Grokking as Compression: A Nonlinear Complexity Perspective

Ziming Liu[12*] **Ziqian Zhong**[1*] **Max Tegmark**[12]
[1] MIT [2] IAIFI
{zmliu, ziqianz, tegmark}@mit.edu

Abstract

We attribute grokking, the phenomenon where generalization is much delayed after memorization, to compression. We define *linear mapping number* (LMN) to measure network complexity, which is a generalized version of linear region number for ReLU networks. LMN can nicely characterize neural network compression before generalization. Although L_2 norm has been popular to characterize model complexity, we argue in favor of LMN for a number of reasons: (1) LMN can be naturally interpreted as information/computation, while L_2 cannot. (2) In the compression phase, LMN has nice linear relations with test losses, while L_2 is correlated with test losses in a complicated nonlinear way. (3) LMN also reveals an intriguing phenomenon of the XOR network switching between two generalization solutions, while L_2 does not. Besides explaning grokking, we argue that LMN is a promising candidate as the neural network version of the Kolmogorov complexity, since it explicitly considers local or conditioned linear computations aligned with the nature of modern artificial neural networks.

1 Introduction

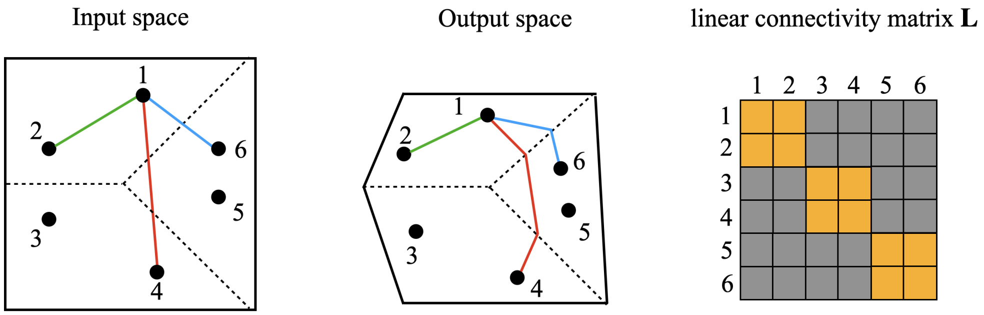
Grokking, the phenomenon where generalization happens long after memorization [1], is challenging our understanding of deep learning. Although there have been a few seemingly independent explanations of grokking [2–12], many of them share a similar high-level idea which is "grokking is compression": There exist a generalization solution and a memorization solution; the memorization solution is easier to be learned so learned at first, but the generalization solution is more efficient so emerges later. Although various measures have been proposed to characterize the process of "compression", e.g., L_2 [4], Fourier gap [6], network efficiency [11], neither of these measures admits a natural interpretation as information/computation complexity (most are, at best, proxies).

We propose a metric called *linear mapping number* (LMN), which measures the complexity of a network (or a subnetwork). In brief, LMN is a generalized version of the linear region number for ReLU networks. ReLU networks are known to represent piecewise linear functions; they partition input space into regions on which the network is a local linear mapping; different regions have different linear mappings, as shown Figure 1. Geometrically, one can think of ReLU networks as origami, i.e., folding flat input space (Figure 1 left) into complicated shapes (Figure 1 middle), and the number of linear regions measures the network complexity. LMN generalizes the concept of linear region number to networks with smooth activations.

We argue that LMN is a better metric than L_2, which has been used to measure network complexity in deep learning, especially for grokking [4]. A conceptual example is linear networks, which can only represent linear mappings even when they are deep. For linear networks, LMN always gives 1, but L_2 can be arbitrary hence not very informative. Moreover, LMN can be naturally interpreted as information: if one wants to compress a network into (input-dependent) linear mappings, then the compressed information is basically LMN times the size of one linear mapping.

[*] Equal contribution

36th Conference on Neural Information Processing Systems (NeurIPS 2022).



