Deep Networks Always Grok and Here is Why

Ahmed Imtiaz Humayun IMTIAZ@RICE.EDU
Rice University

Randall Balestriero RANDALL.BALESTRIERO@GMAIL.COM
Brown University

Richard Baraniuk RICHB@RICE.EDU
Rice University

Abstract

Grokking, or *delayed generalization*, is a phenomenon where generalization in a deep neural network (DNN) occurs long after achieving near zero training error. Previous studies have reported the occurrence of grokking in specific controlled settings, such as DNNs initialized with large-norm parameters or transformers trained on algorithmic datasets. We demonstrate that grokking is actually much more widespread and materializes in a wide range of practical settings, such as training of a convolutional neural network (CNN) on CIFAR10 or a Resnet on Imagenette. We introduce the new concept of *delayed robustness*, whereby a DNN groks adversarial examples and becomes robust, long after interpolation and/or generalization. We develop an analytical explanation for the emergence of both delayed generalization and delayed robustness based on the *local complexity* training dynamics of a DNN's input-output mapping. Our *local complexity* measures the density of so-called "linear regions" (aka, spline partition regions) that tile the DNN input space.

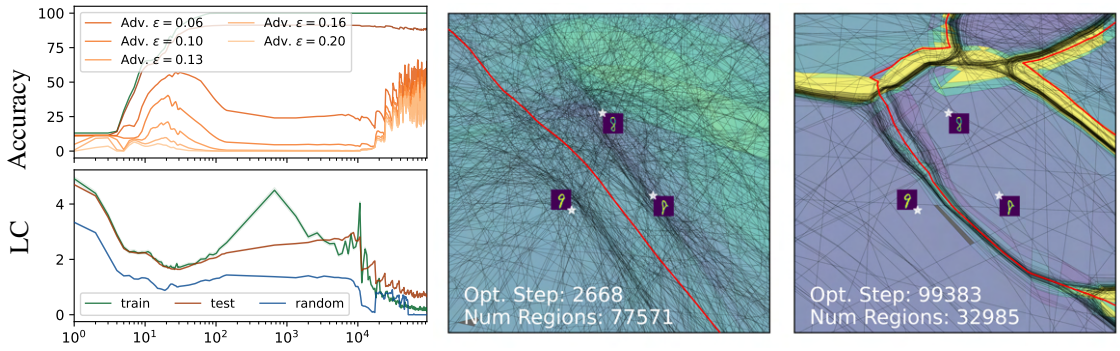

Figure 1: **Emergence of Robust Partition.** We train a 4-layer ReLU MLP of 200 width, on $1K$ samples from MNIST for 10^5 optimization steps, with batch size 200. The network starts grokking adversarial examples after approximately 10^4 optimization steps **(left-top)**. The local complexity around data points **(left-bottom)** follows a double descent curve with the final descent starting approximately around 10^4 optimization steps as well. *Where do the non-linearities migrate to?* In the **middle** and **right** images we present analytically computed visualizations of the DNN input space partition [9] before and after the network groks adversarial examples. The partition or *linear regions* are visualized across a 2D domain in the input space, that intersects three training samples. During the final descent in local complexity, a unique structure emerges in the DNN partition geometry, where many non-linearities (black lines) have concentrated around the decision boundary (red line), allowing the formation of a robust partition.

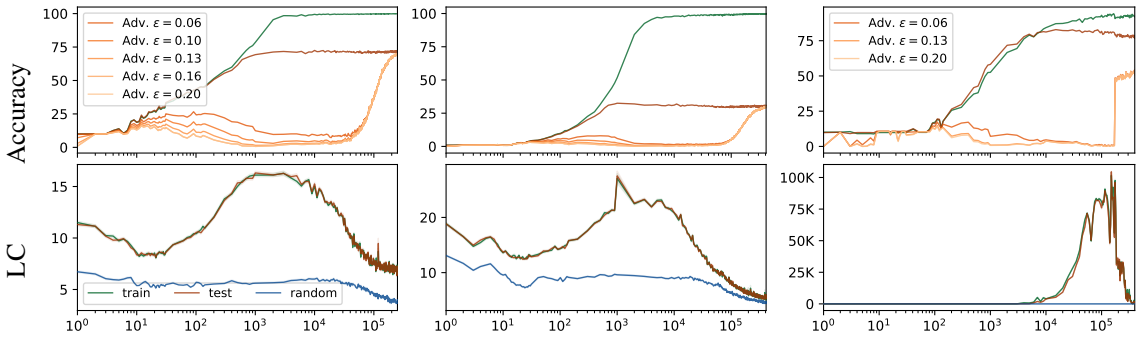

Figure 2: **Grokking across datasets and architectures.** From left to right, examples of delayed robustness emerging late in training for a CNN trained on CIFAR10, CNN trained on CIFAR100, and ResNet18 trained on the Imagenette. Clear double descent behavior visible in the local complexity of CNN with CIFAR10 and CIFAR100. The ResNet18 trained with Imagenette obtains a very high local complexity during the ascent phase of the complexity dynamics. To compute local complexity we consider 25 dimensional neighborhoods centered on 1024 train, test or random samples. We use $r = 0.005$ for CNN and $r = 10^{-4}$ for ResNet18.

1. Introduction

Grokking is a surprising phenomenon related to representation learning in Deep Neural Networks (DNNs) whereby DNNs may learn generalizing solutions to a task long after interpolating the training dataset, i.e., reaching near zero training error. It was first demonstrated by [19] on simple Transformer architectures performing modular addition or division. Subsequently, multiple studies have reported instances of grokking for settings outside of modular addition, e.g., DNNs initialized with large weight norms for MNIST, IMDb [14], or XOR cluster data [25]. For all the reported instances, DNNs that grok show a standard behavior in the training loss/accuracy curves approaching zero error as training progresses. The test error however, remains high even long after training error reaches zero. After a large number of training iterations, the DNN starts grokking–or generalizing–to the test data. This paper concerns the following question:

Question. *How subjective is the onset of grokking on the test data? When grokking does not manifest as a measurable change in the test set performance, could there exist an alternate test dataset for which grokking would occur?*
Observation. *For a number of training settings, with standard initialization with or without weight decay, DNNs grok adversarial samples long after generalizing on the test dataset. We dub this phenomenon* ***delayed robustness****, a novel form of grokking previously unreported.*
Question. *How can we explain both delayed generalization and delayed robustness?*
Observation. *To explain grokking, we propose a novel complexity measure based on the local non-linearity of the DNN. Our novel measure does not rely on the dataset, labels, or loss function that is used during training. We show that DNNs undergo a phase change in the local complexity (LC) averaged over data points during training. The phase change coincides with grokking.*
We come to the following conclusion: Grokking occurs due to the emergence of a robust input space partition by a DNN, through a linearization of the DNN function around training points as a consequence of training dynamics.

2. Measuring Local Complexity using the Deep Network Spline Partition

Barak et al. [3] introduced the notion of *progress measures* for DNN training, as scalar quantities that are causally linked with the training state of a network. We will introduce a novel measure based on the spline formulation of DNNs, that we review in Appendix A.1.

Suppose a domain is specified as the convex hull of a set of vertices $\boldsymbol{V} = [\boldsymbol{v}_1, \dots \boldsymbol{v}_p]^T$ in the DNN's input space. We wish to compute the local complexity or smoothness [7] for neighborhood $\mathcal{V} = conv(\boldsymbol{V})$. Let's denote the DNN layer weight as $W^{(\ell)} \triangleq [\boldsymbol{w}_1^{(\ell)}, \dots, \boldsymbol{w}_{D^{(\ell)}}^{(\ell)}]$, $b^{(\ell)}$ where ℓ is the layer index, $\boldsymbol{w}_i^{(\ell)}$ is the i-th row of $W^{(\ell)}$ or weight of the i-th neuron, and $D^{(\ell)}$ is the output space dimension of layer ℓ. The forward pass through this layer for $\boldsymbol{V}$ can be considered an inner product with each row of the weight matrix $W^{(\ell)}$ followed by a continuous piecewise linear activation function. Without loss of generality, let's consider ReLU as the activation function in our network. The partition at the input space of layer ℓ can therefore be expressed as the set of all hyperplane equations formed via the neuron weights such as $\partial\Omega = \bigcup_{i=1}^{D^{(\ell)}} \mathcal{H}_i^{(\ell)}$ and $\mathcal{H}_i^{(\ell)} = \left\{ \boldsymbol{x} \in \mathbb{R}^{D^{(\ell-1)}} : \langle \boldsymbol{w}_i^{(\ell)}, \boldsymbol{x} \rangle + \boldsymbol{b}_i^{(\ell)} = 0 \right\}$ which is also the set of layer ℓ non-linearities. Let, $\Phi = f_{1:\ell-1}(\mathcal{V})$ be the embedded representation of the neighborhood $\mathcal{V}$ by layer $\ell - 1$ of the network. Therefore, approximating the local complexity of $\mathcal{V}$ induced by layer ℓ, would be equivalent to counting the number of linear regions in $\Phi \cap \partial\Omega = \bigcup_{i=1}^{D^{(\ell)}} \Phi \cap \mathcal{H}_i^{(\ell)}$. The local partition inside Φ results from an arrangement of hyperplanes; therefore the number of regions is of the order $\mathcal{N}^{D^{(\ell-1)}}$ [22], where $\mathcal{N} = |\{i : i = 1, 2..D^{(\ell)} \text{ and } \mathcal{H}_i^{(\ell)} \cap \Phi \neq \emptyset\}|$ is the number of hyperplanes from layer ℓ intersecting Φ. We consider $\mathcal{N}$ as a proxy for local complexity for any neighborhood Φ. To make computation tractable, let, $\Phi \approx \widehat{\Phi} = conv(f_{1:\ell-1}(\boldsymbol{V}))$. Therefore, for $\widehat{\Phi}$, any sign changes in layer ℓ pre-activations is due to the corresponding neuron hyperplanes intersecting $conv(\boldsymbol{V})$. Therefore for a single layer, the local complexity (LC) for a sample in the input space can be approximated by the number of neuron hyperplanes that intersect $\boldsymbol{V}$ embedded to that layers input space. If we consider input space neighborhoods with the same volume, then local complexity measures the un-normalized density of non-linearity in an input space locality. We highlight that this is tied to the VC-dimension of (ReLU) DNN [4] where the more regions are present the more expressive the decision boundary can be [15]. In Figure 7, we provide a visual explanation of our method for local complexity approximation through a cartoon schematic diagram. To summarize, we consider randomly oriented P dimensional ℓ_1 norm balls with radius r, i.e., cross-polytopes centered on any given data point x as a frame defining the neighborhood. We therefore follow the steps entailed in Figure 7 in a layerwise fashion, to approximate the local complexity in the prescribed neighborhood. Experimental setups are described in Appendix B

3. Local Complexity Training Dynamics and Grokking

In all our experiments either involving delayed generalization or robustness, we see three distinct phases in the dynamics of local complexity:

• *The first descent*, when the local complexity start by descending after initialization. This phase is subject to the network parameterization as well as initialization, e.g., when grokking is induced in the MLP-MNIST case with scaled initialization, we do not see the first descent (Figure 20, Figure 4).

