Investigation into Grokking

许盛昶
2100010811@stu.pku.edu.cn

Abstract

This report focuses on reproducing the experiments in the paper [1]: *Grokking: Generalization Beyond Over-fitting on Small Algorithmic Datasets*. We investigate the grokking phenomenon for different models, data fractions and optimizers and find it commonly exists. Moreover, we find that the grokking of Transformer is closely related to its generalization on different parts of the validation set. There are also some interesting phenomena regarding the frequencies of the output layer. All the codes and supplementary materials are avaliable at: https://github.com/Wshxxsh/ML-Grokking.

1 Introduction

"Grokking" is an interesting phenomenon when a neural network suddenly learns a pattern in the dataset and jumps from random chance generalization to perfect generalization. In this project, we concentrate on the learning of modular addition: $(\overline{x}, \overline{y}) \mapsto \overline{x+y}$ for $\overline{x},\ \overline{y} \in \mathbb{Z}_p$, where p is a prime. This can be treated as a sequence prediction model: the goal is to predict the last token of the sequence $(\overline{x}, +, \overline{y}, =, \overline{x+y})$. Therefore, it is natural to use Transformers or RNN architectures for training.

Our contributions are as follows:

- We use Transformers, MLP and LSTM separately to learn the modular addition and find that the grokking phenomenon is widely existing in different models and optimizers.
- We show that lower data fractions will lead to more obvious grokking phenomenon, and weight decay is a useful regularization to resolve this for most optimizers.
- We explore the learning process of the properties of $\mathbb{Z}_p$ as an abelian group and observe that the grokking of Transformer is closely related to the learning of commutativity.
- We attempt to monitor the frequency changes in the final layer of the neural network model and find there may be a relationship between grokking and the changes in frequency.

2 Problem Settings

Following [1], all of our experiments are based on datasets of equations of the form $\langle\overline{x}\rangle\langle+\rangle\langle\overline{y}\rangle\langle=\rangle\langle\overline{x+y}\rangle$, where $\langle a\rangle$ is the token corresponding to a. Specifically, for $\mathbb{Z}_p$ we have

$$\langle\overline{x}\rangle = x \pmod{p}, \quad \langle+\rangle = p, \quad \langle=\rangle = p+1 \tag{1}$$

Therefore, the dataset for $\mathbb{Z}_p$ takes the form $\mathcal{D} = \{(x, p, y, p+1) \mid 0 \leqslant x, y \leqslant p-1\}$. In experiment, $\mathcal{D}$ is partitioned into two subsets $\mathcal{T}$ and $\mathcal{V}$, with $\mathcal{T}$ designated for training and $\mathcal{V}$ for testing (or validation). We will investigate the impact of the training data fraction α on the performance.

In our project, we explore the grokking phenomenon on three models: Transformer, MLP and LSTM. The details about our models are shown in A.2.

Preprint. Under review.

3 Grokking on Different Models

We discuss the grokking phenomenon on different models with different p: Transformer, MLP and LSTM. We also investigate the impact of the training data fraction on the results. If not specifically pointed out, we use AdamW with $\beta_1 = 0.9, \beta_2 = 0.98$, weight decay $= 0.05$ as our default optimizer. In all experiments, We stop the iteration once the test accuracy attains 99% 10 times.

3.1 Grokking on Transformers

We first test the grokking phenomenon on the Transformer for $p = 97, 149$ and $\alpha = 0.2, 0.3, 0.4, 0.5$. The learning rate η is set 10^{-3} in all cases. The results are shown in Fig 1.

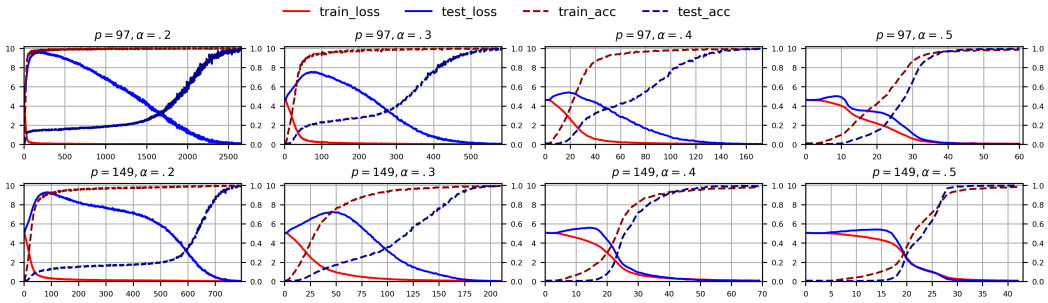

Figure 1: Grokking on Transformers

3.2 Grokking on MLP and LSTM

We test the grokking phenomenon on the MLP and LSTM for $p = 97, 149$ and $\alpha = 0.3, 0.4, 0.5, 0.6$. η is set 10^{-3} in most cases and is set a bit smaller in several tests to get convergent results. In detail, η for (MLP, $p = 97$, $\alpha = 0.3$) is 7×10^{-4} and for (MLP, $p = 149$, $\alpha = 0.6$) is 9×10^{-4}. The results are shown in Fig 2 and 3. We can see all models grok in learning with some difference as follows:

