On the Neural Feature Ansatz for Deep Neural Networks

Edward Tansley
University of Oxford

Estelle Massart
UCLouvain

Coralia Cartis
University of Oxford

Abstract

Understanding feature learning is an important open question in establishing a mathematical foundation for deep neural networks. The Neural Feature Ansatz (NFA) states that after training, the Gram matrix of the first-layer weights of a deep neural network is proportional to some power $\alpha > 0$ of the average gradient outer product (AGOP) of this network with respect to its inputs. Assuming gradient flow dynamics with balanced weight initialization, the NFA was proven to hold throughout training for two-layer linear networks with exponent $\alpha = 1/2$ (Radhakrishnan et al., 2024). We extend this result to networks with $L \geq 2$ layers, showing that the NFA holds with exponent $\alpha = 1/L$, thus demonstrating a depth dependency of the NFA. Furthermore, we prove that for unbalanced initialization, the NFA holds asymptotically through training if weight decay is applied. We also provide counterexamples showing that the NFA does not hold for some network architectures with nonlinear activations, even when these networks fit arbitrarily well the training data. We thoroughly validate our theoretical results through numerical experiments across a variety of optimization algorithms, weight decay rates and initialization schemes.

1 INTRODUCTION

Deep neural networks (DNNs) typically operate in the overparametrized regime, where the number of parameters to tune in the model outweighs the size of the training data. While overparametrization endows DNNs with extreme expressivity, allowing exact interpolation of the data even when the latter are noisy realizations (Zhang et al., 2021), the good performance of DNNs observed in practice calls for *implicit biases* that prevent the model from overfitting the data (Vardi, 2023).

This work addresses recent developments, exploring specific biases in the feature learning mechanisms, namely, the process through which neural networks extract information from high-dimensional input data. Recently, the Neural Feature Ansatz (NFA) was proposed as a possible explanation of feature extraction, in which model weights reflect the importance that features have on model predictions (quantified by the magnitude of the partial derivatives of the model output with respect to its input). The NFA was shown empirically to hold in a wide range of models including feedforward, convolutional, recurrent neural networks as well as transformers (Radhakrishnan et al., 2024), and was used as a hypothesis to shed light on other behaviors such as neural collapse (Beaglehole et al., 2024), training instabilities (Zhu et al., 2024), and grokking (Mallinar et al., 2025). Developing a theoretical foundation for the NFA is however still an open question.

Beyond the NFA, the presence of low-dimensional structures was identified in Parkinson et al. (2023, 2025) in the specific setting of deep linear neural networks (with a single final ReLU layer) trained with weight decay; this type of architecture is known to provide insight into the effect of network depth (Arora et al., 2018), despite the fact that adding linear layers does not increase model expressivity. Parkinson et al. (2025) showed both theoretically and empirically that increasing model depth leads to some form of implicit bias on the linear layers that induces the learned model to exhibit some low-dimensional behavior, by varying mostly along a subset of directions of the input space and being almost constant along its orthogonal complement. Functions that are only varying along a low-dimensional subspace of their input space are often referred to as *multi-index*, or *low-rank* functions. This implicit bias also improves model generalization,

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. Copyright 2026 by the author(s).

which, in the case where data is generated using a multi-index target function, is higher when the low-dimensional subspace of variation is aligned with the subspace of variation of the target function. In a similar vein, Guth et al. (2024) explored feature extraction mechanisms across layers in random feedforward neural networks, and identified some low-rank structure and dimensionality reduction mechanisms within layers. Furthermore, the ubiquity of multi-index models motivated the development of dedicated training strategies, see Bruna and Hsu (2025) for a survey. In particular, it was shown in Mousavi-Hosseini et al. (2025) that learning the low-rank structure can remove the dependence on the ambient dimension in high-dimensional settings.

While these two lines of work both aim to advance the understanding of feature learning mechanisms, many open questions remain. In this work, inspired by Parkinson et al. (2023, 2025) and Radhakrishnan et al. (2024), we aim to uncover the role of depth in the NFA. More precisely, we prove that for deep linear neural networks trained under gradient flow dynamics with balanced initialization, the NFA holds exactly for all time. We further prove that this result holds asymptotically in the case of unbalanced initialization, in the presence of weight decay regularization. For nonlinear neural networks, we propose a counterexample that illustrates that the NFA does not always hold, and a second one illustrating the limitations of the NFA to account for model generalization. We conclude the paper with numerical experiments supporting our theoretical findings as well as the dimensionality reduction mechanism resulting from the NFA when learning multi-index functions.

Meanwhile, the recent work of Ziyin et al. (2025) derived a unifying framework involving variants of the NFA under assumptions involving alignment of gradients, features and/or weights. Even more recently, and concurrently to our work, the authors of Boix-Adsera et al. (2025) proposed an alternative to the NFA that can be derived from first-order optimality conditions. Our results here add further understanding to this growing body of works.

Notation. We consider a neural network $f_{\boldsymbol{\theta}}$, parametrized by a set of parameters $\boldsymbol{\theta}$, aiming to approximate some function f^* using a set of N data points $\{(\boldsymbol{x}_i, y_i)\}_{1\leq i\leq N}$, with $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i = f^*(\boldsymbol{x}_i)$. This paper considers L-layer feedforward neural networks, whose set of parameters $\boldsymbol{\theta}$ contains weight matrices $\boldsymbol{W}_1, \dots, \boldsymbol{W}_L$ and biases $\boldsymbol{b}_1, \dots, \boldsymbol{b}_L$. The network $f_{\boldsymbol{\theta}}$ is learned by minimizing the empirical loss $\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^N l_{\boldsymbol{\theta}}(\boldsymbol{x}_i, y_i)$, for some $l_{\boldsymbol{\theta}} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}$. We denote by $\boldsymbol{W}_{l,t}$ the weight matrix associated with the lth layer at iteration t of the optimization algorithm, and $\boldsymbol{W}_{l,0}$ its corresponding value at initialization. For a given function f, we denote by $\boldsymbol{A}_f$ the AGOP of f, which we define in section 2. We use $\|\cdot\|_2$ and $\|\cdot\|_F$ to refer to the 2-norm and the Frobenius norm, respectively. We use $\mathrm{Tr}(\cdot)$ for the trace and $\ker(\cdot)$ for the kernel of a matrix. We use bold lower and uppercase characters (e.g., $\boldsymbol{v}$, $\boldsymbol{W}$) for vectors and matrices, respectively.

2 PRELIMINARIES

Low-Rank Functions. Low-rank functions, also referred to as *multi-index* (Bruna and Hsu, 2025), *multi-ridge* (Tyagi and Cevher, 2014), functions with *active subspaces* (Constantine, 2015), or functions of *low effective dimensionality* (Cartis et al., 2024), are functions that vary only within a (low-dimensional and unknown) linear subspace and are constant along its orthogonal complement. These functions satisfy the following equivalent properties.

Definition 2.1 (Cartis et al. (2024))**.** *A continuously differentiable*[1] *function $f : \mathbb{R}^d \to \mathbb{R}$ has rank $r \leq d$ if it satisfies one of the following equivalent conditions:*

1. *There exists a subspace $\mathcal{T}$ of dimension r such that $f(\boldsymbol{x}_\top + \boldsymbol{x}_\perp) = f(\boldsymbol{x}_\top)$ for all $\boldsymbol{x}_\top \in \mathcal{T}$ and $\boldsymbol{x}_\perp \in \mathcal{T}^\perp$.*
2. *There exists a subspace $\mathcal{T}$ of dimension r such that $\nabla f(\boldsymbol{x}) \in \mathcal{T}$ for all $\boldsymbol{x} \in \mathbb{R}^d$.*
3. *There exists a matrix $\boldsymbol{A} \in \mathbb{R}^{r\times d}$ and a map $g : \mathbb{R}^r \to \mathbb{R}$ such that $f(\boldsymbol{x}) = g(\boldsymbol{A}\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathbb{R}^d$.*

Average Gradient Outer Product (AGOP). We recall the definition of the *average gradient outer product (AGOP)* of the network with respect to the data, which we write $\boldsymbol{A}_f \in \mathbb{R}^{d\times d}$:

$$\boldsymbol{A}_f := \frac{1}{N}\sum_{i=1}^N \nabla f(\boldsymbol{x}_i)\nabla f(\boldsymbol{x}_i)^\top. \tag{2.1}$$

Note that the eigenvectors associated with the eigenvalues of $\boldsymbol{A}_f$ with largest magnitude correspond to the directions in which perturbations to the data have the largest effect on the network output, when averaged over the data points. Indeed, let $\boldsymbol{z}$ be some arbitrary vector, then

$$\frac{1}{N}\sum_{i=1}^N \|\nabla f(\boldsymbol{x}_i)^\top \boldsymbol{z}\|_2^2 = \boldsymbol{z}^\top \boldsymbol{A}_f \boldsymbol{z}; \tag{2.2}$$

[1] Note that some of the neural networks we consider are not continuously differentiable, for example due to the classical ReLU activation, but as these models are continuously differentiable almost everywhere this does not raise any practical issue.

choosing $\boldsymbol{z}$ as the eigenvector of $\boldsymbol{A}_f$ associated with its largest eigenvalue will maximize the right-hand side of equation 2.2 over all unit-norm vectors (by the definition of the Rayleigh quotient), hence, the left hand-side of equation 2.2. Note that, if the network has a multivariate output, a similar expression involving the Jacobian can be used (Radhakrishnan et al., 2024). Noting that the rows of the Jacobian are themselves the transposes of the gradients, for $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))^\top$, we have that $\boldsymbol{J}_f(\boldsymbol{x})^\top \boldsymbol{J}_f(\boldsymbol{x}) = \sum_{j=1}^m \nabla f_j(\boldsymbol{x}) \nabla f_j(\boldsymbol{x})^\top$.

The Neural Feature Ansatz (NFA). The *Neural Feature Ansatz* (see Radhakrishnan et al. (2024)) states that the weight matrix associated to the first layer can explain the structure of the AGOP matrix:

$$\boldsymbol{W}_1^\top \boldsymbol{W}_1 \propto (\boldsymbol{A}_f)^\alpha \tag{2.3}$$

for some $\alpha > 0$. A value of $\alpha = 1/2$ is proposed and proven in the case of a 2-layer linear network (under gradient flow and balanced initialization) (Radhakrishnan et al., 2024). We use the *cosine similarity*, donated by ρ, to measure the extent to which the NFA holds. For two matrices $\boldsymbol{M}$ and $\boldsymbol{N}$, this is given by

$$\rho(\boldsymbol{M}, \boldsymbol{N}) := \mathrm{Tr}(\boldsymbol{M}^\top \boldsymbol{N}) / (\|\boldsymbol{M}\|_F \cdot \|\boldsymbol{N}\|_F). \tag{2.4}$$

Here, $\rho(\boldsymbol{W}_1^\top \boldsymbol{W}_1,\ (\boldsymbol{A}_f)^\alpha) \approx 1$ indicates that the NFA approximately holds for that particular power of α.

In Figure 1, we include an illustration of the NFA. We see that after training a 5-layer network with weight decay, $\boldsymbol{W}_1^\top \boldsymbol{W}_1 \propto (\boldsymbol{A}_f)^{1/5}$. Furthermore, these two matrices are approximately equal, supporting our results in section 3.

3 THE NFA FOR DEEP LINEAR NETWORKS

In this section, we prove that the NFA holds (at least asymptotically) for deep linear neural networks when the latter are trained with gradient flow dynamics and weight decay regularization. Thus we extend the results of Radhakrishnan et al. (2024), which were restricted to 2-layer NNs. Throughout this section we shall consider deep linear networks of the form

$$f(\boldsymbol{x}) = \boldsymbol{W}_L \boldsymbol{W}_{L-1} \cdot \ldots \cdot \boldsymbol{W}_1 \boldsymbol{x} \tag{3.1}$$

for some $L \geq 2$. The Jacobian of such a network is given by $\boldsymbol{J}_f(\boldsymbol{x}) = \boldsymbol{W}_L \boldsymbol{W}_{L-1} \cdot \ldots \cdot \boldsymbol{W}_1$ for all $\boldsymbol{x} \in \mathbb{R}^d$. We therefore write $\boldsymbol{J}_f$ instead of $\boldsymbol{J}_f(\boldsymbol{x})$. Due to this constant Jacobian, we have $\boldsymbol{A}_f = \boldsymbol{J}_f^\top \boldsymbol{J}_f$.

Here we assume the gradient flow dynamics, namely,

$$\frac{\partial \boldsymbol{W}_{l,t}}{\partial t} = -\frac{\partial \mathcal{L}(\boldsymbol{\theta}_t)}{\partial \boldsymbol{W}_{l,t}}, \tag{3.2}$$

where $\boldsymbol{W}_{l,t}$ and $\boldsymbol{\theta}_t$ are respectively the weights of the lth layer and the total set of parameters at time $t \geq 0$.

