Final Project: Grokking Phenomenon Reproduction

Kecen Sha * 2200010611
Yuziheng Wu * Your ID
Di Yue * 2100012961

Abstract

Grokking, a phenomenon that the generalization of a neural network happens much later than the convergence of its training loss, has received increasing attention from both learning theory and application since it was first introduced by [Power et al., 2022]. In this project, we reproduce the grokking phenomenon for modular addition problem, and provide an explanation based on [Kumar et al., ICLR 2024]. [2]

1 Introduction

When using neural network to train certain problems, generalization may occur long after the model overfits training set. This striking phenomenon is called grokking [1]. In this project, we study the grokking phenomenon on training the modular addition task. Specifically, given a fixed prime number p, we consider equations in the form of modular arithmetic, where each equation consists of K summands and is represented as a string:

$$a_1 + a_2 + \cdots + a_K = b,$$

where both the individual summands a_i and the sum b are elements of the finite field $\mathbb{Z}/p\mathbb{Z}$, comprising integers from 0 to $p-1$.

The input to our model is such an equation string. Each character within the string, including digits and the addition operator, is transformed into a one-hot vector representation. This encoding scheme assigns a unique binary vector to every possible character.

The output of the model is also designed as a one-hot vector, corresponding to the predicted value of b after decoding. During the training phase, the target output is the actual sum b, computed as the sum of K modular numbers modulo p. The objective of the training process is to adjust the model parameters so that it learns to accurately predict the correct modular sum b given any valid input equation string.

We mainly focus on the phenomenon when $K = 2$. In this case, we first generate all p^2 equations and shuffle them randomly. Given the training data fraction $\alpha \in (0, 1)$, we take the first α data as training set and the rest as validation set.

Our experiments are divided into 4 parts in section 3. 3.1 reproduces the grokking phenomenon stated in [1] and study the effect of α on grokking phenomenon. 3.2 uses other models such as LSTM and MLP to conduct the same task. 3.3 investigates the impact of different optimizers(with different hyper-Parameters such as learning rate, weight decay and dropout) to grokking phenomenon. 3.4 explores the grokking phenomenon for different K.

At the end of our report we give some explanations on grokking phenomenon, mainly based on [2].

*Equal contribution.

[2] All the codes and supplementary materials are available at: `https://github.com/tnediserp/MIML-grokking`

Preprint. Under review.

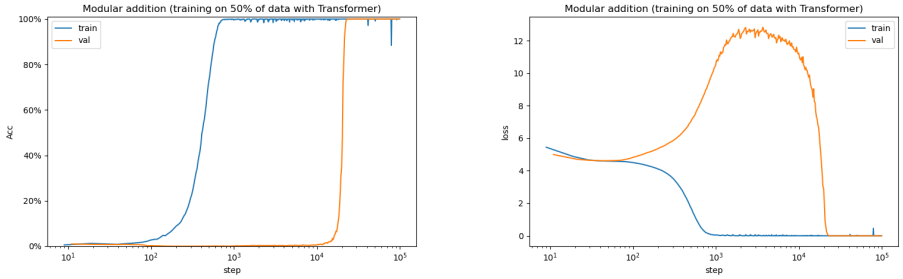
2 Preliminary

3 Experiments

3.1 Reproducing the Grokking Curve

In this task, we investigate the grokking phenomenon in modular addition with the transformer model. Specifically, we use a decoder-only transformer with 2 layers, width $d_{\text{model}} = 128$, 4 attention heads and dropout 0.1. We use the AdamW optimizer with $\beta_1 = 0.9, \beta_2 = 0.98$, learning rate 10^{-3}, weight decay 0.1, and linear learning rate warmup over the first 10 updates. For $p = 97$ and $\alpha = 0.5$, we randomly separate an α fraction from all p^2 equations in $\mathbb{Z}_p$ as the training set; the rest serves as the validation set. The model is trained with minibatch size 512 for 10^5 steps.

The accuracy and loss throughout the training process are plotted in Figure 1.



