Final Project: Grokking Phenomenon Reproduction

Kecen Sha [*]
2200010611

Yuziheng Wu [*]
2200010878

Di Yue [*]
2100012961

Abstract

We reproduce the grokking phenomenon [Power et al., 2022], that a neural network generalizes long after it memorizes the training data, for modular addition problem, and provide an explanation based on [Kumar et al., ICLR 2024]. [2]

1 Introduction

In the field of machine learning, it is an important goal to understand the generalization dynamics of neural networks. Typically, the model's generalization ability can be well reflected by its performance on the training data. However, when training neural networks for certain problems, generalization may occur long after the model overfits the training data. This striking phenomenon is called *grokking* [1]. Ever since it was first introduced, exhausted efforts have been made to understand grokking from a theoretical viewpoint. For example, grokking is explained by the hardness of representation learning [2], the effect of weight decay and weight norm decrease [3, 4] and the transition between different training regimes [5, 6].

Among those learning tasks where grokking can be observed, the most well-studied one might be the *modular addition problem* [1, 5, 6, 7]. Specifically, given a fixed prime number p, consider learning the output of the modular arithmetic:

$$a_1 + a_2 + \cdots + a_K = b, \tag{1}$$

where both the individual summands a_i and the sum b are elements of the finite field $\mathbb{Z}/p\mathbb{Z}$. We model (1) as a classification problem, where classes are labeled by integers $b \in \{0, 1, \ldots, p-1\}$. We mainly focus on the case $K = 2$, and discuss general K-wise addition in Section 3.4.

2 Notations and Problem Setup

Throughout, we let p be a prime and denote $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ to be the additive group modulo p. Initially, each number or operand is encoded by a token, and each arithmetic equation in $\mathbb{Z}_p$ is represented by a token string `['a', '+', 'b', '=', 'c']`. Let d_{tok} be the total number of different tokens. For each token t, denote $e_t \in \mathbb{R}^{d_{\text{tok}}}$ to be the one-hot encoding of t, (i.e., e_t has only the t-th coordinate being 1). For $a \in \mathbb{Z}_p$, we sometimes denote its corresponding token also by a, without causing any confusion.

Our whole dataset includes all p^K (tokenized) equations in $\mathbb{Z}_p$. Given the training data fraction $\alpha \in (0, 1)$, we randomly take an α-fraction of the dataset as the training set and the rest serves as the validation set.

Our classification model f takes as input $(e_{a_1}, e_{a_2}, \ldots, e_{a_K}) \in \mathbb{R}^{K \cdot d_{\text{tok}}}$, where $a_i \in \mathbb{Z}_p$, and outputs a prediction vector $\mathbf{y} \in \mathbb{R}^{d_{\text{tok}}}$. Throughout the model is evaluated by cross-entropy loss with the target output $e_{a_1+a_2+\cdots+a_K}$.

[*]Equal contribution.

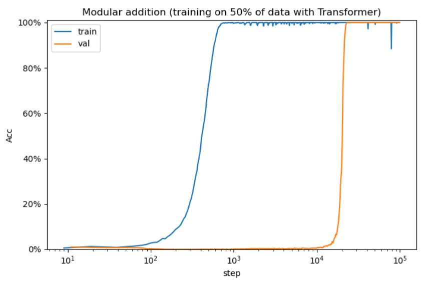
[2]All the code and supplementary materials are available at: `https://github.com/tnediserp/MIML-grokking`

3 Experiments

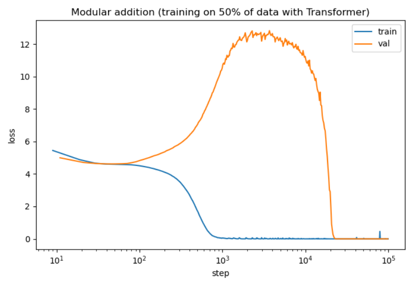
3.1 Reproducing the Grokking Curve

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. We set the prime to be $p = 97$ and the training data fraction to be $\alpha = 0.5$. 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

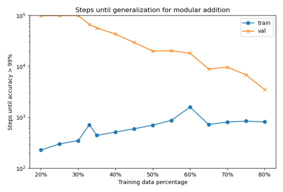


(b) Training and validation loss

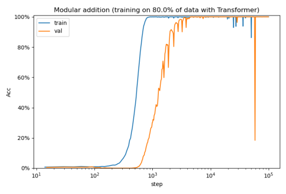
Figure 1: The learning curves of transformer model

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.

