Unveiling Grokking in a sparse parity learning task: Analyzing Feature Learning Dynamics During Training

Javier Sanguino Bautiste JSANGUINO@ETHZ.CH
ETH, Zürich, Switzerland

Gregor Bachmann
ETH, Zürich, Switzerland

Bobby He
ETH, Zürich, Switzerland

Lorenzo Noci
ETH, Zürich, Switzerland

Thomas Hofmann
ETH, Zürich, Switzerland

Abstract

In this paper, we analyze the phenomenon of Grokking in a sparse parity task trained with Deep Neural Networks through the lens of feature learning. In particular, we analyze the evolution of the Neural Tangent Kernel (NTK) matrix. We show that during the initial overfitting phase, the NTK's eigenfunctions are not aligned with the predictive input features. On the other hand, at a later stage the NTK's top eigenfunctions evolve to focus on the features of interest, which corresponds to the onset of the delayed generalization typically observed in Grokking. Our experiments can be viewed as a mechanistic interpretation of feature learning during training through the NTK eigenfunctions' evolution.

1. Introduction

Deep learning has garnered significant attention for its ability to effectively solve a wide range of tasks [4, 8]. However, its theoretical understanding still lags behind the empirical success. For example, the mechanisms behind feature learning, crucial to the practical effectiveness of these models, remain an active area of research [1, 23]. One paradigmatic case of this is the surprising phenomenon of grokking, linked to feature learning by Kumar et al. [9]. The authors establish the link by controlling the rate of feature learning during training and measuring how it affects grokking.

Grokking was first identified by Power et al. [18]. Since then, it has been observed in arithmetic tasks [2, 12, 17, 20, 22] and even in vision and language [11, 14] with various NN architectures or in regression [13]. As shown in Figure 1, grokking describes a scenario where, long after overfitting, validation accuracy improves from chance level to perfect generalization. Notably, no hyperparameters are altered; the network is simply trained for a longer period.

We analyze the feature learning of networks that exhibit grokking through the lens of the Neural Tangent Kernel (NTK) [7], and its evolution though training [5]. The NTK can be understood as a linearization of the NN function space [10], making it a valuable tool for analytically studying NN training dynamics. However, in typical "lazy" learning regimes [3] that cannot capture rich NN training behaviour, the NTK is static.

© J. Sanguino Bautiste, G. Bachmann, B. He, L. Noci & T. Hofmann.

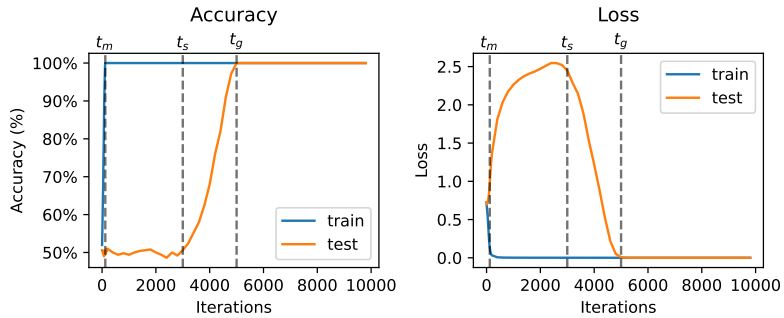

Figure 1: Grokking can be identified by comparing the train and test accuracy or loss. Time points: t_0 is the initialization time, t_m is the first iteration with full training accuracy, t_s is the beginning of an increase in test accuracy, and t_g is the first iteration with full test accuracy.

In particular, in this work we (1) confirm the occurrence of grokking when the network begins learning meaningful features (as suggested by Kumar et al. [9]) in a sparse parity task, measuring changes in the NTK matrix, and (2) investigate how these meaningful features are encoded in the NTK matrix as training progresses showing the emergence of structure. More generally, the tools used in this paper can be seen as a mechanistic interpretability framework for studying feature learning during training.

2. Experimental Setup and NTK framework

In this section we will introduce the necessary background and notation used throughout this work.

2.1. Task Definition

We solve a (k, d)-sparse parity task. Let $\mathcal{D}_{k,d} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ be training set, where $\boldsymbol{x_i} \in \{0,1\}^d$ is a binary vector and the target $y_i = \left(\sum_{l=1}^k x_i\right) \bmod 2$ is the parity of the first k digits (clean digits). Notice that the remaining $d-k$ entries (noisy digits) of $\boldsymbol{x}_i$ do not contribute to y_i. Further, let $\boldsymbol{\mathcal{X}} = \{\boldsymbol{x}_i\}_{i=1}^n$ and $\boldsymbol{\mathcal{Y}} = \{y_i\}_{i=1}^n$. Finally, let $\boldsymbol{x} \in \{0,1\}^d$ denote an arbitrary test point.

2.2. Model Definition

To solve the task, we consider an equal-width L-layer Multi-Layer Perceptron $f_\theta(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$, where $\boldsymbol{\theta}$ represents the weights. The network is trained using full-batch Adam and includes L1-norm regularization on the weights, as it helps to induce grokking in the task [17].

2.3. Neural Tangent Kernel and Feature Learning link

From a theoretical perspective, if we consider continuous updates of the parameters instead of discrete updates ($\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}; \boldsymbol{\theta}_t)$, where $\mathcal{L}(x, y; \boldsymbol{\theta})$ is the empirical loss, the evolution of f_θ can be described with the gradient flow equation:

$$\frac{\partial f_t(\boldsymbol{\mathcal{X}})}{\partial t} = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{X}})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \frac{\partial \boldsymbol{\theta}_t}{\partial t} = -\eta \hat{\boldsymbol{\Theta}}_t(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}}) \nabla_{f_t(\mathcal{X})} \mathcal{L}(\theta) \tag{1}$$

where $\hat{\boldsymbol{\Theta}}_t$ is the empirical Neural Tangent Kernel. In general, Eq. 1 is hard to solve, as $\hat{\boldsymbol{\Theta}}_t$ depends on t.

If we consider $f_t^{lin}(\boldsymbol{x}) := f_0(\boldsymbol{x}) + \nabla_{\boldsymbol{\theta}} f_0(\boldsymbol{x})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} \boldsymbol{\Delta}_t$ where $\boldsymbol{\Delta}_t := \boldsymbol{\theta}_t - \boldsymbol{\theta}_0$, using the MSE loss as $\mathcal{L}(x, y; \boldsymbol{\theta})$, the ODE in equation 1 has a closed solution (as $\nabla_{\boldsymbol{\theta}} f_0(\boldsymbol{x})$ is now constant through training, see Appendix 3). In particular, for an arbitrary test point $\boldsymbol{x}$, $f_t^{lin}(\boldsymbol{x}) = \mu_t(\boldsymbol{x}) + \gamma_t(\boldsymbol{x})$, where

$$\mu_t(\boldsymbol{x}) = \hat{\boldsymbol{\Theta}}_0(\boldsymbol{x}, \boldsymbol{\mathcal{X}})\hat{\boldsymbol{\Theta}}_0^{-1}(I - e^{-\mu\hat{\boldsymbol{\Theta}}_0 t})\boldsymbol{\mathcal{Y}} \tag{2}$$

$$\gamma_t(\boldsymbol{x}) = f_0(\boldsymbol{x}) - \hat{\boldsymbol{\Theta}}_0(\boldsymbol{x}, \boldsymbol{\mathcal{X}})\hat{\boldsymbol{\Theta}}_0^{-1}(I - e^{-\mu\hat{\boldsymbol{\Theta}}_0 t})f_0(\boldsymbol{\mathcal{X}}) \tag{3}$$

Thus, the predictive function has a closed-form solution during training. We follow Hu et al. [6] by letting our initialization be small so the initial output is small ($f_0(\boldsymbol{x}) \approx 0$) to simplify further the equation. Then, $f_t^{lin}(\boldsymbol{x}) = \mu_t$. We will consider this case for our next derivations.

Therefore, $f_t^{lin}(\boldsymbol{x})$ can be understood as kernel learning, meaning no feature learning is involved. If we start training the network normally, obtain the NTK at an arbitrary checkpoint ($t = T$) and then calculate the linearised predictive function at $t \to \infty$, we will effectively freeze feature learning from $t = T$. The resulting predictive function can thus be described as:

$$f_\infty^{lin}(\boldsymbol{x}; T) = \hat{\boldsymbol{\Theta}}_T(\boldsymbol{x}, \boldsymbol{\mathcal{X}})\hat{\boldsymbol{\Theta}}_T^{-1}\boldsymbol{\mathcal{Y}} \tag{4}$$

where $\hat{\boldsymbol{\Theta}}_T$ is the NTK of the checkpoint from which the linearised training is induced.