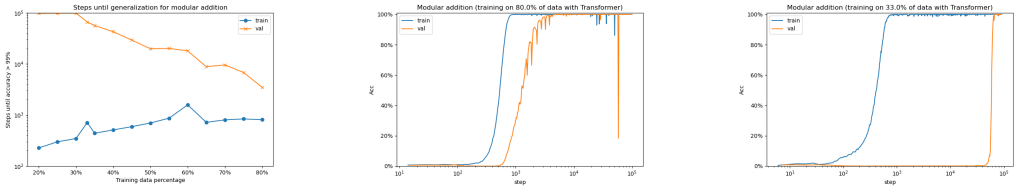
(a) Training and validation accuracy

(b) Training and validation loss

Figure 1: The grokking curves of transformer model

As is shown in Figure 1a, the model overfits the training data within 10^3 updating steps. Nevertheless, generalization does not happen until after 10^4 steps. Figure 1b further illustrates that the decrease of loss is consistent with the increase of accuracy, both for training and validation phases. Interestingly, while the training loss decreases monotonically, an increase of the validation loss is observed before it begins to converge.

We further study the effect of the training data fraction α. For each α we sample, we train the model on a random α fraction of all p^2 equations for at most 10^5 steps, and determine the minimum number of steps required to achieve validation accuracy $\geq 99\%$. The results are plotted in Figure 2a. As a comparison, we plot the accuracy curves for $\alpha = 80\%$ and $\alpha = 33\%$ in Figures 2b and 2c, respectively.



(a) Minimal number of training steps required to achieve validation accuracy $\geq 99\%$

(b) The accuracy curve for $\alpha = 80\%$

(c) The accuracy curve for $\alpha = 33\%$

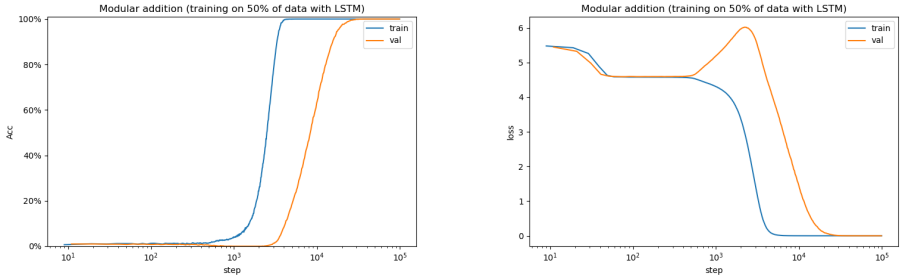
Figure 2: Effect of the training data fraction α

When $\alpha = 80\%$, the model generalizes in 10^4 steps. As α decreases, it becomes easier for the model to overfit the training data, while the number of steps required for generalization increases rapidly. When $\alpha \leq 30\%$, the model would not generalize in 10^5 steps.

3.2 Grokking Phenomenon of Other Models

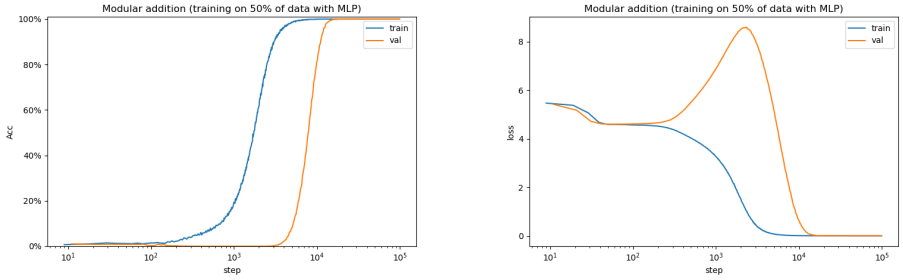
We study the grokking phenomenon for two other network architectures, long short-term memory (LSTM) and multilayer perceptron (MLP), suggesting that grokking is not unique for transformer. Specifically, our LSTM has 2 layers, a hidden state of size $d_{\text{hidden}} = 128$, and dropout 0.1. Our MLP has 2 hidden layers of size $d^{(1)}_{\text{hidden}} = d^{(2)}_{\text{hidden}} = 256$, both of which have dropout 0.1. Other hyperparameters remain the same as Section 3.1.

The grokking curves for LSTM and MLP are shown in Figure 3 and Figure 4, respectively.



(a) Training and validation accuracy (b) Training and validation loss

Figure 3: The grokking curves of LSTM model



(a) Training and validation accuracy (b) Training and validation loss

Figure 4: The grokking curves of MLP model

In Figures 3a and 4a, we again observe a delay of generalization which, however, is not as significant as that in Figure 1a. Besides, a similar increase of validation loss could be observed in both Figure 3b and Figure 4b.