• *The ascent phase*, when the local complexity accumulates around both training and test data points. The ascent phase happens ubiquitously, and the local complexity generally keeps ascending

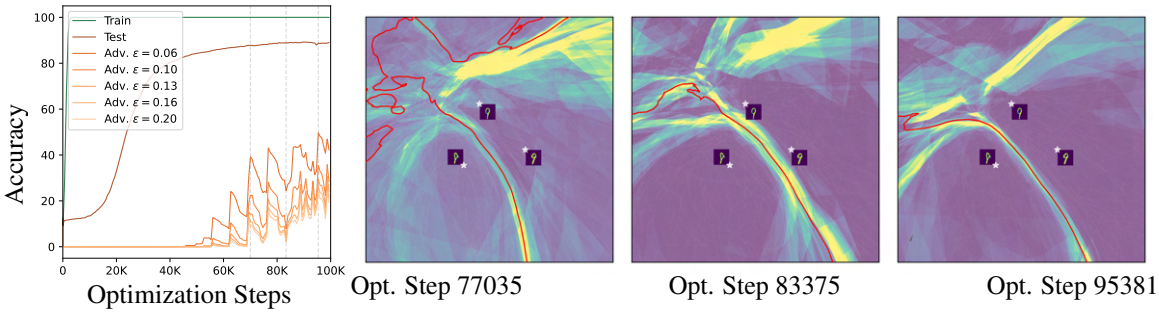

Figure 3: **Grokking visualized.** We induce grokking by randomly initializing a 4 depth 200 width ReLU MLP and scaling the initialized parameters by eigth following [14]. In the leftmost figure, we can see that the grokking is visible for both the test samples as well as adversarial examples generated using the test set. We see that the network robustness, periodically increases. By visualization the partition and curvature of the function across a 2D slice of the input space [9], we see that the network periodically increases the concentration of non-linearity around its decision boundary, making the boundary sharper at each robustness peak. This occurs even when the network doesn't undergo delayed generalization (Figure 1). As the local complexity around the decision boundary increases, the local complexity around data points farther from the decision boundary decreases (Figure 17). Anonymized video showing periodic accumulation https://bit.ly/grok-splinecam.

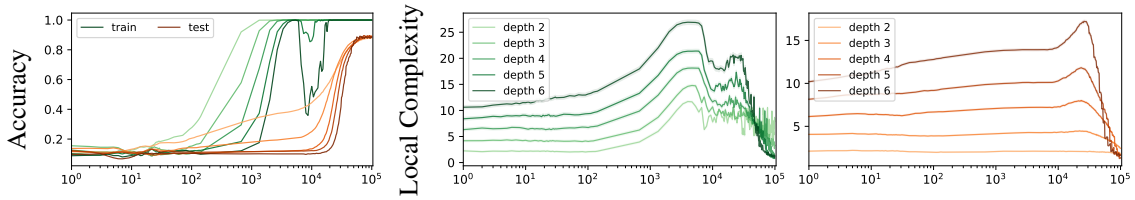

Figure 4: **Local complexity across depths.** From left to right, accuracy, local complexity around training and local complexity around test data points, for an MLP trained on MNIST with width 200 and varying depth. *As depth is increased the max LC during ascent phase becomes larger and second descent is expedited. We can also see a distinct second peak right before the descent phase.*

until training interpolation is reached (e.g., Figure 2). During the ascent phase, the training local complexity may be higher for training data points than for test data points, indicating an accumulation of non-linearities around training data compared to test data Figure 1.

• *The second descent phase* or region migration phase, during which the network moves the linear regions or non-linearities away from the training and test data points. Focusing on Figure 1-bottom-left and Figure 20 for the MLP-MNIST setting, one perplexing observation that we make is that the local complexity around random points – uniformly sampled from the domain of the data – also decreases during the final descent phase. This would mean that the non-linearities are not randomly moving away from the training data, but moving to a part of the input space where under expectation, we do not get samples for LC measurement. To better understand the phenomenon, we consider a square domain $\mathbb{D}$ that passes through three MNIST training points, and use Splinecam [9] to analytically compute the input space partition on $\mathbb{D}$. In short, Splinecam uses the weights of the network to exactly compute the input space representation of each neuron's pre-activation zero-level set on $\mathbb{D}$ e.g., black lines in Figure 1, Figure 3, and Figure 17. Through these visualizations, we see clear evidence that *during the second descent phases of training, linear regions or the non-linearities of the network, migrate close to the decision boundary creating a robust partition in the input space.*

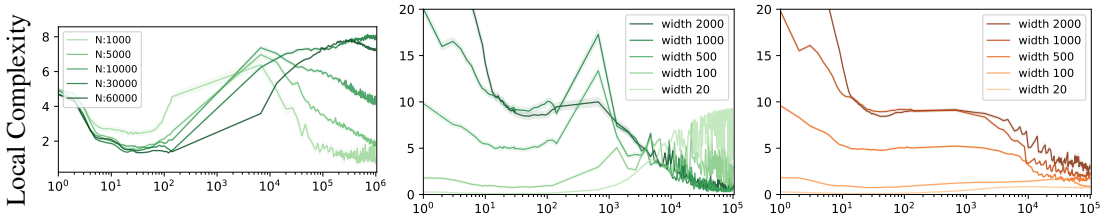

Figure 5: **Left: Memorization requirement delays grok.** When training an MLP on varying number of randomly labeled MNIST samples, we see that with increase in the number of samples, the ascent phase gets delayed. This shows that with increased memorization requirement the network takes longer to complete ascent and later undergo region migration.**Middle & Right: Increasing width hastens region migration.** LC dynamics while training an MLP with **varying width** on MNIST. For the peak LC during the ascent phase, we see an initial increase and then decrease as the network gets overparameterized.

Moreover, during region migration, *the network intends to lower the local complexity around training points*, resulting in a decrease in local complexity around training even compared to test data points.

4. What Affects the Progress Measure?

Parameterization In Figures 5, 18 and 20, we see that increasing the number of parameters either by increasing depth, or by increasing width of the network in our MNIST-MLP experiments, hastens region migration, therefore makes grokking happen earlier.
Weight Decay. We train a CNN with depth 5 and width 32 on CIFAR10 with varying weight decay. In Figure 21 we present the train, test and random LC for our experiments for neighborhoods of different radius. Weight decay does not seem to have a monotonic behavior as it both delays and hastens region migration, based on the amount of weight decay.
Batch normalization removes grokking. In Appendix D, we show that at each layer ℓ of a DN, BN explicitly adapts the partition so that the partition boundaries are as close to the training data as possible. This is confirmed by our experiments in Figure 14 where we see that grokking adversarial examples ceases to occur compared to the non-batchnorm setting in Figure 2.
Activation function. While most of our experiments use ReLU activated networks, in Figure 25 we present results for a GeLU activated MLP, as well as in Figure 10 we present results for a GeLU activated Transformer. For both settings we see similar training dynamics as with ReLU.
Effect of Training Data. We control the training dataset to either induce higher generalization on higher memorization. We increase the number of samples in our dataset to monitor the effect of grokking Figure 19 and LC Figure 23. We see that increasing the size of the dataset hastens grokking.

5. Conclusions and Limitations

We proposed a thorough empirical study of grokking, both on the test dataset and adversarial examples generated using the test dataset. We obtained new observations hinting that grokking is a common phenomenon for DNNs. Delving into DNN geometry, we isolate the root cause of both delayed generalization and robustness as region migration, i.e., a descent of local complexity around training data points that occurs in the latest phase of training. Again, the observation of such migration of the DNN partition is a new discovery of its own right. We hope that our analysis has provided novel insights into DNNs training dynamics from which grokking naturally emerges.

References

[1] Randall Balestriero and Richard Baraniuk. A spline theory of deep networks. In *Proc. ICML*, pages 374–383, 2018.

[2] Randall Balestriero and Richard G Baraniuk. Batch normalization explained. *arXiv preprint arXiv:2209.14778*, 2022.

[3] 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.

[4] Peter L Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *The Journal of Machine Learning Research*, 20(1):2285–2301, 2019.

[5] Matteo Gamba, Adrian Chmielewski-Anders, Josephine Sullivan, Hossein Azizpour, and Marten Bjorkman. Are all linear regions created equal? In *AISTATS*, pages 6573–6590, 2022.

[6] Christian Garbin, Xingquan Zhu, and Oge Marques. Dropout vs. batch normalization: an empirical study of their impact to deep learning. *Multimedia Tools and Applications*, 79: 12777–12815, 2020.

[7] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. *arXiv preprint arXiv:1901.09021*, 2019.

[8] Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Polarity sampling: Quality and diversity control of pre-trained generative networks via singular values. In *CVPR*, pages 10641–10650, 2022.

[9] Ahmed Imtiaz Humayun, Randall Balestriero, Guha Balakrishnan, and Richard G. Baraniuk. Splinecam: Exact visualization and characterization of deep network geometry and decision boundaries. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3789–3798, June 2023.

[10] Ahmed Imtiaz Humayun, Randall Balestriero, Guha Balakrishnan, and Richard G Baraniuk. Splinecam: Exact visualization and characterization of deep network geometry and decision boundaries. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3789–3798, 2023.

[11] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[12] Xu Ji, Razvan Pascanu, R Devon Hjelm, Balaji Lakshminarayanan, and Andrea Vedaldi. Test sample accuracy scales with training sample density in neural networks. In *Conference on Lifelong Learning Agents*, pages 629–646. PMLR, 2022.

[13] Masayoshi Kubo, Ryotaro Banno, Hidetaka Manabe, and Masataka Minoji. Implicit regularization in over-parameterized neural networks. *arXiv preprint arXiv:1903.01997*, 2019.