	1	2	3	4	5	6
1	🟧	🟧	⬛	⬛	⬛	⬛
2	🟧	🟧	⬛	⬛	⬛	⬛
3	⬛	⬛	🟧	🟧	⬛	⬛
4	⬛	⬛	🟧	🟧	⬛	⬛
5	⬛	⬛	⬛	⬛	🟧	🟧
6	⬛	⬛	⬛	⬛	🟧	🟧

Figure 1: Linear mapping number (LMN) is a generalized version of linear region number for ReLU networks. A ReLU network partitions input space into piece-wise linear regions. If two points lie in the same linear region or different linear regions, the line connecting them in the input space (left) will remain linear (green) or turn into non-linear curves (red and blue) in the output space (middle). We can construct a linear connectivity matrix (right) to characterize whether two points lie on the same linear region, which is applicable to networks with any activations. Based on the Von Neumann entropy of the matrix, we can estimate the number of linear mappings (details in Section 2).

We use LMN to characterize the compression process of grokking on three algorithmic tasks: modular additon, permutation group S_4 and multi-digit XOR. After memorization and before generalization, the LMN decreases steadily, and has a strong linear relation with test loss. By contrast, L_2 is correlated with test losses in a complicated nonlinear way. For modular addition and permutation, the LMN starts to level off after grokking, as expected. For multi-digit XOR, LMN displays an unexpected double-descent after grokking. This reveals something intriguing about the XOR case, which has two (rather than one) generalization solutions which are almost degenerate, so the network jumps between these two solutions.

This paper is organized as follows: In Section 2, we define linear mapping number (LMN). In Section 3, we use LMN to explain grokking, showing that it is related to L_2 but also better than L_2 in serveral senses. We discuss related works in Section 4.

2 Linear Mapping Number (LMN)

The linear mapping number (LMN) is a generalization of the linear region number for ReLU networks. For simplicity, let us first consider ReLU networks. A ReLU network partitions input space into linear regions, where in each region the ReLU network behaves like a linear mapping locally, although different linear regions correspond to different linear mappings (see Figure 1). The number of linear regions has been proposed to measure network complexity for ReLU networks [13, 14].

While the linear region number is only defined for networks with ReLU activations, our proposed linear mapping number is defined for networks with *any* activations, including smooth ones. However, ReLU networks point to a route for how to define LMN generally. As illustrated in Figure 1, if two samples lie in the same or different linear regions, a straight line connecting them in input space (Figure 1 left) will remain linear or become non-linear in output space (Figure 1 middle). This inspires us to measure "linear connectivity" between two samples: The more linear the output line is, the larger the linear connectivity is. For a network $\mathbf{f}:\mathbb{R}^{d_1}\to\mathbb{R}^{d_2}$, and two input samples $\mathbf{x}^{(i)},\mathbf{x}^{(j)}\in\mathbb{R}^{d_1}$, $i,j\in[N]$, we denote the linear connectivity of them as $L_{ij}\in\mathbb{R}$. We interpolate linearly between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ in input space:

$$\mathbf{x}^{(i,j)}(\lambda)=\mathbf{x}^{(i)}+\lambda(\mathbf{x}^{(j)}-\mathbf{x}^{(i)}),\ \lambda\in[0,1], \tag{1}$$

which corresponds to the output curve $\mathbf{y}^{(i,j)}(\lambda)=\mathbf{f}(\mathbf{x}^{(i,j)}(\lambda))\in\mathbb{R}^{d_2}$. The $k^{\rm th}$ dimension $\mathbf{y}_k^{(i,j)}(\lambda)$ is simply a scalar function of λ, so we can evaluate its linearity by doing linear regression and calculating r^2 (the square of the Pearson correlation coefficient). We define L_{ij} as the average of r^2 over dimensions k, i.e.,

$$L_{ij}\equiv\frac{1}{d_2}\sum_{k=1}^{d_2}r^2(\mathbf{y}_k^{(i,j)}(\lambda),\lambda). \tag{2}$$