As a first step, we assume balanced initialization of the weight matrices in the model, where balancedness is defined as follows.

Definition 3.1 (Balanced matrices)**.** *Two weight matrices $\boldsymbol{W}_l \in \mathbb{R}^{k \times m}$ and $\boldsymbol{W}_{l+1} \in \mathbb{R}^{n \times k}$ are said to be balanced if $\boldsymbol{W}_l \boldsymbol{W}_l^\top = \boldsymbol{W}_{l+1}^\top \boldsymbol{W}_{l+1}$.*

We say that a network is balanced if the weight matrices of each pair of adjacent layers are balanced and an initialization of the network weights is balanced if the network is balanced at time $t = 0$.

The following result states that if this balancedness property holds at initialization, it holds for all $t \geq 0$ assuming the weights follow gradient flow dynamics.

Lemma 3.1 (Arora et al. (2018))**.** *For time $t \geq 0$, let us define $f_t(\boldsymbol{x}) = \boldsymbol{W}_{L,t} \boldsymbol{W}_{L-1,t} \cdots \boldsymbol{W}_{1,t} \boldsymbol{x}$ for $\boldsymbol{x} \in \mathbb{R}^d$. Suppose that $\boldsymbol{W}_{1,t}, \boldsymbol{W}_{2,t}, \ldots, \boldsymbol{W}_{L,t}$ follow the gradient flow dynamics given by Equation 3.2, then for any $t \geq 0$ and $1 \leq l < L$,*

$$\boldsymbol{W}_{l+1,t}^\top \boldsymbol{W}_{l+1,t} - \boldsymbol{W}_{l,t} \boldsymbol{W}_{l,t}^\top = \boldsymbol{W}_{l+1,0}^\top \boldsymbol{W}_{l+1,0} - \boldsymbol{W}_{l,0} \boldsymbol{W}_{l,0}^\top \tag{3.3}$$

Hence if the layers in a network are balanced at $t = 0$, they are balanced for all $t \geq 0$ when the network is trained under gradient flow.

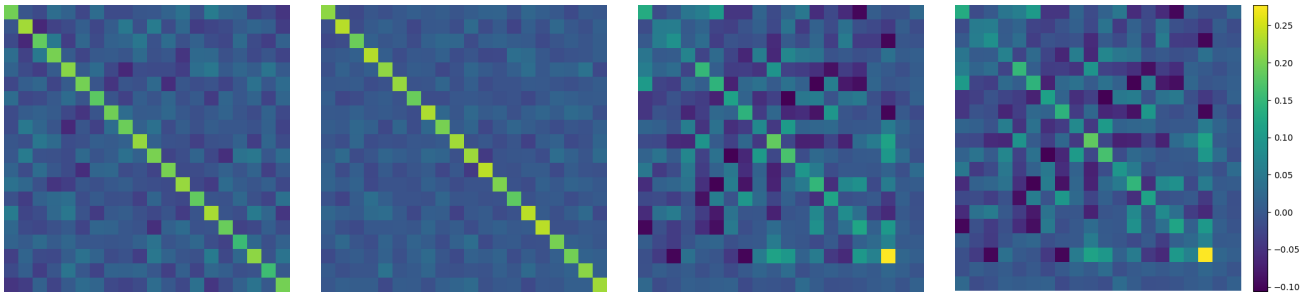
We now present a result proving that the NFA holds for L-layer linear networks, extending the result of Radhakrishnan et al. (2024), where 2-layer linear networks are considered. This result suggests that the α value in the Neural Feature Ansatz has a depth dependency. Noting that the Jacobian of the network is exactly equal to the product of matrices, Theorem 3.1 follows directly Equation (24) in Arora et al. (2018). The derivation below can be found in Radhakrishnan et al. (2025), although our own results were derived independently.

Theorem 3.1 (NFA for deep linear networks)**.** *For $t \geq 0$, let $f_t(\boldsymbol{x}) = \boldsymbol{W}_{L,t} \boldsymbol{W}_{L-1,t} \cdots \boldsymbol{W}_{1,t} \boldsymbol{x}$ for $\boldsymbol{x} \in \mathbb{R}^d$. Suppose that $\boldsymbol{W}_{1,t}, \boldsymbol{W}_{2,t}, \ldots, \boldsymbol{W}_{L,t}$ follow the gradient flow dynamics given by Equation 3.2. Suppose additionally that $\boldsymbol{W}_{1,0}, \boldsymbol{W}_{2,0}, \ldots, \boldsymbol{W}_{L,0}$ are balanced ($\boldsymbol{W}_{l,0} \boldsymbol{W}_{l,0}^\top = \boldsymbol{W}_{l+1,0}^\top \boldsymbol{W}_{l+1,0}$ for $l = 1, \ldots, L-1$) then at any time $t \geq 0$, we have*

$$\boldsymbol{W}_{1,t}^\top \boldsymbol{W}_{1,t} = (\boldsymbol{A}_{f,t})^{1/L}. \tag{3.4}$$

where $\boldsymbol{A}_{f,t} = \boldsymbol{J}_{f_t}(\boldsymbol{x})^\top \boldsymbol{J}_{f_t}(\boldsymbol{x})$.

Proof. By first applying Lemma 3.1 and expanding the



(a) (b) (c) (d)

Figure 1: **The NFA holds after training under weight decay, even with unbalanced initialization.** (a) $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ before training. (b) $(\boldsymbol{A}_f)^{1/L}$ before training. (c) $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ after training. (d) $(\boldsymbol{A}_f)^{1/L}$ after training, with $L = 5$ linear layers. Alignment as measured by cosine similarity (Equation 2.4): before training $= 0.915$; after training $= 1.000$. We plot the same experiment as in Figure 3 (a) which is described in section 5.

brackets, we have, for all $k \geq 1$, $t \geq 0$ and $1 \leq l < L$:

$$\begin{aligned} \boldsymbol{W}_{l,t}^\top (\boldsymbol{W}_{l+1,t}^\top \boldsymbol{W}_{l+1,t})^k \boldsymbol{W}_{l,t} &= \boldsymbol{W}_{l,t}^\top (\boldsymbol{W}_{l,t} \boldsymbol{W}_{l,t}^\top)^k \boldsymbol{W}_{l,t} \\ &= (\boldsymbol{W}_{l,t}^\top \boldsymbol{W}_{l,t})^{k+1}. \end{aligned} \tag{3.5}$$

As f_t is linear, we have $\boldsymbol{J}_{f_t}(\boldsymbol{x}) = \boldsymbol{W}_{L,t} \boldsymbol{W}_{L-1,t} \cdot \ldots \cdot \boldsymbol{W}_{1,t}$. We may repeatedly apply the equality of Equation 3.5 to see that[2]

$$\begin{aligned} \left(\boldsymbol{J}_{f_t}(\boldsymbol{x})^\top \boldsymbol{J}_{f_t}(\boldsymbol{x})\right)^{1/L} &= \left((\boldsymbol{W}_{1,t}^\top \boldsymbol{W}_{1,t})^L\right)^{1/L} \\ &= \boldsymbol{W}_{1,t}^\top \boldsymbol{W}_{1,t}. \end{aligned} \tag{3.6}$$

□

Therefore, for deeper linear networks, the NFA holds with $\alpha = 1/L$, where L is the number of layers in the model.

Remark: We assumed that the network output was multivariate. The assumption that the layers are all balanced means that in the case of a univariate output, all of the layers would have to be rank 1 both at initialization and throughout training. Indeed, it has been shown that the weight matrices in deep linear networks converge to be rank 1 for classification tasks (Ji and Telgarsky, 2018).

3.1 Removing the Balancedness Assumption

For unbalanced initializations, Lemma 3.1 states that the weights in adjacent layers shall remain unbalanced through training. By introducing weight decay into the gradient flow dynamics, it can be shown that the weights in adjacent layers will become asymptotically balanced (Kobayashi et al., 2024). We recall the gradient flow dynamics with weight decay:

$$\frac{\partial \boldsymbol{W}_{l,t}}{\partial t} = -\frac{\partial \mathcal{L}(\boldsymbol{\theta}_t)}{\partial \boldsymbol{W}_{l,t}} - \lambda \boldsymbol{W}_{l,t}, \tag{3.7}$$

where $\lambda > 0$ is the weight decay parameter.

We also recall the following lemma which we will use to prove a similar result to Theorem 3.1 for unbalanced initialization.

Lemma 3.2 (Kobayashi et al. (2024))**.** *Suppose $\boldsymbol{W}_l$, $\boldsymbol{W}_{l+1}$ are the weight matrices of two adjacent layers of a neural network, that has a loss function differentiable with respect to $\hat{\boldsymbol{W}}_{l+1,l} := \boldsymbol{W}_{l+1} \cdot \boldsymbol{W}_l$. Suppose that the layers follow the gradient flow dynamics given by Equation 3.7 for $\lambda > 0$, then $\boldsymbol{W}_{l+1,t}^\top \boldsymbol{W}_{l+1,t} - \boldsymbol{W}_{l,t} \boldsymbol{W}_{l,t}^\top$ converges exponentially quickly to zero. In particular, $\boldsymbol{W}_{l,t} \boldsymbol{W}_{l,t}^\top - \boldsymbol{W}_{l+1,t}^\top \boldsymbol{W}_{l+1,t} = e^{-2\lambda t} \boldsymbol{C}_l$ where $\boldsymbol{C}_l = \boldsymbol{W}_{l,0} \boldsymbol{W}_{l,0}^\top - \boldsymbol{W}_{l+1,0}^\top \boldsymbol{W}_{l+1,0}$.*

Letting $c_{\max} := \max_l \|\boldsymbol{C}_l\|_F$ in Lemma 3.2, we can prove the following theorem, the proof of which is included in Appendix A.

Theorem 3.2 (NFA for deep linear networks, without balanced initialization)**.** *For time $t \geq 0$, let $f_t(\boldsymbol{x}) = \boldsymbol{W}_{L,t} \boldsymbol{W}_{L-1,t} \cdots \boldsymbol{W}_{1,t} \boldsymbol{x}$ for $\boldsymbol{x} \in \mathbb{R}^d$. Suppose that $\boldsymbol{W}_{1,t}, \boldsymbol{W}_{2,t}, \ldots, \boldsymbol{W}_{L,t}$ follow the gradient flow dynamics given by Equation 3.7 for $\lambda > 0$. Defining $c_{\max}$ as above, at any time $t > 0$, we have*

$$\|\boldsymbol{A}_{f,t} - \left(\boldsymbol{W}_{1,t}^\top \boldsymbol{W}_{1,t}\right)^L\|_F = \mathcal{O}(c_{\max} e^{-2\lambda t}). \tag{3.8}$$

where $\boldsymbol{A}_{f,t} = \boldsymbol{J}_{f_t}^\top \boldsymbol{J}_{f_t}$ and $\lambda > 0$ is the weight decay constant.

According to Wihler (2009), for two $d \times d$ positive semidefinite matrices $\boldsymbol{X}, \boldsymbol{Y}$,

$$\|\boldsymbol{X}^{1/L} - \boldsymbol{Y}^{1/L}\|_F \leq d^{(L-1)/(2L)} \|\boldsymbol{X} - \boldsymbol{Y}\|_F^{1/L}. \tag{3.9}$$

This result allows us to quantify directly the gap between the neural feature matrix $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ and the L-th principal square root of the AGOP matrix.

[2]For brevity, we include the explicit steps in Appendix A.

Corollary 3.1. *For time $t \geq 0$, let $f_t(\boldsymbol{x}) = \boldsymbol{W}_{L,t}\boldsymbol{W}_{L-1,t}\cdots\boldsymbol{W}_{1,t}\boldsymbol{x}$ for $\boldsymbol{x} \in \mathbb{R}^d$. Suppose that $\boldsymbol{W}_{1,t}, \boldsymbol{W}_{2,t}, \ldots, \boldsymbol{W}_{L,t}$ follow the gradient flow dynamics given by Equation 3.7 for $\lambda > 0$. Defining $c_{\max}$ as above, at any time $t > 0$, we have*

$$\|(\boldsymbol{A}_{f,t})^{1/L} - \boldsymbol{W}_{1,t}^\top \boldsymbol{W}_{1,t}\|_F = \mathcal{O}(c_{\max} e^{-2\lambda t/L}). \quad (3.10)$$

where $\boldsymbol{A}_{f,t} = \boldsymbol{J}_{f_t}^\top \boldsymbol{J}_{f_t}$ and $\lambda > 0$ is the weight decay constant.

