\ell\cdots\widetilde{B}_2B_{1,[:,i]}x_q\right)^\top\right].\end{aligned}$$

We take the limit as $k\to\infty$ from the outermost layer to the innermost to derive,

$$\begin{aligned}\lim_{k\to\infty}\frac{1}{k}\sum_{i}(B_{0,i}^{(t+1)}-B_{0,i}^{(t)})(B_{0,i}^{(t+1)}-B_{0,i}^{(t)})^\top &= 2^{-\ell}\eta^2\sum_{p,q=1}^{n}y_py_qx_px_q^\top\lim_{k\to\infty}\frac{1}{k}\sum_{i}\phi'((B_{0,i}^{(t)})^\top x_p)\phi'((B_{0,i}^{(t)})^\top x_q)\\ &= 2^{-\ell}\eta^2\sum_{p,q=1}^{n}y_py_qx_px_q^\top\mathbb{E}_{B_{0,i}^{(t)}}\left[\phi'((B_{0,i}^{(t)})^\top x_p)\phi'((B_{0,i}^{(t)})^\top x_q)\right].\end{aligned}$$

Now, we consider the right hand side of the desired equation.

$$\widetilde{g}^{(t+1)}(z) = (a_\ell^{(t)})^\top\phi_{(\ell:1,t)}\left(\phi\left(\frac{\eta}{\sqrt{k}}\sum_{p=1}^{n}\left(\widetilde{a}^{(t)}\widetilde{B}_\ell^{(t)}\cdots\widetilde{B}_2^{(t)}B_1^{(t)}\phi'((B_0^{(t)})^\top x_p)\right)y_px_p^\top z\right)\right)$$

The gradient with respect to the input is then:

$$\begin{aligned}\nabla_x\widetilde{g}^{(t+1)}(z) &= \left(\frac{\eta}{k^{\ell/2}}\widetilde{a}^{(t+1)}\widetilde{B}_\ell^{(t+1)}\cdots\widetilde{B}_2^{(t+1)}\widetilde{B}_1^{(t+1)}(B_0^{(t+1)}-B_0^{(t+1)})\right)^\top\\ &= \left(\frac{\eta}{k^\ell}\widetilde{a}^{(t+1)}\widetilde{B}_\ell^{(t+1)}\cdots\widetilde{B}_2^{(t+1)}\widetilde{B}_1^{(t+1)}\left(\sum_{p=1}^{n}\widetilde{a}^{(t)}\widetilde{B}_\ell^{(t)}\cdots\widetilde{B}_2^{(t)}B_1^{(t)}\phi'(B_0^{(t)}x_p)y_p\right)^\top x_p^\top\right)^\top\end{aligned}$$

Expanding the outer product, eliminating cross terms, and taking the infinite width limit from the outermost to the innermost layer,

$$\mathbb{E}_z\left[\lim_{k\to\infty}\left(\nabla_z g^{(t+1)}(z)\right)\left(\nabla_z g^{(t+1)}(z)\right)^\top\right] = 2^{-2\ell-1}\eta^2\sum_{p,q=1}^{n}y_py_qx_px_q^\top\mathbb{E}_{B_{0,i}^{(t)}}\left[\phi'(B_{0,i}^{(t)}x_p)\phi'(B_{0,i}^{(t)}x_q)\right].$$

This concludes the proof. □

Proof of Theorem 3. The proof of this theorem follows as a special case of the previous theorem. □

B Background on Kernel Ridge Regression

We here provide a brief review of kernel ridge regression [57]. 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 an non-parametric estimator given by

$$\widehat{f}_{n,\lambda} = \operatorname*{argmin}_{f\in\mathcal{H}}\sum_{i=1}^{n}(f(x_i)-y_i)^2+\lambda\|f\|_{\mathcal{H}}^2\,; \tag{3}$$

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 [74], this problem has a unique solution in the span of the data given by

$$\widehat{f}_{n,\lambda} = \sum_{i=1}^{n} \widehat{\alpha}_i K(x, x_i) \qquad \text{where} \quad \widehat{\alpha} = y(K(X,X) + \lambda I_n)^{-1}; \tag{4}$$

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 [43] to approximately solve kernel regression via early-stopped, preconditioned-SGD that can run on the GPU. For $\lambda \to 0^+$, we recover the pseudo-inverrse solution $\widehat{\alpha} = yK(X,X)^\dagger$. For multi-class and multi-variate problems, y_i are vector valued and we consider each class/target variable as a separate problem.