Note that $L_{ij} \in [0,1]$. The r^2 is measured using uniform points on $\lambda \in [0,1]$ [2]. When $\mathbf{y}^{(i,j)}(\lambda)$ is a straight line, $L_{ij} = 1$; when $\mathbf{y}^{(i,j)}(\lambda)$ resembles a symmetric parabola, $L_{ij} = 0$. We define self-connectivity $L_{ii} \equiv 1$. In summary, larger L_{ij} means that the network behaves more like a linear mapping for sample i and j (i.e., two samples need only one shared linear mapping), while smaller L_{ij} means the network behaves non-linearly in-between sample i and j. We can stack L_{ij} into a matrix $\mathbf{L}$ such that $\mathbf{L}_{ij} = L_{ij}$, and call $\mathbf{L}$ the linear connectivity matrix (Figure 1 right).

If we say linearly connected samples belong to the same linear mapping, then the problem of counting linear mappings boils down to the problem of clustering: given the sample similarity matrix $\mathbf{L}$, how many clusters are there? Since the number of clusters is a discrete quantity and determining it may be non-robust or hyper-parameter dependent, we use a soft estimator leveraging the eigenvalue structure of the similarity matrix inspired by Von Neumann entropy [15]. Define λ_i $(i = 1, \cdots, N)$ as the eigenvalues of $\mathbf{L}$. Note that $\mathbf{L}$ is symmetric ($\mathbf{L}_{ij} = \mathbf{L}_{ji}$) hence all eiganvalues are real. $\mathbf{L}$ is almost semi-positive definite, i.e., all eigenvalues large in magnitude are positive, but there might be a few small negative eigenvalues (see Appendix B), which we take their absolute values. We define normalized eigenvalues $\tilde{\lambda}_i = |\lambda_i|/(\sum_{j=1}^{N}|\lambda_j|)$. Then we treat the normalized eigenvalue vector $(\tilde{\lambda}_1, \tilde{\lambda}_2, \cdots, \tilde{\lambda}_N)$ as a probability distribution. We define the nonlinear complexity of the distribution (measured in bits) as

$$S_{\rm NL} \equiv -\sum_i \tilde{\lambda}_i {\rm log}_2 \tilde{\lambda}_i \tag{3}$$

and define the number of linear mappings LMN as $\text{LMN} \equiv 2^{S_{\rm NL}}$. Note that given a data set $\mathbf{x}^{(i)}$, the quantity $S_{\rm NL}$ defines a measure of the nonlinear complexity of *any* function, regardless of whether it is defined as a neural network or not, and that $S_{\rm NL} = 0$ for any linear or affine function.

To get some intuition of the definition above, let us consider a case where there are c clusters with each cluster having the equal size N/c, and samples are perfectly linearly connected to other samples within the cluster. In this case, $\mathbf{L}$ is a block-diagonal matrix with c blocks ($c = 3$ illustrated in Figure 1 right), each block being an all-one matrix. The normalized eigenvalue vector is then $\tilde{\lambda}_i = 1/c$ $(1 \le i \le c)$ and $\tilde{\lambda}_i = 0$ $(c < i \le N)$, whose entropy is $S = \log(c)$, resulting in $\text{LMN} = c$, as expected. Note that LMN does not only apply to the whole network, but also to any sub-network. In particular, LMN between an intermediate layer and the output layer is of interest.

3 Using LMN to explain grokking

In this Section, we show that LMN is able to characterize the compression process of network complexity before grokking. LMN steadily decreases between memorization and generalization.

Experiment setup We train three-layer fully-connected networks with SiLU activations [16] to perform algorithmic tasks, including {addition modulo 31, permutation composition on S_4, 5-digit bitwise XOR}. The neural network parameters (including embeddings) are trained with the AdamW optimizer (learning rate 10^{-3}, weight decay 0.2) on cross-entropy loss for 20000 steps. The embedding dimension is 32, the hidden dimension is 100, and the output dimension is {31, 24, 32}. An 80-20 train-test split is performed on all possible inputs.