Remark: Note that Lemma 3.2 holds for the linear part of networks of the form

$$f(\boldsymbol{x}) = \boldsymbol{a}^\top \phi(\boldsymbol{W}_L \boldsymbol{W}_{L-1} \cdot \ldots \cdot \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + b_2 \quad (3.11)$$

where ϕ is a differentiable activation (which can be seen as a classification head on top of the linear feature extraction layers). We consider this type of architecture in our numerical experiments (although with a ReLU activation function which is not differentiable at the origin). Specifically, the result is applicable to the evolution of $\tilde{f}(\boldsymbol{x}) = \boldsymbol{W}_L \cdot \ldots \cdot \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1$.

The Importance of Weight Decay. To illustrate our $\lambda > 0$ requirement for unbalanced initialization, suppose we have a two-layer network $f(x) = \boldsymbol{W}_2 \boldsymbol{W}_1 \boldsymbol{x}$, with $\boldsymbol{W}_1 = \mathrm{Diag}(a, 1, c)$, $\boldsymbol{W}_2 = \mathrm{Diag}(1/a, 1, 0)$ for some $a > 0$ and for arbitrary non-zero constant c, so that $f(\boldsymbol{x}) = x_1 + x_2$. Suppose further that f interpolates our training set. Then without weight decay, this network will not update under gradient flow as the loss function will have zero gradient. In this case, we have $\boldsymbol{W}_1^\top \boldsymbol{W}_1 = \mathrm{Diag}(a^2, 1, c^2)$ while $\boldsymbol{A}_f = \mathrm{Diag}(1, 1, 0)$. Thus proportionality cannot hold for any $\alpha > 0$. Supposing that weight decay was used, Theorem 3.2 would apply and the gradient flow would converge to a solution that satisfies the NFA (with equality rather than just proportionality), with $\alpha = 1/2$; in particular, we may expect the first elements of $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ (a and $1/a$ at initialization) to tend to a value close to 1 and c to go to 0 under training with weight decay.

4 THE NFA FOR NONLINEAR NETWORKS

Previous sections showed that the NFA holds for deep linear neural networks under suitable assumptions on the training process. We now provide a simple target function that can be exactly represented by a 2-layer network with ReLU activations in the hidden layer, but for which any such network would violate the NFA.

Example 1. *Suppose that $f^* : \mathbb{R}^2 \to \mathbb{R}$ is defined by $f^*(\boldsymbol{x}) = [x_1]_+ + [x_2]_+$ and that we have some data set $\{(\boldsymbol{x}_i, y_i)\}_{1 \leq i \leq N}$, with the $\boldsymbol{x}_i$s drawn from some distribution $\boldsymbol{X}$ that has equal probability for each of the four quadrants (e.g. the uniform distribution $U([-1,1]^2)$). We observe that f^* can be expressed exactly by a one-hidden-layer bias-free[3] neural network f with ReLU activation in the hidden layer:*

$$f(x) = \boldsymbol{a}^\top \phi(\boldsymbol{W}\boldsymbol{x}),$$

with

$$\boldsymbol{W} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}; \quad \boldsymbol{a} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}; \quad \boldsymbol{W}^\top \boldsymbol{W} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Moreover, since f^* has gradient discontinuities on the lines $x_1 = 0$ and $x_2 = 0$, the preactivations in the hidden layer will also have to align with these directions so that this is the only one-hidden-layer representation of this function up to rescaling of the rows of $\boldsymbol{W}$ and corresponding entries of $\boldsymbol{a}$. For any $\boldsymbol{x} \in \mathbb{R}^2$, we have

$$\nabla f(\boldsymbol{x}) = \begin{pmatrix} \mathbb{1}_{\{x_1 > 0\}} \\ \mathbb{1}_{\{x_2 > 0\}} \end{pmatrix};$$

$$\nabla f(\boldsymbol{x}) \nabla f(\boldsymbol{x})^\top = \begin{pmatrix} \mathbb{1}_{\{x_1 > 0\}} & \mathbb{1}_{\{x_1 > 0;\ x_2 > 0\}} \\ \mathbb{1}_{\{x_1 > 0;\ x_2 > 0\}} & \mathbb{1}_{\{x_2 > 0\}} \end{pmatrix},$$

where we use $\mathbb{1}$ to denote an indicator function. By our assumption on the training data distribution we have

$$\boldsymbol{E}_f := \mathbb{E}_{\boldsymbol{x} \sim \boldsymbol{X}} \left[\nabla f(\boldsymbol{x}) \nabla f(\boldsymbol{x})^\top \right] = \frac{1}{4} \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

Assuming N is large, we will have $\boldsymbol{A}_f \approx \boldsymbol{E}_f$. In fact, by the strong law of large numbers, we have $\boldsymbol{A}_f \to \boldsymbol{E}_f$ as $N \to \infty$. On the other hand, there is no power of $\alpha > 0$ such that $\boldsymbol{W}^\top \boldsymbol{W} \propto (\boldsymbol{E}_f)^\alpha$, and so the NFA does not hold in this setting.

From this counterexample, we may deduce that the NFA does not always hold for nonlinear networks (regardless of the value of $\alpha > 0$).

If we removed the ReLU activations from both the target function and network architecture in Example 1, there would be some values for the weights that would satisfy the NFA and some that would not. Here, the target function becomes $f^*(x) = x_1 + x_2$, with $\boldsymbol{A}_{f^*} = (1\ 1; 1\ 1)$. In particular, with the previously specified values of $\boldsymbol{W}$ and $\boldsymbol{a}$, the network function would represent f^*, but the NFA would not hold for any power α. However, we could choose $\boldsymbol{W} = (1\ 1; 1\ 1)$, $\boldsymbol{a} = (1/2\ 1/2)$. In this case, f would represent f^* and we would have $\boldsymbol{W}^\top \boldsymbol{W} = 2\boldsymbol{W} \propto (\boldsymbol{A}_f)^{1/2}$.

[3] The same conclusions could be reached even if biases were included in the target function f^* and the network architecture; in particular, $f^*(\boldsymbol{x}) = [x_1 + c_1]_+ + [x_2 + c_2]_+ + c_3$. This is because the nonlinearities of f^* would still dictate the directions in $\boldsymbol{W}$ in the same way.

What About Wider or Deeper Networks? In the above example, we considered a narrow two-layer network which was not overparametrized. The question remains: what happens for wider or deeper networks? In the case of overparametrized two-layer neural networks, we can add zero-weight connections to find networks for which f^* is interpolated and the NFA holds. For example, suppose that we have

$$\boldsymbol{W} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}; \quad \boldsymbol{a} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}; \quad \boldsymbol{W}^\top \boldsymbol{W} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix},$$

the NFA would hold exactly with $\alpha = 1$. Of course, the third neuron in the hidden layer would have no impact on the network output. By the universal approximation property, it can be shown that any function $f^* : \mathbb{R}^2 \to \mathbb{R}$ can be approximated by a sufficiently wide 2-layer network (with biases).

The NFA and Generalization. We next show that alignment between $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$ is neither necessary nor sufficient for f to be a good fit of f^*. To demonstrate a lack of sufficiency, we consider the following example:

Example 2. *For a given target function f^*, we may set $f(\boldsymbol{x}) := f^*(\boldsymbol{x}) + c$ for some constant c. This gives $\boldsymbol{A}_f = \boldsymbol{A}_{f^*}$, while f is a very poor fit of f^* for large values of c.*

To demonstrate that alignment between $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$ is not necessary for a model to generalize well, we consider the following:

Example 3. *For $n \geq 1$, define $f_n^* : \mathbb{R}^2 \to \mathbb{R}$ by*

$$f_n^*(\boldsymbol{x}) = \frac{1}{n}\cos(n^2 x_1) + x_2, \tag{4.1}$$

for $\boldsymbol{x} = (x_1,\ x_2)$. Note that

$$\nabla f_n^*(\boldsymbol{x}) = \begin{pmatrix} -n\sin(n^2 x_1) \\ 1 \end{pmatrix}$$

$$\nabla f_n^*(\boldsymbol{x}) \nabla f_n^*(\boldsymbol{x})^\top = \begin{pmatrix} n^2 \sin^2(n^2 x_1) & -n\sin(n^2 x_1) \\ -n\sin(n^2 x_1) & 1 \end{pmatrix}.$$

Assuming that we sample data points such that the distribution of x_1 is symmetric around the origin, for example, $x_1 \sim U[-\pi,\ \pi]$, the off-diagonal terms are zero. In this case, we have that

$$\boldsymbol{A}_{f_n^*} = \begin{pmatrix} \mathcal{O}(n^2) & 0 \\ 0 & 1 \end{pmatrix}. \tag{4.2}$$

Suppose we set $f(\boldsymbol{x}) := x_2$, then

$$\boldsymbol{A}_f = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \tag{4.3}$$

and hence the cosine similarity between $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f_n^*}$, $\rho(\boldsymbol{A}_f, \boldsymbol{A}_{f_n^*}) \to 0$ as $n \to \infty$. Notice that $\mathbb{E}[|f(\boldsymbol{x}) - f_n^*(\boldsymbol{x})|] < 1/n \to 0$ as $n \to \infty$.

There is therefore no direct implication between the NFA and model generalization.

Benefits of AGOP Alignment. Whilst there is no necessary or sufficient condition between $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$ alignment and generalization, such alignment is typically beneficial for the network to generalize to unseen data. In Equation 2.2, we see that $\boldsymbol{A}_f$ captures the input space directions that have the largest effect on the network output (estimated from the gradients at the data points). Consequently, alignment between $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$ indicates that the function f represented by the network and the target function f^* vary on average along the same main directions, which is desirable for model generalization. For low-rank (multi-index) functions, the range of $\boldsymbol{A}_{f^*}$ exactly captures the active subspace, here, the subspace in which the function varies (Constantine, 2015). This motivated Parkinson et al. (2025) to measure the largest principal angle between the principal (or leading) subspaces of $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$ to provide an explanation for generalization performance of networks that learn such low-rank functions. In such cases, it is essential for generalization that $\boldsymbol{A}_f$ captures the principal subspace of $\boldsymbol{A}_{f^*}$, so that f is invariant along the same directions in the input space as f^*. Other works such as Zhu et al. (2024) posit that this alignment between $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$ is key for predicting generalization, even in datasets where explicit low-rank structure is not present. They demonstrate empirically that optimization algorithms that lead to a better AGOP alignment typically lead to better performance when tested on unseen data.

Examples 2 and 3 capture worst-case behaviours rather than what may typically be seen in practice. Note that the alignment between $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$ can alternatively be seen as a relaxation of the interpolation condition of the gradients of the target function (which is typically not used in practice except in Sobolev learning, see Czarnecki et al. (2017)): $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_i) \approx \nabla_{\boldsymbol{x}} f^*(\boldsymbol{x}_i)$ at each training point input x_i. Indeed, gradient interpolation at all data points implies the equality between $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$, while the converse is not true since $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$ only provide average information on the alignment between the gradients. This is the idea behind our proposed counterexamples: an alignment between $\boldsymbol{A}_f$ and $\boldsymbol{A}_{f^*}$ does not explain generalization in cases where interpolating the target function gradient does not either. Indeed, in Example 2, for any data point x_i, we have $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_i) = \nabla_{\boldsymbol{x}} f^*(\boldsymbol{x}_i)$, but $f(\boldsymbol{x}_i) \neq f^*(\boldsymbol{x}_i)$, showing that gradient interpolation alone is not sufficient for retrieving the target function

f^*. In Example 3, we have a highly oscillatory function, for which the gradient information is not useful for predictions.

5 NUMERICAL EXPERIMENTS

We now conduct numerical experiments to verify the claims in section 3, as well as exploring further the low-dimensional structure resulting from the NFA when approximating low-rank functions. Across this section, we rely on the data generation mechanisms and training schemes proposed in Parkinson et al. (2025).

Data Generation. We consider low-rank target functions of the form

$$f^*(\boldsymbol{x}) = \boldsymbol{a}^\top g(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}),$$

for some matrix $\boldsymbol{A} \in \mathbb{R}^{r\times d}$, where g is some link function, and for input dimension $d = 20$. In this section, we let $g : x \mapsto [x]_+$ be the ReLU function applied elementwise and $r = 5$; results associated with $r \in \{2,\ 20\}$, and other link functions are presented in Appendix C. The data points are generated according to $\boldsymbol{x}_i \sim U([-1/2, 1/2]^{20})$ for all i, and $y_i = f^*(\boldsymbol{x}_i)$ (also see Appendix C for additional results with label noise). Unless stated otherwise, we use a dataset size of 2048 points. For the stochastic optimization methods, we used a batch size of 64.