2.3.1. EIGENFUNCTION DECOMPOSITION

A very common decomposition of kernel-based predictors is based on the eigenfunction framework. Let us denote the input distribution as $\boldsymbol{x} \sim p_{\boldsymbol{X}}$. The eigenfunctions associated with the NTK $\boldsymbol{\Theta} : \mathbb{R}^n \times \mathbb{R}^n$ are defined through the integral equation

$$\mathbf{d}\int_{\mathbb{R}^d} \Theta(\boldsymbol{x}, \boldsymbol{x'})\phi(\boldsymbol{x'})p_{\boldsymbol{X}}(\boldsymbol{x'})d\boldsymbol{x'} = \omega\phi(\boldsymbol{x}) \tag{5}$$

where $\phi : \mathbb{R}^n \to \mathbb{R}$ is the eigenfunction and $\omega \geq 0$ is its corresponding eigenvalue. To calculate the eigenfunctions, we need to know the input distribution $p_{\boldsymbol{X}}$, which we usually don't have access to. Coming back to our example, let $\hat{\boldsymbol{\Theta}}(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}}) = \boldsymbol{V}\boldsymbol{\Omega}\boldsymbol{V}^T$ with orthogonal $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ and a diagonal $\boldsymbol{\Omega} \in \mathbb{R}^{n\times n}$. We can form an estimator of the eigenfunctions for an arbitrary test point $\boldsymbol{x}$,

$$\hat{\phi}_i(\boldsymbol{x}) = \frac{1}{\omega_i}\sum_{l=1}^{n} \Theta(\boldsymbol{x}, \boldsymbol{x}_l)V_{li} \tag{6}$$

It can be shown that this forms a consistent estimator, i.e. as the sample size n increases, it holds that $\hat{\phi}_i \to \phi_i$ point-wise. Returning back to our predictive equation 16, we can now write it as a linear combination of eigenfunctions,

$$f_t^{lin}(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i^t \hat{\phi}_i(\boldsymbol{x})$$

The weights α_i can be written as $\alpha_i^t = \frac{\omega_i(1-e^{-\eta\omega_i t})}{\omega_i}\boldsymbol{v}_i^T\boldsymbol{\mathcal{Y}} =: \gamma_i^t \boldsymbol{v}_i^T\boldsymbol{\mathcal{Y}}$ where $\boldsymbol{v}_i$ is the i-th eigenvector. Therefore, α_i^t measures how well aligned the i-th eigenvector is with the targets $\boldsymbol{\mathcal{Y}}$ throughout time.

3. Experiments

This section presents the results of an experiment conducted on a $(3, 35)$-sparse parity dataset for several seeds. The rest of training details can be consulted in Appendix B. Results hold for different choices of d in a (k, d)-sparse parity dataset, Appendix D shows them.

It is known that grokking arises when the "right" structure emerges in the network [12, 15]. In our case, this occurs when the network starts focusing on clean digits (see Appendix C.1 for details on how the weights evolve), indicating that it is learning the correct features. In our experiments, we analyze how the NTK shows the evolution of feature learning during training.

3.1. Feature learning happens late during training

As training a model until $t = T$ and then considering a linearized version of the predictive function stops feature learning from $t = T$, we can evaluate the quality of the features learned up to that checkpoint using Equation 4. Additionally, if the NTK matrices at two different checkpoints are similar, it suggests that the true optimization path during training closely follows the linearized approximation, indicating that the network is close to the lazy regime. As suggested by Fort et al. [5] and Tsilivis and Kempe [21], we use the Central Kernel Alignment (CKA) metric:

$$d(\boldsymbol{K}_1, \boldsymbol{K}_2) = 1 - \frac{Tr(\boldsymbol{K}_1 \boldsymbol{K}_2^T)}{\sqrt{Tr(\boldsymbol{K}_1 \boldsymbol{K}_1^T) Tr(\boldsymbol{K}_2 \boldsymbol{K}_2^T)}}$$

As observed in figure 2 (e), the NTK changes slightly until t_m, and then enters a lazy regime (where, in square 1, it does not change at all). As shown in square 2, after t_s, the kernel starts changing, indicating a transition from the lazy regime to a rich feature learning regime. This change continues until a bit after t_g to then stabilize, as shown in square 3.

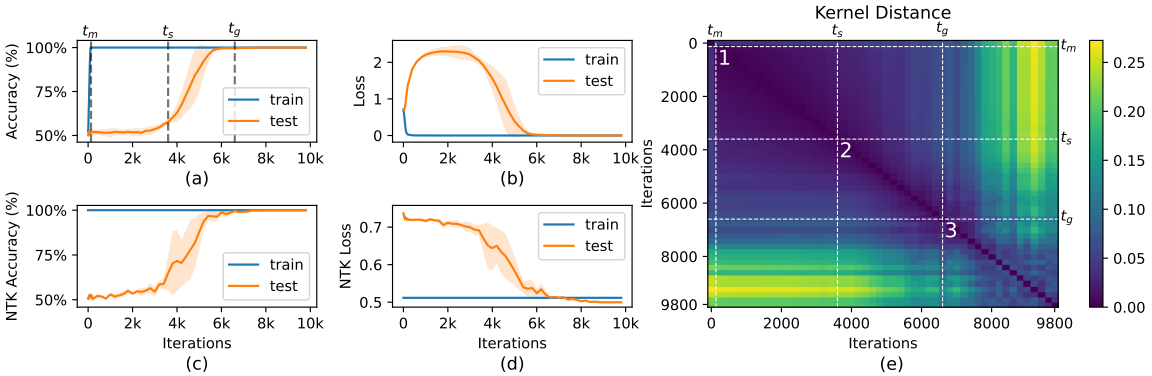

Figure 2: Plots (a) and (b) show the accuracy and loss throughout training while (c) and (d) show the evolution of accuracy and loss when using Equation 4 for that checkpoints. All four plots show the average over several seeds and their variance. Plot (e) shows the average of CKA (over the seeds) between NTKs associated to all checkpoints. Note that the key checkpoints are defined with (a) but (e) includes them to facilitate its interpretation.

Therefore, the rapid increase in generalization occurs when the network enters a rich feature learning regime, this is when the training dynamics diverge from the linearized version, as stated by Kumar et al [9]. Consequently, this experiment confirms their claims by extending the analysis to an unconstrained setting (sparse parity task) and complements them with the study of NTK evolution to understand feature learning throughout training.

3.2. Features are encoded in the NTK

In this section, we investigate the NTK at t_0, t_m, t_s, t_g to understand how it encodes features (refer to Figure 1 for a definition of the time points).

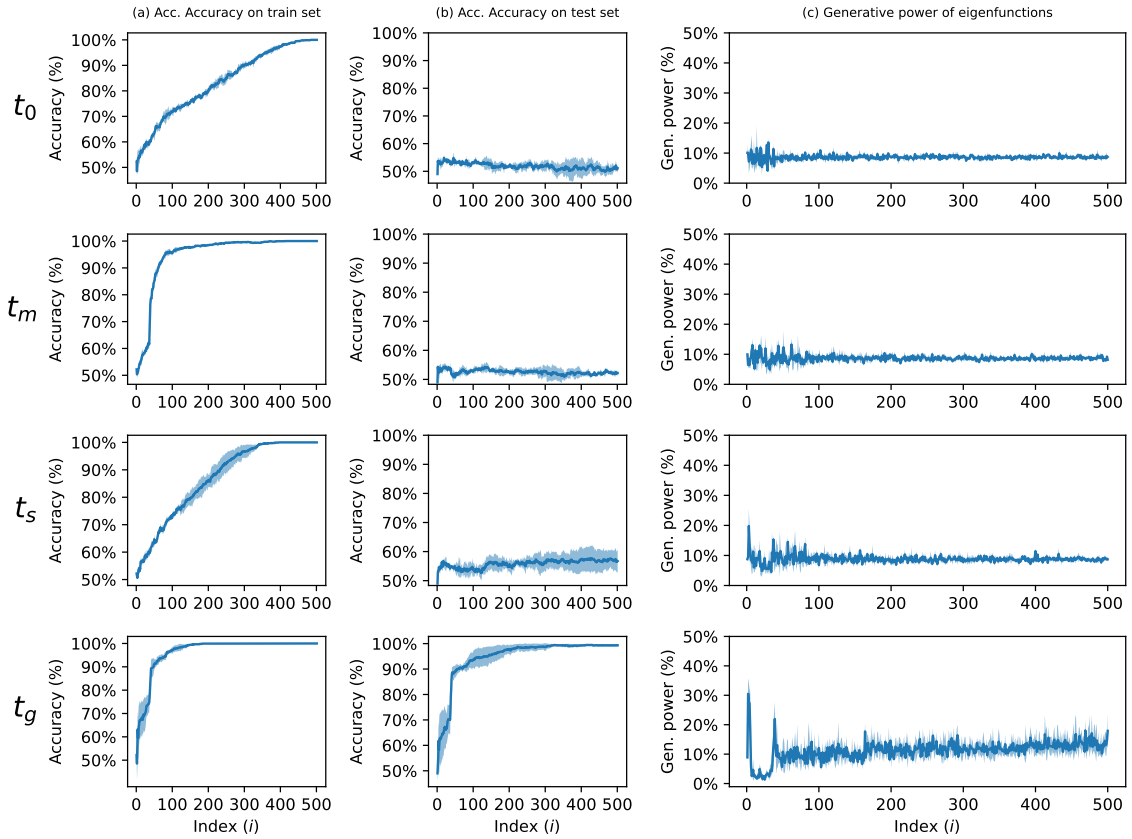

Figure 3: Evolution of training and test accuracy and loss throughoutout training and of the NTK associated to that weights checkpoint following equation 4.