Results LMN is measured between the first hidden layer and the output logit layer [3]. In Figure 2, we plotted the LMN and losses during the training course for the three tasks. We denote the period before training accuracy reaches 100% (overfitting point) the memorizing phase, the period after that but before testing accuracy reaches 100% (generalizing point) the generalizing phase, and the remaining period finalizing phase. We see that the LMN decreases during the generalizing phase, revealing the "hidden" compression process of the network. Furthermore, the LMN is more linearly correlated than the test loss comparing to the L_2 norm of the model parameters.

An intriguing phenomenon in XOR In the 5-digit bitwise XOR task, we discovered a previously undescribed phenomenon: the LMN formed a double-descent-like shape during the finalizing phase;

[2] In practice, we use 21 uniformly spaced points on $\lambda \in [0,1]$, i.e., $\lambda = 0.0, 0.05, 0.1, \cdots, 0.95, 1.0$. The r^2 between variable x and y is $r^2(x,y) = (\langle xy\rangle - \langle x\rangle\langle y\rangle)^2/(\langle x^2\rangle - \langle x\rangle^2)(\langle y^2\rangle - \langle y\rangle^2)$, where $\langle\cdot\rangle$ means averging over samples.

[3] The first hidden layer is the most meaningful one for a three-layer network. The results for the embedding layer and the second hidden layer are shown in Appendix A.

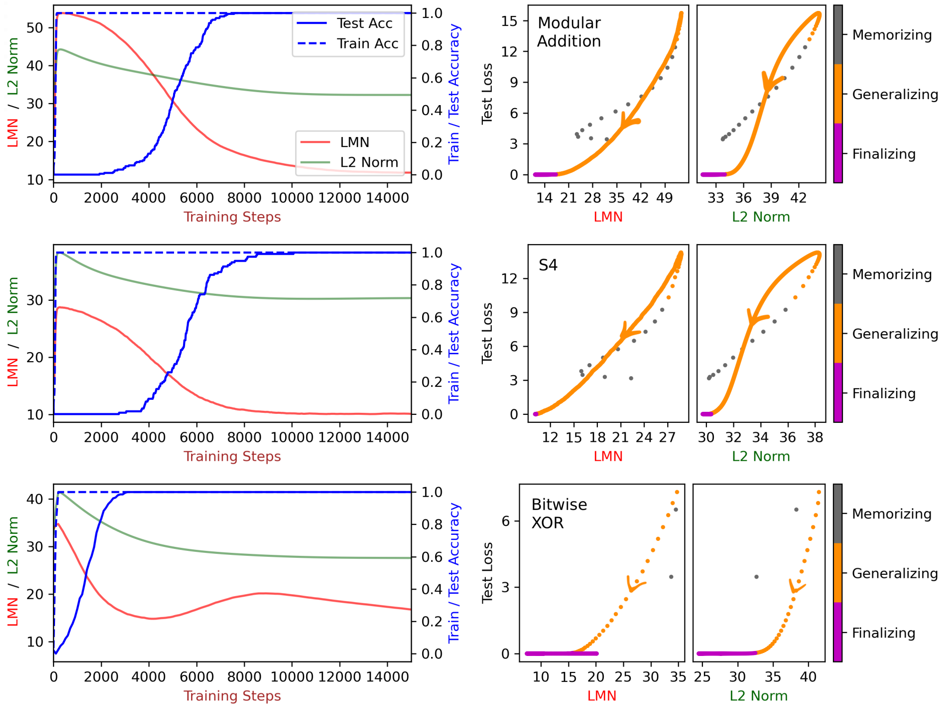


Figure 2: Train & test accuracy, LMN (linear mapping number) after the first layer and L_2 norm of model parameters during the training processes. The three rows correspond to three different algorithmic tasks. *Top:* Modular addition. *Middle:* S4 group operation. *Bottom:* Bitwise XOR.