5.1 Validating our Theoretical Results

Note that the theoretical results derived in section 3 were obtained under the assumption that the model training followed gradient flow dynamics. In practice, this algorithm is discretized and possibly replaced by stochastic optimization methods such as SGD with or without momentum, or Adam. This section aims to assess whether the findings of section 3 still (at least approximately) hold in these settings.

Architectures Considered. Following Parkinson et al. (2025), we considered here deep linear neural networks with a single ReLU final layer[4]:

$$f(\boldsymbol{x}) = \boldsymbol{a}^\top[\boldsymbol{W}_L\boldsymbol{W}_{L-1}\cdot\ldots\cdot\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1]_+ + b_2. \quad (5.1)$$

This network structure allows approximating more general functions than linear ones, but results for deep linear neural networks are provided in Appendix C. The number of layers is variable, but each hidden layer has width 64. To align with the theory of section 3, we consider how the initial linear part of this network evolves through training, namely, $\tilde{f}(\boldsymbol{x}) = \boldsymbol{W}_L\boldsymbol{W}_{L-1}\cdot\ldots\cdot\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1$. The Jacobian of this function is given by $\boldsymbol{J}_{\tilde{f}} = \boldsymbol{W}_L\boldsymbol{W}_{L-1}\cdot\ldots\cdot\boldsymbol{W}_1$ and we let $\boldsymbol{A}_{\tilde{f}} := \boldsymbol{J}_{\tilde{f}}^\top\boldsymbol{J}_{\tilde{f}}$. To assess the validity of the NFA, and in accordance with Radhakrishnan et al. (2024), we calculate the cosine similarity, whose definition is recalled next, between $(\boldsymbol{A}_{\tilde{f}})^\alpha$ and $\boldsymbol{W}_1^\top\boldsymbol{W}_1$ for various powers of α.

Initialization Schemes and Training Algorithms Considered. We consider both balanced and unbalanced initialization schemes. For the unbalanced initialization scheme, we use the default PyTorch initialization for linear layers[5], while the balanced initialization scheme is described in Appendix B.

We consider a variety of optimization algorithms for model training. In this section, we primarily include results for GD and for stochastic gradient descent (SGD), with and without momentum. Additional results for training networks with Adam (Kingma and Ba, 2014), and more results for gradient descent, are included in Appendix C. Following Parkinson et al. (2025), we train each model for $60,000$ epochs before reducing the learning rate by a factor of 10 and running an additional 100 epochs.

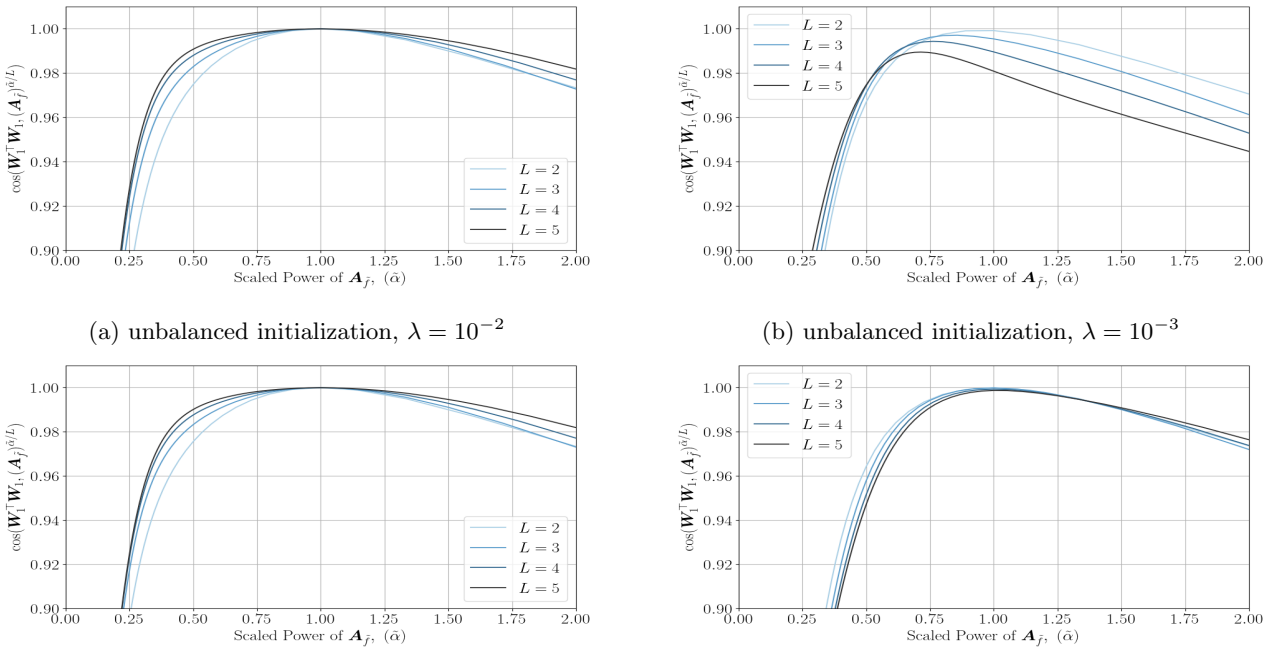
Results. Figure 2 displays the cosine similarity between the neural feature matrix $\boldsymbol{W}_1^\top\boldsymbol{W}_1$ and $\boldsymbol{A}_{\tilde{f}}$ as defined above, with respect to the NFA exponent α. In order to display on the same plot curves obtained from networks with different numbers of layers, we rescale α by the number of linear layers. In other words, the x-axis of Figure 2 is $\tilde{\alpha} := L\alpha$. According to Theorem 3.1, the cosine similarity should be greatest when $\tilde{\alpha} = 1$, which corresponds to $\alpha = 1/L$, which is indeed the case for balanced initialization (regardless of the weight decay parameter value and despite the fact that the training algorithm is SGD instead of a mere gradient flow dynamics in section 3). Note also that, as long as the weight decay parameter λ is sufficiently large, the cosine similarity is also maximal when $\tilde{\alpha} = 1$ for unbalanced initialization.

Figure 3 compares the impact of the choice of the optimizer on the NFA, showing that the NFA holds in both settings, and that furthermore momentum increases the rate at which $\boldsymbol{W}_1^\top\boldsymbol{W}_1$ and $(\boldsymbol{A}_{\tilde{f}})^{1/L}$ align for both SGD and GD.

Finally, regarding depth dependency, note that all our experiments show that the rate at which $\boldsymbol{W}_1^\top\boldsymbol{W}_1$ and $(\boldsymbol{A}_{\tilde{f}})^{1/L}$ align is slower for deeper networks than for shallower networks.

[4]Our notation differs as we have L linear layers prior to the nonlinearity rather than $L-1$.

[5]Each weight w in the lth layer is initialized as $w \sim U(-1/\sqrt{d_l},\ 1/\sqrt{d_l})$ where d_l is the in-degree of the lth layer.



(a) unbalanced initialization, $\lambda = 10^{-2}$

(b) unbalanced initialization, $\lambda = 10^{-3}$

(c) balanced initialization, $\lambda = 10^{-2}$

(d) balanced initialization, $\lambda = 10^{-3}$

Figure 2: Illustration of the impact of initialization and weight decay (λ) on the alignment between $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ and $(\boldsymbol{A}_f)^{1/L}$ at the end of training (SGD). A learning rate of $\eta = 10^{-4}$ was used.

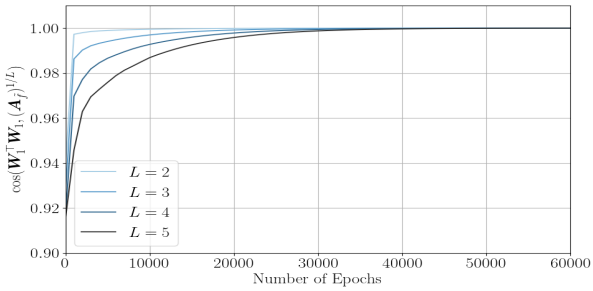
5.2 Recovering the Low-Rank Structure of the Target Function

Note that, since our target function f^* is low-rank, by Definition 2.1 $\nabla f^*(\boldsymbol{x}) \in \mathcal{T}$ for all $\boldsymbol{x} \in \mathbb{R}^d$, where $\mathcal{T}$ is the subspace of variation of f^*. Therefore, $\nabla f^*(\boldsymbol{x})^\top \boldsymbol{x}_\perp = 0$ for all $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{x}_\perp \in \mathcal{T}^\perp$ and hence $(\boldsymbol{x}_\perp)^\top \boldsymbol{A}_{f^*} \boldsymbol{x}_\perp = 0$ for all $\boldsymbol{x}_\perp \in \mathcal{T}^\perp$. As such, the low-rank structure of f^* is captured by $\boldsymbol{A}_{f^*}$ as $\mathcal{T}^\perp \subseteq \ker(\boldsymbol{A}_{f^*})$[6]. Proportionality between $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ and some power of $\boldsymbol{A}_f$ implies that the two matrices must have the same rank. Hence, supposing that the NFA holds and f has the same low-rank structure as the target function f^*, the first-layer weight matrix $\boldsymbol{W}_1$ must have low-rank structure, which we verify numerically in this section for a generic feedforward neural network architecture, with depth ranging from 2 to 5, width 64 and same initialization as above. We however have biases in each layer, as well as ReLU activations. The data generation mechanism is the same as in the previous section, but we also experiment with $N = 8192$ alongside $N = 2048$. The model was trained with SGD with weight decay ($\lambda = 10^{-3}$) and momentum ($\beta = 0.9$). Figure 4 shows the distribution of the singular values of $\boldsymbol{W}_1$, for ReLU networks of various depths. Note that the low-rank structure of the target function f^* is indeed captured by the first-layer weight matrix for all network depths considered, and that this becomes increasingly accurate as the number of data points increases, allowing a better learning of the low-rank structure in the target function.

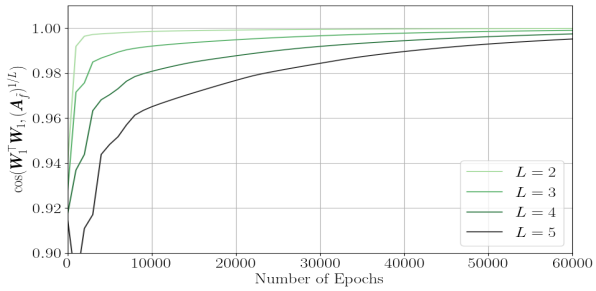
[6]We use a "subset" notation here as the sample data points do not cover the entire domain.

6 CONCLUSION

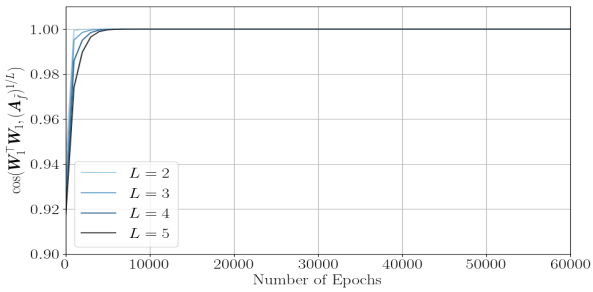
We have shown that under gradient flow and balanced initialization, the NFA holds for deep linear networks with a depth-dependent exponent. Furthermore, assuming that weight decay is applied, the NFA holds asymptotically regardless of initialization. As a barrier to extending the NFA for linear networks to more general feedforward neural networks, in section 4, we show that there exist functions that can be expressed by a given architecture but for which perfect proportionality is not attainable between some power of the AGOP and $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ for that architecture. In section 5, we illustrate that our theoretical results for gradient flow continue to hold when applying diverse training algorithms. We also illustrate that in the case of nonlinear networks, $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ has the same low-rank structure as the AGOP of the target function, which indicates that the low-rank behaviour observed for linear networks may be extendable in the future to nonlinear ones.