To understand which eigenfunctions are essential for the classification task, we have computed the NTK accuracy accumulating eigenfunctions progressively when calculating the predictive function i.e $f_i^{lin}(\boldsymbol{x}) = \sum_{j=1}^{i} \alpha_j \hat{\phi}_j(\boldsymbol{x})$. Figures 3 (a) and (b) show the accuracy with respect of the number of accumulated eigenfunctions. It can be seen that there is an emerging structure in the eigenfunctions as the network generalizes: the initial ones and those around 35 notably boost accuracy. It appears that these eigenfunctions (and therefore, some part of the NTK) encode valuable information relevant to a sparse parity learning task. In other experiments we have observed that changing the number of input features d also shifts the index of relevant eigenvalues to d.

To visualize the features encoded into these important eigenfunctions, we utilize saliency maps, defined as $\boldsymbol{s_i}(\boldsymbol{\mathcal{X}}) = \frac{1}{n}\sum_{j=1}^{n} \nabla_{\boldsymbol{x}} \mathcal{L}(\hat{\phi}_i(\boldsymbol{x_j}), y_j) \in \mathbb{R}^d$ [16, 19]. Saliency maps indicate the sensitivity of the loss to changes in each digit for each eigenfunction. By averaging them over all training samples, we obtain a more robust approximation, leveraging the consistent positioning of the clean digits in every sample of $\mathcal{X}$.

We have also devised a metric (generative power, g) aimed at quantifying the generalizing power of an eigenfunction based on the proportion of its saliency map that concentrates on the clean digits:

$$g_i = \frac{\sum j = 1^k \, [s_i(x)]^j}{\sum_{j=1}^{d} [s_i(x)]^j}$$

where $[s_i(x)]^j$ is the j-th digit of $\boldsymbol{s_i}(x)$.

Figure 3 (c) illustrates how the eigenfunctions exhibit an emerging structure as generalization occurs in the network. The first eigenfunctions focus on the clean digits, while the subsequent ones consistently focus on the noisy digits (generative power close to 0%). From the d-th eigenfunction onward, a small number of eigenfunctions again focus only on the clean digits. For a visualisation of the raw saliency maps, check Appendix C.2. This serves an explanation as why these eigenfunctions boost accuracy.

One last observation is that there appears to be a minimum number of eigenfunctions required for the network to generalize effectively. Figure 4 illustrates that the number of eigenfunctions needed for 99% accuracy on the training set decreases throughout training. As the network achieves good generalization performance, it converges to a minimum. This suggests a critical threshold in the number of eigenfunctions necessary for capturing the essential features of the dataset and facilitating successful generalization by the network.

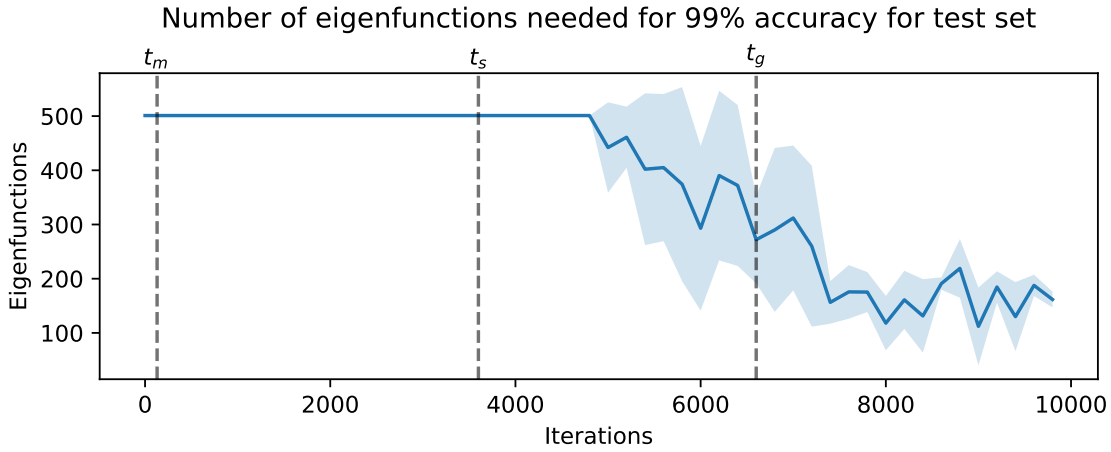

Figure 4: Minimum number of eigenfunctions to reach 99% accuracy on the test set though training. The number seems to converge at some point of training (after t_g). Note that this plot can be seen as summary of Figure 3 (b).

This approach offers an interpretation of the mechanism driving the change of the NTK by examining the evolution of its eigenfunctions throughout the training process. By tracking how these eigenfunctions change, we gain insights into the network's feature learning dynamics.

4. Conclusion

Our study sheds light on the mechanism underlying grokking in deep neural networks, in the context of sparse parity tasks. Our findings are consistent with Kumar et al. [9], where Grokking is attributed to the network's transition from a lazy (close to its linearisation) to a feature learning regime. In this context, our analysis also unveil the emergence of structured eigenfunctions, indicating the acquisition of essential features crucial for generalization. Leveraging saliency maps and a devised metric (the generative power), we demonstrate how these eigenfunctions evolve, with early ones focusing on clean digits and later ones capturing noisy digits. Our findings provide insights into the progression of training dynamics and the encoding of task-relevant information.

Overall, our research compiles several techniques to analyze the NTK throughout training, proposing a mechanistic interpretability framework that can be used to understand the feature learning dynamics during training of Neural Network.

References

[1] Devansh Arpit, Stanisław Jastrzundefinedbski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks. In *Proc. of the International Conf. on Machine Learning*, 2017. URL https://dl.acm.org/doi/10.5555/3305381.3305406.

[2] Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham M. Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: SGD learns parities near the computational limit. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=8XWP2ewX-im.

[3] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In *Advances in neural information processing systems*, 2019. URL https://dl.acm.org/doi/abs/10.5555/3454287.3454551.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019. URL https://api.semanticscholar.org/CorpusID:52967399.

[5] Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In *Advances in Neural Information Processing Systems*, 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/405075699f065e43581f27d67bb68478-Paper.pdf.

[6] Wei Hu, Zhiyuan Li, and Dingli Yu. Simple and effective regularization methods for training on noisily labeled data with generalization guarantee. In *International Conf. on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hke3gyHYwH.

[7] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural Tangent Kernel: convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[9] Tanishq Kumar, Blake Bordelon, Samuel J. Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics. In *International Conf. on Learning Representations*, 2024. URL https://openreview.net/forum?id=vt5mnLVIVo.

[10] Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear

models under gradient descent. *Journal of Statistical Mechanics: Theory and Experiment*, 2020. URL http://dx.doi.org/10.1088/1742-5468/abc62b.

[11] Ziming Liu, Eric J Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *International Conf. on Learning Representations*, 2023. URL https://openreview.net/forum?id=zDiHoIWa0q1.

[12] William Merrill, Nikolaos Tsilivis, and Aman Shukla. A tale of two circuits: Grokking as competition of sparse and dense subnetworks, 2023. URL https://arxiv.org/abs/2303.11873.

[13] Jack William Miller, Charles O'Neill, and Thang D Bui. Grokking beyond neural networks: An empirical exploration with model complexity. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ux9BrxPCl8.

[14] Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher Manning. Grokking of hierarchical structure in vanilla transformers. In *Proc. of the Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://aclanthology.org/2023.acl-short.38.

[15] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *International Conf. on Learning Representations*, 2023. URL https://openreview.net/forum?id=9XFSbDPmdW.

[16] Yeat Jeng Ng, Ainhize Barrainkua, and Novi Quadrianto. Understanding and addressing spurious correlation via neural tangent kernels: A spectral bias perspective, 2024. URL https://openreview.net/forum?id=89AOrk05uy.

[17] Adam Pearce, Asma Ghandeharioun, Nada Hussein, Nithum Thain, Martin Wattenberg, and Lucas Dixon. Do machine learning models memorize or generalize. *People+ AI Research*, 2023. URL https://pair.withgoogle.com/explorables/grokking/.

[18] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177.

[19] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *Computing Research Repository (CoRR)*, abs/1312.6034, 2013. URL https://api.semanticscholar.org/CorpusID:1450294.

[20] Dashiell Stander, Qinan Yu, Honglu Fan, and Stella Biderman. Grokking group multiplication with cosets. In *Proc. of the International Conf. on Machine Learning*, 2024. URL https://openreview.net/pdf?id=hcQfTsVnBo.

[21] Nikolaos Tsilivis and Julia Kempe. What can the neural tangent kernel tell us about adversarial robustness? In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=KBUgVv8z7OA.

[22] Zhiwei Xu, Yutong Wang, Spencer Frei, Gal Vardi, and Wei Hu. Benign overfitting and grokking in ReLU networks for XOR cluster data. In *International Conf. on Learning Representations*, 2024. URL https://openreview.net/pdf?id=BxHgpC6FNv.

[23] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM*, 64(3):107–115, feb 2021. URL https://doi.org/10.1145/3446776.

Appendix A. Definitions and extension of mathematical derivations

This appendix extends the definitions and elaborates on the mathematical derivations used in the main text. We first introduce the dataset and model that we use. Later derive its Neural Tangent Kernel (NTK) equations.

A.1. Dataset

In a sparse parity task, the goal is to calculate the parity of k digits within a d-bit number, where $k \leq d$. It has been demonstrated that gradient-based methods do not perform well on learning parities. Most works take the approach of making favorable assumptions on the input distribution. Nevertheless, we intentionally follow the paper on the example of [2] to not simplify the problem. Let us introduce first the term (d, S)-parity function to later define the dataset we use throughout the experiments.