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

[15] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *NeurIPS*, pages 2924–2932, 2014.

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

[17] Roman Novak, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. *arXiv preprint arXiv:1802.08760*, 2018.

[18] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

[19] 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.

[20] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the expressive power of deep neural networks. In *ICML*, pages 2847–2854, 2017.

[21] Jasper Tan, Daniel LeJeune, Blake Mason, Hamid Javadi, and Richard G Baraniuk. A blessing of dimensionality in membership inference through regularization. In *International Conference on Artificial Intelligence and Statistics*, pages 10968–10993. PMLR, 2023.

[22] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. *Handbook of discrete and computational geometry*. CRC press, 2017.

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

[24] Kexiang Xu, Aleksandar Ilić, Vesna Iršič, Sandi Klavžar, and Huimin Li. Comparing wiener complexity with eccentric complexity. *Discrete Applied Mathematics*, 290:7–16, 2021.

[25] Zhiwei Xu, Yutong Wang, Spencer Frei, Gal Vardi, and Wei Hu. Benign overfitting and grokking in relu networks for xor cluster data. *arXiv preprint arXiv:2310.02541*, 2023.

[26] Haoran You, Randall Balestriero, Zhihan Lu, Yutong Kou, Huihong Shi, Shunyao Zhang, Shang Wu, Yingyan Lin, and Richard Baraniuk. Max-affine spline insights into deep network pruning. *arXiv preprint arXiv:2101.02338*, 2021.

Appendix A. Empirical analysis of our proposed method

A.1. Deep Networks are Affine Spline Operators

DNNs primarily perform a sequential mapping of an input vector $\boldsymbol{x}$ through L nonlinear transformations, i.e., layers, as in

$$f_\theta(\boldsymbol{x}) \triangleq \boldsymbol{W}^{(L)} \dots \boldsymbol{a}\left(\boldsymbol{W}^{(2)}\boldsymbol{a}\left(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) + \boldsymbol{b}^{(2)}\right) \cdots + \boldsymbol{b}^{(L)}, \tag{1}$$

starting with some input $\boldsymbol{x}$. For any layer $\ell \in \{1, \dots, L\}$, the $\boldsymbol{W}^{(\ell)}$ weight matrix, and the $\boldsymbol{b}^{(\ell)}$ bias vector can be parameterized to control the type of operation for that layer, e.g., a circulant matrix as $\boldsymbol{W}^{(\ell)}$ results in a convolutional layer. The operator $\boldsymbol{a}$ is an element-wise nonlinearity, e.g., ReLU, and θ is the set of all parameters of the network. According to Balestriero and Baraniuk [1], for any $\boldsymbol{a}$ that is a continuous piecewise linear function, f_θ is a continuous piecewise affine spline operator. That is, there exists a partition Ω of the DNN's input space $\mathbb{R}^D$ (for example, Figure 6 left) comprised of non-overlapping regions that span the entire input space. On any region of the partition $\omega \in \Omega$, the DNN's input-output mapping is a simple affine mapping with parameters $(\boldsymbol{A}_\omega, \boldsymbol{b}_\omega)$. In short, we can express f_θ as

$$f_\theta(\boldsymbol{x}) = \sum_{\omega \in \Omega} (\boldsymbol{A}_\omega \boldsymbol{x} + \boldsymbol{b}_\omega) \mathbb{1}_{\{\boldsymbol{x} \in \omega\}}, \tag{2}$$

where, $\mathbb{1}_{\{\boldsymbol{x} \in \omega\}}$ is an indicator function that is non-zero for $x \in \omega$.

Curvature and Linear Regions. Formulations like that in Equation (2) that represent DNNs as continuous piecewise affine splines, have previously been employed to make theoretical studies amenable to actual DNNs, e.g. in generative modeling [8], network pruning [26], and OOD detection [12]. Empirical estimates of the density of linear regions in the spline partition have also been employed in sensitivity analysis [17], quantifying non-linearity [5], quantifying expressivity [20] or to estimate the complexity of spline functions [7]. We demonstrate the relationship between function curvature and linear region density through a toy example in Figure 6. In Figure 6-left and Figure 1-(middle,right), any contiguous line is a non-linearity in the input space, corresponding to a single neuron of the network. All the non-linearities re-orient themselves during training to be able to obtain the target function (Figure 6-right). Therefore, in Figure 6, we see that DNN partitions have higher density of linear regions/non-linearities/knots in the spline partition, where the function curvature is higher.

Computing the exact number of linear regions or piecewise-linear hyperplane intersections for an deep network with N-dimensional input space neighborhood has combinatorial complexity and therefore is intractable. This is one of the key motivations behind our approximation method.

MLP with zero bias. To validate our method, we start with a toy experiment with a linear MLP with width 400, depth 50, 784 dimensional input space, initialized with zero bias and random weights. In such a setting all the layerwise hyperplanes intersect the origin at their input space. We compute the LC around the input space origin using our method, for neighborhoods of varying radius $r = \{0.0001, 0.001, 0.01, 0.1, 1, 10\}$ and dimensionality $P = \{2, 10, 25, 50, 100, 200\}$. For all the trials, our method recovers all the layerwise hyperplane intersections, even with a neighborhood dimensionality of $P = 2$.

Non-Zero Bias Random MLP with shifting neighborhood. For a randomly initialized MLP, we expect to see lower local complexity as we move away from the origin [7]. For this experiment

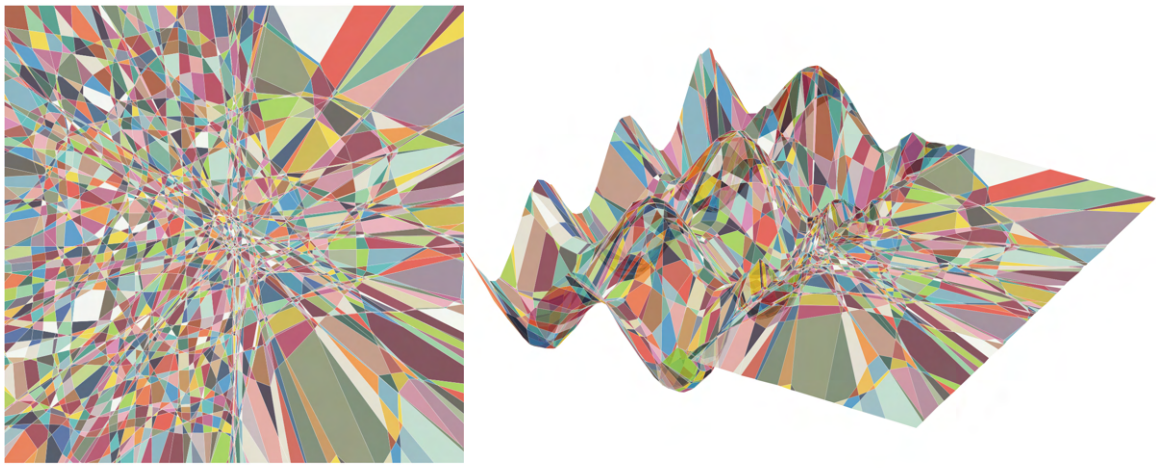

Figure 6: **Curvature and complexity.** Visual depiction of Equation (2) with a toy affine spline $S : \mathbb{R}^2 \to \mathbb{R}$, obtained by training an MLP to regress the piecewise function $f(x_1, x_2) = \{\sin(x_1) + \cos(x_2)\}\mathbb{1}_{x_1<0}$. Regions in the input space partition Ω (left) and the graph of the affine spline function (right) are randomly colored. The spline partition has significantly higher density of non-linearities for $x_1 < 0$, i.e., the local complexity is higher where the function has more curvature.

we take a width 100 depth 18 MLP with input dimensionality $d = 784$, Leaky-ReLU activation with negative slope 0.01. We start by computing LC at the origin $[0]^d$, and linearly shift towards the vector $[10]^d$. We see that for all the settings, shifting away from the origin reduces LC. LC gets saturated with increasing P, showing that lower dimensional neighborhoods can be good enough for approximating LC. Increasing r on the other hand, increases LC and reduces LC variations between shifts, since the neighborhood becomes larger and LC becomes less local.

Trained MLP comparison with SplineCam. For non-linear MLPs, we compare with the exact computation method Splinecam [9]. We take a depth 3 width 200 MLP and train it on MNIST for 100K training steps. For 20 different training checkpoints, we compute the local complexity in terms of the number of linear regions computed via SplineCam and number of hyperplane intersections via our proposed method. We compute the local complexity for 500 different training samples. For both our method and SplineCam we consider a radius of 0.001. For our method, we consider a neighborhood with dimensionality $P = 25$. We present the LC trajectories in Fig. 26. We can see that for both methods the local complexity follows a similar trend with a double descent behavior.

Deformation of neighborhood by deep networks. As mentioned in Appendix A, we compute the local complexity in a layerwise fashion by embedding a neighborhood $conv(V)$ into the input space for any layer and computing the number of hyperplane intersections with $conv(V^\ell)$, where V^ℓ is the embedded vertices at the input space of layer ℓ. The approximation of local complexity is therefore

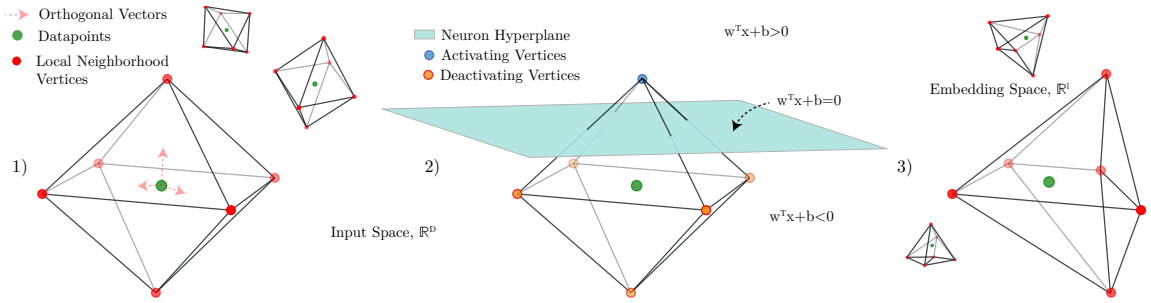

Figure 7: **Local Complexity Approximation.** 1) Given a point in the input space $x \in \mathbb{R}^D$, we start by sampling P orthonormal vectors $\{v_1, v_2, ..., v_P\}$ to obtain cross-polytopal frame $\boldsymbol{V}_x = \{x \pm r * v_p \forall p\}$ centered on x, where r is a radius parameter. We consider the convex hull $conv(\boldsymbol{V}_x)$ as the local neighborhood of x. 2) If any neuron hyperplane intersects the neighborhood $conv(\boldsymbol{V}_x)$ then the pre-activation sign will be different for the different vertices. We can therefore count the number neurons for a given layer, which results in sign changes in the pre-activation of $\boldsymbol{V}_x$ to quantify local complexity x for that layer. 3) By embedding $\boldsymbol{V}_x$ to the input of the next layer, we can obtain a coarse approximation of the local neighborhood of x and continue computing local complexity in a layerwise fashion.