C Dataset and Experimental Details

Below, we provide a description of all datasets, models, and training methodology considered in this work.

Experiments with RFMs and fully connected networks on CelebA. For all binary classification tasks on CelebA in this work, 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 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 [72].

For all classification tasks, 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 [48]. 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).

For RFM-T experiments in Fig. 7, we retrained a Laplace kernel with bandwidth 10 and ridge regularization of 10^{-3} after thresholding the diagonals of the feature matrix with pixel intensity larger than 0.05 after min-max scaling the feature matrix.

SVHN. 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 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. 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 [72]. The test accuracy for RFMs in Fig. 3c is given by training a 2-hidden layer NTK on the feature matrix selected from the iteration of training (iteration 4) that resulted in the best validation accuracy.

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 [48].

Low rank polynomials. We consider the low rank polynomials from [73] and [16]. We use 1000 examples for training and 10000 samples for testing. Following the setup of [73], we sample training inputs from a Rademacher distribution in 30 dimensions and add random noise (see Fig. 11a). 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 in the feature matrix. 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 Fig. 11d, 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 $g(z) = \mathrm{He}_2(z) + \mathrm{He}_4(z)$ where $\mathrm{He}_2, \mathrm{He}_4$ are the second and fourth probabilist's Hermite polynomials. We train a 1 hidden layer neural network using full batch Adam [33] 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 [17]. We first describe the experiments for 120 of the 121 datasets with fewer than 130000 examples since we used EigenPro [43] 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. 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 [23]. We used the repository from [23] 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 [23], "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 $L = 10$. On large regression datasets, we also optimized for bandwidths over $\{1, 5, 10, 15, 20\}$. For the *song release year (year)* and *drug design (yprop_4_1)* datasets, we additionally optimized over large bandwidths $\{50, 100, 500\}$ for the vanilla Laplace kernel. We also optimized over two target transformations - the log transform ($\widehat{y} = |y| \log(1 + |y|)$) and `sklearn.preprocessing.QuantileTransformer`. In both cases, we inverted the transform before testing. We also optimized over data transformations - `sklearn.preprocessing.StandardScaler` and `sklearn.preprocessing.QuantileTransformer`. For non-kernel methods, we compare to the metrics reported in [23]. 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.

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 5×5 square of pixels (rows 2 through 6 and columns 7 through 11) in the upper left corner to all 1s 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 full gradient descent with a learning rate of 0.1. 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 for three iterations with ridge regularization of 10^{-3} and using the Laplace kernel as the base kernel function with a bandwidth of 10.

Simplicity bias. For experiment in Fig. 8, 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. We trained a two hidden layer fully connected ReLU network using SGD with a learning rate of 0.1 and a mini-batch size of 100. The RFM was trained for 1 iteration using the Laplace kernel as the base kernel function and ridge regularization of 10^{-3}.

D Description of Metrics for 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 [17], certain classifiers were missing performance values. To compute the Friedman rank, the authors of [17] 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 [17].

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 $\mathsf{ADTM}_j = \text{Avg}_i \left(\frac{x_{ij} - \min_j x_{ij}}{\max_j x_{ij}} \right)$. Note $\mathsf{ADTM} \in [0, 1]$, with 1 indicating a method is the best among all methods in the collection, and 0 indicating a method is the worst.

E Sample Complexity Improvement with the Expected Gradient Outer Product

We demonstrate that one can obtain a provable improvement in sample complexity using RFM (and applying the expected gradient outer product transformation in general) in the case that the target function depends on only a few relevant directions of the input.