Definition 1 *(d, S)-parity function*

For an natural number d and for a subset [1] $S \subseteq [d]$, let $f_S(\boldsymbol{x})$ be the (d, S)-parity function, $f_S : \{0,1\}^d \to \{0,1\}$ that takes a d-digit binary number $\boldsymbol{x}$ and checks if the number of ones in the index of the digits of S is odd. That is,

$$f_S(\boldsymbol{x}) = \left(\sum_{i \in S} x^i \right) \mod 2 \tag{7}$$

where x^i is the i-th digit of $\boldsymbol{x}$.

Without loss of generality, in this work, we study the parity in the firsts k digits of the problem. Therefore, $S = \{1, 2, \ldots, k\}$ ($k \in \mathbb{N}, k \leq d$). We can then define the (d, k)-parity function $f_k : \{0,1\}^d \to \{0,1\}$ as:

$$f_k(\boldsymbol{x}) = \left(\sum_{i=1}^{k} x^i \right) \mod 2 \tag{8}$$

Definition 2 *Sparse parity Dataset*

Let a Sparse parity Dataset $\mathcal{D}_{(d,k)} = \{(\boldsymbol{x_i}, y_i)^B\}$ of n training samples and n_{test} test samples ($B = n + n_{test}$) be a dataset where $\boldsymbol{x_i}$ is drawn from $Unif(\{0,1\}^d)$ and $y_i = f_k(\boldsymbol{x_i})$. We denote $\boldsymbol{\mathcal{X}} = [\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}] \in \mathbb{R}^{n \times d}$ as the matrix of all training data and $\boldsymbol{\mathcal{Y}} = [y_1, \ldots, y_n] \in \mathbb{R}^d$ the vector of all training labels. $\boldsymbol{X}$ is any subset of test data (including the full test data).

1. We consider $[d]$ to be the superset of sets of less or equal than d elements containing some natural numbers from 1 to d without repetition.

A.2. Model

To solve the task we train a Neural Network (NN), we consider a L-layer MLP with the form $f_\theta(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}$ that can be define recursively by:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{z}^{(L)} = \frac{1}{\sqrt{m_L}}\boldsymbol{\theta}^{(L)}\boldsymbol{x}^{(L-1)}$$
$$\boldsymbol{z}^{(l)} = \frac{1}{\sqrt{m_l}}\boldsymbol{\theta}^{(l)}\boldsymbol{x}^{(l-1)} \quad \longrightarrow \quad \boldsymbol{x}^{(l)} = \sigma\left(\boldsymbol{z}^{(l)}\right)$$

where $\boldsymbol{\theta} = \{\boldsymbol{\theta}^{(l)},\ \forall l = 1, \dots, n\}$ are the trainable weights. $\boldsymbol{\theta}^{(l)} = [\boldsymbol{\theta}_1^{(l)}, \dots, \boldsymbol{\theta}_{m_l}^{(l)}] \in \mathbb{R}^{m_l \times m_{l-1}}$ for $l = 1, \dots, L-1$ are the weights of the hidden layers and $\boldsymbol{\theta}^{(L)} = [\theta_1^{(L)}, \dots, \theta_{m_L}^{(L)}] \in \mathbb{R}^{m_L}$ are the weights of the classification layer, all initialized as $\theta_{ij}^{(l)} \sim \mathcal{N}(0, \sigma_{init})$. $\sigma(x)$ is the ReLU function. We assume that $m_l = m, \forall l$ and we can consider $m_0 = d$.

The network is trained using full-batch gradient descent. With a slight abuse of notation ($\boldsymbol{\theta}_{t'} = \boldsymbol{\theta}\Big|_{t=t'}$ and $f_{t'}$ its corresponding function), the weight update rule in each iteration is[2]

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}; \boldsymbol{\theta}_t) \tag{9}$$

where $\mathcal{L}(x, y; \boldsymbol{\theta})$ is the empirical loss, defined as

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(x, y; \boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} l(f_{\boldsymbol{\theta}}(x_i), y_i) + \lambda_1 \|\boldsymbol{\theta}\|_1 + \lambda_2 \|\boldsymbol{\theta}\|_2 \tag{10}$$

A.3. Neural Tangent Kernel

From a theoretical point of view, if instead of having discrete updates of the parameters like in equation 9, we consider a continuous update, the evolution of the weights of the NN can be described using the gradient flow differential equation.

$$\begin{aligned} \frac{\partial \boldsymbol{\theta}_t}{\partial t} &= -\eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \\ &= -\eta \left(\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{X}})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}\right)^T \nabla_{f_t(\boldsymbol{\mathcal{X}})} \mathcal{L}(\boldsymbol{\theta}) \end{aligned} \tag{11}$$

Then,

$$\begin{aligned} \frac{\partial f_t(\boldsymbol{\mathcal{X}})}{\partial t} &= \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{X}})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \frac{\partial \boldsymbol{\theta}_t}{\partial t} \\ &= -\eta \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{X}})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \left(\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{X}})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}\right)^T \nabla_{f_t(\boldsymbol{\mathcal{X}})} \mathcal{L}(\boldsymbol{\theta}) \\ &= -\eta \hat{\boldsymbol{\Theta}}_t(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}}) \nabla_{f_t(\mathcal{X})} \mathcal{L}(\theta) \end{aligned} \tag{12}$$

2. We have experimented with slight variations of the optimizer update rule, including a momentum term and using the Adam optimizer. However, we have chosen not to include the specific formulas here, as they are not essential to our main objectives.

where $\hat{\boldsymbol{\Theta}}_t$ is the empirical Neural Tangent Kernel

Definition 3 *Empirical Neural Tangent Kernel of the training data (e-NTK) The e-NTK ($\hat{\boldsymbol{\Theta}}_t \in \mathbb{R}^{n\times n}$) is the inner product of the gradients evaluated at $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, where the network is fixed at $\boldsymbol{\theta} = \boldsymbol{\theta}_t$*

$$\hat{\boldsymbol{\Theta}}_t(\boldsymbol{x}_1, \boldsymbol{x}_2) := \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x}_1)\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \left(\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\boldsymbol{x}_2)\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}\right)^T$$

Equation 12 is difficult to solve, as $\hat{\boldsymbol{\Theta}}_t$ depends on t. Nevertheless, there is two approximations that simplify the calculations. In this work, we focus on considering a linearised version of the network but equation 12 can be also solved considering the network to be infinitely wide ($m \to \infty$).

A.3.1. LINEARISED NETWORK

Let us consider the first-order Taylor approximation of the predictive function of the Neural Network:

$$f_t^{lin}(\boldsymbol{x}) := f_0(\boldsymbol{x}) + \nabla_{\boldsymbol{\theta}} f_0(\boldsymbol{x})\Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_0} \boldsymbol{\Delta}_t$$

where $\boldsymbol{\Delta}_t := \boldsymbol{\theta}_t - \boldsymbol{\theta}_0$. Then, equations 11 and 12 get simplified to:

$$\frac{\partial \boldsymbol{\Delta}_t}{\partial t} = -\eta \nabla_{\boldsymbol{\theta}} f_0(\boldsymbol{\mathcal{X}}) \nabla_{f_t^{lin}(\boldsymbol{\mathcal{X}})} \mathcal{L}(\boldsymbol{\theta}) \tag{13}$$

$$\frac{\partial f_t^{lin}(\mathcal{X})}{\partial t} = -\eta \hat{\boldsymbol{\Theta}}_0(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}}) \nabla_{f_t^{lin}(\boldsymbol{\mathcal{X}})} \mathcal{L}(\boldsymbol{\theta}) \tag{14}$$

As $\nabla_{\boldsymbol{\theta}} f_0(\boldsymbol{x})$ (and therefore $\hat{\boldsymbol{\Theta}}_0$) is constant through training, using the MSE loss, the ODE has a closed-form solution:

$$f_t^{lin}(\boldsymbol{\mathcal{X}}) = (I - e^{-\mu \hat{\boldsymbol{\Theta}}_0 t})\boldsymbol{\mathcal{Y}} + e^{-\mu \hat{\boldsymbol{\Theta}}_0 t} f_0(\boldsymbol{\mathcal{X}}) \tag{15}$$

considering $\hat{\boldsymbol{\Theta}}_0 = \hat{\boldsymbol{\Theta}}_0(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{X}})$.