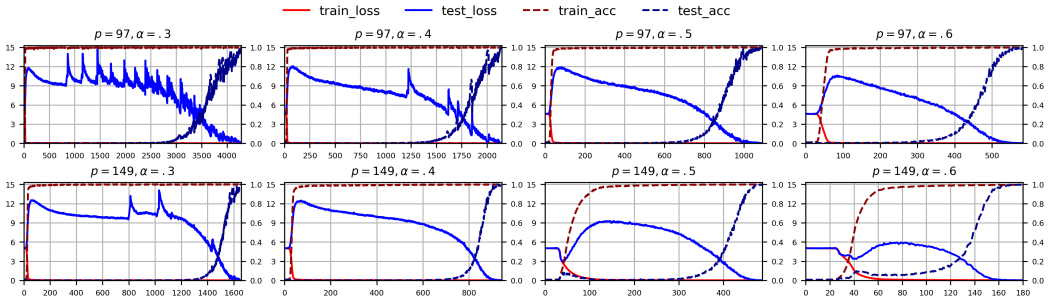

Figure 2: Grokking on MLP

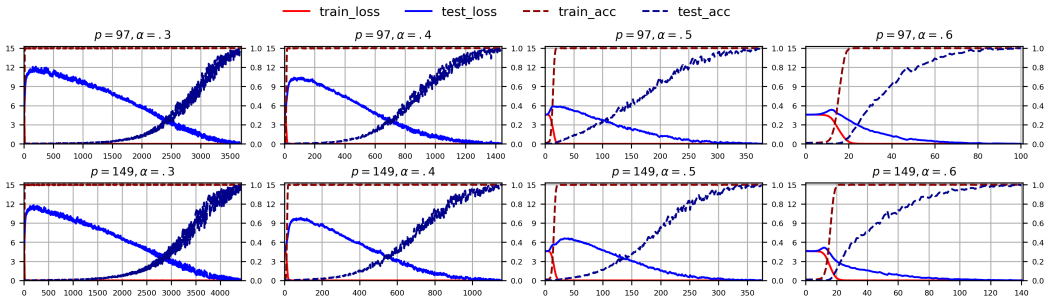

Figure 3: Grokking on LSTM

- Our Transformer and LSTM grok relatively slowly. After grokking, they need many epochs to reach 99% accuracy. But for MLP, it needs much less epochs for the testing accuracy to increase to 99% after grokking. This might be because the tokens are processed sequentially in Transformer and LSTM, while MLP processes them simultaneously.
- Before grokking, the testing accuracy of MLP and LSTM maintains at chance level. In contrast, the accuracy of Transformer remains at a relatively higher level (more than 10%) before it groks. The reason of this phenomenon will be explained in Subsection 6.1.
- In all cases, we see that models with larger p and α grok earlier. This is explicable, because the scale of $\mathcal{T}$ is about αp^2. The larger p or α is, the more data can be used for training. Hence, the model will converge more quickly.

4 Impact of Different Optimizers and Regularization Methods

In this section, we investigate the impact of different optimizers and regularization methods on the grokking phenomenon. We take Transformer as our model with $p = 97$ and $\alpha = 0.3$. For comparative purposes, the parameters are set with a batch size of 60, a drop out of 0.1 and a learning rate of 10^{-3}. And we use AdamW as our benchmark optimizer, with $\beta_1 = 0.9,\ \beta_2 = 0.98$ and 0.05 weight decay. The results are shown in Fig 4. We can preliminarily draw the following conclusion:

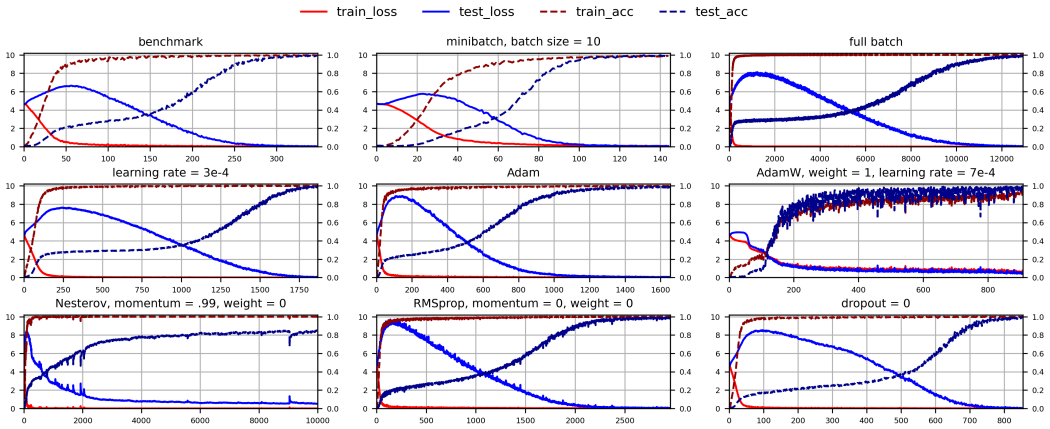

Figure 4: Different Optimizers and Regularization Methods

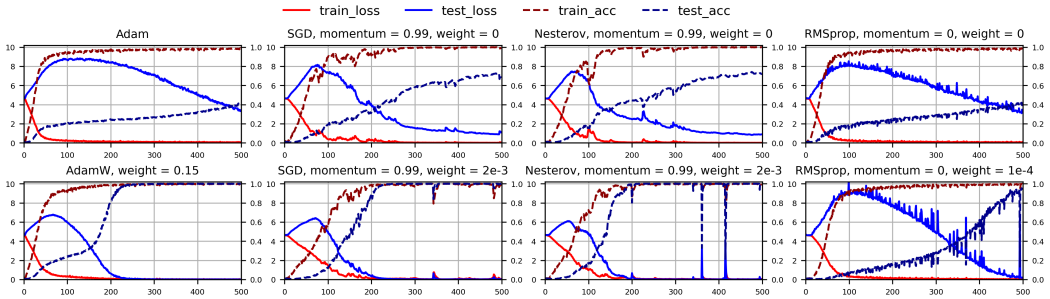

Figure 5: Optimizers without and with Weights

- A smaller batch size leads to faster convergence. This is easy to understand because the steps in an epoch are inversely proportional to the batch size.
- If no drop out is added in the model, it will converge much more slowly. Too small learning rate will cause the same problem.
- We test the grokking phenomenon on AdamW, SGD with Nesterov momentum and RMSprop. These optimizers always achieve convergence in their results over time.

- Moreover, we test the AdamW with different weights, including 0 (AdamW with 0 weight is exactly Adam), 0.05 (benchmark model) and 1 (here the learning rate is set a bit smaller to get faster convergence). The results imply that larger weights help to resolve the grokking. We did a further research into the influence of weights on different optimizers, including Adam, SGD momentum, Nesterov momentum and RMSprop. The total number of epochs is fixed at 500 and the results are plotted in Fig 5. It is obvious that non zero weight decay will bring all optimizers better performance and faster grokking. Although larger weights might induce instability in the training process, it is still an effective method to relieve grokking.