subject to the deformation induced by each layer to $conv(V)$. To measure deformation by layers 1 to $\ell - 1$, we consider the undirected graph formed by the vertices V^ℓ and compute the average eccentricity and diameter of the graphs [24]. Eccentricity for any vertex v of a graph, is denoted by the maximum shortest path distance between v and all the connected vertices in the graph. The diameter is the maximum eccentricity over vertices of a graph. Recall from Appendix A that $conv(V)$ where $V = \{x \pm r v_p : p = 1...P\}$ for an input space point x, is a cross-polytope of dimensionality P, where only two vertices are sampled from any of the orthogonal directions v_p. Therefore, all vertices share edges with each other except for pairs $\{(x + r v_p, x - r v_p) : p = 1...P\}$. Given such connectivity, we compute the average eccentricity and diameter of neighborhoods $conv(V^\ell)$ around 1000 training points from CIFAR10 for a trained CNN (Fig. 11). We see that for larger r both of the deformation metrics exponentially increase, where as for $r \leq 0.014$ the deformation is lower and more stable. This shows that for lower r our LC approximation for deeper CNN networks would be better since the neighborhood does not get deformed significantly.

Appendix B. Experimental Setup

Sensitivity of approximation to P and r One of the possible limitations of local complexity measure is the deformation of the local neighborhood when its passed through a network from layer to layer, as shown in Figure 7. For different radius r of the input space neighborhood $\boldsymbol{V}_x$ centered on any arbitrary data point x, we compute the change of graph eccentricity [24] by different layers of a CNN to measure the degree of deformation by each layer. We present the results in Figure 8 for 1000 different training data points for a CNN trained on CIFAR10. The higher the deformation, the less reliable the approximation. Here, layer index 0 corresponds to the input space. We see that below a certain radius value, deformation by the CNN is limited and does not exponentially increase. In subsequent experiments, e.g., Figure 18, we have also observed that the dynamics of local complexity

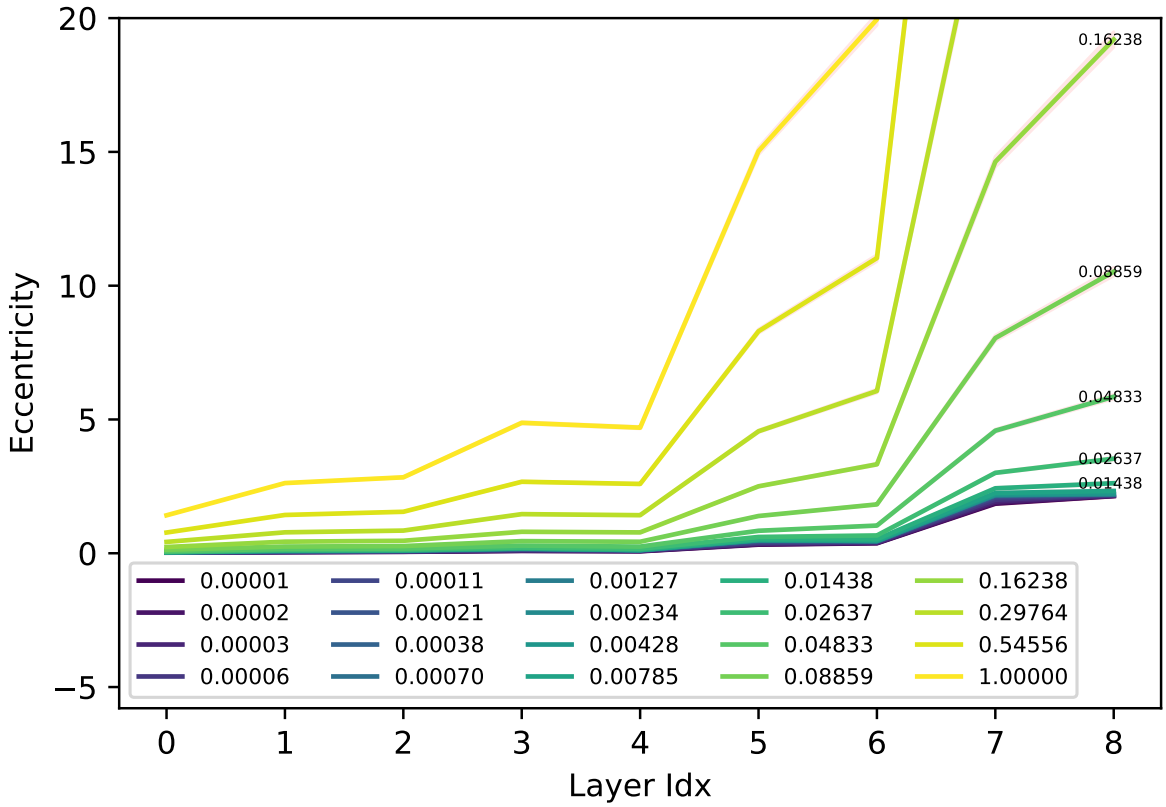

Figure 8: **Deformation with depth.** Change of average eccentricity [24] of the input space neighborhoods $\boldsymbol{V}_x$ by different layers of a CNN trained on the CIFAR10 dataset, for different radius r. We see that, for larger radius, the deformation increases with depth almost exponentially. For $r \leq 0.014$ deformation is low, indicating that smaller radius neighborhoods are reliable for LC computation on deeper networks. Values are averaged over neighborhoods sampled for 1000 training points from CIFAR10. For ResNet18 see Figure 13.

is similar between large and small r neighborhoods. We present more validation experiments in Appendix A.

Experimental Setup For all experiments we sample 1024 train test and random points for local complexity (LC) computation, except for the MNIST experiments, where we use 1000 training points (all of the training set where applicable) and 10000 test and random points for LC computation. We use $r = 0.005$ and $P = 25$ unless specified otherwise and except for the ResNet18 experiments with Imagenette where we use $r = 10^{-4}$. For training, we use the Adam optimizer and a weight decay of 0 for all the experiments except for the MNIST-MLP experiments where we use a weight decay of 0.01. Unless specified, we use CNNs with 5 convolutional layers and two linear layers. For the ResNet18 experiments with CIFAR10, we use a pre-activation architecture with width 16. For the Imagenette experiments, we use the standard torchvision Resnet architecture. For all settings we do not use Batch Normalizaiton, as reasoned in Appendix D. In all our plots, we denote training accuracy/LC using green, test accuracy/LC using orange and random LC using blue colors. We also

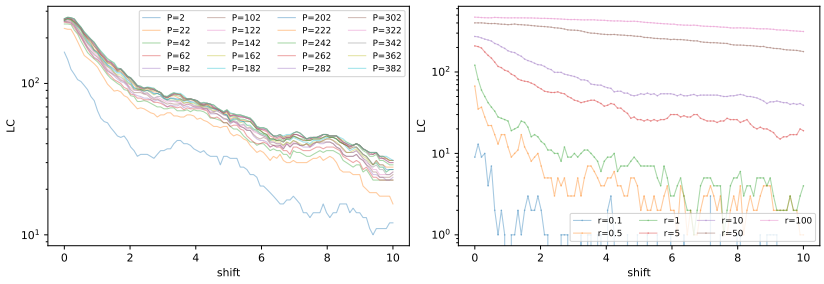

Figure 9: LC for a P dimensional neighborhood with radius r while being shifted from the origin $[0]^d$ to vector $[10]^d$. In **left**, we vary P with fixed $r = 5$ while on **right** we vary r for fixed $P = 20$. We see that for all the settings, shifting away from the origin reduces LC. The increase of LC with the neighborhood dimensionality P gets saturated as we increase P, showing that lower dimensional neighborhoods can be good enough for approximating LC. Increasing r on the other hand, increases LC and reduces LC variations between shifts, since the neighborhood becomes larger and LC becomes less local.

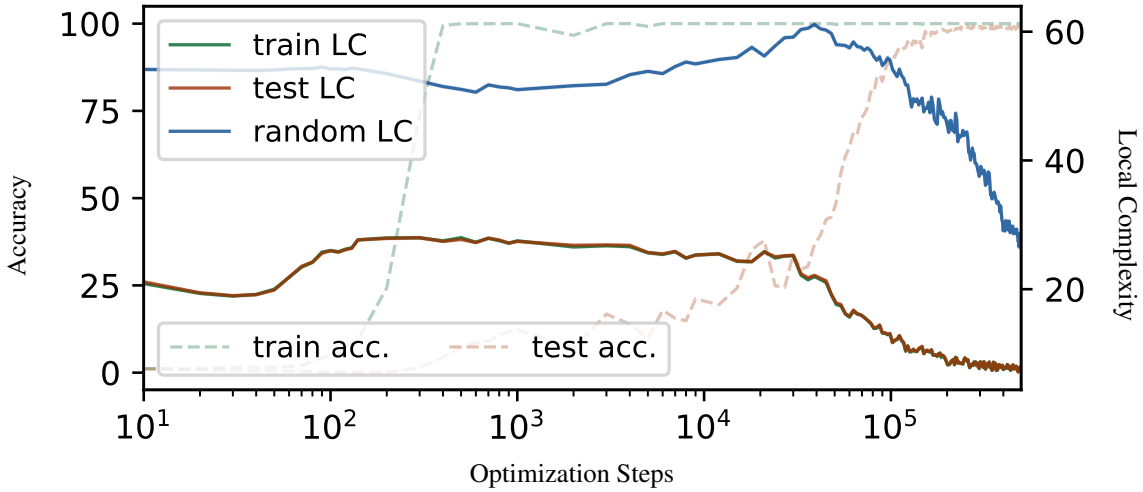

Figure 10: **Region migration in modular addition.** By measuring the local complexity for the GeLU activated fully connected layers of a Transformer architecture, we see that here as well, region migration occurs during grokking.

color curves for adversarial examples using different shades of orange. All local complexity plots show the 99% confidence interval.