the LMN increases briefly after generalization before decreasing again. We believe the phenomenon is due to two possible solutions for handling individual bits: we could create mapping for all the four possible pairs $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$, or reduce the number of pairs to three by symmetry (handling $(0,1)$ and $(1,0)$ identically). While the latter is more efficient in terms of internal representations, the former could produce better results earlier in the finalizing phases, as the model might be unable to handle symmetries perfectly. In the period where the LMN increases after generalizing, the model could be handling asymmetries in the model: adding separate treatments for $(0,1)$ and $(1,0)$ pairs, and only favoring the more symmetric treatment after that. Evidence for the explanation is that the two turning points of the LMN are 15 and 20, which happen to be 5×3 and 5×4 (there are 5 digits in total; for each digit, either memorize 3 samples or 4 samples). Mechanistic investigation of this phenomenon is left for future study.

4 Related Works and Discussions

Grokking is the phenomenon where generalization happens long after overfitting [1]. There are some attempts to understand grokking by studying toy models [2, 9], defining measures to characterize the dynamics [3, 4, 6, 11, 10], and linking to double descent [7] and optimization [8]. This work studies grokking from computation/information complexity.

Complexity measures for deep learning To understand why deep learning generalizes, a number of complexity measures are proposed [17–19]. From the perspective of information (the minimal number of linear mappings required to simulate the network), linear region number is used to measure complexity of ReLU networks [13, 14], and our work extends it to linear mapping number which accommodates general networks with any activation.

Compression and deep learning The theory of information bottleneck [20] suggests a compression phase followed by a fitting phase, although the compression story is sensitive to technical details [21]. Recently the success of language models is also attributed to compression [22]. We agree that the perspectives of information and compression are very likely the key to unlock generalization puzzles of deep learning, and our proposed LMN might be a useful metric in this regard. We would like to test the usability of LMN on a broad range of tasks and architectures in the future.

Acknowledgement

ZL and MT are supported by IAIFI through NSF grant PHY-2019786, the Foundational Questions Institute and the Rothberg Family Fund for Cognitive Science.

References

[1] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

[2] Ziming Liu, Ouail Kitouni, Niklas S Nolte, Eric Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems*, 35:34651–34663, 2022.

[3] Neel Nanda, Lawrence Chan, Tom Liberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.

[4] Ziming Liu, Eric J Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. *arXiv preprint arXiv:2210.01117*, 2022.

[5] William Merrill, Nikolaos Tsilivis, and Aman Shukla. A tale of two circuits: Grokking as competition of sparse and dense subnetworks. *arXiv preprint arXiv:2303.11873*, 2023.

[6] Boaz Barak, Benjamin Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: Sgd learns parities near the computational limit. *Advances in Neural Information Processing Systems*, 35:21750–21764, 2022.

[7] Xander Davies, Lauro Langosco, and David Krueger. Unifying grokking and double descent. *arXiv preprint arXiv:2303.06173*, 2023.

[8] Vimal Thilak, Etai Littwin, Shuangfei Zhai, Omid Saremi, Roni Paiss, and Joshua Susskind. The slingshot mechanism: An empirical study of adaptive optimizers and the grokking phenomenon. *arXiv preprint arXiv:2206.04817*, 2022.

[9] Andrey Gromov. Grokking modular arithmetic. *arXiv preprint arXiv:2301.02679*, 2023.

[10] Pascal Notsawo Jr, Hattie Zhou, Mohammad Pezeshki, Irina Rish, Guillaume Dumas, et al. Predicting grokking long before it happens: A look into the loss landscape of models which grok. *arXiv preprint arXiv:2306.13253*, 2023.

[11] Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency. *arXiv preprint arXiv:2309.02390*, 2023.