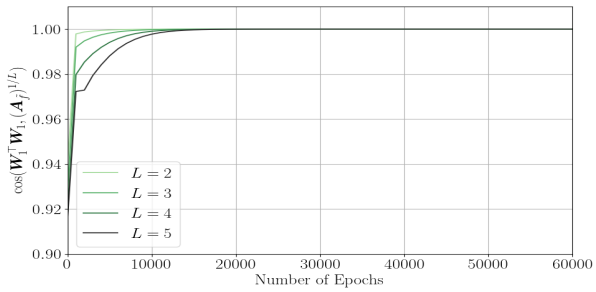
(a) SGD, no momentum



(b) GD, no momentum

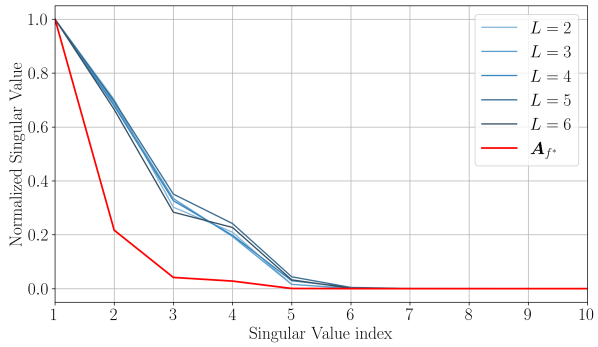


(c) SGD, momentum

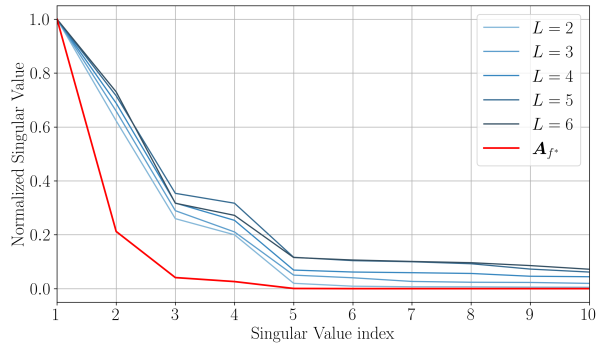


(d) GD, momentum

Figure 3: Illustration of the impact of the optimization algorithm on the NFA, with weight decay ($\lambda = 10^{-2}$). When momentum is used, it is weighted by a parameter $\beta = 0.9$. The learning rates for SGD and GD are $\eta = 10^{-4}$ and $\eta = 10^{-3}$, respectively. Here we plot the first $60,000$ epochs before the learning rate decrease.



(a) 8192 Data points



(b) 2048 Data points

Figure 4: Singular values decay for $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ vs $\boldsymbol{A}_{f^*}$. Normalized Singular Values are computed by dividing by σ_1 for the respective matrix

Acknowledgments

ET's work was supported by the EPSRC Centre for Doctoral Training in Mathematics of Random Systems: Analysis, Modelling and Simulation (EP/S023925/1). CC's work was supported by the Hong Kong Innovation and Technology Commission (InnoHK Project CIMDA) and by the EPSRC grant EP/Y028872/1, Mathematical Foundations of Intelligence: An "Erlangen Programme" for AI.

References

Arora, S., Cohen, N., and Hazan, E. (2018). On the Optimization of Deep Networks: Implicit Acceleration by Overparameterization. In *Proceedings of the 35th International Conference on Machine Learning*, pages 244–253. PMLR.

Beaglehole, D., Súkeník, P., Mondelli, M., and Belkin, M. (2024). Average gradient outer product as a mechanism for deep neural collapse. *Advances in Neural Information Processing Systems*, 37:130764–130796.

Boix-Adsera, E., Mallinar, N., Simon, J. B., and Belkin, M. (2025). The Features at Convergence Theorem: a first-principles alternative to the Neural Feature Ansatz for how networks learn representations. arXiv:2507.05644 [cs].

Bruna, J. and Hsu, D. (2025). Survey on Algorithms for multi-index models. arXiv:2504.05426 [stat].

Cartis, C., Liang, X., Massart, E., and Otemissov, A. (2024). Learning the subspace of variation for global optimization of functions with low effective dimension. arXiv:2401.17825 [math].

Constantine, P. G. (2015). *Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Czarnecki, W. M., Osindero, S., Jaderberg, M., Świrszcz, G., and Pascanu, R. (2017). Sobolev Training for Neural Networks. Version Number: 3.

Guth, F., Ménard, B., Rochette, G., and Mallat, S. (2024). A Rainbow in Deep Network Black Boxes. *Journal of Machine Learning Research*, 25(350):1–59.

Ji, Z. and Telgarsky, M. (2018). Gradient descent aligns the layers of deep linear networks.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs].

Kobayashi, S., Akram, Y., and Oswald, J. V. (2024). Weight decay induces low-rank attention layers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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

Mallinar, N. R., Beaglehole, D., Zhu, L., Radhakrishnan, A., Pandit, P., and Belkin, M. (2025). Emergence in non-neural models: grokking modular arithmetic via average gradient outer product. In *Forty-second International Conference on Machine Learning*.

Meckes, E. S. (2019). *The Random Matrix Theory of the Classical Compact Groups*. Cambridge Tracts in Mathematics. Cambridge University Press, Cambridge.

Mousavi-Hosseini, A., Javanmard, A., and Erdogdu, M. A. (2025). Robust Feature Learning for Multi-Index Models in High Dimensions. In *The Thirteenth International Conference on Learning Representations*.

Parkinson, S., Ongie, G., and Willett, R. (2023). ReLU Neural Networks with Linear Layers are Biased Towards Single- and Multi-Index Models. arXiv:2305.15598 [cs, stat].

Parkinson, S., Ongie, G., and Willett, R. (2025). ReLU Neural Networks with Linear Layers Are Biased towards Single- and Multi-index Models. *SIAM Journal on Mathematics of Data Science*.

Radhakrishnan, A., Beaglehole, D., Pandit, P., and Belkin, M. (2024). Mechanism for feature learning in neural networks and backpropagation-free machine learning models. *Science*, 383(6690):1461–1467.

Radhakrishnan, A., Belkin, M., and Drusvyatskiy, D. (2025). Linear Recursive Feature Machines provably recover low-rank matrices. *Proceedings of the National Academy of Sciences*, 122(13):e2411325122.

Tyagi, H. and Cevher, V. (2014). Learning non-parametric basis independent models from point queries via low-rank methods. *Applied and Computational Harmonic Analysis*, 37(3):389–412.

Vardi, G. (2023). On the Implicit Bias in Deep-Learning Algorithms. *Commun. ACM*, 66(6):86–93.

Wihler, T. P. (2009). On the Hölder Continuity of Matrix Functions for Normal Matrices. *Journal of Inequalities in Pure and Applied Mathematics*, 10(4).

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115.

Zhu, L., Liu, C., Radhakrishnan, A., and Belkin, M. (2024). Catapults in SGD: spikes in the training

loss and their impact on generalization through feature learning. In *Proceedings of the 41st International Conference on Machine Learning*, pages 62476–62509. PMLR.

Ziyin, L., Chuang, I. L., Galanti, T., and Poggio, T. A. (2025). Formation of Representations in Neural Networks. In *The Thirteenth International Conference on Learning Representations*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes
   (b) Complete proofs of all theoretical results. Yes
   (c) Clear explanations of any assumptions. Yes

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Not Applicable
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Not Applicable

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes
   (b) The license information of the assets, if applicable. Yes
   (c) New assets either in the supplemental material or as a URL, if applicable. Yes
   (d) Information about consent from data providers/curators. Not Applicable
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable

A Omitted Proofs

A.1 Derivation of Equation 3.6

Using the definition of $\boldsymbol{J}_{f_t}$, we obtain the first equality below,

$$\begin{aligned}
\left(\boldsymbol{J}_{f_t}(\boldsymbol{x})^\top \boldsymbol{J}_{f_t}(\boldsymbol{x})\right)^{1/L} &= \left(\boldsymbol{W}_{1,t}^\top \cdots \boldsymbol{W}_{L-1,t}^\top \boldsymbol{W}_{L,t}^\top \boldsymbol{W}_{L,t} \boldsymbol{W}_{L-1,t} \cdots \boldsymbol{W}_{1,t}\right)^{1/L} \\
&= \left(\boldsymbol{W}_{1,t}^\top \cdots \boldsymbol{W}_{L-2,t}^\top (\boldsymbol{W}_{L-1,t}^\top \boldsymbol{W}_{L-1,t})^2 \boldsymbol{W}_{L-2,t} \cdots \boldsymbol{W}_{1,t}\right)^{1/L} \\
&= \left(\boldsymbol{W}_{1,t}^\top \cdots \boldsymbol{W}_{L-3,t}^\top (\boldsymbol{W}_{L-2,t}^\top \boldsymbol{W}_{L-2,t})^3 \boldsymbol{W}_{L-3,t} \cdots \boldsymbol{W}_{1,t}\right)^{1/L} \\
&\vdots \\
&= \left(\boldsymbol{W}_{1,t}^\top (\boldsymbol{W}_{2,t}^\top \boldsymbol{W}_{2,t})^{L-1} \boldsymbol{W}_{1,t}\right)^{1/L} \\
&= \left((\boldsymbol{W}_{1,t}^\top \boldsymbol{W}_{1,t})^L\right)^{1/L} \\
&= \boldsymbol{W}_{1,t}^\top \boldsymbol{W}_{1,t},
\end{aligned}$$

where to obtain the remaining equalities, we repeatedly apply equation 3.5.

A.2 Proof of Theorem 3.2

We prove the following two lemmas to simplify the proof of this theorem. The result of Lemma A.1, that the weight matrices will remain bounded through the gradient flow trajectory is known (for example, see "Sufficient Condition 1" in Kobayashi et al. (2024)). Through the proof of the result below, we make the bound C_F explicit as a function of the network weights at initialization.

Lemma A.1. *For $t \geq 0$, let $f_t(\boldsymbol{x}) = \boldsymbol{W}_{L,t}\boldsymbol{W}_{L-1,t}\cdots\boldsymbol{W}_{1,t}\boldsymbol{x}$ for $\boldsymbol{x} \in \mathbb{R}^d$. Suppose that $\boldsymbol{W}_{1,t}, \boldsymbol{W}_{2,t}, \ldots, \boldsymbol{W}_{L,t}$ follow the gradient flow dynamics given by Equation 3.7 for $\lambda > 0$. Assume that a continuously differentiable loss function $\mathcal{L}$ is bounded below, i.e., there exists $\mathcal{L}_{low}$ such that $\mathcal{L}(\boldsymbol{\theta}) \geq \mathcal{L}_{low}$ for all $\boldsymbol{\theta}$. Then there exists some constant C_F such that $\|\boldsymbol{W}_{l,t}\|_F \leq C_F$ for all $l \in \{1, \ldots, L\}$ and for all $t \geq 0$.*

Proof. Note that equation 3.7 is the gradient flow of the regularized loss function

$$\hat{\mathcal{L}}_\lambda(\boldsymbol{\theta}) := \mathcal{L}(\boldsymbol{\theta}) + \frac{\lambda}{2}\sum_{l=1}^{L}\|\boldsymbol{W}_l\|_F^2. \tag{A.1}$$

Therefore, as $\mathcal{L}$ is continuously differentiable, $\mathcal{L}_\lambda\hat{(}\boldsymbol{\theta}_t)$ is monotonically decreasing with respect to t. For all $t \geq 0$, we have that

$$\mathcal{L}(\boldsymbol{\theta}_t) + \frac{\lambda}{2}\sum_{l=1}^{L}\|\boldsymbol{W}_{l,t}\|_F^2 \leq \mathcal{L}(\boldsymbol{\theta}_0) + \frac{\lambda}{2}\sum_{l=1}^{L}\|\boldsymbol{W}_{l,0}\|_F^2, \tag{A.2}$$

so that, for any $l \in \{1, \ldots, L\},\ t \geq 0$,

$$\|\boldsymbol{W}_{l,t}\|_F^2 \leq \frac{2}{\lambda}\left(\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}_t) + \frac{\lambda}{2}\sum_{j=1}^{L}\|\boldsymbol{W}_{j,0}\|_F^2 - \frac{\lambda}{2}\sum_{\substack{j=1\\ j\neq l}}^{L}\|\boldsymbol{W}_{j,t}\|_F^2\right) \tag{A.3}$$

$$\leq \frac{2}{\lambda}\left(\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}_{low} + \frac{\lambda}{2}\sum_{j=1}^{L}\|\boldsymbol{W}_{j,0}\|_F^2\right) =: C_F^2. \tag{A.4}$$

Hence, $\|\boldsymbol{W}_{l,t}\|_F \leq C_F$, for all $l \in \{1, \ldots, L\}$ and for all $t \geq 0$. □

The following lemma may be thought of as a generalization of equation 3.5 to the case of unbalanced layers.