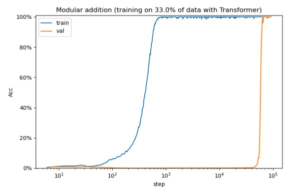
We further study the effect of the training data fraction α. For each α we sample, we train the model for at most 10^5 steps, and determine the minimum number of steps required to achieve validation accuracy $\geq 99\%$. The results are plotted in Figure 2a. As a comparison, we plot the accuracy curves 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

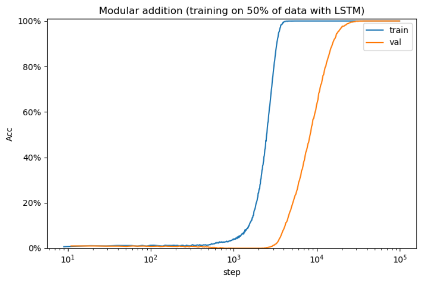
When $\alpha = 80\%$, the model generalizes in 10^4 steps. As α decreases, it becomes easier for the model to overfit the training data, while the number of steps required for generalization increases rapidly. When $\alpha \leq 30\%$, the model would not generalize in 10^5 steps.

3.2 Grokking Phenomenon of Other Models

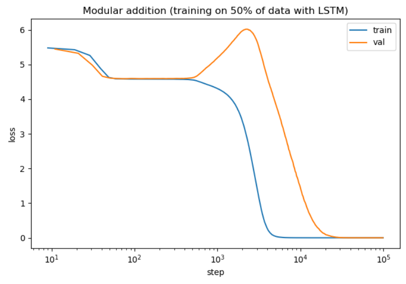
We study the grokking phenomenon for two other network architectures, long short-term memory (LSTM) and multilayer perceptron (MLP), suggesting that grokking is not unique for transformer. Specifically, our LSTM has 2 layers, a hidden state of size $d_{\text{hidden}} = 128$, and dropout 0.1. Our

MLP has 2 hidden layers of size $d_{\text{hidden}}^{(1)} = d_{\text{hidden}}^{(2)} = 256$, 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.

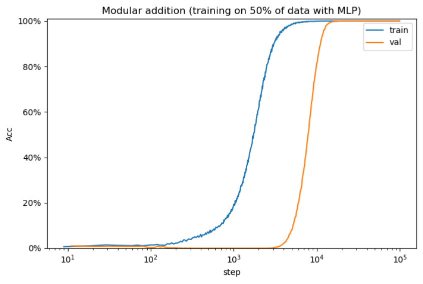


(a) Training and validation accuracy

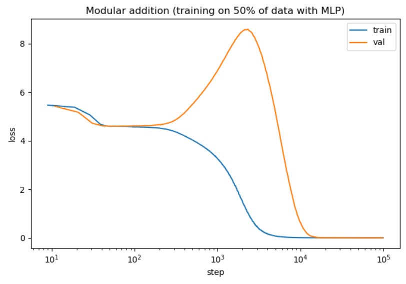


(b) Training and validation loss

Figure 3: The learning curves of LSTM model



(a) Training and validation accuracy



(b) Training and validation loss

Figure 4: The learning 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 Hyperparameters

In this task, we study the effects of hyperparameters such as the optimizer, learning rate, weight decay, dropout and batch size. For all experiments in this section, we set $p = 97$ and use the transformer model. Our baseline setting has the same hyperparameters as mentioned at the beginning of Section 3.1.

It can be seen from Figure 5 that the model generalizes well in the baseline setting for training data fraction $\alpha = 33\%$, but does not generalize even for $\alpha = 60\%$ when the learning rate is changed into 10^{-4}. This is because a small learning rate slows down training. However, big learning rates do not necessarily speed up training, which can be seen by comparing the first two figures on the second row. Removing dropout has little impact on AdamW (top right). Adding weight decay makes SGD's training even worse (middle right). As for the comparison between optimizers, under the same learning rate and dropout settings, AdamW is the best and SGD with Nesterov acceleration is the worst. RMSprop and mini batch Adam have acceptable performance when $\alpha \geq 50\%$, while full batch Adam needs to exceed 70% to reach the same level.