5 Modular K-Summation

5.1 Datasets

In this section we discuss the grokking phenomenon in modular k-summation: $(\overline{x_1}, \overline{x_2}, \ldots, \overline{x_k}) \mapsto \sum_{i=1}^{k} \overline{x_i}$ for $\overline{x_i} \in \mathbb{Z}_p$. So first we formalize the datasets for training and testing. It is worth noting that $+$ is not necessary in Sec 2. So based on the formalization in the case $k = 2$, the dataset takes the form $\mathcal{D}_k = \{(x_1, x_2, \ldots, x_k, =) \mid 0 \leqslant x_i \leqslant p-1\}$, where the token of $=$ is still $p+1$ as before.

5.2 Experiments

We test the modular k-summation for $(k = 3, p = 47)$, $(k = 4, p = 23)$, $(k = 5, p = 11)$. Note that in these cases, the sizes of the datasets are 103,823, 279,841, 161,051, which are of about the same scale. We still use Transformer for training with the same parameters as in Sec 3 with a batch size of 512. The results are displayed in Fig 6. We can see the grokking phenomenon still exists for k-summation, but it arises only when α is sufficient small. In fact, this verifies the hypothesis we proposed in Subsec 3.2 that the grokking phenomenon is related to the scale of $\mathcal{T}$. The size of dataset in this section is about 10 times larger than in Sec 3, hence the α is set about 10 times smaller here.

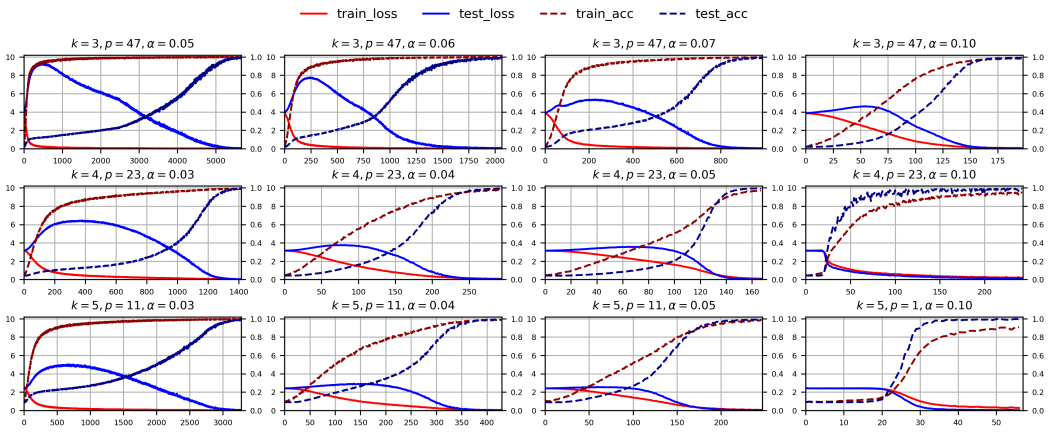

Figure 6: Modular K-Summation

6 Explanation of Grokking

6.1 Learning Process of the Properties of the Abelian Group

Since $(\mathbb{Z}_p, +)$ is an Abelian group, in this subsection we investigate into the learning process of several properties, including commutativity, associativity and the identity element. For commutativity, we test the accuracy of $\overline{x} + \overline{y} = \overline{y} + \overline{x}$ for all $\overline{x}, \overline{y} \in \mathbb{Z}_p$. For identity element, we test the accuracy of $\overline{x} + \overline{0} = \overline{x}$ for all $\overline{x} \in \mathbb{Z}_p$. And for associativity, since it is too expensive to test all triples in $\mathbb{Z}_p^3$, we only randomly choose N_{assoc} triples to test whether the identity $(\overline{x} + \overline{y}) + \overline{z} = \overline{x} + (\overline{y} + \overline{z})$ holds.

We explore the learning process in Transformer and MLP. For the training set $\mathcal{T}$, we define its "complement" as $\mathcal{C}_{\mathcal{T}} = \{(\overline{x}, \overline{y}) \mid (\overline{x}, \overline{y}) \notin \mathcal{T} \text{ and } (\overline{y}, \overline{x}) \in \mathcal{T}\}$. Obviously $\mathcal{C}_{\mathcal{T}} \subset \mathcal{V}$. We test the

accuracy on $\mathcal{C}_\mathcal{T}$ and $\mathcal{V}\backslash\mathcal{C}_\mathcal{T}$ to examine the learning of commutativity. We first test the case $\mathcal{C}_\mathcal{T} = \varnothing$, which means $\mathcal{T}$ is symmetric with respect to $\overline{x}$ and $\overline{y}$. Meanwhile, we design a test to eliminate all tuples containing $\overline{0}$ from $\mathcal{T}$ while maintaining α unchanged. That is, $\mathcal{Z}_0(\mathcal{T} \cup \mathcal{C}_\mathcal{T}) = \varnothing$ where $\mathcal{Z}_q(\mathcal{S}) = \{(\overline{x}, \overline{q}) \in \mathcal{S}\}$. Our purpose is to see how the structure of $\mathcal{T}$ effects the learning. We use (Transformer, $p = 97, \alpha = 0.25$) and (MLP, $p = 97, \alpha = 0.5$) with the same parameters as in Sec 3. N_{assoc} is set 10 000 and results are shown in Fig 7. We observe some interesting phenomena here.