Lemma A.2. *For any $l \in \{1, \dots, L\}$, let $\boldsymbol{D}_{l,t} := (\prod_{j=1}^{l-1} \boldsymbol{W}_{j,t}^\top)(\boldsymbol{W}_{l,t}^\top \boldsymbol{W}_{l,t})^{L-l+1}(\prod_{j=1}^{l-1} \boldsymbol{W}_{j,t}^\top)^\top$, then $\|\boldsymbol{D}_{l+1,t} - \boldsymbol{D}_{l,t}\|_F \leq 2^{(L-l)} e^{-2\lambda t} c_{\max} C_F^{2(L-l)}$, where C_F is defined above and $c_{\max} := \max_l \|\boldsymbol{C}_l\|_F$, for $\boldsymbol{C}_l := \boldsymbol{W}_{l,0}\boldsymbol{W}_{l,0}^\top - \boldsymbol{W}_{l+1,0}^\top \boldsymbol{W}_{l+1,0}$.*

Proof. From the definitions of $\boldsymbol{D}_{l,t}$, $\boldsymbol{D}_{l+1,t}$, we have

$$\boldsymbol{D}_{l+1,t} - \boldsymbol{D}_{l,t} = (\prod_{j=1}^{l-1} \boldsymbol{W}_{j,t}^\top)[\boldsymbol{W}_{l,t}^\top(\boldsymbol{W}_{l+1,t}^\top \boldsymbol{W}_{l+1,t})^{L-l}\boldsymbol{W}_{l,t} - (\boldsymbol{W}_{l,t}^\top \boldsymbol{W}_{l,t})^{L-l+1}](\prod_{j=1}^{l-1} \boldsymbol{W}_{j,t}^\top)^\top. \tag{A.5}$$

Then, by the submultiplicity of the Frobenius norm and by Lemma A.1,

$$\|\boldsymbol{D}_{l+1,t} - \boldsymbol{D}_l\|_F \leq (\prod_{j=1}^{l-1} \|\boldsymbol{W}_{j,t}^\top\|_F) \cdot \|\boldsymbol{W}_{l,t}^\top(\boldsymbol{W}_{l+1,t}^\top \boldsymbol{W}_{l+1,t})^{L-l}\boldsymbol{W}_{l,t} - (\boldsymbol{W}_{l,t}^\top \boldsymbol{W}_{l,t})^{L-l+1}\|_F \cdot (\prod_{j=1}^{l-1} \|\boldsymbol{W}_{j,t}^\top\|_F) \tag{A.6}$$

$$\leq C_F^{2(l-1)} \|\boldsymbol{W}_{l,t}^\top(\boldsymbol{W}_{l+1,t}^\top \boldsymbol{W}_{l+1,t})^{L-l}\boldsymbol{W}_{l,t} - (\boldsymbol{W}_{l,t}^\top \boldsymbol{W}_{l,t})^{L-l+1}\|_F. \tag{A.7}$$

From Lemma 3.2, we have that $\boldsymbol{W}_{l+1,t}^\top \boldsymbol{W}_{l+1,t} = \boldsymbol{W}_{l,t}\boldsymbol{W}_{l,t}^\top - e^{-2\lambda t}\boldsymbol{C}_l$. It follows

$$\|\boldsymbol{W}_{l,t}^\top(\boldsymbol{W}_{l+1,t}^\top \boldsymbol{W}_{l+1,t})^{L-l}\boldsymbol{W}_{l,t} - (\boldsymbol{W}_{l,t}^\top \boldsymbol{W}_{l,t})^{L-l+1}\|_F = \|\boldsymbol{W}_{l,t}^\top(\boldsymbol{W}_{l,t}\boldsymbol{W}_{l,t}^\top - e^{-2\lambda t}\boldsymbol{C}_l)^{L-l}\boldsymbol{W}_{l,t} - (\boldsymbol{W}_{l,t}^\top \boldsymbol{W}_{l,t})^{L-l+1}\|_F \tag{A.8}$$

$$= \|\sum_{j=1}^{2^{(L-l)}-1} \boldsymbol{W}_{l,t}^\top \boldsymbol{T}_{j,t} \boldsymbol{W}_{l,t}\|_F \tag{A.9}$$

$$\leq C_F^2 \sum_{j=1}^{2^{(L-l)}-1} \|\boldsymbol{T}_{j,t}\|_F, \tag{A.10}$$

where the $\boldsymbol{T}_{j,t}$s correspond to the terms in the binomial expansion of $(\boldsymbol{W}_{l,t}\boldsymbol{W}_{l,t}^\top + e^{-2\lambda t}\boldsymbol{C}_l)^{L-l}$ that have at least one power of $e^{-2\lambda t}\boldsymbol{C}_l$, which is all terms other than $(\boldsymbol{W}_{l,t}\boldsymbol{W}_{l,t}^\top)^{L-l}$ (indeed, the term $(\boldsymbol{W}_{l,t}\boldsymbol{W}_{l,t}^\top)^{L-l}$ will cancel with the second term of the right-hand side of equation A.8). There are $2^{(L-l)} - 1$ such terms. For each of these terms, assuming that $e^{-2\lambda t} c_{\max} \leq C_F$, which holds asymptotically in time, we have $\|\boldsymbol{T}_{j,t}\|_F \leq e^{-2\lambda t} c_{\max} C_F^{2(L-l-1)}$. It follows that for t sufficiently large

$$\|\boldsymbol{D}_{l+1,t} - \boldsymbol{D}_l\|_F \leq C_F^2(2^{(L-l)} - 1)e^{-2\lambda t} c_{\max} C_F^{2(L-l-1)} \leq 2^{(L-l)} e^{-2\lambda t} c_{\max} C_F^{2(L-l)}. \tag{A.11}$$

□

Using these two above Lemmas, we are now able to prove Theorem 3.2.

Proof of Theorem 3.2. Observing that $\boldsymbol{A}_{f_t} = \boldsymbol{D}_{L,t}$ and $(\boldsymbol{W}_{1,t}^\top \boldsymbol{W}_{1,t})^L = \boldsymbol{D}_{1,t}$, we may form a telescoping sum and use the triangle inequality, asymptotically through time to show that we have

$$\|\boldsymbol{A}_{f,t} - \left(\boldsymbol{W}_{1,t}^\top \boldsymbol{W}_{1,t}\right)^L\|_F = \|\sum_{l=1}^{L-1}(\boldsymbol{D}_{l+1,t} - \boldsymbol{D}_{l,t})\|_F \tag{A.12}$$

$$\leq \sum_{l=1}^{L-1} \|\boldsymbol{D}_{l+1,t} - \boldsymbol{D}_{l,t}\|_F \tag{A.13}$$

$$\leq \sum_{l=1}^{L-1} 2^{(L-l)} e^{-2\lambda t} c_{\max} \hat{C}_F^{2(L-l)} \tag{A.14}$$

$$\leq e^{-2\lambda t} c_{\max} \hat{C}_F^{2L} \sum_{l=1}^{L-1} 2^{(L-l)} \tag{A.15}$$

$$= 2^L e^{-2\lambda t} c_{\max} \hat{C}_F^{2L}, \tag{A.16}$$

where $\hat{C}_F = \max(C_F,\ 1)$. As $\lambda > 0$ by assumption, this expression decays exponentially to zero as $t \to \infty$. $\square$

B Experiment Setup

In this section, we include further details on the settings for our numerical experiments. Our code is available at https://github.com/edtansley/nfa-experiments

Forcing Balanced Initialization The proof of Theorem 3.1 relies upon a balanced initialization of the weight matrices. We describe here the scheme we used to ensure that the initialization of the network is balanced, which is used in the experiments to produce Figure 2. Let d_l denote the in-degree of the lth layer. In our experiments on synthetic data, we have $d_1 = 20$, but $d_l = 64$ for $l = 2, \ldots, L$. We state the following Lemma, which for completeness, we prove in Appendix D[7].

Lemma B.1. *Suppose the entries of the weight matrix $\boldsymbol{W}_l \in \mathbb{R}^{d_{l+1} \times d_l}$ are drawn independently according to the uniform distribution $U(-1/\sqrt{d_l}, 1/\sqrt{d_l})$, then*

$$\mathbb{E}[\|\boldsymbol{W}_l^\top \boldsymbol{W}_l\|_F^2] = \frac{d_{l+1}}{d_l}\left(\frac{1}{5} + \frac{d_l + d_{l+1} - 2}{9}\right), \tag{B.1}$$

As a result, $\mathbb{E}\left[\|\boldsymbol{W}_1 \boldsymbol{W}_1^\top\|_F^2\right] \neq \mathbb{E}\left[\|\boldsymbol{W}_2^\top \boldsymbol{W}_2\|_F^2\right]$. Thus we use the following scaling of the weights of $\boldsymbol{W}_1$:

$$\tilde{\boldsymbol{W}}_1 = \sqrt{\frac{d_1 d_3\,(5d_2 + 5d_3 - 1)}{d_2^2\,(5d_1 + 5d_2 - 1)}} \cdot \boldsymbol{W}_1. \tag{B.2}$$

This correction ensures that $\mathbb{E}[\|\tilde{\boldsymbol{W}}_1 \tilde{\boldsymbol{W}}_1^\top\|_F^2] = \mathbb{E}[\|\boldsymbol{W}_2^\top \boldsymbol{W}_2\|_F^2]$.

We apply this correction at the start of our procedure to enforce a balanced initialization. We use $\texttt{Haar}(d_{l+1}, d_1)$ to denote the leading d_1 columns of a $d_{l+1} \times d_{l+1}$ Haar-distributed (Meckes, 2019) random matrix. This assumes that $d_{l+1} \geq d_1$, which holds for the architectures for which we experiment with balanced initialization. Note that we only balance the linear section of the networks. The output layer is not modified.

Algorithm 1 Force balancedness between layers of a network

Input: $\boldsymbol{W}_1$
Output: $\tilde{\boldsymbol{W}}_1, \ldots, \tilde{\boldsymbol{W}}_L$, with $\tilde{\boldsymbol{W}}_l \tilde{\boldsymbol{W}}_l^\top = \tilde{\boldsymbol{W}}_{l+1}^\top \tilde{\boldsymbol{W}}_{l+1} \quad \forall l \leftarrow 1, \ldots, L-1$
  $\tilde{\boldsymbol{W}}_1 \leftarrow \boldsymbol{W}_1$ ▷ (Using the above formula)
  $\boldsymbol{U}_1, \boldsymbol{\Sigma}, \boldsymbol{V}_1 \leftarrow \texttt{SVD}(\tilde{\boldsymbol{W}}_1)$ ▷ (Using the reduced SVD)
  for $l \leftarrow 2$ to L **do**
    $\boldsymbol{U}_l \leftarrow \texttt{Haar}(d_{l+1}, d_1)$
    $\tilde{\boldsymbol{W}}_l \leftarrow \boldsymbol{U}_l \boldsymbol{\Sigma} \boldsymbol{U}_{l-1}^\top$
  return $\tilde{\boldsymbol{W}}_1, \ldots, \tilde{\boldsymbol{W}}_L$

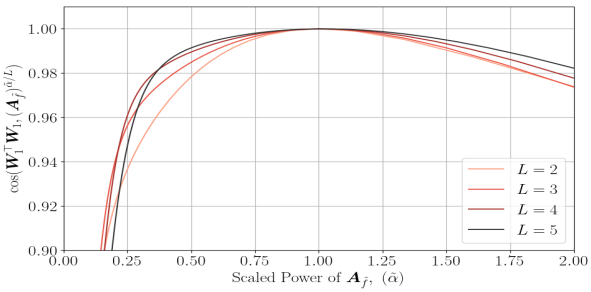
To confirm that this initialization is balanced, observe that for $l = 1, \ldots, L-1$, we have

$$\begin{aligned}
\tilde{\boldsymbol{W}}_{l+1}^\top \tilde{\boldsymbol{W}}_{l+1} &= (\boldsymbol{U}_{l+1} \boldsymbol{\Sigma} \boldsymbol{U}_l^\top)^\top \boldsymbol{U}_{l+1} \boldsymbol{\Sigma} U_l^\top \\
&= \boldsymbol{U}_l \boldsymbol{\Sigma} \boldsymbol{U}_{l+1}^\top \boldsymbol{U}_{l+1} \boldsymbol{\Sigma} \boldsymbol{U}_l^\top \\
&= \boldsymbol{U}_l \boldsymbol{\Sigma} \boldsymbol{U}_{l-1}^\top \boldsymbol{U}_{l-1} \boldsymbol{\Sigma} \boldsymbol{U}_l^\top \\
&= \boldsymbol{U}_l \boldsymbol{\Sigma} \boldsymbol{U}_{l-1}^\top (\boldsymbol{U}_l \boldsymbol{\Sigma} \boldsymbol{U}_{l-1}^\top)^\top \\
&= \tilde{\boldsymbol{W}}_l \tilde{\boldsymbol{W}}_l^\top.
\end{aligned}$$

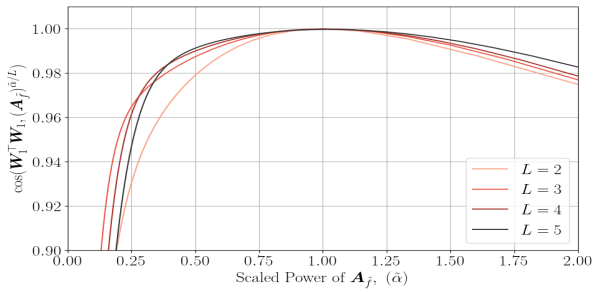
C Additional numerical experiments

In this section, we present additional numerical experiments. In section 5, we include results for SGD and GD with and without momentum, learning rank-5 functions with ReLU link functions. We include here results

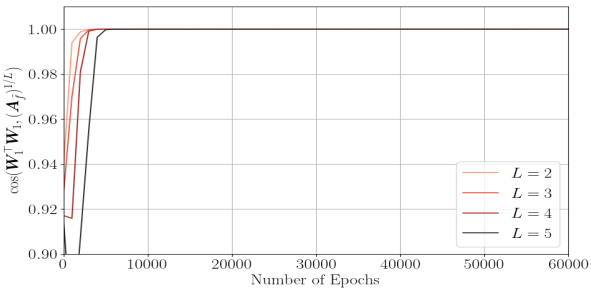
[7] We consider the square of the Frobenius norm to simplify the algebraic expressions.