Appendix C. Impact of Batch-Normalization and Relation to Circuits

Local complexity as progress measure While we don't quite understand why the network goes from accumulation to repelling of non-linearities around the training data between the ascent and second descent phases, we see that the second descent always precedes the onset of delayed generalization or delayed robustness. In Figure 3-middle and right, we present splinecam visualizations for a network during grokking. The colors denote the norm of the slope parameter $\boldsymbol{A}_\omega$ for each

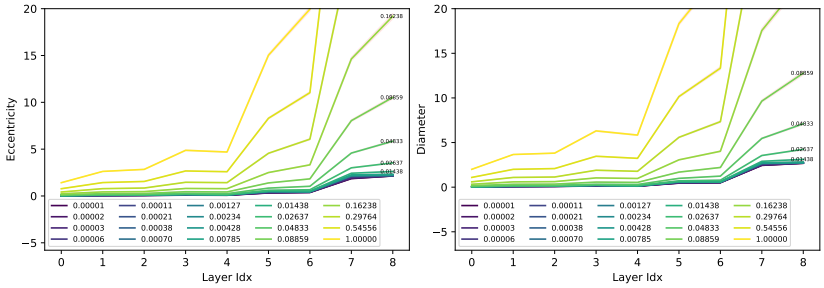

Figure 11: Change of avg. eccentricity and diameter [24] of the input space neighborhood by different layers of a CNN trained on the CIFAR10 dataset. For different sampling radius r of the sampled input space neighborhood V, the change of eccentricity and diameter denotes how much deformation the neighborhood undergoes between layers. Here, layer 0 corresponds to the input space neighborhood. Numbers are averaged over neighborhoods sampled for 1000 training points from CIFAR10. For larger radius the deformation increases with depth exponentially. For $r \leq 0.014$ deformation is lower, indicating that smaller radius neighborhoods are reliable for LC computation on deeper networks. Confidence interval shown in red, is almost imperceptible.

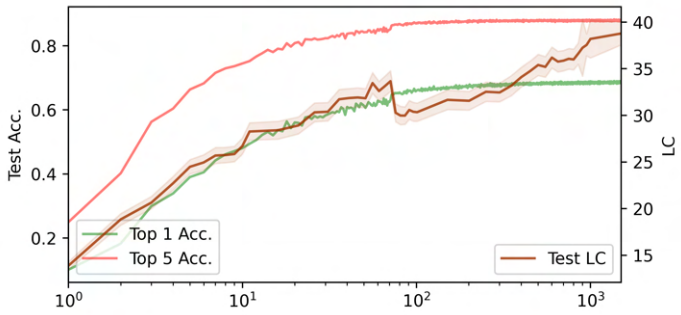

Figure 12: Training a ResNet18 with batchnorm on Imagenet Full. LC is computed only on test points using 1000 test set samples. Computing LC 1000 samples takes approx. 28s on an RTX 8000.

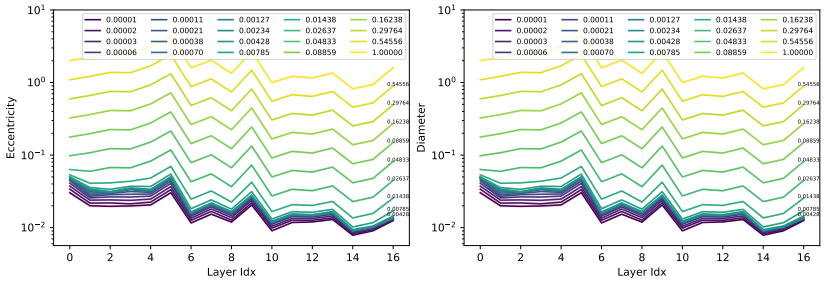

Figure 13: Change of avg. eccentricity and diameter [24] of the input space neighborhood by different layers of a ResNet18 trained on the CIFAR10 dataset, similar to the setting of Fig. 11. Resnet deforms the input neighborhood by reducing the avg. eccentricity and diameter of the neighborhood graphs. For $r \leq 0.014$ deformation is lower, indicating that smaller radius neighborhoods are reliable for LC computation on deeper networks.

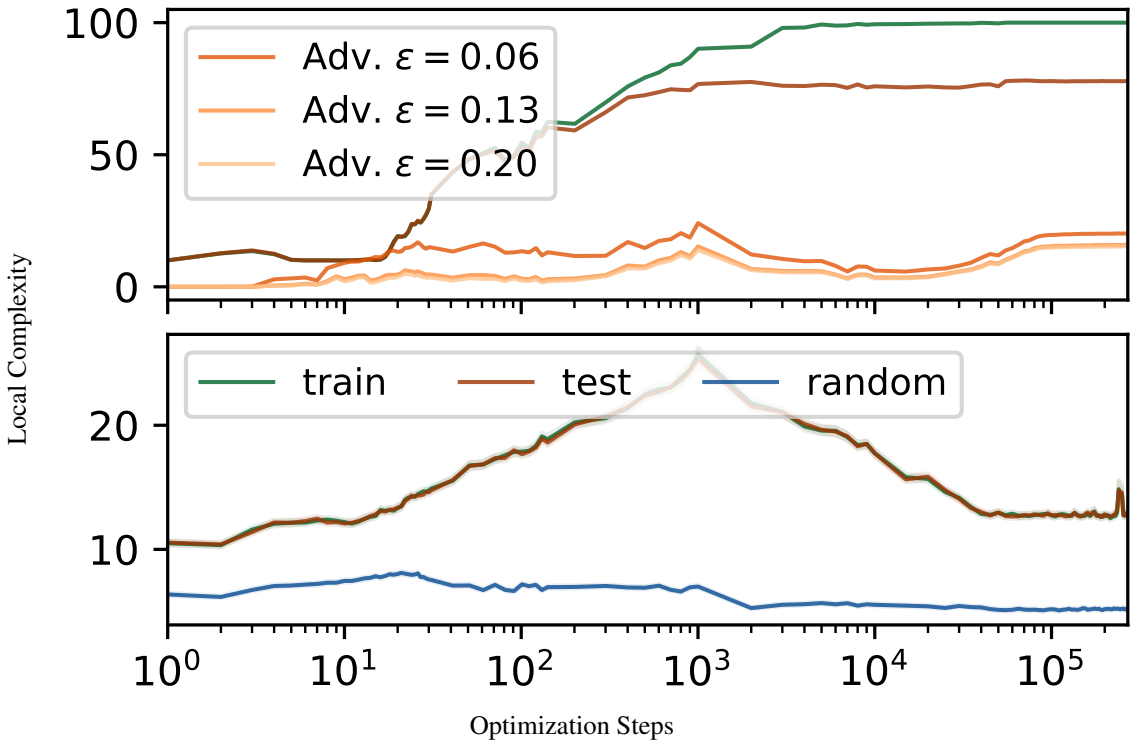

Figure 14: **Batch-norm removes grokking.** Training a CNN with an identical setting as in Figure 2-left, except the CNN now has Batch Normalization layers after every convolution. With the presence of batchnorm, the LC values increase, the initial descent gets removed and most importantly, grokking does not occur for adversarial samples.

region ω computed obtained via SplineCam. We see that while a network groks, the regions start concentrating around the decision boundary where the network has the highest norm. This is intuitive because in such classification settings, an increase of local complexity around the decision boundary allows the function to sharply transition from one class to another. Therefore, therefore the more the non-linearites converge towards the decision boundary, the higher the function norm can be while smoothly transitioning as well. We have provided an animation showing the evolution of partition geometry and emergence of the robust partition during training here[1]. In the animation, we can see that the partition periodically switches between robust configurations during region migration. As time progresses we see increasing accumulation of the non-linearities around the decision boundary. These results undoubtedly show that the local non-linearity or local complexity dynamics is directly tied to the partition geometry and emergence of delayed generalization/robustness.

Relationship with Circuits A common theme in mechanistic interpretability, especially when it comes to explaining the grokking phenomenon, is the idea of 'circuit' formation during training [16, 18, 23]. A circuit is loosely defined as a subgraph of a deep neural network containing neurons (or linear combination of neurons) as nodes, and weights of the network as edges. Recall that Equation (2) expresses the operation of the network in a region-wise fashion, i.e., for all input vectors

1. https://vimeo.com/907898050

$\{x : x \in \omega\}$, the network performs the same affine operation using parameters $(\boldsymbol{A}_\omega, \boldsymbol{b}_\omega)$ while mapping x to the output. The affine parameters for any given region, are a function of the active neurons in the network as was shown by Humayun et al. [9] (Lemma 1). Therefore for each region, we necessarily have a circuit or subgraph of the network performing the linear operation. Between two neighboring regions, only one node of the circuit changes. From this perspective, our local complexity measure can be interpreted as a way to measure the density of unique circuits formed in a locality of the input space as well. While in practice this would result in an exponential number of circuits, the emergence of a robust partition show that towards the end of training, the number of unique circuits get drastically reduced. This is especially true for sub-circuits corresponding to deeper layers only. In Figure 15, we show the robust partition in a layerwise fashion. We can see that for deeper layers, there exists large regions, i.e., embedding regions with only one circuit operation through the layer. This result, matches with the intuition provided by Nanda et al. [16] on the cleanup phase of circuit formation late in training.

