An In-depth Exploration and Replication of the Grokking Phenomenon in Deep Learning

Hualing Zhong (钟华玲) 2301991750
Graduate School of China Academy of Engineering Physics
zhonghualing24@gscaep.ac.cn

Abstract

This study delves into the intriguing phenomenon of Grokking, where neural networks generalize after extensive training on small algorithmic datasets. By replicating key findings from [1], we explore the effects of varying training data fractions, model architectures, optimizers, and regularization techniques on the Grokking process. Our analysis also extends to multiple modular addition datasets, examining how the number of summands influences learning outcomes. Ultimately, we provide a comprehensive explanation for the Grokking phenomenon, emphasizing the role of regularization in guiding models towards effective generalization. The code can refer to https://github.com/zhonghl945/Grokking.

1 Introduction

The Grokking phenomenon represents a critical yet elusive stage in machine learning where models transition from merely memorizing training data to achieving true understanding and generalization. First noted by Power et al.[1], this phenomenon underscores the complex interplay between overfitting and generalization, highlighting the importance of regularization techniques in fostering robust learning behaviors. This project aims to deepen our understanding of Grokking through systematic experimentation with modular arithmetic tasks, thereby contributing valuable insights into the mechanisms that govern neural network learning. In this project, our aim is to explore the Grokking phenomenon in the context of learning modular addition : $(x, y) \mapsto (x + y) mod\ p,\ x, y \in \mathbb{Z}_p$. Since "integers" are not assumed to be known to the model, they need to be one-hot encoded. Furthermore, we consider the tokens for the operators and the equality sign, which are also treated as input.

The remainder of the report is organized as follows. In Section 2, we replicate the core results of [1] using a Transformer model and AdamW optimizer, while also examining how varying training data fraction α influence the Grokking phenomenon. Section 3 and Section 4 address different models, optimizers, and regularization techniques. In Section 5, we conduct experiments on modular additions that involve multiple numbers. Finally, in Section 6, we provide an explanation for the Grokking phenomenon.

2 Task 1

The Transformer model we use includes the self-attention mechanism and the feed-forward network and uses position coding and token embedding. More details please refer to Appendix A.

The hyperparameter settings are as follows: the batch size is 256, the learning rate in AdamW is 5×10^{-4}, with $\beta_1 = 0.9$, $\beta_2 = 0.98$, and weight decay set to 1. Additionally, the learning rate is decayed every 5000 steps, with a decay rate of 0.8. Last, the dropout is set 0.

2.1 Grokking with transformer and AdamW

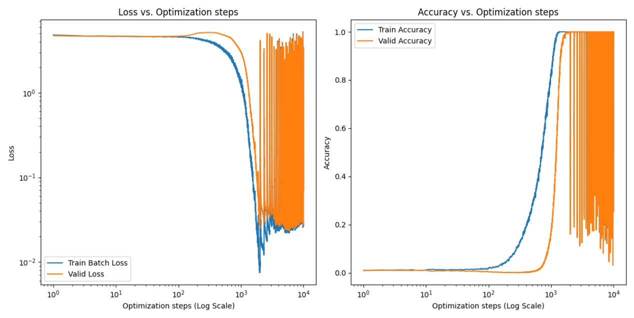
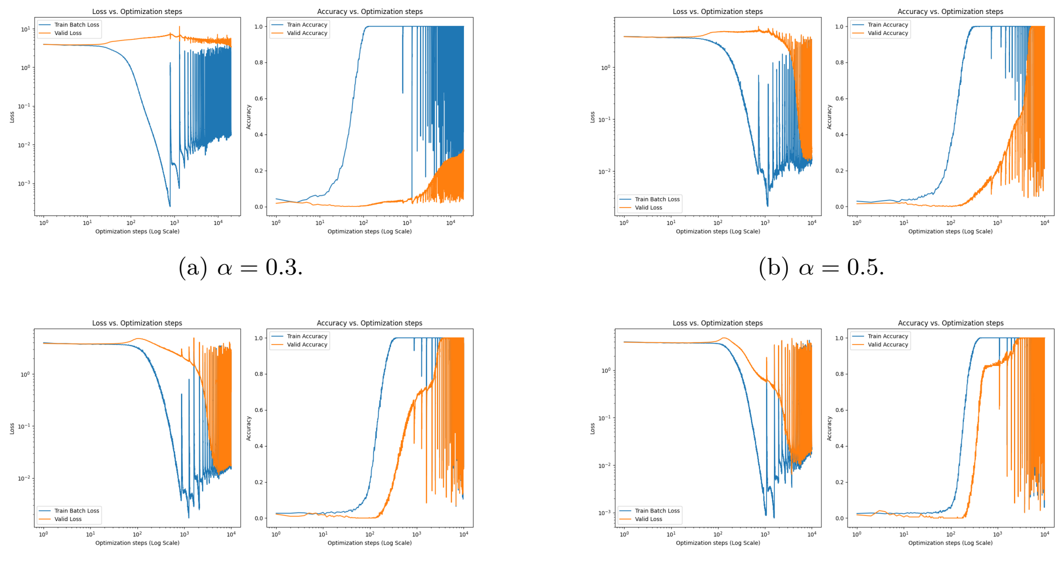