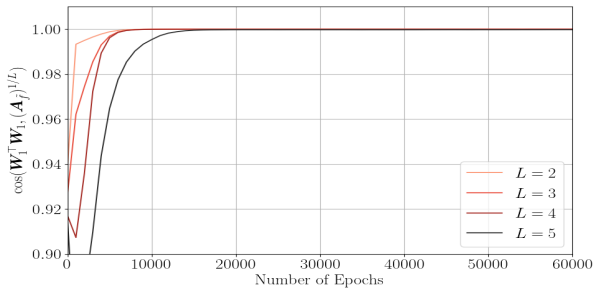
(a) $\lambda = 10^{-2}$



(b) $\lambda = 10^{-3}$



(c) $\lambda = 10^{-2}$



(d) $\lambda = 10^{-3}$

Figure 5: Illustration of the impact of the weight decay (λ) on the alignment between $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ and $(\boldsymbol{A}_f)^{1/L}$ at the end of training (Adam). A learning rate of $\eta = 10^{-4}$ was used.

for the Adam optimization algorithm, as well as results for rank-2 and rank-20 target functions (note that the latter are full rank) and target functions with different link functions. In addition, we conduct experiments on the MNIST (Lecun et al., 1998) dataset, to test whether we see similar results when testing on non-synthetic datasets.

Results for Adam. Here, we replicate Figures 2 and 3, replacing SGD by the Adam optimization algorithm. The results are included in Figure 5. Compared to SGD in Figure 2, we see that Adam has better alignment after training, when $\lambda = 10^{-3}$, which may be explained by the effect of momentum. When comparing Adam with SGD with momentum (see Figure 3), we see that the alignment happens for the former at a slightly slower rate than for the latter when $\lambda = 10^{-2}$ (seen in plot (c) of both figures). Since the results for Adam and SGD with momentum are comparable, this suggests that the theoretical results from section 3 may be applicable to more general training schemes than GD (which is a discretization of the gradient flow) and vanilla SGD.

Changing the Link Function. Instead of the ReLU function, we now show experiments where the data is generated through a Gaussian function, which is given by $g(x) = \exp(-x^2)$, applied element-wise, in the same manner as the ReLU function in section 5. The results are included in Figure 6. When training the networks with a weight decay rate of $\lambda = 10^{-2}$, the weights of the network tend to zero, and the network does not learn the target function. Additionally, for both GD and SGD, momentum is required to successfully learn the target function in this instance. We see that the alignment is slower for the Gauss function than the ReLU link function.

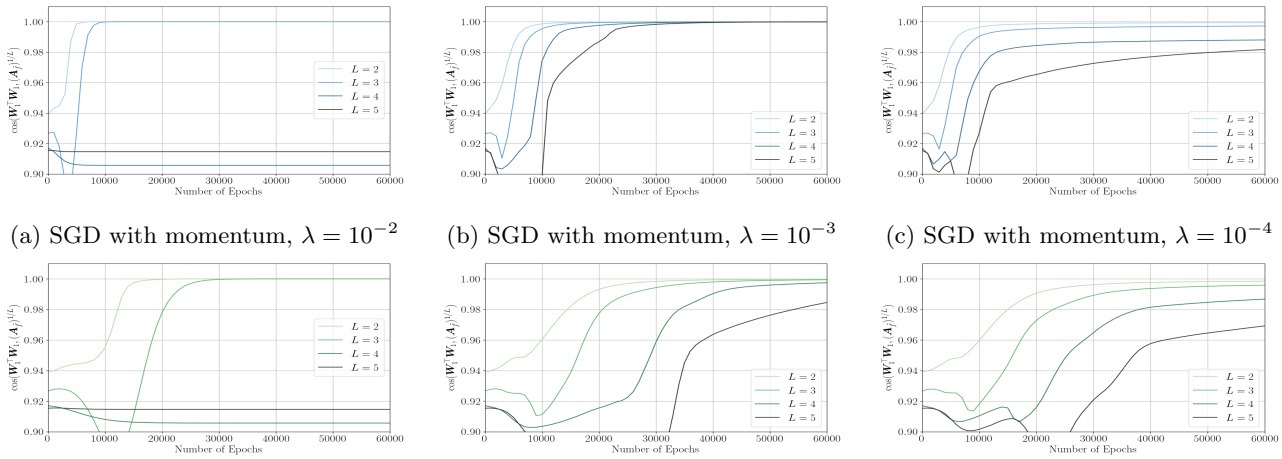
Fully Linear Networks. We now include results for fully linear networks, as these match the theoretical results from section 3. Here the architecture is given by:

$$f(\boldsymbol{x}) = \boldsymbol{W}_L \boldsymbol{W}_{L-1} \cdot \ldots \cdot \boldsymbol{W}_1 \boldsymbol{x} + b_1. \tag{C.1}$$

As target function, we take

$$f^*(\boldsymbol{x}) = g(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}), \tag{C.2}$$

where we either let g be the identity function, so that $g(x) = x$, or we let $g : x \mapsto [x]_+$ be the ReLU function applied elementwise. In both cases, $\boldsymbol{A} \in \mathbb{R}^{21 \times 20}$ is a rank-5 matrix, meaning the output dimension is 21, but



(a) SGD with momentum, $\lambda = 10^{-2}$ (b) SGD with momentum, $\lambda = 10^{-3}$ (c) SGD with momentum, $\lambda = 10^{-4}$

(d) GD with momentum, $\lambda = 10^{-2}$ (e) GD with momentum, $\lambda = 10^{-3}$ (f) GD with momentum, $\lambda = 10^{-4}$

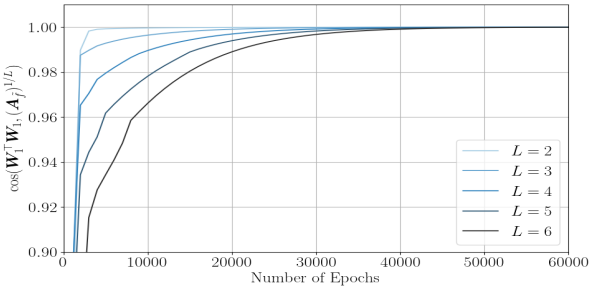
Figure 6: Illustration of the impact of the optimization algorithm on the NFA, varying the weight decay, for the Gauss link function. The momentum parameter is set to $\beta = 0.9$. The learning rates for SGD with momentum and GD with momentum are $\eta = 10^{-4}$ and $\eta = 10^{-3}$, respectively. When the weight decay is set to $\lambda = 10^{-2}$, the network weights tend to zero and the function is not learnt.

there is low-rank structure in the problem. In the case of the ReLU function, the linear network cannot express the target data exactly, but will learn a linear function that best fits the data.

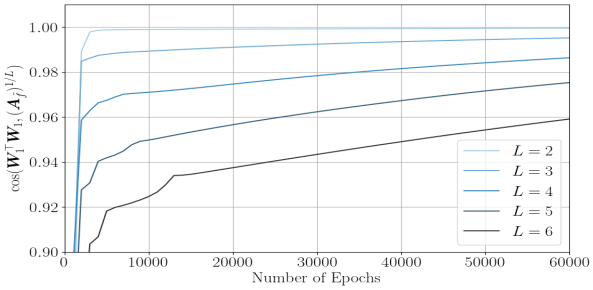
The results are included in Figure 7. The results appear to be similar when learning the two functions. Furthermore, compared to Figure 3 (a), the results are qualitatively similar to the results for networks with a ReLU layer, although with slightly slower convergence.

Experiments on MNIST. In addition to our experiments on synthetic data, we also present results on the MNIST dataset of handwritten digits. Note that this is a classification task, in contrast with the regression tasks that were addressed in section 5, as such the loss function and architectures used shall differ from our earlier experiments. We use the same optimization algorithms and hyperparameters as our other numerical experiments. However, we now run our experiments for 200 epochs, as we observe that this is typically sufficient for the training on this data set. In our experiments, training accuracy for networks trained with Adam is typically $\geq 97\%$, whilst the training accuracy for those trained with SGD is typically $\geq 95\%$. We see that the weight matrices align more quickly when Adam is used than they do when SGD with momentum is employed and that the rate depends on the weight decay parameter λ. We include the results in Figure 8.

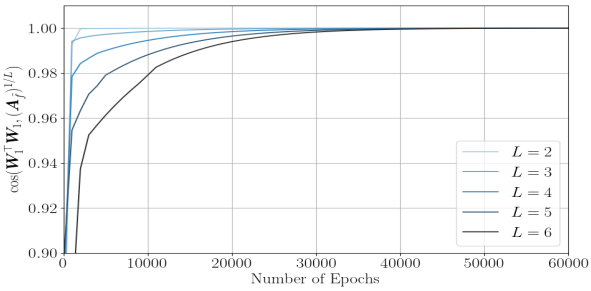
As we obtain similar results when learning to classify the MNIST dataset as we do for the regression task on synthetic data, this suggests that our theoretical results for the NFA on deep networks in section 3 may be applicable to a broader range of tasks than those considered in section 5.



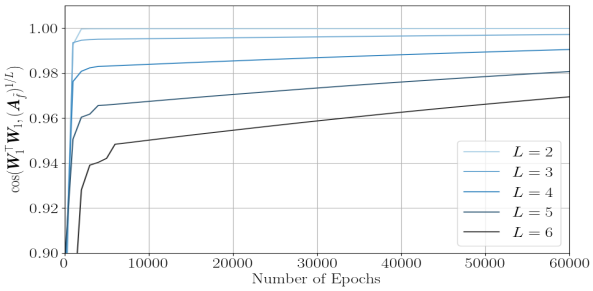
(a) $g(x) = x$, $\lambda = 10^{-2}$



(b) $g(x) = x$, $\lambda = 10^{-3}$

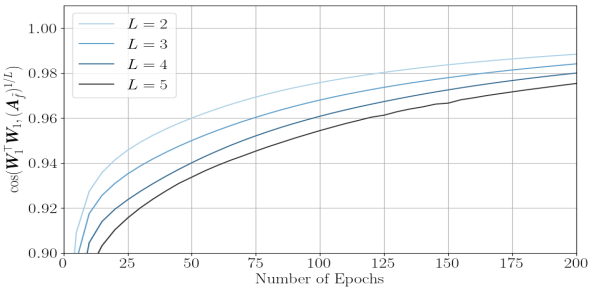


(c) $g(x) = [x]_+$, $\lambda = 10^{-2}$

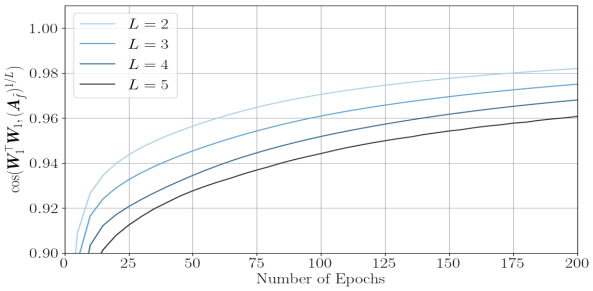


(d) $g(x) = [x]_+$, $\lambda = 10^{-3}$

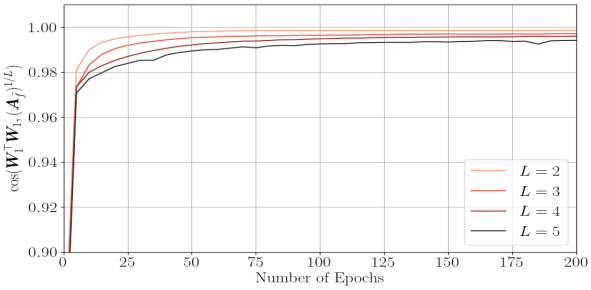
Figure 7: Testing the NFA on fully linear networks. We throughout use SGD with $\eta = 10^{-3}$. We vary the objective function between the top and bottom rows.