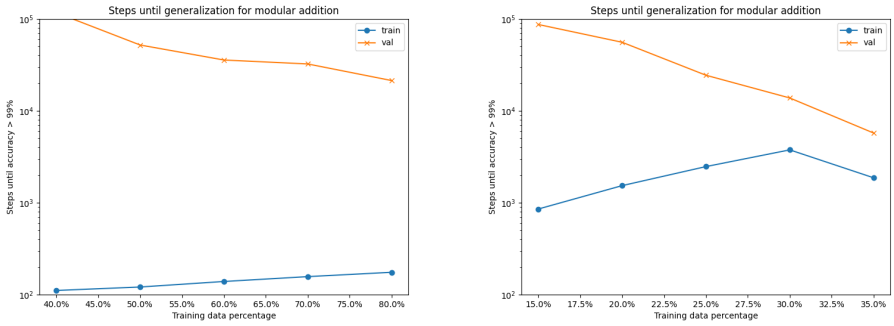
3.3 Effects of Different Hyper-Parameters

3.4 Grokking for K-Wise Modular Addition

In this task, we investigate the grokking phenomenon of K-wise modular addition. Due to the limitation of GPU memory, we only perform experiments on $2 \leq K \leq 5$, and $p = 31$. Specifically, we attempt to discover grokking phenomenon by adjusting training data fraction α, and use techniques such as weight decay and drop out to lower down the scale of training data as much as possible. We use the model Transformer which has the default parameters as shown in 3.1.

The minimal training steps for memorization and generalization on different alpha when $K = 2, 3$ is plotted in Figure 5. Comparing Figures 5a and 5b,we discover that when $K = 3$, the grokking could happen at lower α than when $K = 2$, but the gap between memorization and generalization is not as obvious as when $K = 2$.

This above conclusion is also applied for $K = 4$, but the grokking phenomenon is harder to discover. Figure 6 illustrates the behaviors for $K = 4$ with small variance α. Figures 6a and 6b show that

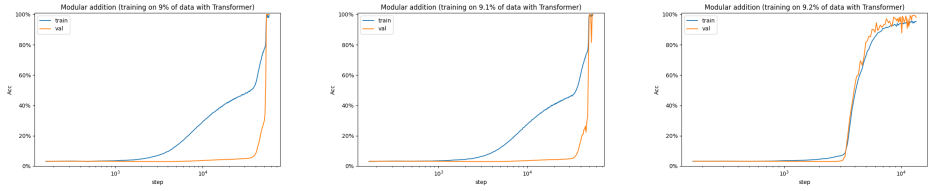


(a) Grokking phenomenon for $K = 2$ (b) Grokking phenomenon for $K = 3$

Figure 5: Grokking phenomenon for different K and α

grokking phenomenon would happen before the training accuracy reaches 60%, and Figure 6c shows that it is also possible that grokking would not happen with a small disturbance to α. Besides, we don't find grokking phenomenon within 10^5 steps when $\alpha \leq 8\%$.

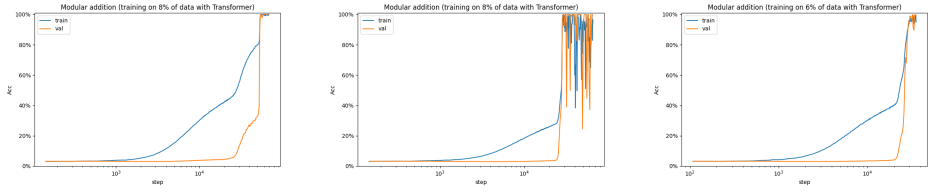
In summary, the experiments imply that the model has better generalization performance when K is larger, with smaller training data fraction, but the grokking phenomenon is less apparent.



(a) $\alpha = 9.0\%$ (b) $\alpha = 9.1\%$ (c) $\alpha = 9.2\%$

Figure 6: Grokking phenomenon for $K = 4$ with small variance α

In addition, as Figure 7 illustrates, we find that with proper adjustment to weight decay and dropout, the model could generalize with fewer training data, but sacrifice training stability.