Definition 1 (Rank-r function)**.** *A function $f : \mathbb{R}^d \to \mathcal{Y}$ has **rank** r if there exists a matrix $U \in \mathbb{R}^{r \times d}$ such that $f(x) = g(Ux)$ for some other function $g : \mathbb{R}^r \to \mathcal{Y}$ for all $x \in \mathbb{R}^d$. The matrix U is called the **target subspace** and its **span** is the span of its rows.*

Proposition 2 (Expected Gradient Outer Product for Low Rank functions)**.** *Suppose the labels $Y \in \mathbb{R}^N$ are generated from a target function $f^* : \mathbb{R}^n \to \mathcal{Y}$ is a degree d polynomial of rank r. Let $M \in \mathbb{R}^{n \times n}$ be the expected gradient outer product, i.e., $M = \mathbb{E}_{x \in X} \left[\nabla f^*(x) \nabla f^*(x)^T \right]$. Then, kernel ridge regression on the transformed data (MX, Y) requires $\text{poly}(n^r)$ samples to learn the target function.*

Remark. *Without the expected gradient outer product transformation, $\Omega(n^d)$ samples are required to achieve better error than the trivial 0-function [22].*

Proof of Proposition 2. 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 $x' \in \mathbb{R}^n$, as

$$Mx' = \mathbb{E}_{x \in X_N} \left[\nabla f^*(x) \nabla f^*(x)^T x' \right] ,$$

we have that Mx' is also in the span of U. Therefore, the transformed data MX_N lies in an r-dimensional subspace and have an equivalent representation in an r-dimensional coordinate space. In other words, for all $i, j \in [n]$, 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 directly from the generalization error bound of [22] for kernel ridge regression with r coordinates. □