[12] Bojan Žunkovič and Enej Ilievski. Grokking phase transitions in learning local rules with gradient descent. *arXiv preprint arXiv:2210.15435*, 2022.

[13] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.

[14] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning*, pages 2596–2604. PMLR, 2019.

[15] John Von Neumann. *Mathematische grundlagen der quantenmechanik*, volume 38. Springer-Verlag, 2013.

[16] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

[17] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. *arXiv preprint arXiv:1912.02178*, 2019.

[18] Silviu-Marian Udrescu and Max Tegmark. Symbolic pregression: Discovering physical laws from distorted video. *Physical Review E*, 103(4):043307, 2021.

[19] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *international conference on machine learning*, pages 2847–2854. PMLR, 2017.

[20] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[21] Andrew Michael Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan Daniel Tracey, and David Daniel Cox. On the information bottleneck theory of deep learning. In *International Conference on Learning Representations*, 2018.

[22] Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, et al. Language modeling is compression. *arXiv preprint arXiv:2309.10668*, 2023.

[23] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17:395–416, 2007.

Appendix

A LMN for all layers

In Figure 2, we plotted LMN for the first hidden layer. Note that LMN can be defined for any layer, including the embedding layer and the second hidden layer. For modular addition, we show the evolution of LMN for all layers in Figure 3. It is clear that only the first hidden layer is sensitive to the hidden progress of the network after memorization and before generalization. The embedding layer and the second hidden layer are less meaningful. The embeddings are not processed by network yet, so they are not related to outputs in a meaningful way. The second layer, on the other hand, is highly correlated with the output logits, hence basically synchronizes with the training curve.

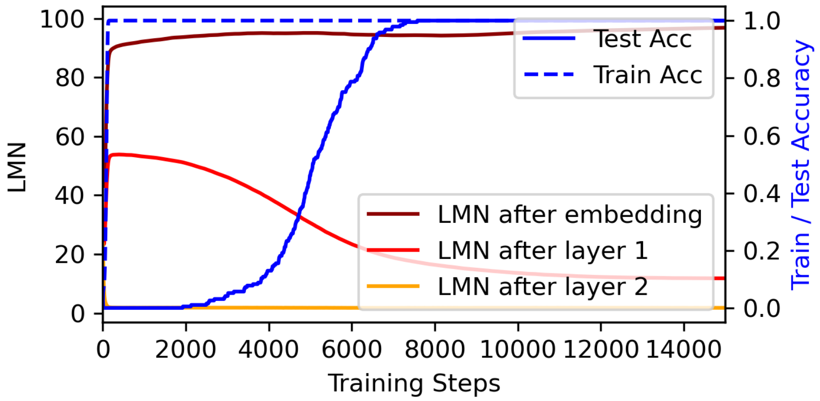


Figure 3: Evolution of LMN for all layers. Only the first hidden layer is meaningful to characterize the hidden progress before grokking, while the LMN of the embedding layer and the second hidden layer plateau quickly after memorization.

B Linear connectivity matrix and eigenvalue distribution

In the main paper, we defined linear connectivity matrix $\mathbf{L}$ in Eq. (2). Here in Figure 4, we visualize it and show its eigenvalues for three snapshots in training (for modular addition): at initialization (step 0), memorization (step 200) and generalization (step 7600). Comparing generalization to memorization, off-diagonal elements of $\mathbf{L}$ are on average larger for generalization, meaning that samples are more linearly connected, hence the network is simpler for generalization. At initialization, linear connectivity is also strong, due to the simplicity inductive bias at initialization (the network is close to be a linear network at initialization).