Appendix D. Understanding Batch Normalization and its effect on the partition

Suppose the usual layer mapping is

$$\boldsymbol{z}_{\ell+1} = \boldsymbol{a}\left(\boldsymbol{W}_\ell \boldsymbol{z}_\ell + \boldsymbol{c}_\ell\right), \quad \ell = 0, \ldots, L-1 \tag{3}$$

While a host of different DNN architectures have been developed over the past several years, modern, high-performing DNNs nearly universally employ *batch normalization* (BN) [11] to center and normalize the entries of the feature maps using four additional parameters $\mu_\ell, \sigma_\ell, \beta_\ell, \gamma_\ell$. Define $z_{\ell,k}$ as k^{th} entry of feature map $\boldsymbol{z}_\ell$ of length D_ℓ, $\boldsymbol{w}_{\ell,k}$ as the k^{th} row of the weight matrix $\boldsymbol{W}_\ell$, and $\mu_{\ell,k}, \sigma_{\ell,k}, \beta_{\ell,k}, \gamma_{\ell,k}$ as the k^{th} entries of the BN parameter vectors $\mu_\ell, \sigma_\ell, \beta_\ell, \gamma_\ell$, respectively. Then we can write the BN-equipped layer ℓ mapping extending (1) as

$$z_{\ell+1,k} = a\left(\frac{\langle \boldsymbol{w}_{\ell,k}, \boldsymbol{z}_\ell \rangle - \mu_{\ell,k}}{\sigma_{\ell,k}} \gamma_{\ell,k} + \beta_{\ell,k}\right), k = 1, \ldots, D_\ell. \tag{4}$$

The parameters μ_ℓ, σ_ℓ are computed as the element-wise mean and standard deviation of $\boldsymbol{W}_\ell \boldsymbol{z}_\ell$ for each mini-batch during training and for the entire training set during testing. The parameters β_ℓ, γ_ℓ are learned along with $\boldsymbol{W}_\ell$ via SGD.[2] For each mini-batch $\mathbb{B}$ during training, the BN parameters μ_ℓ, σ_ℓ are *calculated directly* as the mean and standard deviation of the current mini-batch feature maps $\mathcal{B}_\ell$

$$\mu_\ell \leftarrow \frac{1}{|\mathbb{B}_\ell|} \sum_{\boldsymbol{z}_\ell \in \mathbb{B}_\ell} \boldsymbol{W}_\ell \boldsymbol{z}_\ell, \qquad \sigma_\ell \leftarrow \sqrt{\frac{1}{|\mathcal{B}_\ell|} \sum_{\boldsymbol{z}_\ell \in \mathbb{B}_\ell} \left(\boldsymbol{W}_\ell \boldsymbol{z}_\ell - \mu_\ell\right)^2}, \tag{5}$$

where the right-hand side square is taken element-wise. After SGD learning is complete, a final fixed "test time" mean $\overline{\mu}_\ell$ and standard deviation $\overline{\sigma}_\ell$ are computed using the above formulae over all of the training data,[3] i.e., with $\mathbb{B}_\ell = \mathbb{X}_\ell$.

2. Note that the DNN bias $\boldsymbol{c}_\ell$ from (1) has been subsumed into μ_ℓ and β_ℓ.
3. or more commonly as an exponential moving average of the training mini-batch values.

The Euclidean distance from a point $\boldsymbol{v}$ in layer ℓ's input space to the layer's k^{th} hyperplane $\mathbb{H}_{\ell,k}$ is easily calculated as

$$d(\boldsymbol{v}, \mathbb{H}_{\ell,k}) = \frac{|\langle \boldsymbol{w}_{\ell,k}, \boldsymbol{v} \rangle - \mu_{\ell,k}|}{\|\boldsymbol{w}_{\ell,k}\|_2} \tag{6}$$

as long as $\|\boldsymbol{w}_{\ell,k}\| > 0$.

Then, the average squared distance between $\mathbb{H}_{\ell,k}$ and a collection of points $\mathbb{V}$ in layer ℓ's input space is given by

$$Ł_k(\mu_{\ell,k}, \mathbb{V}) = \frac{1}{|\mathbb{V}|} \sum_{\boldsymbol{v} \in \mathbb{V}} d\left(\boldsymbol{v}, \mathbb{H}_{\ell,k}\right)^2 = \frac{\sigma^2_{\ell,k}}{\|\boldsymbol{w}_{\ell,k}\|_2^2}, \tag{7}$$

Appendix E. What affects the robust partition? *Reprise*

Depth. In Figure 18 we plot LC during training on MNIST for Fully Connected Deep Networks with depth in $\{2, 3, 4, 5\}$ and width 200. In each plot, we show both LC as well as train-test accuracy. For all the depths, the accuracy on both the train and test sets peak during the first descent phase. During the ascent phase, we see that the train LC has a sharp ascent while the test and random LC do not.

The difference as well as the sharpness of the ascent is reduced when increasing the depth of the network. This is visible for both fine and coarse r scales. For the shallowest network, we can see a second descent in the coarser scale but not in the finer r scale. This indicates that for the shallow network some regions closer to the training samples are retained during later stages of training. One thing to note is that during the ascent and second descent phase, there is a clear distinction between the train and test LC. *This is indicative of membership inference fragility especially during latter phases of training.* It has previously been observed in membership inference literature [21], where early stopping has been used as a regularizer for membership inference. We believe the LC dynamics can shed a new light towards membership inference and the role of network complexity/capacity.

In Figure 14, we plot the local complexity during training for CNNs trained on CIFAR10 with varying depths with and without batch normalization. The CNN architecture comprises of only convolutional layers except for one fully connected layer before output. Therefore when computing LC, we only take into account the convolutional layers in the network. Contrary to the MNIST experiments, we see that in this setting, the train-test LC are almost indistinguishable throughout training. We can see that the network train and test accuracy peaks during the ascent phase and is sustained during the second descent. It can also be noticed that increasing depth increases the max LC during the ascent phase for CNNs which is contrary to what we saw for fully connected networks on MNIST. The increase of density during ascent is all over the data manifold, contrasting to just the training samples for fully connected networks.

In Appendix, we present layerwise visualization of the LC dynamics. We see that shallow layers have sharper peak during ascent phase, with distinct difference between train and test. For deeper layers however, the train vs test LC difference is negligible.

Width. In Figure 5 we present results for a fully connected DNN with depth 3 and width $\{20, 100, 500, 1000, 2000\}$. Networks with smaller width start from a low LC at initialization compared to networks that are

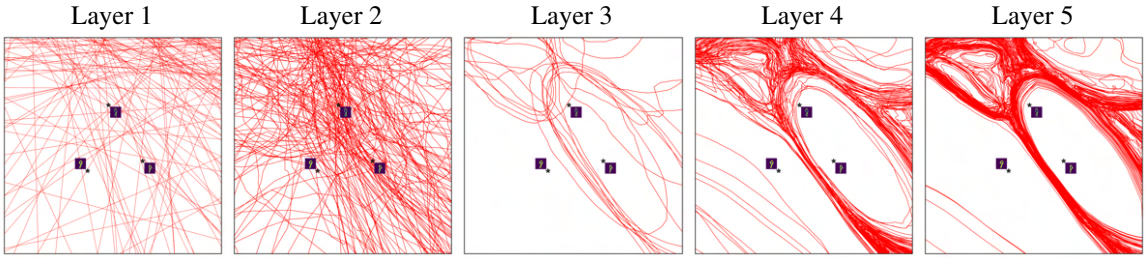

Figure 15: Layerwise visualization of the input space partition for a 2D domain passing through a training set triad, after robust partition formation. The partition is visualized for an MLP with depth 6 and width 200, trained on 1000 samples from MNIST, similar to the setting described in Figure 1. We see that deeper layer neurons partake more in the formation of the robust partition, compared to shallower layers. This is due to the fact that deeper layer neurons can be more localized in the input space due to the non-linearity induced by preceding layers.

wider. Therefore for small width networks the initial descent becomes imperceptible. We see that as we increase width from 20 to 1000 the ascent phase starts earlier as well as reaches a higher maximum LC. However overparameterizing the network by increasing the width further to 2000, reduces the max LC during ascent, therefore reducing the crowding of neurons near training samples. *This is a possible indication of how overparameterization performs implicit regularization [13], by reducing non-linearity or local complexity concentration around training samples.*

Weight Decay regularizes a neural network by reducing the norm of the network weights, therefore reducing the per region slope norm as well. We train a CNN with depth 5 and width 32 and varying weight decay. In Fig. 21 we present the train and random LC for our experiments. We can see that increasing weight decay also delays or removes the second descent in training LC. Moreover, strong weight decay also reduces the duration of ascent phase, as well as reduces the peak LC during ascent. This is dissimilar from BN, which removes the second descent but increases LC overall.

Batch Normalization. It has previously been shown that Batch normalization (BN) regularizes training by dynamically updating the normalization parameters for every mini-batch, therefore increasing the noise in training [6]. In fact, we recall that BN replaces the per-layer mapping from Equation (1) by centering and scaling the layer's pre-activation and adding back the learnable bias $\boldsymbol{b}^{(\ell)}$. The centering and scaling statistics are computed for each mini-batch. After learning is complete, a final fixed "test time" mean $\overline{\mu}^{(\ell)}$ and standard deviation $\overline{\sigma}^{(\ell)}$ are computed using the training data. Of key interest to our observation is a result tying BN to the position in the input space of the partition region from [2]. In particular, it was proved that at each layer ℓ of a DN, BN explicitly adapts the partition so that the partition boundaries are as close to the training data as possible. This is confirmed by our experiments in Fig. 14 we present results for CNN trained on CIFAR10, with and without BN.

Appendix F. Extra Figures

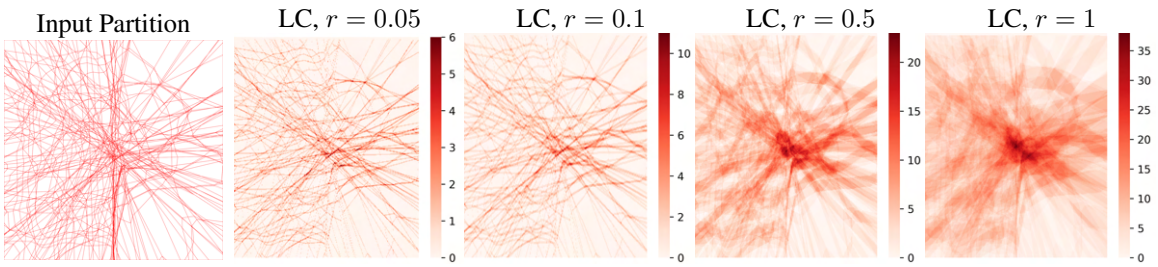

Figure 16: Input space partition computed analytically via SplineCam [10] for the 2D toy setting presented in Figure 6 (left). Regions are colored by white and knots are colored by red. The partition is computed for the input space domain $[-10, 10]^2$, induced by an MLP of depth 5 and width 30. We take a meshgrid of 300×300 points over the input domain, and measure the local complexity at each point with radius, $r \in \{0.05, 0.1, 0.5, 1\}$ (rest). We see that our proposed method can locate the non-linearities for small r. As r is increased our method provides a coarser estimate of the local density of non-linearities, i.e., number of non-linearities intersecting the a fixed volume defined by the local neighborhood.