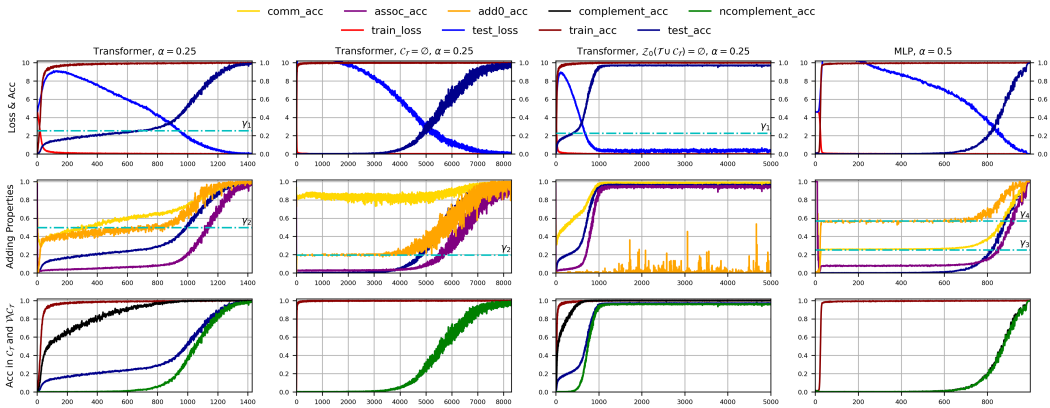

Figure 7: Learning of Adding Properties. **Top.** Loss and accuracy. $\gamma_1 = \#\mathcal{C}_\mathcal{T}/\#\mathcal{V}$. **Mid.** Accuracy of the three properties: commutativity, associativity and the identity element. $\gamma_2 = \#\mathcal{Z}_0(\mathcal{T} \cup \mathcal{C}_\mathcal{T})/\#\mathcal{Z}_0(\mathcal{D})$, $\gamma_3 = \#\mathcal{A}_\mathcal{T}/\#\mathcal{D}$, $\gamma_4 = \#\mathcal{Z}_0(\mathcal{T})/\#\mathcal{Z}_0(\mathcal{D})$. **Bottom.** Accuracy on $\mathcal{T}, \mathcal{V}, \mathcal{C}_\mathcal{T}, \mathcal{V}\backslash\mathcal{C}_\mathcal{T}$

- **Transformer learns the modular addition is Abelian before knowing what the modular addition is**. This sounds like a classic French joke, but it just works. From row 1 in Fig 7 we see the accuracy on $\mathcal{C}_\mathcal{T}$ improves much more quickly that on $\mathcal{V}\backslash\mathcal{C}_\mathcal{T}$. Recall that in Sec 3 we see the testing accuracy of Transformer maintains at a relatively high level before it groks. Now we that this accuracy comes from the $\mathcal{C}_\mathcal{T}$ part, and the level is exactly $\#\mathcal{C}_\mathcal{T}/\#\mathcal{V}$.

  This helps to give an explanation of the grokking phenomenon on Transformer. At first, the model learns to fit $\mathcal{T}$ and generalizes to $\mathcal{C}_\mathcal{T}$ quickly. In this stage, the testing accuracy gradually improves to $\gamma_1 = \#\mathcal{C}_\mathcal{T}/\#\mathcal{V}$. When the accuracy on $\mathcal{C}_\mathcal{T}$ is sufficient high, the model begins to generalize to $\mathcal{V}\backslash\mathcal{C}_\mathcal{T}$. That is when the model groks. The whole learning process can be divided into three parts as follows:

$$\mathcal{T} \to \mathcal{C}_\mathcal{T} \xrightarrow{\text{Grok!}} \mathcal{V}\backslash\mathcal{C}_\mathcal{T} \tag{2}$$

  In fact, this phenomenon in row 1 of Fig 7 is not accidental, and two learning steps (2) widely exists in experiments. We repeat several experiments in A.3 to demonstrate this.

- **The learning process depends on $\mathcal{T}$**. We first have a look at the case $\mathcal{C}_\mathcal{T} = \varnothing$ (row 2 in Fig 7). From the results we see the testing accuracy is zero before grokking. This is consistent with (2), since $\gamma_1 = \#\mathcal{C}_\mathcal{T}/\#\mathcal{V} = 0$. Besides, it takes more epochs for the model to converge, because $\mathcal{T}$ is closed under symmetry and it is more difficult to generalize from $\mathcal{T}$ to $\mathcal{V}$. However, despite the low testing accuracy, the accuracy of commutativity is extremely high in the whole process. This means although the model does not know what $\overline{x} + \overline{y}$ is, it learns quickly that $\overline{x} + \overline{y} = \overline{y} + \overline{x}$.

  For a more precise observation, we focus on the model's prediction on each sample in $\mathcal{D}$. We use a colored $p \times p$ grid to represents $\mathbb{Z}_p^2$. The samples in $\mathcal{T}$ and $\mathcal{C}_\mathcal{T}$ are represented by ■ and ■ separately. The correctly predicted samples are represented by ■. We say a sample $(\overline{x}, \overline{y})$ is "correct in commutativity" if the model $\mathcal{M}$ predicts that $\mathcal{M}(\overline{x}, \overline{y}) = \mathcal{M}(\overline{y}, \overline{x})$, which is represented by ■. We test $\mathcal{C}_\mathcal{T} \neq \varnothing$ and $\mathcal{C}_\mathcal{T} = \varnothing$ for $\alpha = 0.25$ and plot the results in Fig 8.

  For $\mathcal{C}_\mathcal{T} \neq \varnothing$, the model fits $\mathcal{T}$ at about 100 epochs and generalizes to $\mathcal{C}_\mathcal{T}$ at about 600 epochs (see A.4 for details). This can be verified by Fig 8. At 100 epochs, the model performs well only on $\mathcal{T}$. At 600 epochs, almost all samples in $\mathcal{C}_\mathcal{T}$ are predicted correctly. This again confirms that the learning is progressive and step-by-step. And when $\mathcal{C}_\mathcal{T} = \varnothing$, it only needs 100 epochs to obtain a high accuracy in commutativity, but the model generalizes to $\mathcal{V}$ only after thousands of epochs. To get faster convergence, $\mathcal{T}$ should not be very "symmetric".