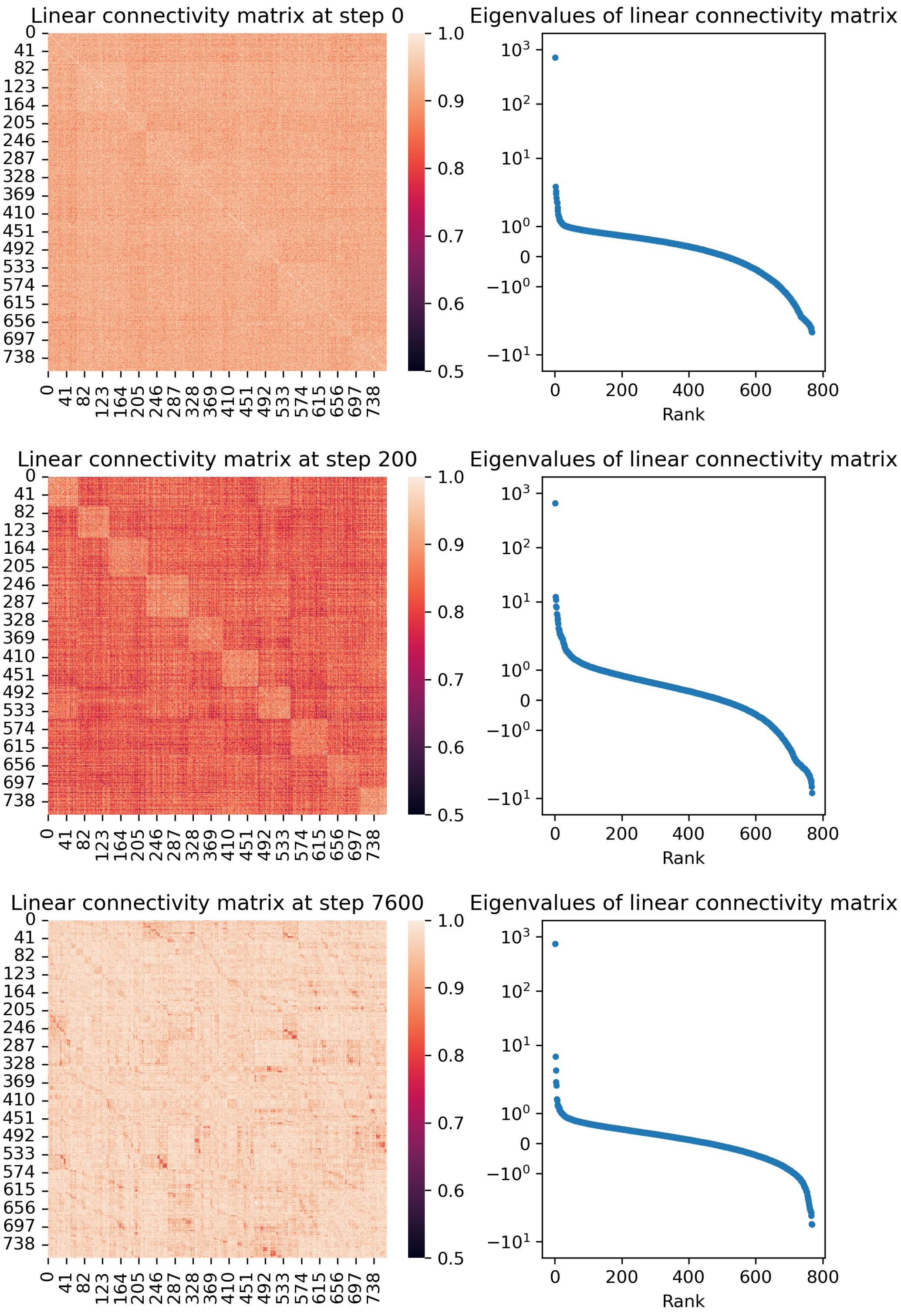


Figure 4: The evolution of the linear connectivity matrix (left) and its eigenvalues (right) at initialization (top), right after memorization (middle) and right after generalization (bottom). For display, we rearranged the input axes of the linear connectivity matrices into 10 clusters via spectral clustering (e.g. [23]).