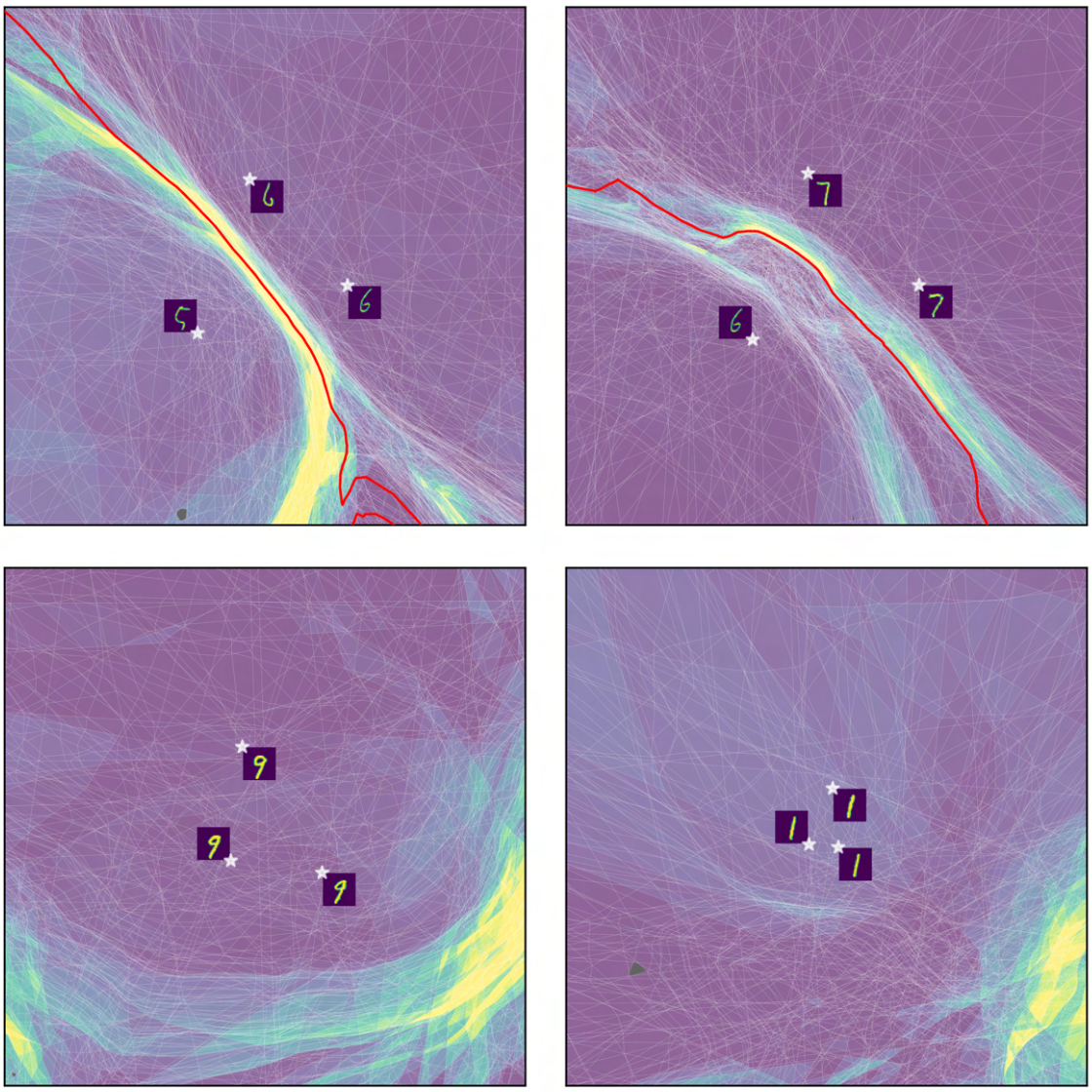

Figure 17: Partition visualization for 2D domains localized around the decision boundary (top) and away from the decision boundary (bottom) for the grokking setup presented in Figure 3. All the plots are show for the optimization step 95381. Number of regions in the partition for top-right, top-left, bottom-right, and bottom-left are 123156, 88362, 33273, and 32018 respectively. The domain used for all of the plots has the same area/volume. Therefore, close to the decision boundary, the region density is much higher compared to away from the decision boundary. This is evidence of region migration happening during the latter phases of training.

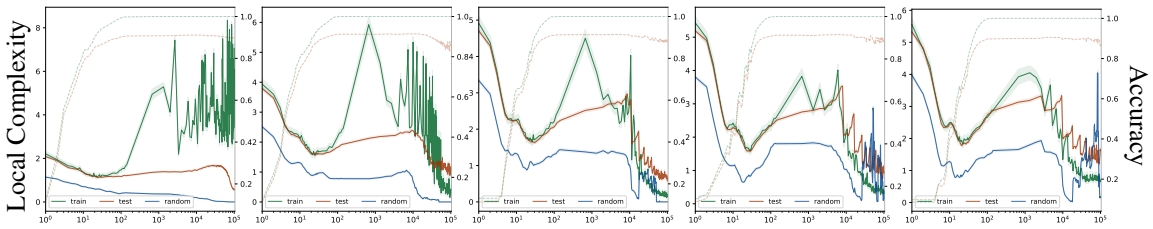

Figure 18: MLP with width 200 and varying depth being trained on 1000 samples from MNIST. Increasing the depth of the network decreases the sharpness of the LC peak during ascend phase. Deeper networks also tend to have a sharper decline in the training LC during region migration.

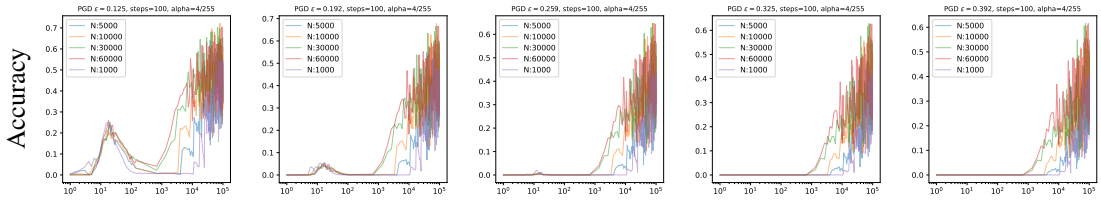

Figure 19: For an MLP with depth 4 and width 200, we train with varying training set sizes and evaluate the adversarial performance after each training iteration. We see that with increasing dataset size, the network groks earlier in time, as can be visible in the adversarial grokking curves for all the different epsilon values.

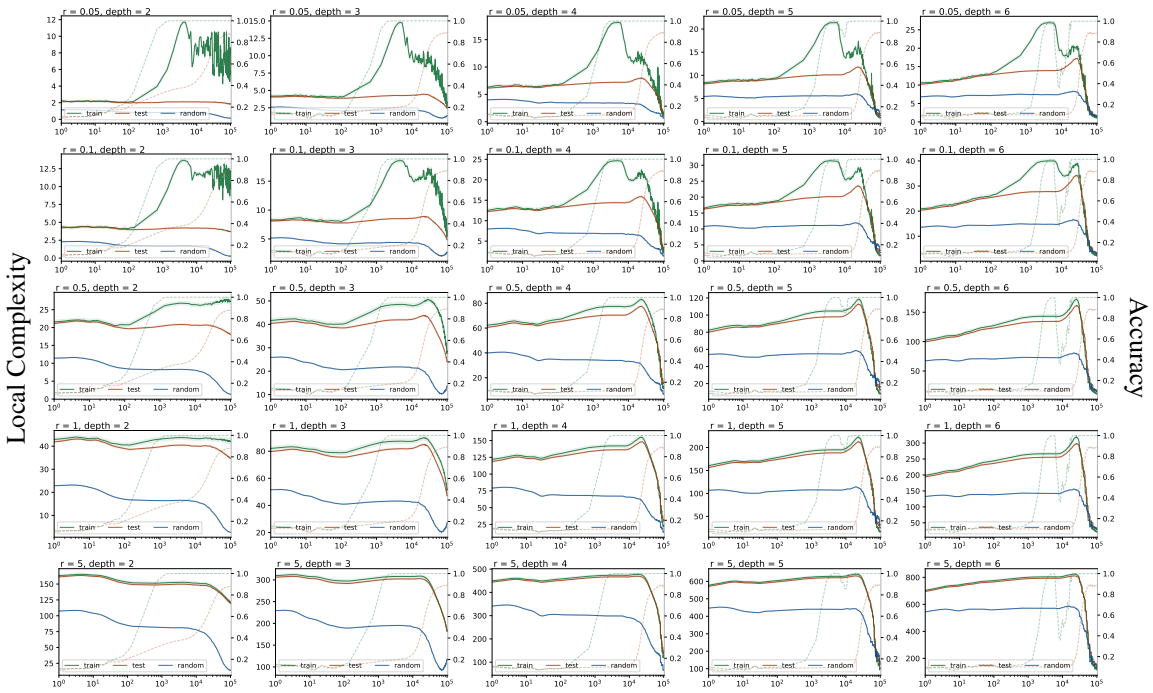

Figure 20: Training a 200 width MLP on MNIST with initialization scaling of 8 and varying depths. Along the row, we consider larger and larger radius neighborhoods for local complexity approximation.