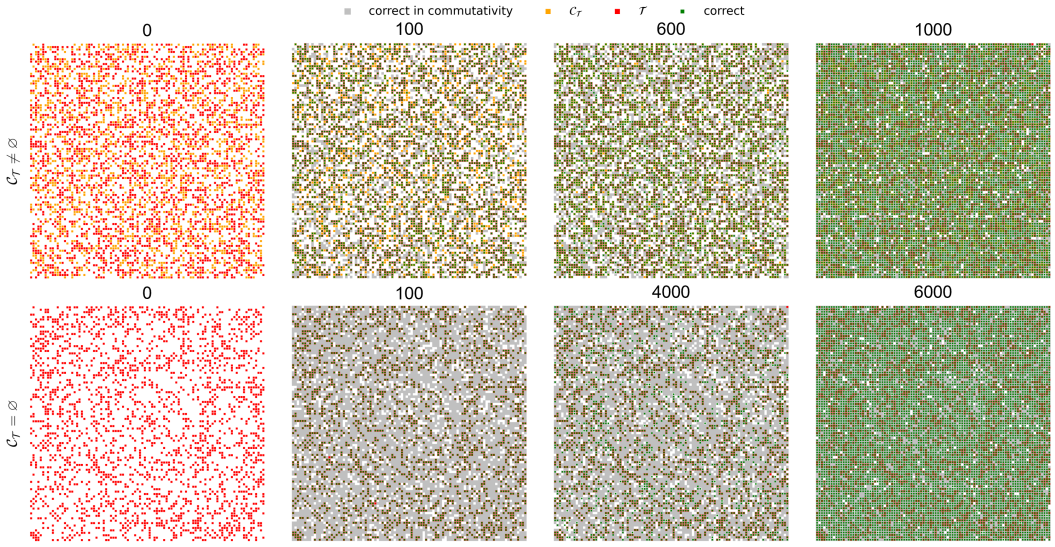

Figure 8: Prediction on $\mathcal{D}$ when $\mathcal{C}_\mathcal{T} \neq \varnothing$ and $\mathcal{C}_\mathcal{T} = \varnothing$

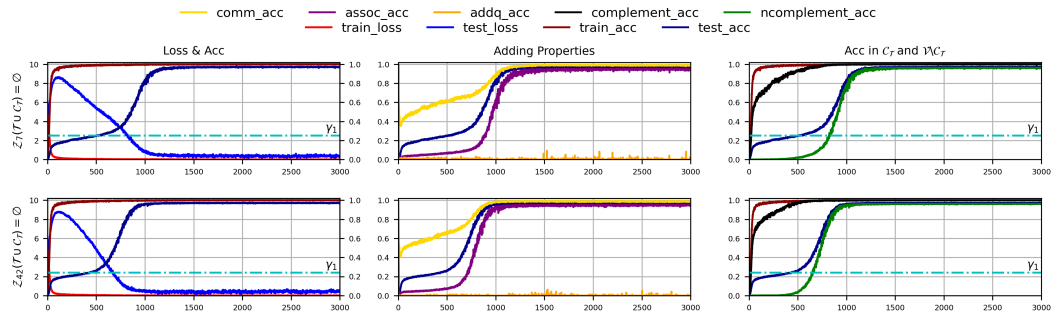

Figure 9: $\mathcal{Z}_7(\mathcal{T} \cup \mathcal{C}_\mathcal{T}) = \varnothing$ and $\mathcal{Z}_{42}(\mathcal{T} \cup \mathcal{C}_\mathcal{T}) = \varnothing$

As of $\overline{0}$ addition, we can see that for general $\mathcal{T}$ (row 1, 2 in Fig 7), the accuracy of $\overline{0}$ addition increases to $\gamma_2 = \#\mathcal{Z}_0(\mathcal{T} \cup \mathcal{C}_\mathcal{T})/\#\mathcal{Z}_0(\mathcal{D})$ in the first stage. This is consistent with (2). However, as we see in the case $\mathcal{Z}_0(\mathcal{T} \cup \mathcal{C}_\mathcal{T}) = \varnothing$, the accuracy of $\overline{0}$ addition remains zero in the whole experiment. Since $\mathcal{Z}_0(\mathcal{T} \cup \mathcal{C}_\mathcal{T}) = \varnothing$, it is reasonable the accuracy is zero at first. But it is interesting that the model never learns what $\overline{0}$ is or how to calculate $\overline{x} + \overline{0}$. In fact, this is also true for $\overline{q}$ other than $\overline{0}$. We test the case $\mathcal{Z}_7(\mathcal{T} \cup \mathcal{C}_\mathcal{T}) = \varnothing$ and $\mathcal{Z}_{42}(\mathcal{T} \cup \mathcal{C}_\mathcal{T}) = \varnothing$ on (Transformer, $p = 97, \alpha = 0.25$) as in Fig 9. Again the model has no idea what $\overline{q}$ is and the accuracy of $\overline{q}$ addition maintains at chance level.

To some extent, these experiments reveal that a proper $\mathcal{T}$ is necessary for the model to generalize fast and perform well. At least all elements in $\mathbb{Z}_p$ must appear once in $\mathcal{T}$.

- **MLP groks differently from Transformer**. In contrast to the Transformer, we can see that the accuracy of $\mathcal{C}_\mathcal{T}$ and $\mathcal{V}\backslash\mathcal{C}_\mathcal{T}$ changes simultaneously for MLP. If we let $\mathcal{A}_\mathcal{T} = \{(\overline{x}, \overline{y}) \mid (\overline{x}, \overline{y}) \in \mathcal{T} \text{ and } (\overline{y}, \overline{x}) \in \mathcal{T}\}$, we can see that the accuracy for commutativity keeps $\gamma_3 = \#\mathcal{A}_\mathcal{T}/\#\mathcal{D}$ and the accuracy for $\overline{0}$ addition maintains at $\gamma_4 = \#\mathcal{Z}_0(\mathcal{T})/\#\mathcal{Z}_0(\mathcal{D})$, which means that MLP cannot generalize any of the properties to the testing dataset before grokking. This partly explains why the testing accuracy of MLP remains at chance level and verifies the advantages of Transformer over MLP.

6.2 Frequencies of the Last Layer

From the Fig 3 in [1], we see that the t-SNE projection of output layer weights is regularly arranged in $\mathbb{R}^2$. We have a detailed exploration into this phenomenon. For all models in Sec 2, the output layer is linear with weights $\boldsymbol{W} \in \mathbb{R}^{p \times d}$. Each row of $\boldsymbol{W}$ can be viewed as a representation of $\overline{x}$ in

latent space. We denote the representation for $\bar{i}$ as $\boldsymbol{w}_i$. By t-SNE, $\boldsymbol{w}_i$ is transformed into $\widehat{\boldsymbol{w}}_i \in \mathbb{R}^2$. Consider $\widehat{\Delta}_i = \widehat{\boldsymbol{w}}_i - \widehat{\boldsymbol{w}}_{i-1}$, [1] shows that approximately we have $\widehat{\Delta}_i = re^{\frac{2ik\pi}{p}}$ for some k and r. Hence, we consider to do DCT on the x component of $\widehat{\Delta}_i$ and monitor how the frequencies change in the training process. It is hoped that all frequencies approach zero except two $(\frac{2k\pi}{p}, \frac{2(p-k)\pi}{p})$. In fact, we have $\widehat{\lambda}_s = \widehat{\lambda}_{p-s}$ (see A.1) and we only need to focus on the low frequencies for $s \leqslant \lceil \frac{p}{2} \rceil$.