(a) $\alpha = 8.0\%$ weight decay = 0.1 dropout = 0.2

(b) $\alpha = 8.0\%$ weight decay = 0.2 dropout = 0.3

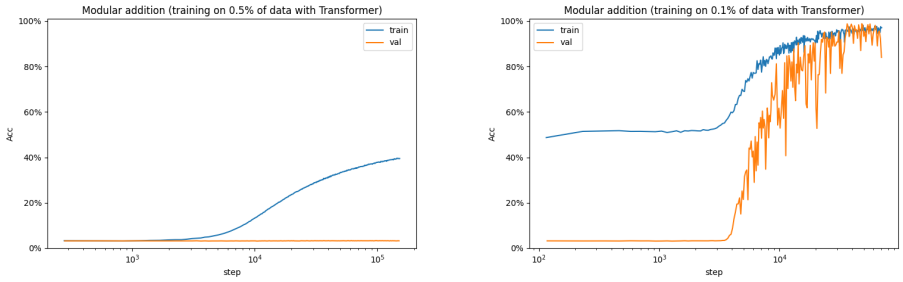
(c) $\alpha = 6.0\%$ weight decay = 0.2 dropout = 0.3

Figure 7: Grokking phenomenon for $K = 4$ with different weight decay and dropout

We further investigate the situation when $K = 5$. Since it has an amount of more than 2×10^7 data, we could only let $\alpha \leq 0.5\%$[3]. This greatly limits our model's generalization ability and Figure 8a suggests no sign of grokking within 10^5 training steps. Then we try to modify training dataset by adding all equations of $K = 2, 3$ to it. Surprisingly the model successfully generalizes within 10^5 steps with no more than 0.1% training data fraction (not include equations we added) as Figure 8b

[3]Validation dataset is also limited. The dataset has been split into a training set and a validation set in a ratio of 1:3.

shows.[4] This suggests that the modification to training set may be effective. We look forward to further study about this technique on the larger K.



(a) Train and val accuracy of $\alpha = 0.5\%$ without modified data

(b) Train and val accuracy of $\alpha = 0.1\%$ with modified data

Figure 8: Grokking phenomenon for $K = 5$

4 An Explanation of the Grokking Phenomenon

In this section, we provide an explanation of the grokking phenomenon based on [3], which claims that grokking happens as a transition between different regimes of training. We first briefly review the kernel regime and rich regime defined in [3] in Section 4.1, and illustrate how they help explain the grokking phenomenon for modular addition in Section 4.2.

4.1 Kernel Regime And Rich Regime

We have the following definition of neural tangent kernel (NTK).

Definition 4.1 (Neural Tangent Kernel [2, 3])**.** Let Θ be the parameter space and $\mathcal{X}$ be the input space. Let $f\colon \Theta \times \mathcal{X} \to \mathcal{Y}$ be a neural network. For $\theta \in \Theta$, the *neural tangent kernel* of $f(\theta, \cdot)$ is defined as

$$K_\theta(\mathbf{x}, \mathbf{x}') := \nabla_\theta f(\theta, \mathbf{x}) \nabla_\theta f(\theta, \mathbf{x}')^\top, \quad \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}.$$

In the *kernel regime*, we have the following approximation of the model output,

$$f(\theta, \mathbf{x}) \approx f(\theta_0, \mathbf{x}) + \langle \nabla_\theta f(\theta_0, \mathbf{x}), \theta - \theta_0 \rangle .$$

Note that the right-hand side is linear in θ, which means that $f(\theta, \cdot)$ has approximately the same expressivity as the kernel method with respect to K_{θ_0}, as defined in Definition 4.1. Under gradient descent, the model's parameters θ are restricted to the affine subspace $W := \mathrm{span}\{\nabla_\theta f(\theta_0, \mathbf{x}_i)\}_{i=1}^n$. Therefore, the model will first converge to the local optimal solution $\theta^*_W \in W$ in the kernel regime. *This corresponds to the stage where the model overfits the training data but does not generalize.*

When the model begins to learn important non-linear features, it will escape W and enter the *rich regime*, where it finally converges towards the global optimal solution θ^*. *This corresponds to the generalization phase in the grokking phenomenon.*

It is worth mentioning that normalization methods such as weight decay accelerates generalization by encouraging the model to escape from the kernel regime, which is consistent with our experiment results in Section 3.3.