For an arbitrary test point $\boldsymbol{x}$, doing the same theoretical elaboration, $f_t^{lin}(\boldsymbol{x}) = \mu_t(\boldsymbol{x}) + \gamma_t(\boldsymbol{x})$, where

$$\mu_t(\boldsymbol{x}) = \hat{\boldsymbol{\Theta}}_0(\boldsymbol{x}, \boldsymbol{\mathcal{X}}) \hat{\boldsymbol{\Theta}}_0^{-1} (I - e^{-\mu \hat{\boldsymbol{\Theta}}_0 t}) \boldsymbol{\mathcal{Y}} \tag{16}$$

$$\gamma_t(\boldsymbol{x}) = f_0(\boldsymbol{x}) - \hat{\boldsymbol{\Theta}}_0(\boldsymbol{x}, \boldsymbol{\mathcal{X}}) \hat{\boldsymbol{\Theta}}_0^{-1} (I - e^{-\mu \hat{\boldsymbol{\Theta}}_0 t}) f_0(\boldsymbol{\mathcal{X}}) \tag{17}$$

Note that then, the evolution of the neural network through training has also closed-form solution and in the end of training ($t \to \infty$) the predictive function is

$$f_\infty^{lin}(\boldsymbol{x}) = \hat{\boldsymbol{\Theta}}_0(\boldsymbol{x}, \boldsymbol{\mathcal{X}}) \hat{\boldsymbol{\Theta}}_0^{-1} (\boldsymbol{\mathcal{Y}} - f_0(\boldsymbol{\mathcal{X}})) + f_0(\boldsymbol{x}) \tag{18}$$

From here, similar to Hu et al. [6], we will consider the case when we initialise the parameters of the network so $f_0(\boldsymbol{x}) \approx 0$ (σ_{init} is small). Therefore:

$$f_t^{lin}(\boldsymbol{x}) = \mu_t(\boldsymbol{x}) = \hat{\boldsymbol{\Theta}}_0(\boldsymbol{x}, \boldsymbol{\mathcal{X}})\hat{\boldsymbol{\Theta}}_0^{-1}(I - e^{-\mu\hat{\boldsymbol{\Theta}}_0 t})\boldsymbol{\mathcal{Y}} \tag{19}$$

$$f_\infty^{lin}(\boldsymbol{x}) = \hat{\boldsymbol{\Theta}}_0(\boldsymbol{x}, \boldsymbol{\mathcal{X}})\hat{\boldsymbol{\Theta}}_0^{-1}\boldsymbol{\mathcal{Y}} \tag{20}$$

Note that this equations have the form of a Kernel Learning algorithm with the transformation of the inputs given by the Neural Tangent Kernel. Therefore, a linearised Neural Network has no feature learning involved in model. Consequently, this is a useful tool for theoretically studying NN training dynamics but also takes away much of the power of NN.

Appendix B. Experiment details

In this section, we outline the details of the experiments for reproducibility of the results. Grokking appears consistently for a sparse parity task, the choice of training hyperparameters was based on a purely time basis, so grokking would appear soon during training.

We use a sparse parity dataset $\mathcal{D}_{(d,k)} = \{(\boldsymbol{x_i}, y_i)^B\}$ with 500 training samples and 500 test samples. x_i is chosen so samples are not repeated in any of the $B = 1000$ samples: $\boldsymbol{x_i} \neq \boldsymbol{x_j}\ \forall i \neq j$. $k = 3$ is fixed and $d = 10$ is variable. In the main text, only results using $k = 3$ and $d = 30$ are shown but other results can be checked in D .

The network used to solve the task has depth $L = 2$ and width $m = 64$. No bias term is used in neither of the layers. The weights are initialized following a normal distribution with $\sigma_{init} = 1$ ($\theta_{ij}^{(l)} \sim \mathcal{N}(0, 1)$).

The optimizer used was full-batch Adam with learning rate $\mu = 0.03$ and $(\beta_1, \beta_2) = (0.9, 0.999)$ over a binary cross-entropy loss with a $l1$-regularisation term over the weights of $\lambda_1 = 10^{-3}$.

Appendix C. Complementary results

This section offers complementary results to the ones shown in the main text for a $(3, 35)-$sparse parity dataset with the experiment setup described in B.

C.1. Emergence of the structure in the weights of the network

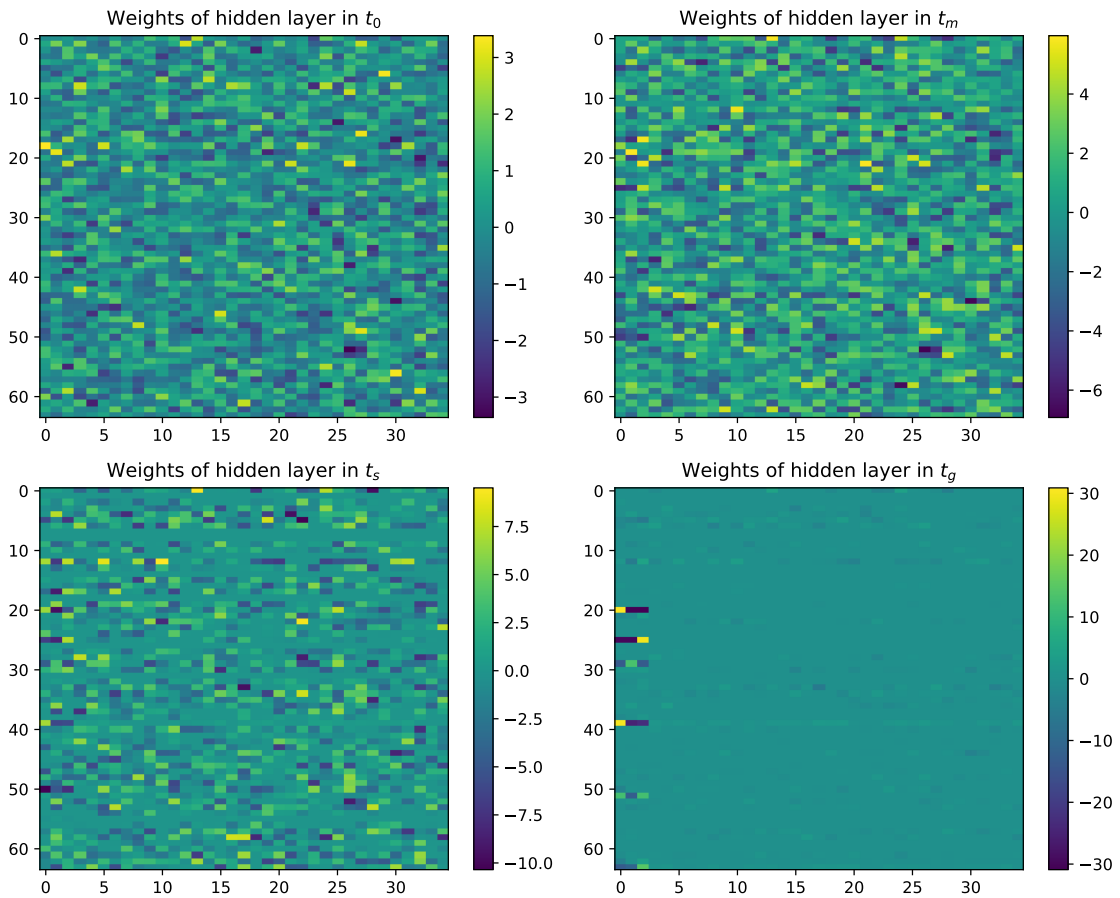

Figure 5: Evolution of the weights of the first layer of a 2-layer MLP. Each image represents $\boldsymbol{W}^{(1)} \in \mathbb{R}^{m \times d}$ for different time steps of training. The weights progressively focus on the first three digits (clean digits).

As it can be seen, generalisation happens because the networks starts focusing on the clean digits. Nevertheless, during the memorisation period ($t = [t_m, t_s]$) this structure does not seem to arise. It is noteworthy that the scale of the weights increases from t_0 to t_m. This increase in scale contributes to the rise in the loss function during the memorization phase. Moreover, the logits of the function also scale up due to this weight scaling.

C.2. Visualization of generative power

In this section, we show the saliency maps used to calculate the generative power metric.