Besides, we try to investigate the "frequency" change of $\boldsymbol{w}_i$ directly. It is not that convenient to do DCT in $\mathbb{R}^p$, so we turn to do DCT on the inner production between $\boldsymbol{w}_i$s. In detail, let $\Delta_i = \boldsymbol{w}_i - \boldsymbol{w}_{i-1}$, $a_{ij} = \langle \Delta_i, \Delta_j \rangle_2$ be the inner production and $l_k = \sum_{\bar{j}-\bar{i}=\bar{k}} a_{ij}/p$ be the average. l_k is induced because the matrix $A = (a_{ij})$ is becoming "circulant" finally (see A.5). Again, we do DCT on l_is to get the frequencies λ_i, and we still only need to focus on the low frequencies.

We use two gauges to measure the similarity between $\boldsymbol{\lambda}$ (or $\widehat{\boldsymbol{\lambda}}$) and the unit vector $\boldsymbol{e}_i$: Signal Noise Ratio (SNR) and Entropy. They take the form

$$\text{SNR}(\boldsymbol{\lambda}) = \frac{\|\boldsymbol{\lambda}\|_1}{\|\boldsymbol{\lambda}\|_2}, \ \text{Entropy}(\boldsymbol{\lambda}) = -\sum_{i=0}^{\lceil p/2 \rceil} \frac{|\lambda_i|}{\|\boldsymbol{\lambda}\|_1} \log \frac{|\lambda_i|}{\|\boldsymbol{\lambda}\|_1} \tag{3}$$

We use (Transformer, $p = 97$, $\alpha = 0.2$) for experiment. The perplexity and iteration of t-SNE are set to $p - 1$ and 500, respectively. Given the inherent randomness in the t-SNE process, we employ a moving average to achieve smoother results. A total of 10,000 epochs are executed. In Fig 10, we separately present the outputs both before convergence and throughout the entire process. From this we conclude that

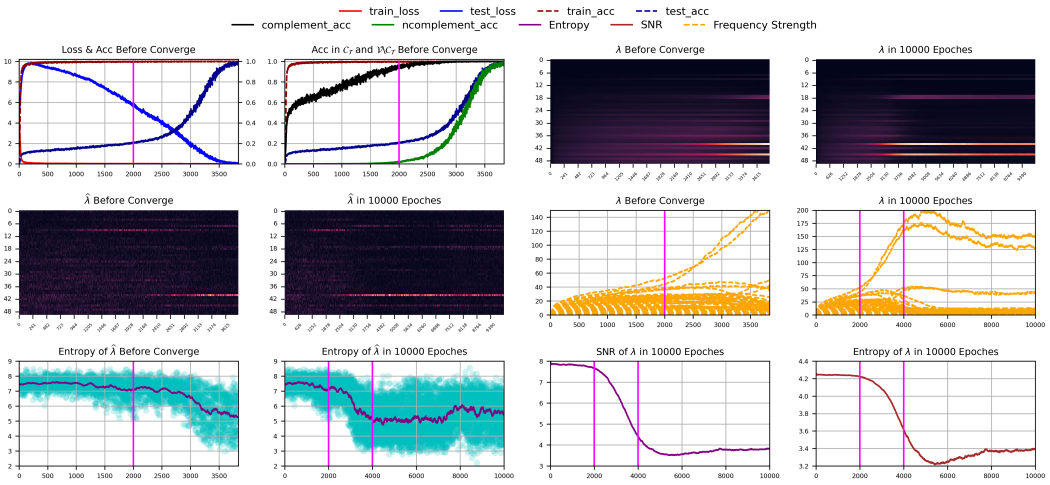

Figure 10: Outputs Both Before Convergence and Throughout the Entire Process

- Before about 2000 epochs, the model is learning to fit $\mathcal{T}$ and generalize to $\mathcal{C}_\mathcal{T}$, which is consistent with (2). In this stage, we see that all λ_i are of about the same level. Both the SNR and Entropy of $\boldsymbol{\lambda}$ and $\widehat{\boldsymbol{\lambda}}$ are almost unchanged. It seems that $\boldsymbol{W}$ is a mixture of all frequencies with no apparent structure.
- During 2000-4000 epochs, the model begins to generalize to $\mathcal{V}\backslash\mathcal{C}_\mathcal{T}$. The heatmap indicates that approximately two frequencies begin to dominate, with all others tending towards zero. Or equivalently, $\boldsymbol{\lambda}$ is evolving into a "two-hot" vector. As a result, the SNR and Entropy of $\boldsymbol{\lambda}$ decrease rapidly in this stage. Moreover, from the heatmap of $\widehat{\boldsymbol{\lambda}}$ we deduce that $\widehat{\boldsymbol{\lambda}}$ is becoming a one-hot vector. This partly explains why the $\widehat{\boldsymbol{w}}_i$ is arranged circularly in $\mathbb{R}^2$.
- After 4000 epochs, the model converges and the accuracy on $\mathcal{V}$ oscillates around 99%. In this stage, we can see that all but four λ_i converge to zero. Among the four frequencies, two has been increasing since grokking, while the other two start to increase as the model approaches convergence. In the end, all frequencies tend to stabilize, resulting in the model exhibiting a regular structure.

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.

A Appendix

A.1 Notation

We first list the notations about the datasets.