We further study the impact of weight decay by training the transformer model with AdamW optimizer with different weight decay. The results are plotted in Figure 6. As the weight decay increases, the generalization becomes easier, and the grokking phenomenon becomes less obvious. However, it is worth noting that when the weight decay exceeds 0.2, there are occasional "rollback" phenomena in training accuracy and validation accuracy on certain training steps, which occur more frequently

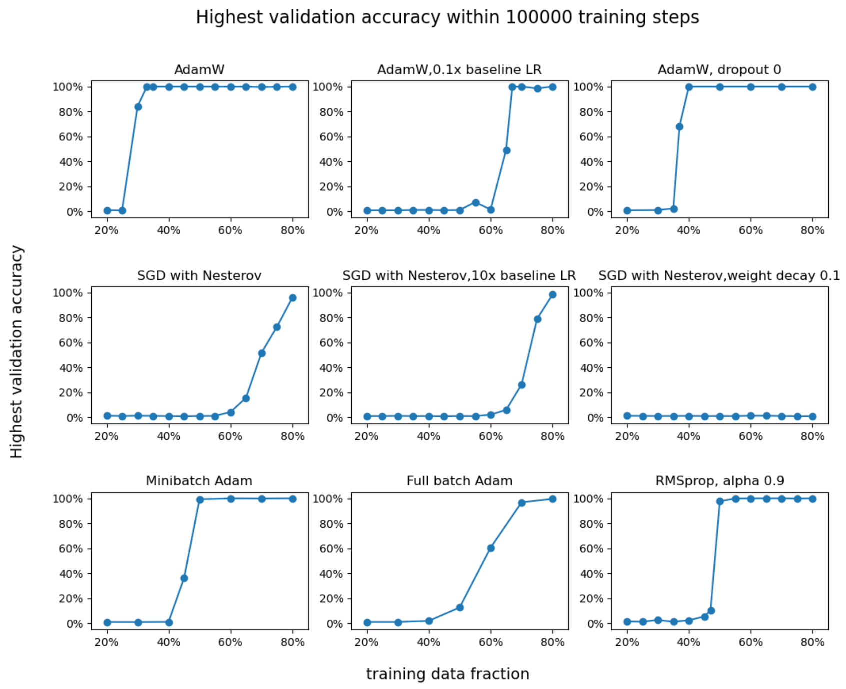


Figure 5: Grokking phenomenon for different settings. Top left: Baseline setting (see Section 3.1). Top center: Baseline setting with learning rate 10^{-4}. Top right: Baseline setting with dropout 0. Middle left: SGD with Nesterov, weight decay 0, learning rate 10^{-3}. Center: SGD with Nesterov, weight decay 0, learning rate 10^{-2}. Middle right: SGD with Nesterov, weight decay 0.1, learning rate 10^{-3}. Bottom left: Mini batch Adam with weight decay 0. Bottom center: Full batch Adam with weight decay 0. Bottom right: RMSprop with $\alpha = 0.9$, weight decay 0.