4.2 Regime Transition in Modular Addition Problem

We further illustrate how grokking is related to the transition between the aforementioned two regimes in the modular addition problem. We first rewrite the target function f^* as

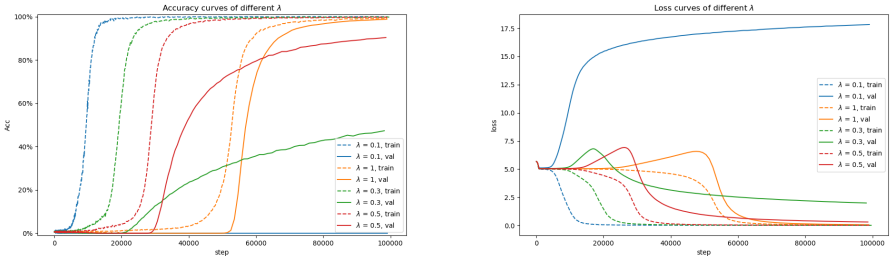
$$f^*(e_a, e_b) = e_{a+b} = H^a e_b = H^b e_a,$$

[4] The training accuracy quickly raises to about 50% because the added equations account about 50% in the training set. It seems that the model quickly learns the case of $K = 2, 3$.

where $H = \sum_{j=0}^{p-1} e_{j+1} e_j^\top$ is the $p \times p$ cyclic permutation matrix. When the first (second) coordinate of f^* is fixed, it becomes a simple linear function of the second (first) coordinate. Following this observation, we consider the following generalized version of f^*.

Let A be any non-empty subset of $\mathbb{Z}_p$, and $E(A) := \{e_a : a \in A\}$. Denote $E(\mathbb{Z}_p) := \{e_0, e_1, \ldots, e_{p-1}\}$. Define $f_A^* : E(A) \times E(\mathbb{Z}_p) \to E(\mathbb{Z}_p)$ to be the restriction of f^* on $E(A) \times E(\mathbb{Z}_p)$. Intuitively, the smaller $|A|$ is, the closer f_A^* is to a linear map. We define $\lambda := \frac{|A|}{p} \in (0, 1]$ to be a measure of the "non-linearity" of f_A^*. Specifically, $f^* = f_{\mathbb{Z}_p}^*$, and thus has non-linearity 1.

To prove the explanations in Section 4.1, we study how the non-linearity λ influences the grokking phenomenon. For each given λ, we sample a random subset $A \subseteq \mathbb{Z}_p$ of size λp. We then train a 2-layer MLP on $A \times \mathbb{Z}_p$ with training data fraction $\alpha = 0.6$, SGD optimizer and weight decay 0 to learn the target function f_A^*. The results are shown in Figure 9.



(a) Training and validation accuracy

(b) Training and validation loss

Figure 9: Learning curves of different target functions f_A^*.

Figure 9a shows that it becomes slower to overfit the training data as non-linearity λ increases. Perhaps surprisingly, generalization nevertheless becomes easier as λ increases. For $\lambda = 0.1$, the model does not generalize at all and the validation loss blows up. For $\lambda = 0.3$ and 0.5, some generalization happens, but the validation loss does not fully converge within 10^5 steps. For $\lambda = 1.0$, the model quickly generalizes after it begins to fit the training data, and the grokking phenomenon almost vanishes.

We believe this interesting result reflects the mechanism of regime transition. For smaller λ's, the target function f_A^* behaves more like a linear function. Hence, the model tends to first learn these linear features. It is then trapped in the kernel regime, and have to take a long time to escape from the affine subspace W and enter the rich regime; this is how grokking happens. To be specific, the peaks of the loss curves in Figure 9b approximately corresponds to the kernel regime, and the similar shapes can also be observed in Figures 1b, 3b and 4b. In contrast, For a larger λ, the function f_A^* is very far from a linear map, hence forcing the model to learn other useful features. Therefore, the model quickly moves to the rich regime and converges to the global optimal solution. Since the transition happens immediately, grokking almost disappears in this setting.

References

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

[2] Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, pages 8580–8589, 2018.

[3] Tanishq Kumar, Blake Bordelon, Samuel J. Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics. In *ICLR*. OpenReview.net, 2024.

A Appendix

Optionally include extra information (complete proofs, additional experiments and plots) in the appendix. This section will often be part of the supplemental material.