- $\mathcal{D}$: The dataset for modular addition on $\mathbb{Z}_p$. $\mathcal{D} = \{(x, p, y, p+1) \mid 0 \leqslant x, y \leqslant p-1\}$. We usually treat $\mathcal{D}$ and $\mathbb{Z}_p^2$ as identical, and view $(x, p, y, p+1) \in \mathcal{D}$ the same as $(\overline{x}, \overline{y}) \in \mathbb{Z}_p^2$.
- $\mathcal{D}_k$: The dataset for modular k-addition on $\mathbb{Z}_p$. $\mathcal{D}_k = \{(x_1, x_2, \ldots, x_k, =) \mid 0 \leqslant x_i \leqslant p-1\}$.
- $\mathcal{T}$: The training set.
- $\mathcal{V}$: The testing set (validation set).
- $\mathcal{C}_\mathcal{T}$: The complement of $\mathcal{T}$ in the sense of symmetry. $\mathcal{C}_\mathcal{T} = \{(\overline{x}, \overline{y}) \mid (\overline{x}, \overline{y}) \notin \mathcal{T} \text{ and } (\overline{y}, \overline{x}) \in \mathcal{T}\}$. That is, if we define the operation $S : \mathcal{D} \to \mathcal{D},\ S((\overline{x}, \overline{y})) = (\overline{y}, \overline{x})$, then $\mathcal{C}_\mathcal{T} = S(\mathcal{T}) \backslash \mathcal{T}$. $\mathcal{C}_\mathcal{T} = \varnothing$ is equivalent to $\mathcal{T} = S(\mathcal{T})$, which means $\mathcal{T}$ is invariant under S. Namely, $\mathcal{T}$ is symmetric with respect to $\overline{x}$ and $\overline{y}$.
- $\mathcal{A}_\mathcal{T}$: The invariant part of $\mathcal{T}$ in the sense of symmetry. $\mathcal{A}_\mathcal{T} = \{(\overline{x}, \overline{y}) \mid (\overline{x}, \overline{y}) \in \mathcal{T} \text{ and } (\overline{y}, \overline{x}) \in \mathcal{T}\}$. This can be expressed as $\mathcal{A}_\mathcal{T} = \mathcal{T} \cap S(\mathcal{T})$.
- $\mathcal{Z}_q$: The set of tuples of the form $(\overline{x}, \overline{q})$. $\mathcal{Z}_q(\mathcal{S}) = \{(\overline{x}, \overline{q}) \in \mathcal{S}\}$. Note that $(\overline{q}, \overline{x})$ is not contained, because when testing the $\overline{q}$ addition we only check whether $\overline{x} + \overline{q} = \overline{x + q}$ instead of $\overline{q} + \overline{x} = \overline{q + x}$.

Then we list the γs we use in Subsection 6.1.

- $\gamma_1 = \#\mathcal{C}_\mathcal{T} / \#\mathcal{V}$. If the testing accuracy maintains at γ_1, it is probably that the model has generalized to $\mathcal{C}_\mathcal{T}$ and is waiting for grokking.
- $\gamma_2 = \#\mathcal{Z}_0(\mathcal{T} \cup \mathcal{C}_\mathcal{T}) / \#\mathcal{Z}_0(\mathcal{D})$. When the model generalized to $\mathcal{C}_\mathcal{T}$, the accuracy of $\overline{0}$ addition will reach γ_2.
- $\gamma_3 = \#\mathcal{A}_\mathcal{T} / \#\mathcal{D}$. If the accuracy of commutativity maintains at γ_3, it means that the model cannot learn anything except the training set $\mathcal{T}$.
- $\gamma_4 = \#\mathcal{Z}_0(\mathcal{T}) / \#\mathcal{Z}_0(\mathcal{D})$. If the accuracy of $\overline{0}$ addition maintains at γ_3, it means that the model cannot learn anything except the training set $\mathcal{T}$.

Then we list the notations we use when discussing the frequencies.

- $\boldsymbol{W}$: The weight of the output layer in our models.
- $\boldsymbol{w}_i$: The ith row of $\boldsymbol{W}$. $\boldsymbol{w}_i$ can be viewed as the representation for $\overline{i}$ in latent space.
- $\widehat{\boldsymbol{w}}_i$: The t-SNE projection of $\boldsymbol{w}_i$ in $\mathbb{R}^2$.
- Δ_i: The difference between two neighboring $\boldsymbol{w}_i$. $\Delta_i = \boldsymbol{w}_i - \boldsymbol{w}_{i-1}$. Specifically we have $\Delta_0 = \boldsymbol{w}_0 - \boldsymbol{w}_{p-1}$.
- $\widehat{\Delta}_i$: The difference between two neighboring $\widehat{\boldsymbol{w}}_i$. $\widehat{\Delta}_i = \widehat{\boldsymbol{w}}_i - \widehat{\boldsymbol{w}}_{i-1}$.
- a_{ij}: The inner product between two Δ_i. $a_{ij} = \langle \Delta_i, \Delta_j \rangle_2$.
- l_k: The average of a_{ij} along the diagonal. $l_k = \sum_{\overline{j-i}=\overline{k}} a_{ij}/p$. If we let A to be the matrix with elements a_{ij}, then l_k is just the average of a_{ij} along the kth diagonal of A.
- λ_i: The ith frequency of $\boldsymbol{l}$. $\lambda_s = \sum_{i=0}^{p-1} l_i \cos i\theta_s$ where $\theta_s = \frac{2s\pi}{p}$. Since $\theta_s + \theta_{p-s} = 2\pi$, we have $\lambda_s = \lambda_{p-s}$.
- $\widehat{\lambda}_i$: The ith frequency of $\boldsymbol{\Delta}$. $\widehat{\lambda}_s = \sum_{i=0}^{p-1} \Delta_i \cos i\theta_s$ where $\theta_s = \frac{2s\pi}{p}$. Since $\theta_s + \theta_{p-s} = 2\pi$, we have $\widehat{\lambda}_s = \widehat{\lambda}_{p-s}$.

A.2 Models

The structure of our models is shown in Fig 11. All models take the general scheme on the left with different processing modules. The modules for Transformers and MLPs are displayed in detail, and for LSTM we directly use the modules implemented in Pytorch. The transformer we use has 2 layers of processing modules with 4 heads, while MLP has 3 layers and LSTM has 2 layers.