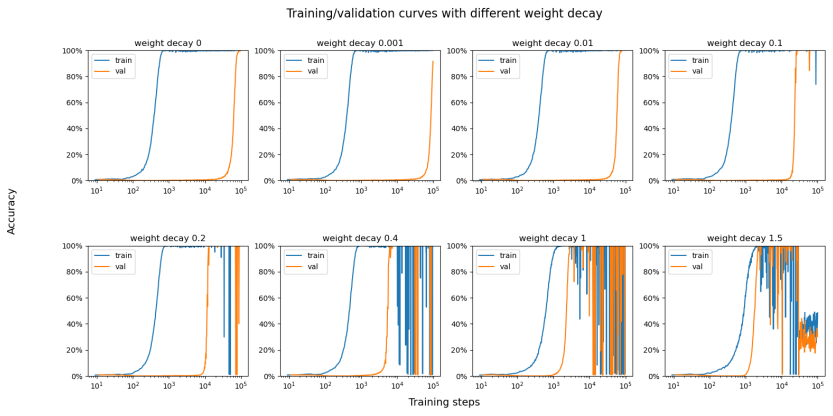


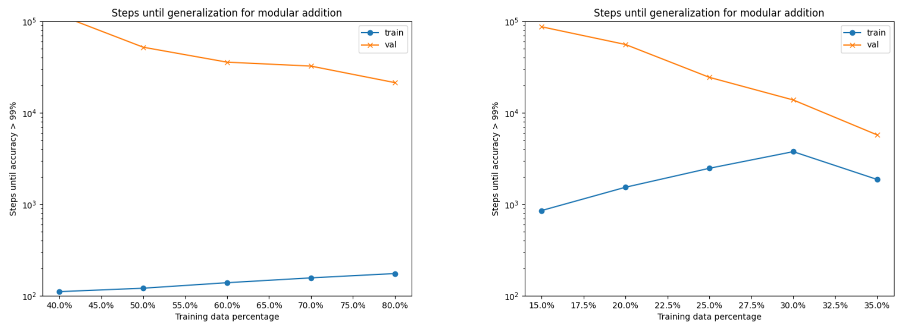
Figure 6: Grokking phenomenon for different weight decay

as the weight decay increases. When the weight decay is 1.5 and the number of training steps exceeds 3×10^4, accuracy cannot even be restored. The reason is worth further exploration, but at least it tells us that the selection of training steps does not need to be too large.

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 $2 \le K \le 5$, and $p = 31$. Specifically, we attempt to discover grokking phenomenon by adjusting training data fraction α, and use techniques such as weight decay and dropout to lower down the scale of training data as much as possible. We use the transformer model which has the default parameters as shown in Section 3.1.

The minimal training steps required for memorization and generalization on different α is plotted in Figure 7, for $K = 2, 3$ respectively. Comparing Figure 7a with Figure 7b, we discover that when $K = 3$, generalization could happen at a smaller α than when $K = 2$, but the gap between memorization and generalization is not as obvious as when $K = 2$.

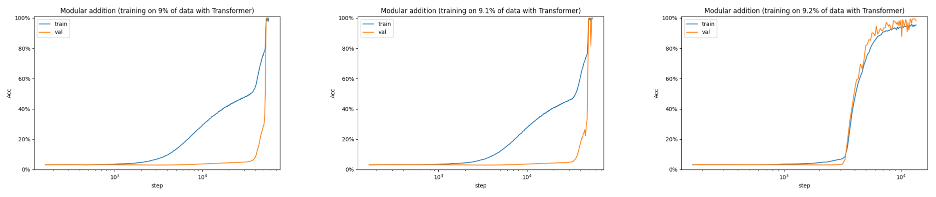


(a) Grokking phenomenon for $K = 2$ (b) Grokking phenomenon for $K = 3$

Figure 7: Grokking phenomenon for different K and α

This above conclusion is also applied for $K = 4$, but the grokking phenomenon is harder to discover. Figure 8 illustrates the behaviors for $K = 4$ with small variance α. Figures 8a and 8b show that generalization would happen before the training accuracy reaches 60%, and Figure 8c shows that it is also possible that grokking would not happen with a small disturbance to α. Besides, we do not find grokking phenomenon within 10^5 steps when $\alpha \leq 8\%$.

In summary, the experiments imply that for a larger K, the model has better generalization performance, which could occur even for a small training data fraction. Nevertheless, the grokking phenomenon becomes less apparent. We provide an explanation of this in Section 4.2.



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

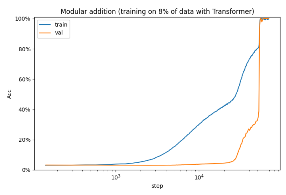
Figure 8: Grokking phenomenon for $K = 4$ with small variance α

In addition, as Figure 9 illustrates, we find that with proper adjustment to weight decay and dropout, the model could generalize with fewer training data, but sacrifice training stability.

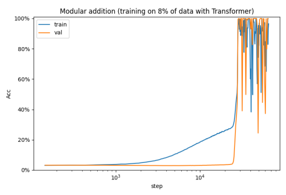
We further investigate the situation when $K = 5$. Since it has an amount of more than 2×10^7 data, we could only let $\alpha \leq 0.5\%$[3]. This greatly limits our model's generalization ability and Figure 10a suggests no sign of generalization within 10^5 training steps. Then we try to modify the 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 10b 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.

[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.