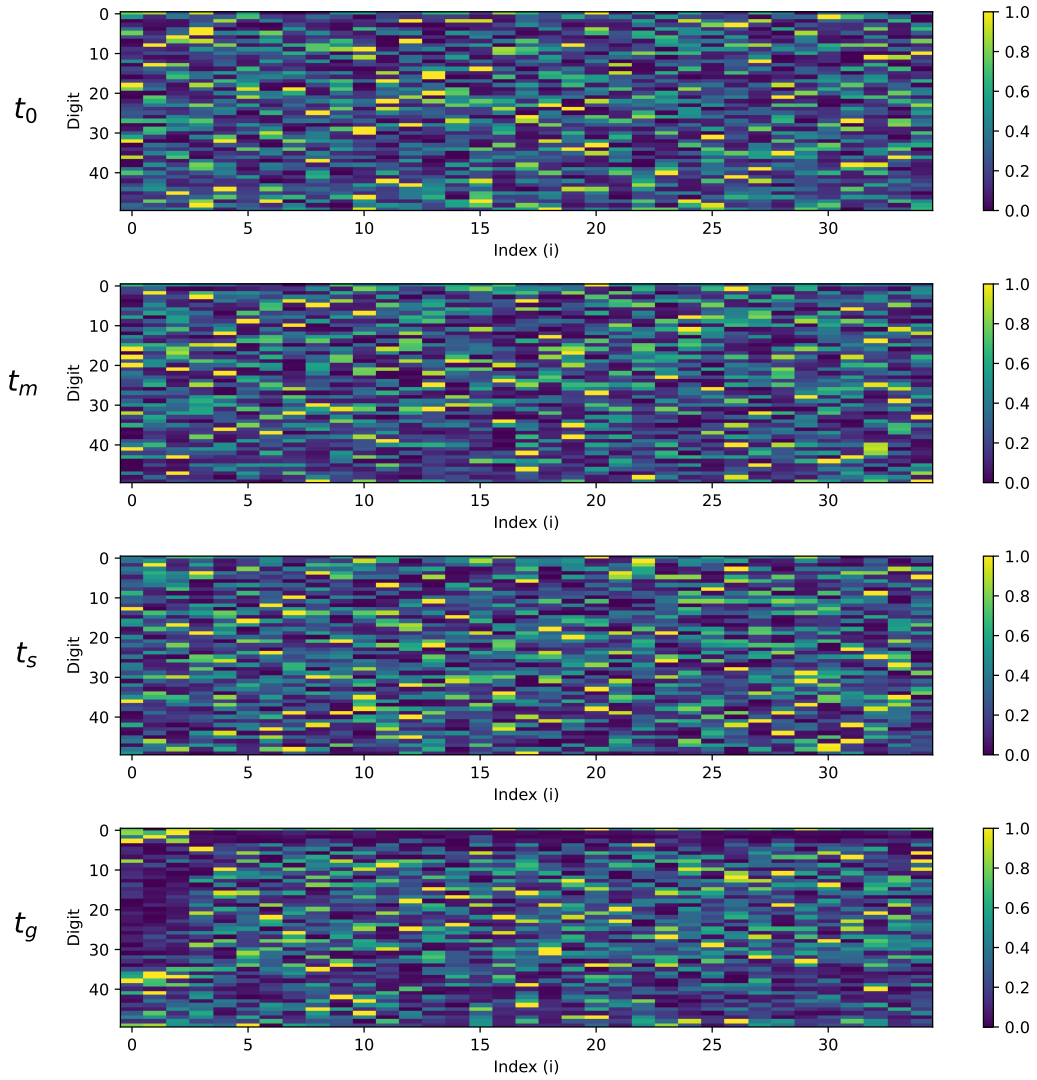

Figure 6: Normalized saliency map of the first 50 eigenfunctions with respect to the loss function $\boldsymbol{s_i}(\boldsymbol{\mathcal{X}}) = \frac{1}{n}\sum_{j=1}^{n} \nabla_{\boldsymbol{x}} \mathcal{L}(\hat{\phi}_i(\boldsymbol{x_j}), y_j)$. Rows indicate the index of the eigenfunction (i). Highest value of each row is mapped to 1 and lowest to 0, the rest are interpolated.

As it can be also devised with the generative power metric (see Figure 3), the first eigenfunctions concetrate their saliency map only in the first 3 digits (clean digits) as training progresses. The following 32 eigenfunctions focus only on the noisy digits. And around the 35 eigenfunction, they focus again only on the clean digits.

Appendix D. Results for other choices of d

D.1. 3 clean digits and 30 noisy

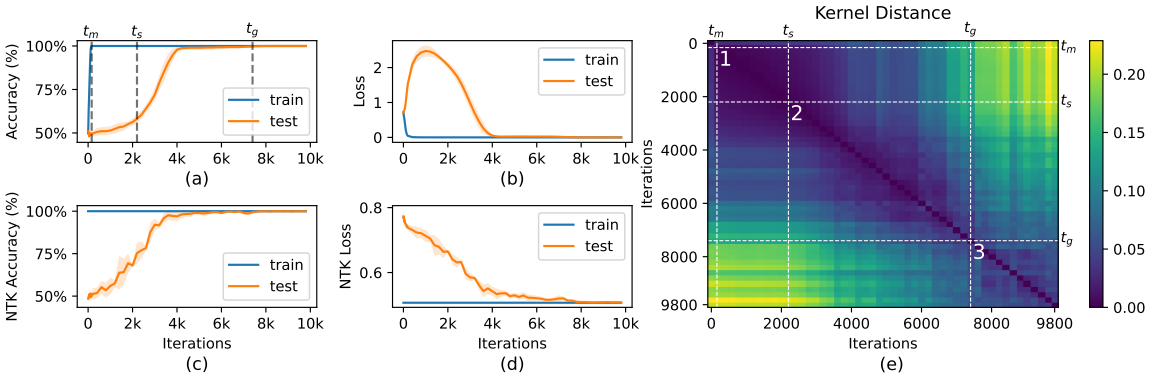

Figure 7: Training metrics and NTK evolution through training for a $(3, 30)$-sparse parity dataset. The rest of the experimental set-up is identical to what was described in B.

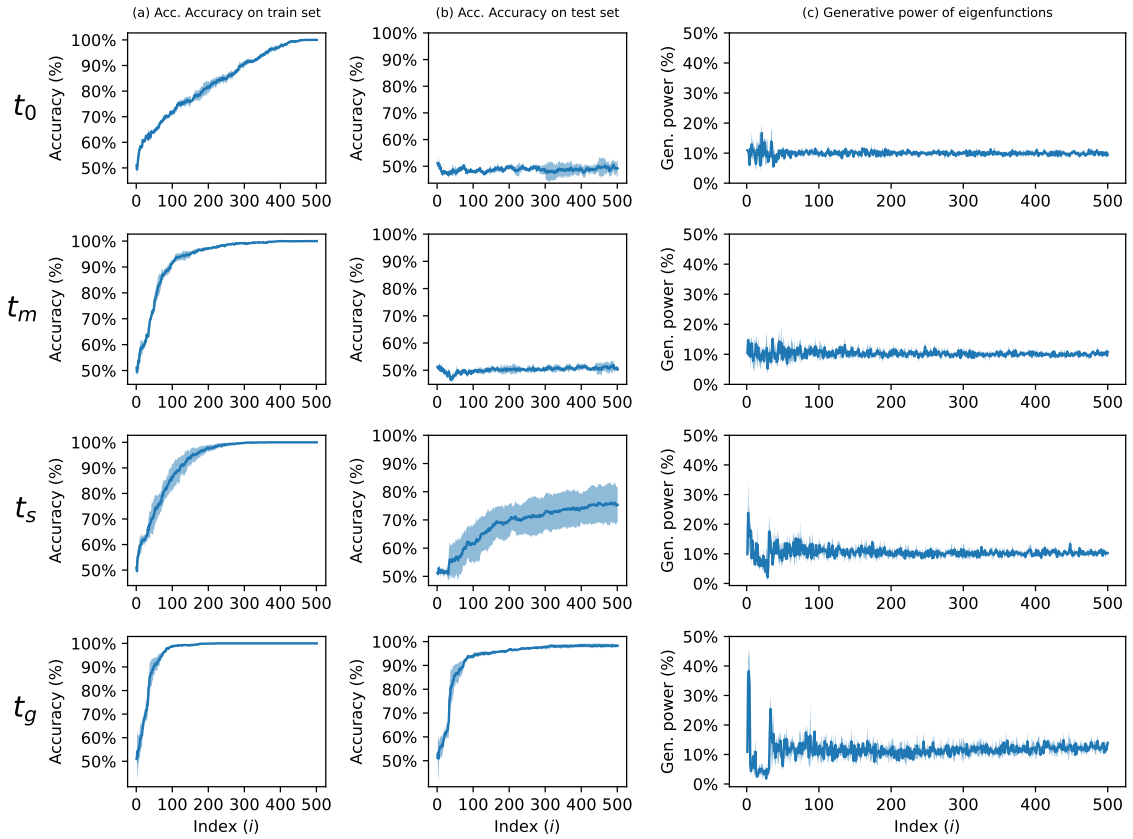

Figure 8: Accumulative accuracy and generative power of each eigenfunction for a $(3, 30)$-sparse parity dataset. The rest of the experimental set-up is identical to what was described in B.