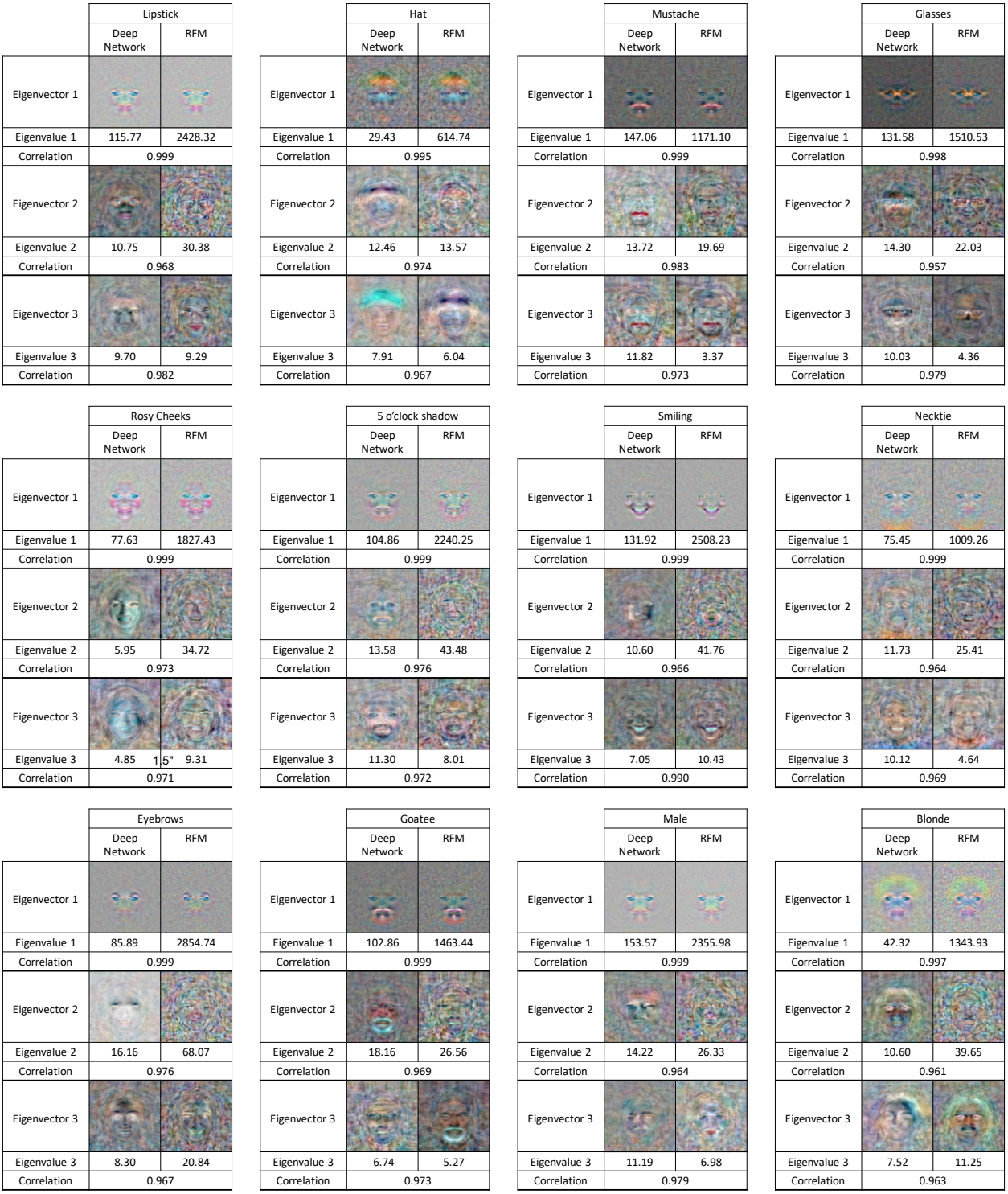


Figure 10: Top three eigenvectors and corresponding eigenvalues of feature matrices of deep networks and RFMs trained for the 12 CelebA classification tasks considered in this work. Overall, we observe (1) Pearson correlation greater than 0.99 between the top eigenvectors of deep network feature matrices and RFM feature matrices and (2) a large gap between the top eigenvalue and second eigenvalue for all feature matrices. These two observations together provide further evidence that RFMs are accurately capturing the features learned by deep fully connected neural networks. In addition, we interestingly observe some cases such as hat, goatee, and blonde classification where the third eigenvector of RFM feature matrices appears visually identical to the second eigenvector of deep network feature matrices.

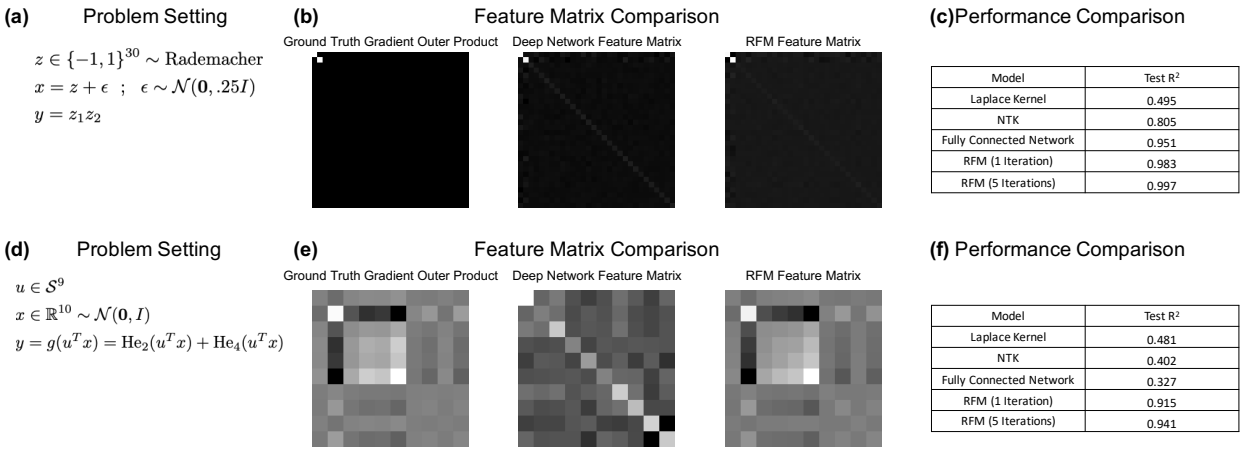


Figure 11: RFMs close the gap and surpass neural networks on low rank polynomial tasks from previous work [16, 73]. All experimental details are provided in Appendix C. (a) We consider the low rank setup from [73] in which the targets, y, are generated as a product of the first two coordinates of Rademacher random variables z. (b) Since we know the target function, we can compare the ground truth gradient outer product against the feature matrices of a trained 1 hidden layer ReLU fully connected network and a trained RFM. We observe that both models learn to select the top two coordintaes. (c) The performance of RFMs and neural networks far exceeds of NTKs and Laplace kernels since these methods learn to select relevant coordinates for prediction. (d) The low rank setup from [16] 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 $\mathrm{He}_2, \mathrm{He}_4$ denote the second and fourth probabilist's Hermite polynomials. (e) While RFMs learn to accurately approximate the ground truth gradient outer product, fully connected networks require additional training modifications, as discussed in [16]. (f) As they learn the relevant subspace, RFMs far outperform 1 hidden layer ReLU fully connected networks, NTKs, and the Laplace kernel.