Figure 1: Transformer, $p = 97$, $\alpha = 0.5$.

As shown in Figure 1, It can be observed that severe oscillations occurred in the later stages of training, which may be related to the learning rate decay strategy. However, this did not affect the occurrence of the Grokking phenomenon. In the subsequent experiments, we observe more pronounced instances of Grokking.

2.2 Impact of the training data fraction



(a) $\alpha = 0.3$. (b) $\alpha = 0.5$.

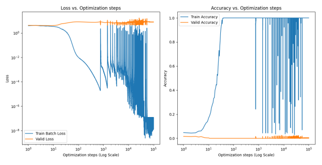
(c) $\alpha = 0.7$. (d) $\alpha = 0.9$.

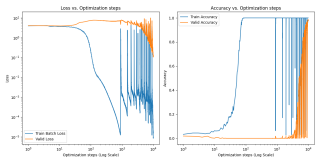
Figure 2: Transformer, $p = 41$, different α.

To save on computational costs, we chose a smaller $p = 41$. From the results, it can be seen that the smaller the data fraction α, the later the Grokking phenomenon occurs, and it may even fail to generalize. Reducing p, which is equivalent to reducing the training set size, makes the Grokking phenomenon more pronounced.

3 Task 2

For details on the network architecture settings of MLP and LSTM, please refer to Appendix B. And the hyperparameter settings are the same as mentioned above.


(a) $\alpha = 0.3$.


(b) $\alpha = 0.5$.

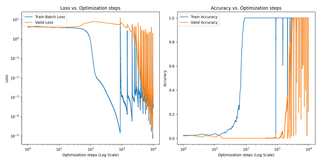
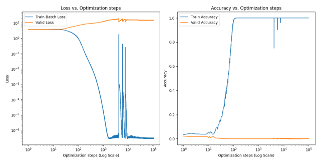
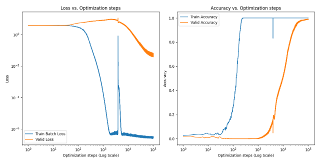

(c) $\alpha = 0.7$.

Figure 3: MLP, $p = 41$, different α.


(a) $\alpha = 0.3$.


(b) $\alpha = 0.5$.

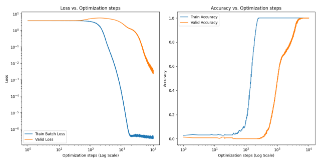

(c) $\alpha = 0.7$.

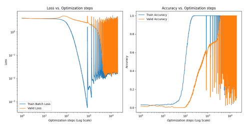
Figure 4: LSTM, $p = 41$, different α.

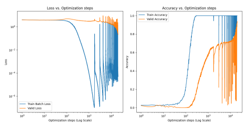
From the results, it can be seen that at $\alpha = 0.3$, the performance of MLP and LSTM is similar to that of Transformer, with all models requiring more time to generalize. The occurrence of the Grokking phenomenon happens in the following order, from earliest to latest: Transformer $<$ MLP $<$ LSTM. Besides, we observe that LSTM shows smoother training progress.

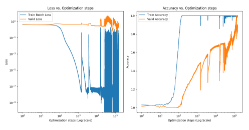
4 Task 3

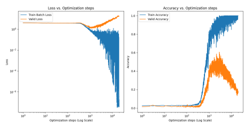
For different optimizers, we tried AdamW with various weight decay (WD)and batch sizes(BS), as well as SGD and RMSprop with different learning rates(LR). In terms of the model, we also experimented with different dropout rates(DR). All of the above changes were made within the Transformer model and data fraction $\alpha = 0.7$. The hyperparameters not mentioned were kept consistent with the settings in Task 1.