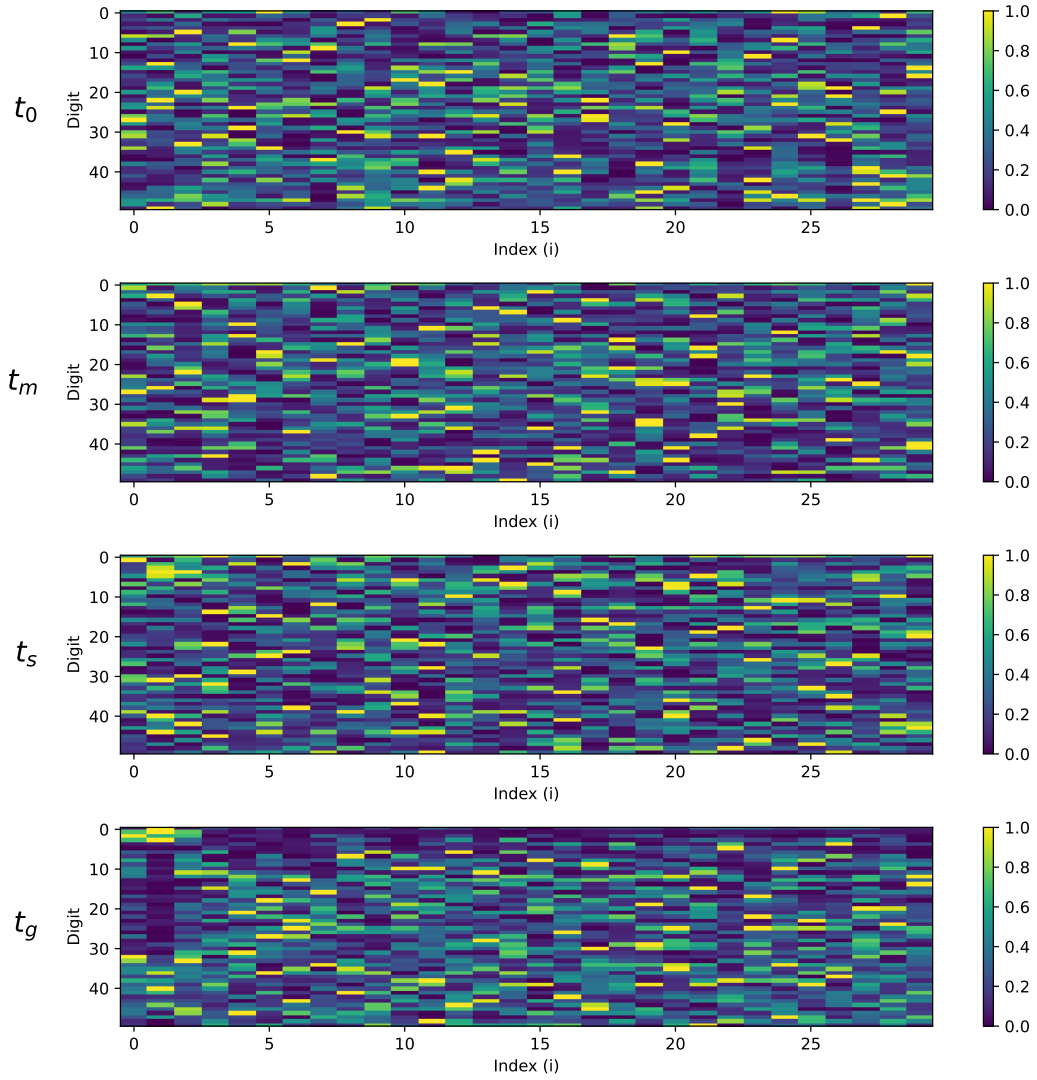

Figure 9: Accumulative accuracy and generative power for each for a $(3, 30)$-sparse parity dataset. The rest of the set-up is identical to what was described in B.

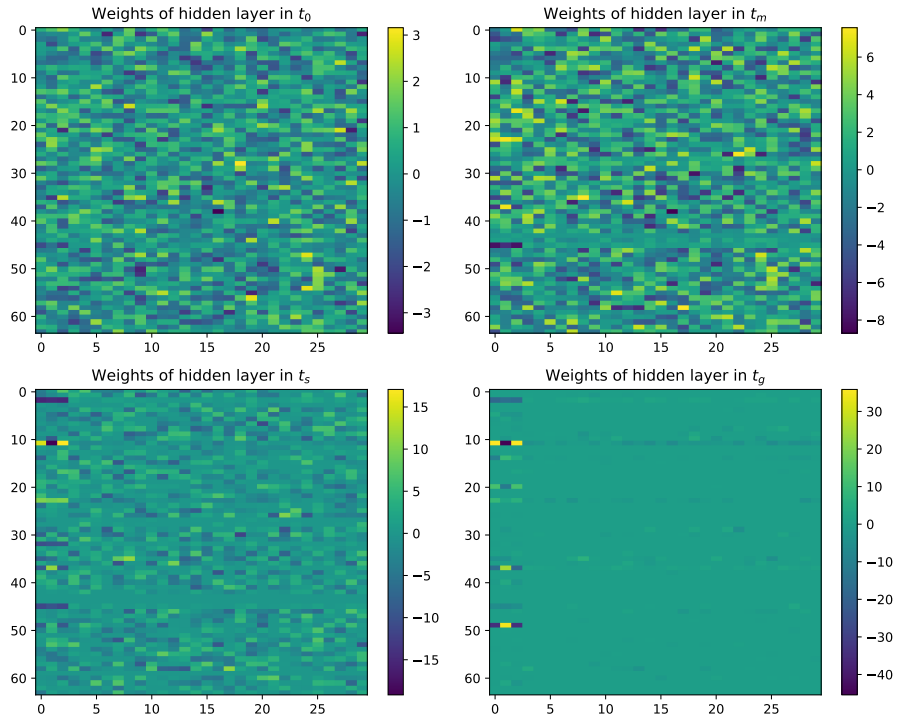

Figure 10: Evolution of the weights of the hidden layer for a $(3, 30)$-sparse parity dataset. The rest of the set-up is identical to what was described in B.

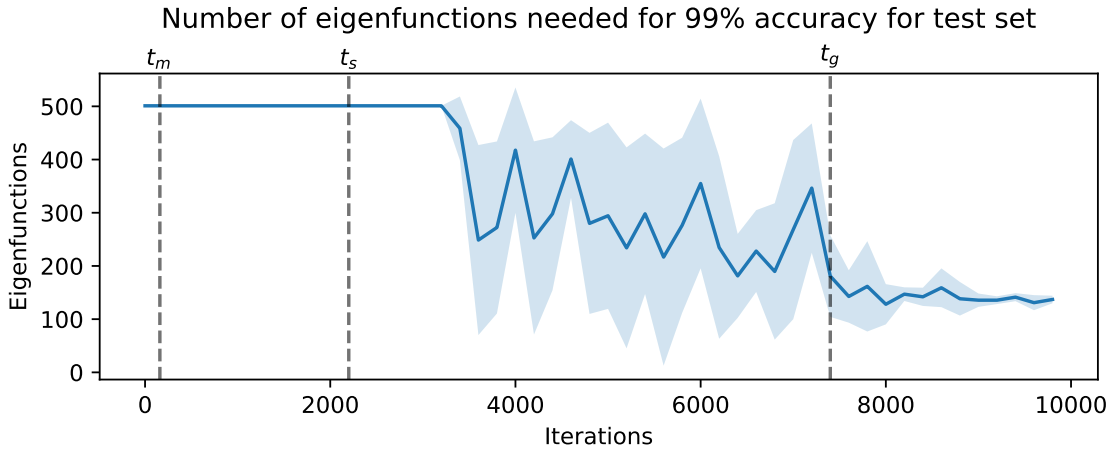

Figure 11: Eigenfunctions needed to reach 99% accuracy through training for a $(3, 30)$-sparse parity dataset. The rest of the set-up is identical to what was described in B.

D.2. 3 clean digits and 40 noisy

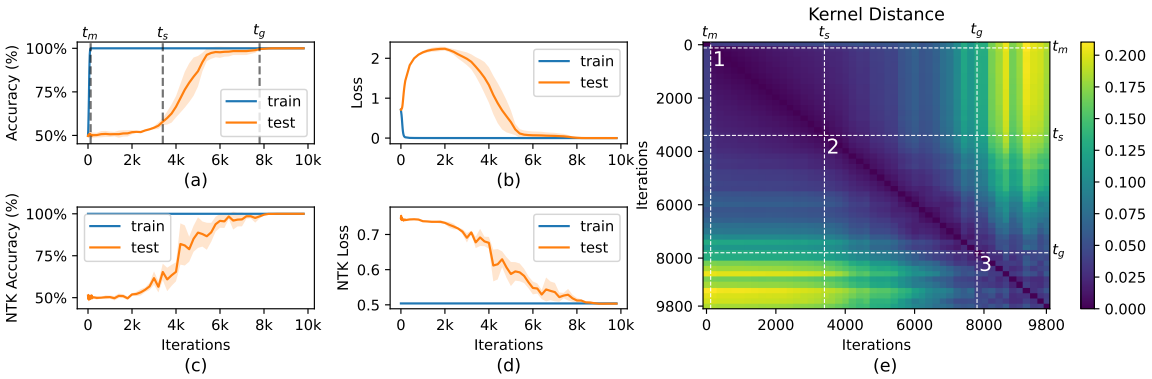

Figure 12: Training metrics and NTK evolution through training for a $(3, 40)$-sparse parity dataset. The rest of the experimental set-up is identical to what was described in B.

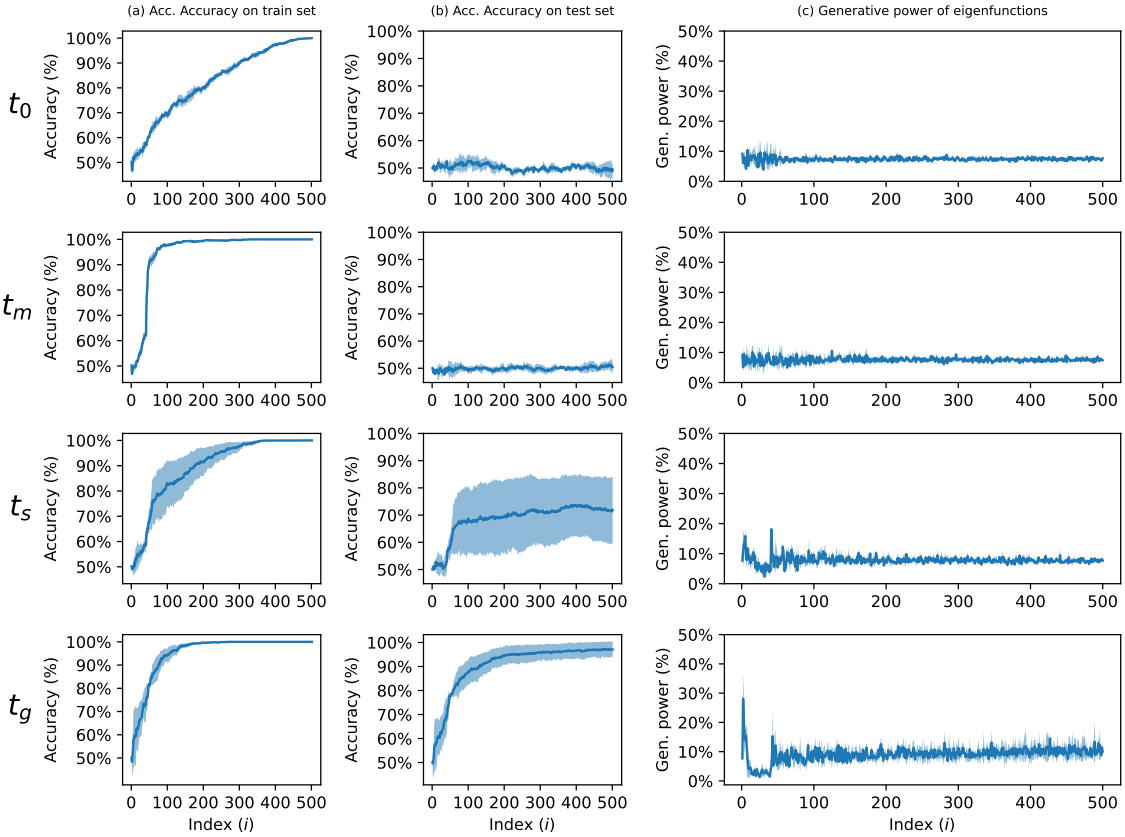

Figure 13: Accumulative accuracy and generative power of each eigenfunction for a $(3, 40)$-sparse parity dataset. The rest of the experimental set-up is identical to what was described in B.