Training samples	Dataset	FT Transformer	GradientBoostingTree	RandomForest	Resnet	SAINT	XGBoost	RFM	best method
4547.0	winequality	-	0.458	0.504	-	-	0.498	0.497	RandomForest
5457.0	isolet	-	-	-	-	-	-	0.868	RFM
5734.0	cpuact	-	0.985	0.983	-	-	0.986	0.986	XGBoost
7056.0	sulfur	-	0.806	0.859	-	-	0.865	0.913	RFM
7484.0	Brazilianhouses	-	0.996	0.993	-	-	0.998	0.993	XGBoost
9625.0	Ailerons	-	0.843	0.839	-	-	0.844	0.841	XGBoost
9752.0	MiamiHousing2016	-	0.924	0.925	-	-	0.936	0.926	XGBoost
10000.0	BikeSharingDemand	-	0.69	0.687	-	-	0.695	0.689	XGBoost
10000.0	california	-	0.846	0.83	-	-	0.861	0.858	XGBoost
10000.0	diamonds	-	0.945	0.945	-	-	0.946	0.944	XGBoost
10000.0	elevators	-	0.863	0.841	-	-	0.908	0.923	RFM
10000.0	fifa	-	0.663	0.655	-	-	0.668	0.652	XGBoost
10000.0	house16H	-	0.541	0.486	-	-	0.548	0.489	XGBoost
10000.0	housesales	-	0.884	0.871	-	-	0.889	0.875	XGBoost
10000.0	houses	-	0.84	0.829	-	-	0.852	0.856	RFM
10000.0	medicalcharges	-	0.979	0.979	-	-	0.979	0.979	GradientBoostingTree
10000.0	nyc-taxi-green-dec-2016	-	0.554	0.563	-	-	0.553	0.53	RandomForest
10000.0	pol	-	0.99	0.989	-	-	0.99	0.991	RFM
10000.0	superconduct	-	0.905	0.91	-	-	0.911	0.911	RFM
10000.0	year	-	0.271	0.241	-	-	0.282	0.303	RFM
37758.0	diamonds	0.945	0.947	-	0.941	0.945	0.948	0.949	RFM
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	RFM

Table 2: Regression R^2 without categorical variables

Training samples	Dataset	FT Transformer	GradientBoostingTree	MLP	RandomForest	Resnet	SAINT	XGBoost	RFM	best method
1787.0	wine	77.54	78.77	77.62	78.96	78.03	77.09	79.85	80.67	RFM
2220.0	phoneme	85.28	86.78	84.95	88.55	86.21	85.58	87.08	88.1	RandomForest
3631.0	kddipumsla97-small	88.96	88.38	87.95	87.95	88.07	89.02	88.26	88.32	SAINT
5325.0	eyemovements	58.62	63.75	56.89	65.04	57.41	58.93	65.54	59.79	XGBoost
7057.0	pol	98.47	98.21	94.27	98.21	94.81	98.14	98.05	98.27	FT Transformer
7404.0	bank-marketing	80.42	80.27	79.18	79.82	79.55	79.09	80.44	79.51	XGBoost
9363.0	MagicTelescope	85.82	85.88	84.7	85.6	85.78	85.1	85.92	85.55	XGBoost
9441.0	house16H	88.16	88.2	87.78	87.8	87.5	88.16	88.83	87.82	XGBoost
10000.0	Higgs	70.7	71.08	68.86	70.76	69.44	70.92	71.42	70.14	XGBoost
10000.0	MiniBooNE	93.74	93.29	93.54	92.71	93.68	93.52	93.62	93.66	FT Transformer
10000.0	california	88.62	89.82	86.0	89.21	87.57	89.07	90.17	89.39	XGBoost
10000.0	covertype	81.32	81.87	78.89	82.75	80.26	80.31	81.9	84.93	RFM
10000.0	credit	76.53	77.22	75.99	77.4	76.29	75.99	77.38	77.66	RFM
10000.0	electricity	82.0	86.16	81.04	86.14	80.86	81.77	86.83	81.5	XGBoost
10000.0	jannis	76.55	77.55	74.57	77.27	74.6	77.26	78.0	75.8	XGBoost
40306.0	jannis	79.74	79.47	76.45	78.85	78.59	79.77	79.56	79.83	RFM
50000.0	Higgs	73.13	72.55	71.15	71.98	72.39	72.75	72.87	72.19	FT Transformer
50000.0	MiniBooNE	94.56	94.14	94.32	93.53	94.61	94.32	94.47	94.72	RFM
50000.0	covertype	90.69	89.76	87.51	90.58	89.39	89.59	89.74	93.78	RFM