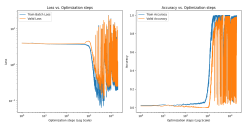
4.1 Effects of different optimizers

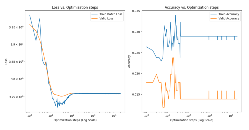

(a) AdamW, LR=5e-4, BS=1024, WD=1.

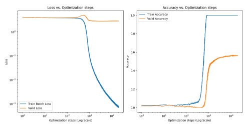

(b) AdamW, LR=5e-4, BS=256, WD=0.3.


(c) AdamW, LR=5e-4, BS=512, WD=0.01.


(d) RMSprop, LR=5e-4, BS=256, WD=0.


(e) SGD, LR=5e-3, BS=256, WD=0.01.


(f) SGD, LR=5e-3, BS=256, WD=0.5.


(g) SGD, LR=5e-3, BS=256, WD=0.

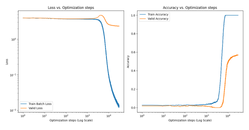
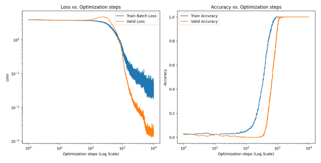

(h) SGD, LR=5e-4, BS=256, WD=0.

Figure 5: Transformer on different optimizers, $p = 41$, $\alpha = 0.7$.

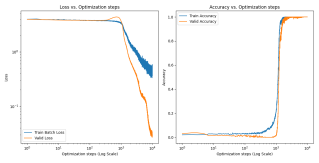
Analyzing the results in the figure, the following preliminary conclusions can be drawn under the Transformer model:

- RMSprop and SGD optimizers are not as effective as AdamW in this experiment.
- The batch size has little impact on the results.
- A smaller learning rate leads to slower convergence, but it does not have a decisive effect on Grokking.
- A larger weight decay is more conducive to the occurrence of Grokking.

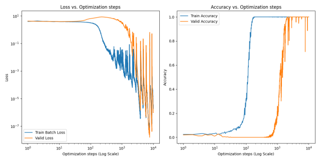
4.2 Effects of dropout



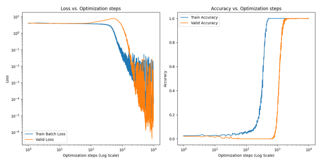
(a) Transformer, DR=0.1.



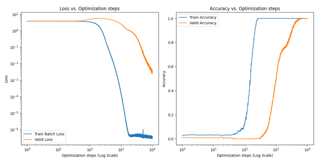
(b) Transformer, DR=0.3.



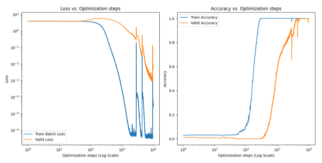
(c) MLP, DR=0.1



(d) MLP, DR=0.3.



(e) LSTM, DR=0.1.



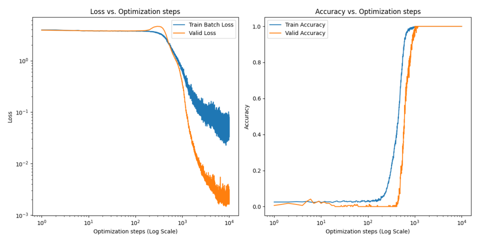
(f) LSTM, DR=0.3

Figure 6: Different model with different dropout, $p = 41$, $\alpha = 0.7$.

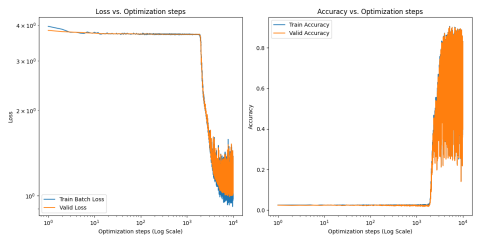
In Task 1 and Task 2, we have already demonstrated how Grokking occurs when dropout $=$ 0. The results above show the outcomes when dropout is set to 0.1 and 0.3. We can observe that dropout is a crucial parameter for Grokking. The larger the dropout, the earlier the model generalizes.

5 Task 4

Consider harder problem $(x_1, x_2, \ldots, x_K) \mapsto \left(\sum_{k=1}^{K} x_k\right) mod\ p$, where $x_k \in [p]$ for $k = 1, \ldots, K$. We continue to use the Transformer and hyperparameter settings from Task 1. The results for $K = 3$ and $K = 4$ are provided below.



(a) $K = 3$.



(b) $K = 4$.