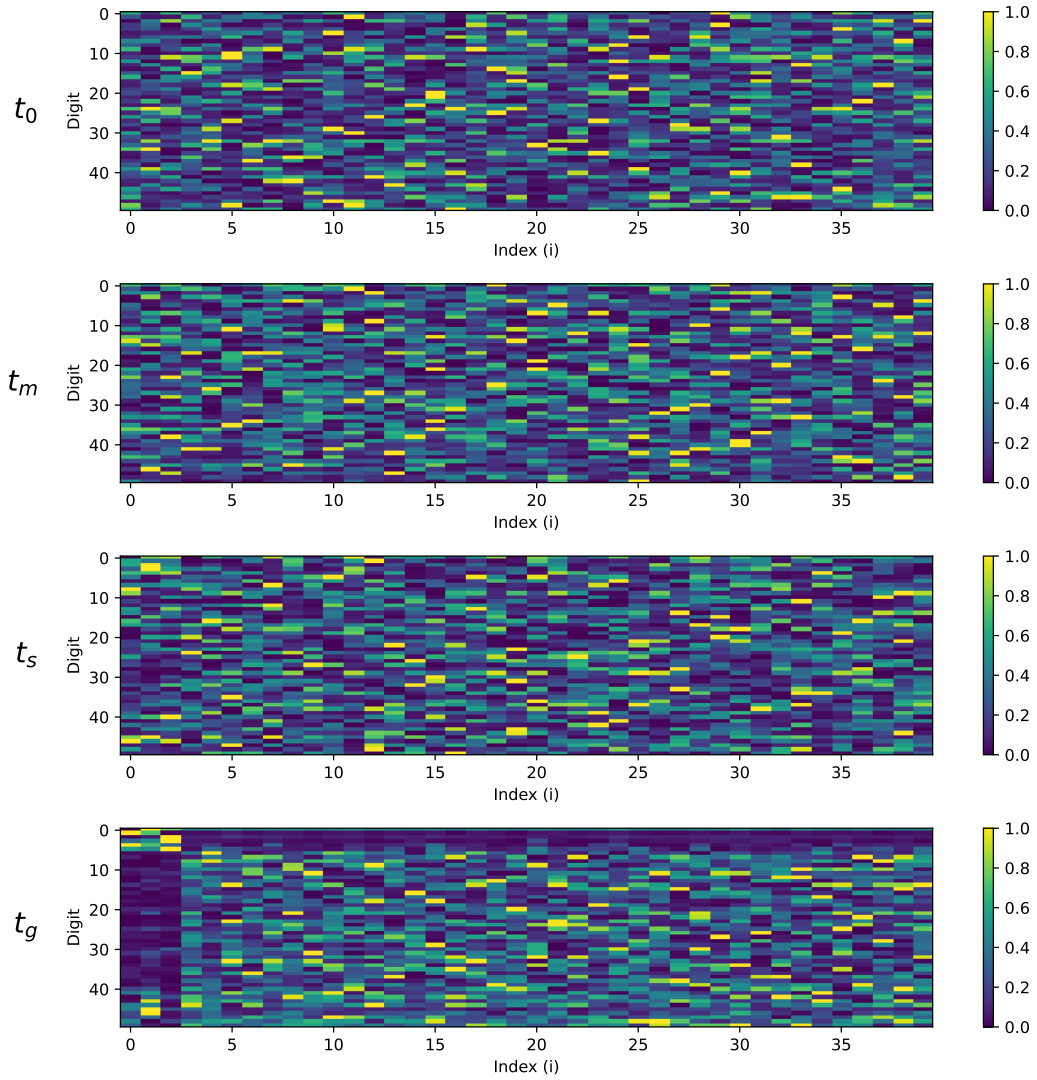

Figure 14: Accumulative accuracy and generative power for each for a $(3, 40)$-sparse parity dataset. The rest of the set-up is identical to what was described in B.

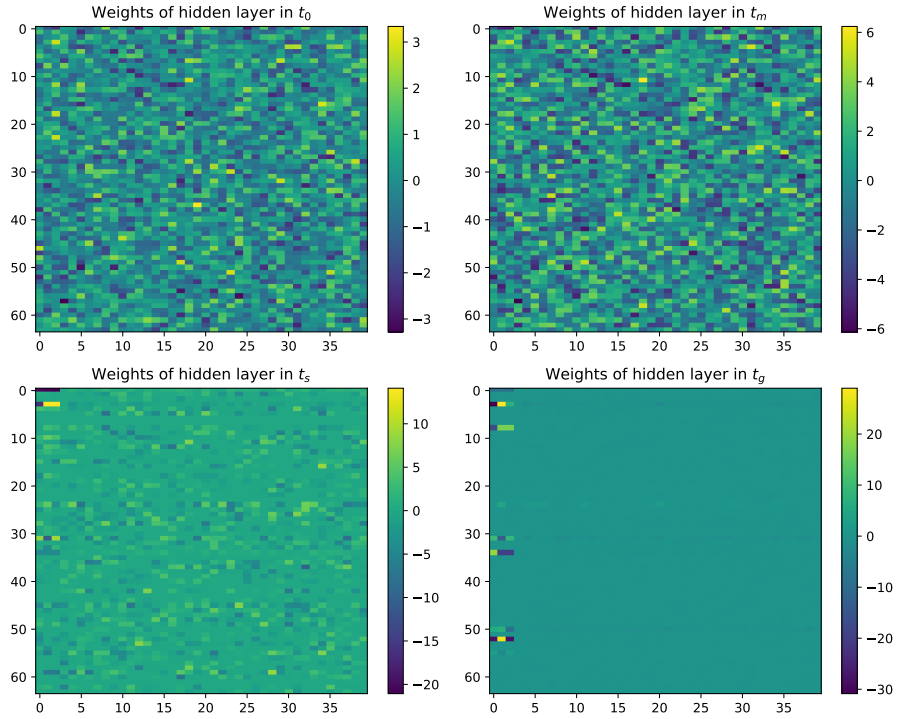

Figure 15: Evolution of the weights of the hidden layer for a $(3, 40)$-sparse parity dataset. The rest of the set-up is identical to what was described in B.

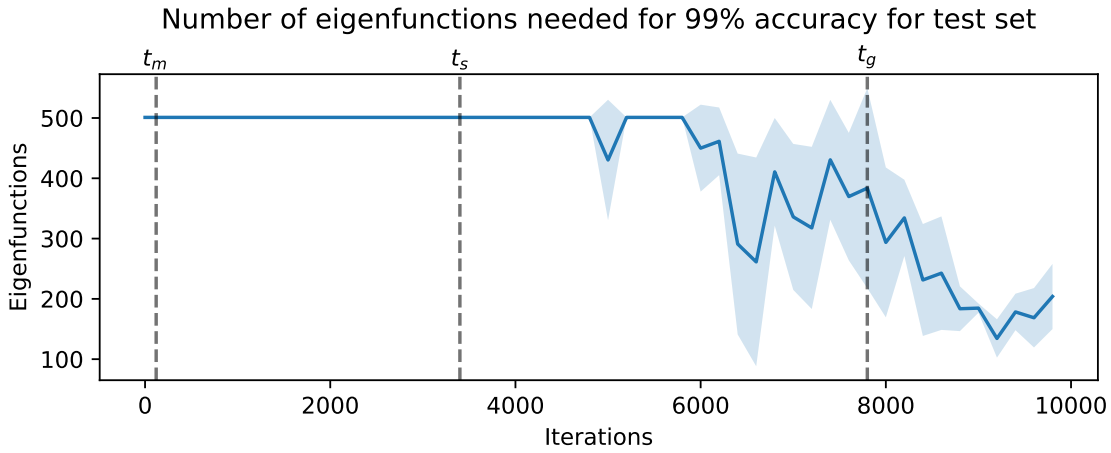

Figure 16: Eigenfunctions needed to reach 99% accuracy through training for a $(3, 40)$-sparse parity dataset. The rest of the set-up is identical to what was described in B.