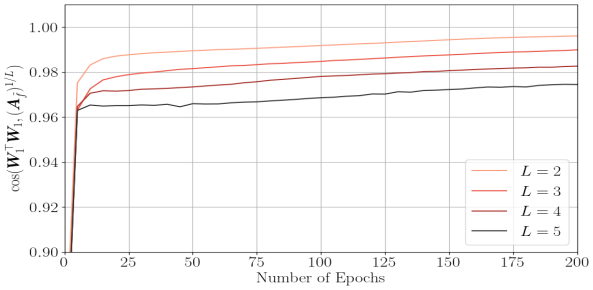
(a) SGD with momentum, $\lambda = 10^{-3}$



(b) SGD with momentum, $\lambda = 10^{-4}$



(c) Adam, $\lambda = 10^{-3}$



(d) Adam, $\lambda = 10^{-4}$

Figure 8: Illustration of the impact of the optimization algorithm on the NFA, on the MNIST dataset.

C.1 Tables of Results

We include tables of results for all of the optimization algorithms mentioned in section 5. We additionally include our results for the Adam optimization algorithm. In these tables, we include the cosine similarity between $\boldsymbol{W}_1^\top \boldsymbol{W}_1$ and $(\boldsymbol{A}_{\tilde{f}})^{1/L}$ after training for the number of initial linear layers L of the network. We include results that have no label noise ($\sigma = 0$) as well as those that have label noise of $\sigma = 1$. In each of these tables we use $N = 2048$ data points. gdm and sgdm refer to GD and SGD with momentum parameter $\beta = 0.9$, respectively. In these tables, we round all results to 2 decimal places.

Table 1: Rank 2

sigma	Linear Layers	lambda	adam	gd	gdm	sgd	sgdm
0	2	10^{-5}	**1.00**	0.99	**1.00**	0.99	**1.00**
0	2	10^{-4}	**1.00**	0.99	**1.00**	0.99	**1.00**
0	2	10^{-3}	**1.00**	0.99	**1.00**	**1.00**	**1.00**
0	2	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	3	10^{-5}	0.98	0.97	**0.99**	0.97	0.98
0	3	10^{-4}	**1.00**	0.97	0.99	0.97	0.99
0	3	10^{-3}	**1.00**	0.98	**1.00**	0.99	**1.00**
0	3	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	4	10^{-5}	**0.99**	0.94	0.97	0.95	0.96
0	4	10^{-4}	**1.00**	0.94	0.97	0.95	0.98
0	4	10^{-3}	**1.00**	0.95	0.99	0.97	**1.00**
0	4	10^{-2}	**1.00**	0.99	**1.00**	**1.00**	**1.00**
0	5	10^{-5}	**0.96**	0.91	**0.96**	0.92	0.94
0	5	10^{-4}	**0.99**	0.92	0.97	0.92	0.96
0	5	10^{-3}	**1.00**	0.92	0.99	0.95	**1.00**
0	5	10^{-2}	**1.00**	0.99	**1.00**	**1.00**	**1.00**
1	2	10^{-5}	**1.00**	0.99	**1.00**	**1.00**	**1.00**
1	2	10^{-4}	**1.00**	0.99	**1.00**	**1.00**	**1.00**
1	2	10^{-3}	**1.00**	0.99	**1.00**	**1.00**	**1.00**
1	2	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	3	10^{-5}	0.99	0.98	**1.00**	0.99	0.98
1	3	10^{-4}	**1.00**	0.98	**1.00**	0.99	0.99
1	3	10^{-3}	**1.00**	0.98	**1.00**	0.99	**1.00**
1	3	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	4	10^{-5}	0.97	0.95	**0.99**	0.98	0.92
1	4	10^{-4}	**0.99**	0.95	**0.99**	0.98	0.93
1	4	10^{-3}	**1.00**	0.96	**1.00**	0.98	0.98
1	4	10^{-2}	**1.00**	0.99	**1.00**	**1.00**	**1.00**
1	5	10^{-5}	0.90	0.92	**0.98**	0.96	0.85
1	5	10^{-4}	0.93	0.92	**0.98**	0.96	0.88
1	5	10^{-3}	0.98	0.93	**0.99**	0.97	0.94
1	5	10^{-2}	**1.00**	0.99	**1.00**	**1.00**	0.99

Table 2: Rank 5

sigma	Linear Layers	lambda	adam	gd	gdm	sgd	sgdm
0	2	10^{-5}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	2	10^{-4}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	2	10^{-3}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	2	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	3	10^{-5}	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**
0	3	10^{-4}	**1.00**	0.99	0.99	0.99	0.99
0	3	10^{-3}	**1.00**	0.99	**1.00**	**1.00**	**1.00**
0	3	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	4	10^{-5}	0.97	**0.98**	**0.98**	**0.98**	0.97
0	4	10^{-4}	**1.00**	0.98	0.98	0.98	0.98
0	4	10^{-3}	**1.00**	0.98	**1.00**	0.99	**1.00**
0	4	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	5	10^{-5}	0.95	0.96	0.96	**0.97**	0.95
0	5	10^{-4}	**0.99**	0.96	0.96	0.97	0.97
0	5	10^{-3}	**1.00**	0.97	0.99	0.98	**1.00**
0	5	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	2	10^{-5}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	2	10^{-4}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	2	10^{-3}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	2	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	3	10^{-5}	0.99	0.99	**1.00**	0.99	0.97
1	3	10^{-4}	0.99	0.99	**1.00**	**1.00**	0.98
1	3	10^{-3}	**1.00**	0.99	**1.00**	**1.00**	**1.00**
1	3	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	4	10^{-5}	0.97	0.98	**0.99**	**0.99**	0.94
1	4	10^{-4}	0.98	0.98	0.98	**0.99**	0.94
1	4	10^{-3}	0.99	0.99	**1.00**	0.99	0.98
1	4	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	5	10^{-5}	0.94	0.97	**0.98**	**0.98**	0.88
1	5	10^{-4}	0.96	0.97	**0.99**	0.98	0.89
1	5	10^{-3}	0.99	0.97	**1.00**	0.99	0.96
1	5	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	0.99

Table 3: Rank 20. **nan** indicate that the training failed resulting in nan values in the weight matrices.

sigma	Linear Layers	lambda	adam	gd	gdm	sgd	sgdm
0	2	10^{-5}	0.99	**1.00**	**1.00**	**1.00**	0.99
0	2	10^{-4}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	2	10^{-3}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	2	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	3	10^{-5}	0.97	**1.00**	0.98	0.99	0.92
0	3	10^{-4}	**1.00**	**1.00**	0.98	0.99	0.95
0	3	10^{-3}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	3	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
0	4	10^{-5}	0.94	**0.97**	0.84	**0.97**	0.88
0	4	10^{-4}	**0.98**	0.97	0.87	0.97	0.92
0	4	10^{-3}	**1.00**	0.98	0.97	0.98	**1.00**
0	4	10^{-2}	**1.00**	0.99	**1.00**	**1.00**	0.99
0	5	10^{-5}	0.94	0.97	**nan**	0.94	0.83
0	5	10^{-4}	0.95	0.97	**nan**	0.95	0.87
0	5	10^{-3}	**0.98**	0.97	0.89	0.97	0.92
0	5	10^{-2}	**0.99**	0.98	**0.99**	**0.99**	0.93
1	2	10^{-5}	0.98	**1.00**	**1.00**	**1.00**	0.98
1	2	10^{-4}	0.98	**1.00**	**1.00**	**1.00**	0.99
1	2	10^{-3}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	2	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	3	10^{-5}	0.95	**1.00**	0.98	0.99	0.89
1	3	10^{-4}	0.96	**1.00**	0.99	0.99	0.91
1	3	10^{-3}	0.99	**1.00**	**1.00**	**1.00**	0.98
1	3	10^{-2}	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
1	4	10^{-5}	0.92	**0.97**	0.84	**0.97**	0.86
1	4	10^{-4}	0.94	0.96	0.87	**0.97**	0.87
1	4	10^{-3}	**0.98**	0.96	0.95	**0.98**	0.94
1	4	10^{-2}	**1.00**	0.99	**1.00**	**1.00**	0.98
1	5	10^{-5}	0.94	0.97	**nan**	0.94	0.83
1	5	10^{-4}	0.94	0.97	**nan**	0.95	0.83
1	5	10^{-3}	0.97	0.97	**nan**	0.97	0.89
1	5	10^{-2}	**0.99**	0.98	0.98	**0.99**	0.93

D Proof of Lemma B.1

We state the following standard result on moments of the uniform distribution.

Lemma D.1. *Suppose $\boldsymbol{X} \sim U(-1,1)$, then $\mathbb{E}[\boldsymbol{X}^k]$ is given by:*

$$\mathbb{E}[\boldsymbol{X}^k] = \begin{cases} 0 & \text{for } k \text{ is odd} \\ \frac{1}{k+1} & \text{for } k \text{ even} \end{cases}$$

Lemma D.2. *Suppose that $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ has entries drawn $a_{ij} \sim U(-1,1)$, then $\mathbb{E}[\|\boldsymbol{A}^\top \boldsymbol{A}\|_F^2] = mn\left(\frac{1}{5} + \frac{m+n-2}{9}\right)$.*

Proof. From definitions, we have

$$\|\boldsymbol{A}^\top \boldsymbol{A}\|_F^2 = \sum_{i=1}^n \sum_{j=1}^n (\boldsymbol{A}^\top \boldsymbol{A})_{ij}^2 = \sum_{i=1}^n \sum_{j=1}^n \left(\sum_{k=1}^n a_{ki}a_{kj}\right)^2 = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^m \sum_{l=1}^m a_{ki}a_{kj}a_{li}a_{lj},$$

and hence by linearity of expectation, we have that

$$\mathbb{E}[\|\boldsymbol{A}^\top \boldsymbol{A}\|_F^2] = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^m \sum_{l=1}^m \mathbb{E}[a_{ki}a_{kj}a_{li}a_{lj}].$$

We calculate the expectation of each term in this sum depending on various cases on i, j, k, l. We throughout use the result of Lemma D.1.

1. $i = j$ and $k = l$, **(mn such terms)** :

$$\mathbb{E}[a_{ki}a_{kj}a_{li}a_{lj}] = \mathbb{E}[a_{ki}^4] = \frac{1}{5}$$

2. $i = j$, but, $k \neq l$ **($m(m-1)n$ such terms)**:

$$\mathbb{E}[a_{ki}a_{kj}a_{li}a_{lj}] = \mathbb{E}[a_{ki}^2 a_{li}^2] = \mathbb{E}[a_{ki}^2]\mathbb{E}[a_{li}^2] = \frac{1}{9},$$

   where for the third inequality, we use the independence of elements

3. $k = l$, but, $i \neq j$ **($mn(n-1)$ such terms)**:

$$\mathbb{E}[a_{ki}a_{kj}a_{li}a_{lj}] = \mathbb{E}[a_{ki}^2 a_{kj}^2] = \mathbb{E}[a_{ki}^2]\mathbb{E}[a_{kj}^2] = \frac{1}{9},$$

   where we again use the independence of elements

4. $k \neq l$ and $i \neq j$ **($m(m-1)n(n-1)$ such terms)**:

$$\mathbb{E}[a_{ki}a_{kj}a_{li}a_{lj}] = \mathbb{E}[a_{ki}]\mathbb{E}[a_{kj}]\mathbb{E}[a_{li}]\mathbb{E}[a_{lj}] = 0,$$

   where we again use the independence of elements.

By summing over these elements and rearranging, we have that

$$\begin{aligned}\mathbb{E}[\|\boldsymbol{A}^\top \boldsymbol{A}\|_F^2] &= \frac{mn}{5} + \frac{m(m-1)n}{9} + \frac{mn(n-1)}{9} \\ &= mn\left(\frac{1}{5} + \frac{m+n-2}{9}\right)\end{aligned}$$

□

Noting the rescaling of the uniform distribution, we may use this result to prove Lemma B.1.

Proof of Theorem 3.2. We may apply Lemma D.2 with $m = d_{l+1},\ n = d_l$ and divide by d_l^2 to account for the fact that the entries are drawn from $U(-1/\sqrt{d_l}, 1/\sqrt{d_l})$ rather than $U(-1,1)$. □