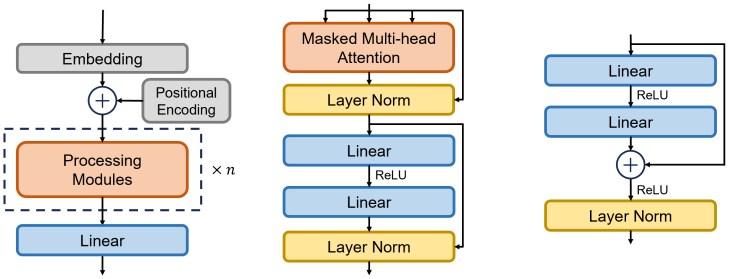

Figure 11: Models for Training. **Left.** General model scheme where the processing modules are to be specified. **Center.** Modules of Transformers. **Right.** Modules of MLPs.

In addition, for Transformer and LSTM, the predicted token is the last token of the output. But since MLP cannot process sequences directly, we concatenate the embedded vectors into one before sent into MLP modules. Then the predicted token of MLP is just the output.

A.3 Replication of Figure 7

We use $(p = 97, \alpha = 0.2, \text{Transformer})$ with the same optimizer and parameters as in Sec 3. We execute four experiments and plot the results in Fig 12. γ_1 in these four tests is $\frac{1478}{7528}, \frac{1470}{7528}, \frac{1491}{7528}, \frac{1511}{7528}$ separately, and γ_2 is $\frac{27}{97}, \frac{29}{97}, \frac{42}{97}, \frac{45}{97}$. We observe the same phenomena as in Subsec 6.1. In the first step of (2), the model generalizes to $\mathcal{C}_\mathcal{T}$. In this phase, the accuracy on $\mathcal{C}_\mathcal{T}$ steadily reaches to nearly 100%, the testing accuracy attains γ_1 and the accuracy of $\overline{0}$ addition attains γ_2. When the accuracy on $\mathcal{C}_\mathcal{T}$ is sufficient high, the model generalizes to $\mathcal{V}\backslash\mathcal{C}_\mathcal{T}$. All accuracies in row 2 of Fig 12 increase simultaneously in this stage.

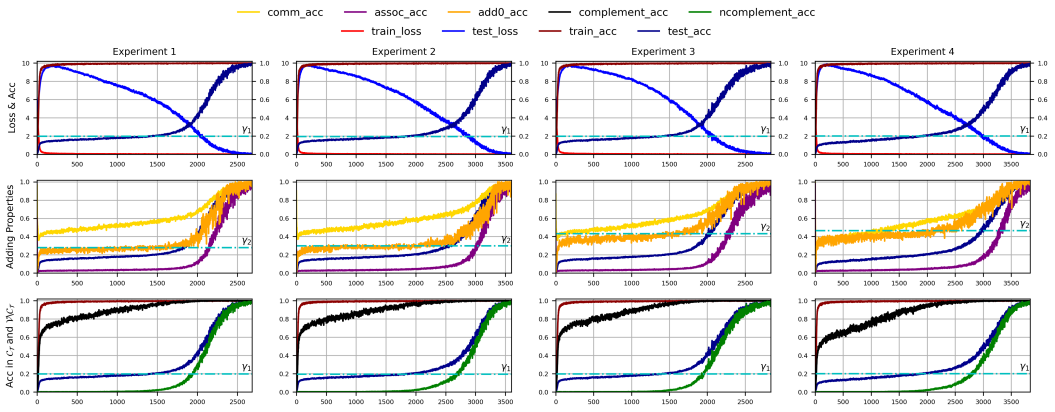

Figure 12: Results of Repeat Tests

A.4 Learning Curves for Figure 8

We display the learning curves corresponding to Fig 8 in Fig 13. γ_1 for $\mathcal{C}_\mathcal{T} \neq \varnothing$ and $\mathcal{C}_\mathcal{T} = \varnothing$ is $\frac{1737}{7057}, 0$ and γ_2 is $\frac{41}{97}, \frac{23}{97}$.

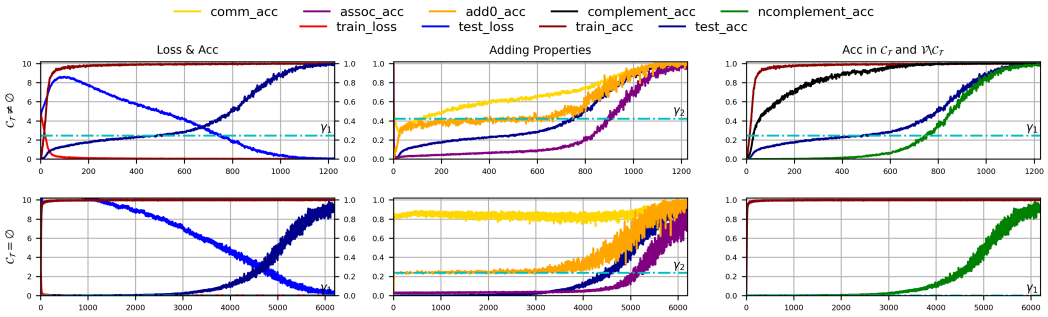

Figure 13: Learning Curves for Figure 8

A.5 Frequency

In Fig 14, we show the change of $A = (a_{ij})$ in an experiment. We use $(\text{Transformer}, p = 97, \alpha = 0.2)$ as our model. It is apparent that A is becoming approximately circulant and the variance of diagonals of A is getting larger as evolving.

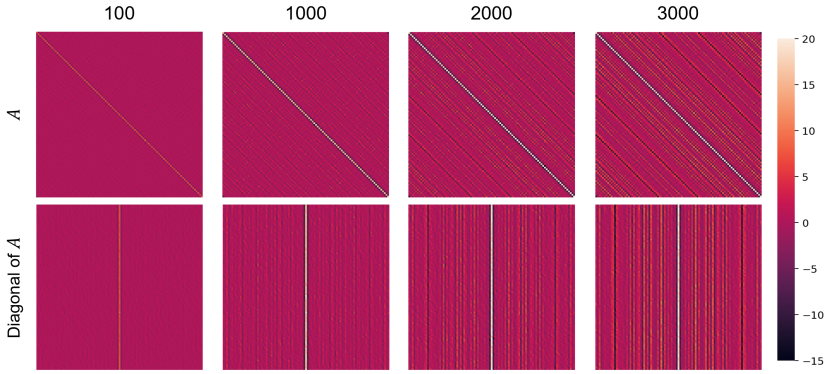

Figure 14: Changes of A