Table 3: Classification accuracy without categorical variables

Training samples	Dataset	FT Transformer	GradientBoostingTree	HistGradientBoostingTree	RandomForest	Resnet	XGBoost	RFM	best method
2836.0	analcatdatasupreme	0.977	0.982	0.982	0.982	0.98	0.984	0.986	RFM
2946.0	MercedesBenzGreenerManufacturing	0.548	0.578	0.576	0.575	0.545	0.578	0.55	GradientBoostingTree
6048.0	visualizingsoil	1.0	1.0	1.0	1.0	1.0	1.0	1.0	RFM
6219.0	yprop41	0.037	0.056	0.063	0.095	0.021	0.08	0.065	RandomForest
7484.0	Brazilianhouses	0.883	0.996	0.994	0.993	0.879	0.998	0.992	XGBoost
9999.0	OnlineNewsPopularity	0.143	0.153	0.156	0.151	0.13	0.162	0.13	XGBoost
10000.0	BikeSharingDemand	0.937	0.942	0.942	0.938	0.936	0.946	0.933	XGBoost
10000.0	SGEMMGPUkernelperformance	1.0	1.0	1.0	1.0	1.0	1.0	1.0	RFM
10000.0	blackfriday	0.381	0.615	0.616	0.609	0.36	0.619	0.607	XGBoost
10000.0	diamonds	0.99	0.99	0.991	0.988	0.989	0.991	0.991	XGBoost
10000.0	housesales	0.891	0.891	0.89	0.875	0.881	0.896	0.884	XGBoost
10000.0	nyc-taxi-green-dec-2016	0.511	0.573	0.539	0.585	0.451	0.579	0.536	RandomForest
10000.0	particulate-matter-ukair-2017	0.673	0.684	0.69	0.675	0.658	0.691	0.659	XGBoost
37758.0	diamonds	-	0.992	0.993	-	-	0.993	0.993	RFM
50000.0	SGEMMGPUkernelperformance	-	1.0	1.0	-	-	1.0	1.0	RFM
50000.0	blackfriday	-	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	0.711	-	-	0.712	0.682	XGBoost

Table 4: Regression R^2 with categorical variables

Training samples	Dataset	FT Transformer	GradientBoostingTree	HistGradientBoostingTree	RandomForest	Resnet	SAINT	XGBoost	RFM	best method
3479.0	rl	70.31	77.62	76.05	79.79	70.56	68.2	79.63	70.53	RandomForest
3522.0	KDDCup09upselling	78.05	-	-	-	76.98	77.8	-	-	FT Transformer
3589.0	KDDCup09upselling	-	80.36	80.61	80.02	-	-	79.56	79.0	HistGradientBoostingTree
5325.0	eyemovements	59.83	63.94	63.58	65.73	57.93	58.54	66.77	61.12	XGBoost
10000.0	compass	75.34	74.09	75.15	79.28	74.46	71.87	76.91	77.5	RandomForest
10000.0	covertype	86.74	85.56	84.53	85.92	85.27	84.95	86.42	88.98	RFM
10000.0	electricity	84.16	87.97	88.15	87.76	82.64	83.35	88.69	85.43	XGBoost
10000.0	road-safety	76.74	76.21	76.45	75.88	76.08	76.43	76.73	75.48	FT Transformer
50000.0	covertype	93.61	93.22	92.09	93.35	92.24	92.58	93.31	95.38	RFM
50000.0	road-safety	78.95	78.73	78.85	78.13	78.41	77.96	80.29	77.86	XGBoost

Table 5: Classification accuracy with categorical variables