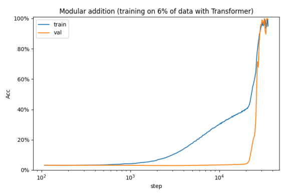
[4]The training accuracy quickly raises to about 50% because the added equations account about 50% in the training set. It seems that the model quickly learns the case of $K = 2, 3$.



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

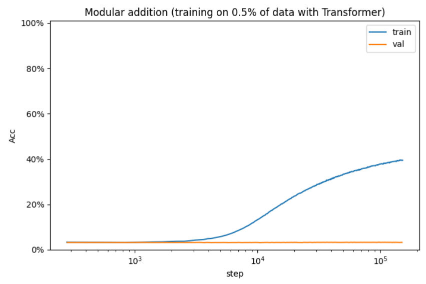


(b) $\alpha = 8.0\%$
weight decay = 0.2
dropout = 0.3

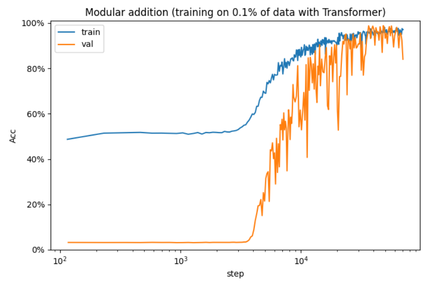


(c) $\alpha = 6.0\%$
weight decay = 0.2
dropout = 0.3

Figure 9: Grokking phenomenon for $K = 4$ with different weight decay and dropout



(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 10: 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 [5], 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 [5] 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 [8, 5])**.** 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,

$$f(\theta, \mathbf{x}) \approx f(\theta_0, \mathbf{x}) + \langle \nabla_\theta f(\theta_0, \mathbf{x}), \theta - \theta_0 \rangle . \tag{2}$$

Note that the right-hand side of (2) 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 $W := \operatorname{span}\{\nabla_\theta f(\theta_0, \mathbf{x}_i)\}_{i=1}^n$. 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 (see Figure 6).

4.2 Regime Transition in Modular Addition Problem

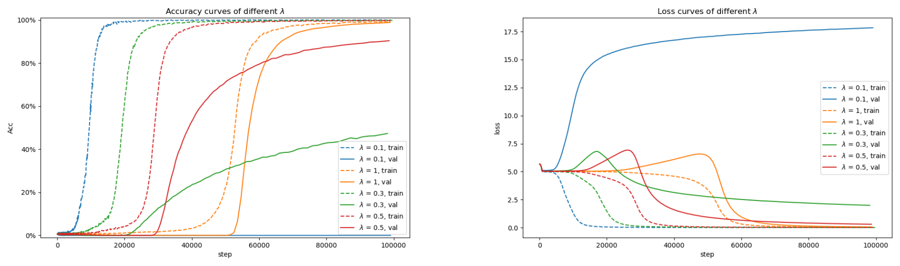
We further illustrate how grokking is related to the transition between the two aforementioned regimes in the modular addition problem. We first rewrite the target function f^* as

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

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 11.



(a) Training and validation accuracy

(b) Training and validation loss

Figure 11: Learning curves of different target functions f_A^*.

Figure 11a 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 11b 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.

Finally, we note that the influence of non-linearity λ is consistent with our results in Section 3.4. For $K \geq 3$, let f^K and f^{K-1} be the target functions of K-wise and $(K-1)$-wise modular addition, respectively. No hard to see that f^{K-1} is equivalent to $f_{\{0\}}^K$, which has non-linearity $\lambda = 1/p$ compared with f^K. Therefore, it would be easier for f^K to generalize than f^{K-1}. This explains our experiment results for $K = 2, 3, 4$ in Figures 7 and 8.

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] Ziming Liu, Ouail Kitouni, Niklas Nolte, Eric J. Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. In *NeurIPS*, 2022.

[3] Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency. *CoRR*, abs/2309.02390, 2023.

[4] Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *ICLR*. OpenReview.net, 2023.

[5] 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.

[6] Mohamad Amin Mohamadi, Zhiyuan Li, Lei Wu, and Danica J. Sutherland. Why do you grok? A theoretical analysis on grokking modular addition. In *ICML*. OpenReview.net, 2024.

[7] Andrey Gromov. Grokking modular arithmetic. *CoRR*, abs/2301.02679, 2023.

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