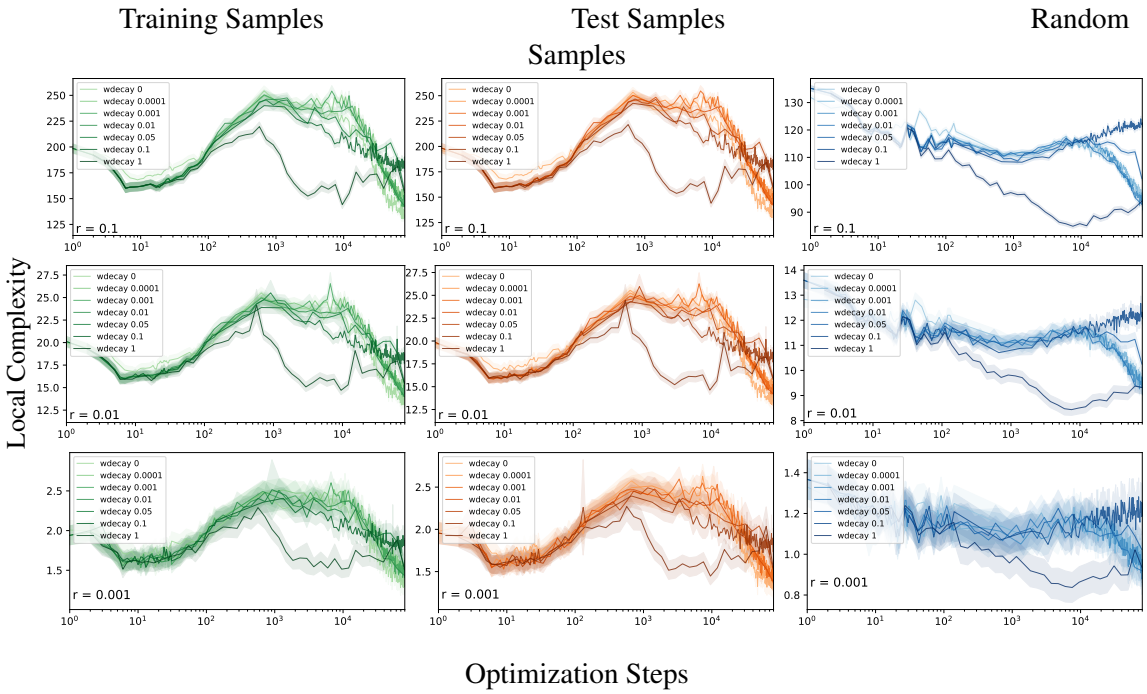

Figure 21: Local complexity dynamics training an MLP on MNIST with weight decay

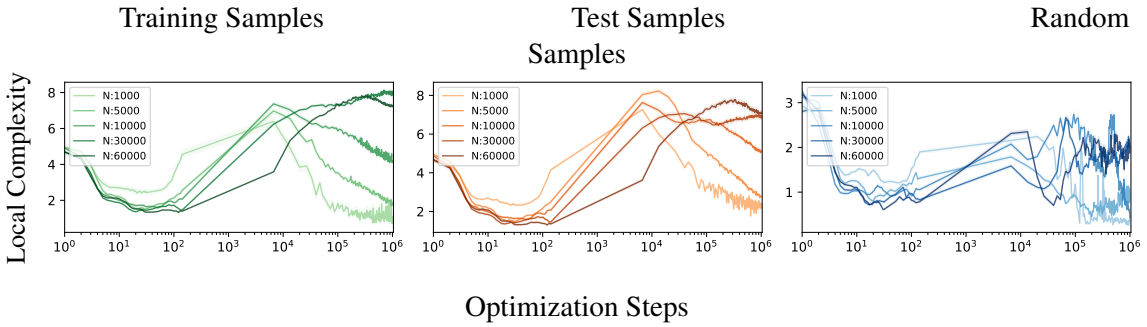

Figure 22: **Increasing the volume of randomly labeled training data.** Continued from Figure 5. Increasing the number of randomly labeled training samples delays the ascent phase of the LC training dynamics for both training and test samples. For random samples the behavior is not affected as much.

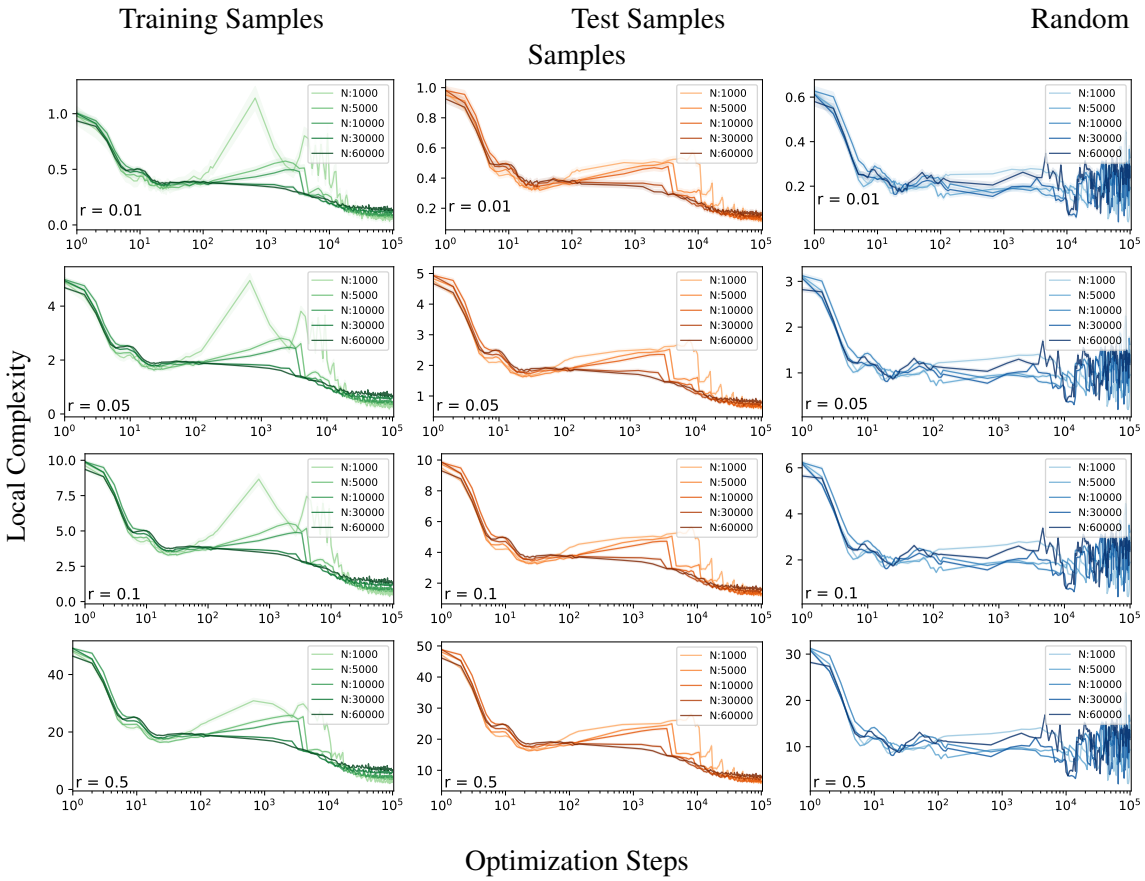

Figure 23: **Dataset size does not affect the onset of region migration.** Local complexity dynamics training an MLP on MNIST with weight decay

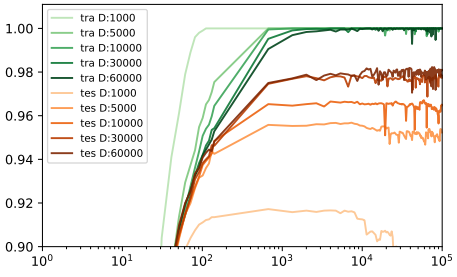

Figure 24: Training and Test accuracy for the different datset sizes presented in Figure 23.

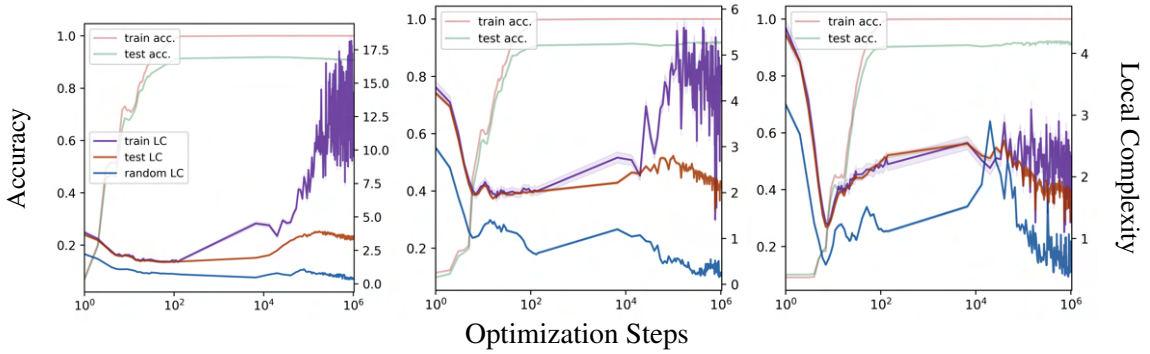

Figure 25: LC dynamics for a GeLU-MLP with width 200 and depth $\{3, 4, 5\}$ presented from left to right. LC is calculated at 1000 training points and 10000 test and random points during training on MNIST.

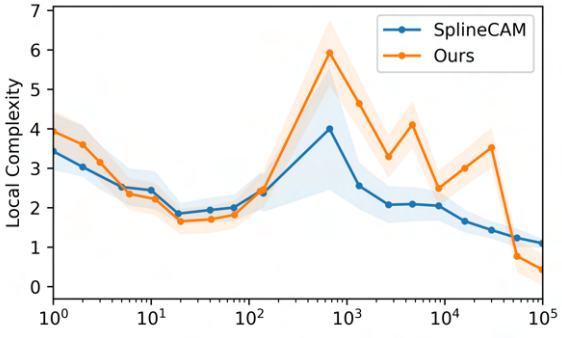

Figure 26: Comparing the local complexity measured in terms of the number of linear regions computed exactly by SplineCAM [9] and number of hyperplane cuts by our proposed method. Both methods exhibit the double descent behavior.

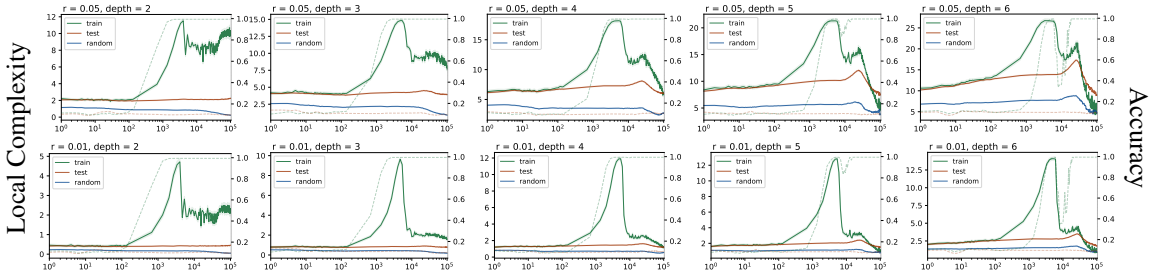

Figure 27: Random label radius and depth Sweep