Figure 7: Transformer, $p = 41$, $\alpha = 0.7$, different K.

Combined with the experimental results with $K = 2$ in Task 2, our experiments show that as K increases, Grokking becomes more difficult to achieve. This may be related to the fact that we fixed the data score. Specifically, increasing K corresponds to more training data, which means that generalization and fitting occur simultaneously.

6 Task 5

Building upon our empirical findings and drawing parallels with recent advancements in the field [2, 3], the phenomenon of grokking can be understood as a special behavior in the neural network learning process. This process can be divided into three stages: **memory**, **algorithm formation**, and **optimization**.

- In this initial stage, the model gradually memorizes the training data by repeatedly encountering the data points. This is clearly reflected in the rapid increase in training accuracy. At this point, the model has not yet learned how to generalize to unseen data; it simply memorizes the patterns in the training set.
- As training progresses, regularization techniques such as weight decay begin to take effect, forcing the model to search for broader rules rather than relying solely on the memorization of specific training samples. Our experiments show that during this stage, the model starts to construct effective algorithmic structures capable of handling a broader range of situations. For example, in a modular arithmetic task, the network might discover mathematical properties related to modular operations and start using them to make predictions, rather than simply relying on the memorized data points.
- Finally, there is a sudden and significant improvement in performance on the validation set. This leap marks the point at which the model has learned a generalized solution and can begin applying it effectively to new test cases. At this stage, regularization continues to help eliminate unnecessary complexity or redundant connections, making the model more streamlined and efficient, focusing on the key features that truly contribute to solving the problem.

It is important to note that our research indicates that appropriate regularization is one of the key factors in ensuring the occurrence of grokking. Without regularization methods (such as weight decay or dropout), the model may fall into overfitting and fail to transition from memorization to true understanding.

In summary, grokking is a process that involves memory, exploration of effective algorithms, and ultimately optimization for good generalization. Regularization plays a crucial role in this process, not only preventing overfitting but also guiding the model to evolve in the right direction, enabling it to provide accurate answers when faced with new data.

References

[1] A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. arXiv preprint arXiv:2201.02177, (2022).

[2] Andriushchenko, M. and Flammarion, N. Towards understanding sharpness-aware minimization. In International Conference on Machine Learning, pp. 639–668. PMLR, 2022.

[3] Li, B., Jin, J., Zhong, H., Hopcroft, J., and Wang, L. Why robust generalization in deep learning is difficult: Perspective of expressive power. Advances in Neural Information Processing Systems, 35:4370–4384, 2022.

Appendix

Appendix A

Each `DecoderBlock` contains a multi-head self-attention layer (`nn.MultiheadAttention`), a feed-forward network (`nn.Linear`), layer normalization (`nn.LayerNorm`), and dropout (`nn.Dropout`). The self-attention layer employs an upper triangular mask (`attn_mask`) to ensure that each time step only attends to the current and previous time steps, preventing information leakage. The feed-forward network consists of two fully connected layers with an activation function (`GELU`) to enhance the model's expressive power.

The `Transformer` model is composed of several stacked `DecoderBlock`s, forming a standard decoder. The input consists of token embeddings and position embeddings, where the token embeddings represent the vocabulary of each input, and the position embeddings inject the positional information of the sequence. The input embeddings are passed through multiple decoder layers, and the final output is mapped to the output space through a linear layer.

The hyperparameters of the Transformer are set as follows: 2 encoder layers, 128-dimensional hidden layers, 4 attention heads, a default input sequence length of 5, a vocabulary size of $p + 2$, and dropout regularization (default is 0).

Appendix B

The `MLP` model is constructed by stacking multiple `MLPBlock` layers. The input consists of token embeddings and position embeddings. After embedding, the input data passes through several `MLPBlock` layers and is finally mapped to token predictions through a linear layer.

`MLPBlock` is a module with two fully connected layers (Linear) and a ReLU activation function, including layer normalization (`LayerNorm`) and a dropout layer. During the forward pass, the input is passed through the dropout layer, then through the feed-forward network (`ffn`), and finally through a residual connection and layer normalization.

The `LSTM` model uses an LSTM layer to handle sequential data. The token embeddings and position embeddings are passed through the LSTM layer, and the last output is passed through a fully connected layer for prediction.

The hyperparameters of the MLP (Multilayer Perceptron) are set as follows: 2 layers, 128-dimensional hidden layers, 4 attention heads, a vocabulary size of $p + 2$, a default input sequence length of 5, and dropout regularization (default is 0).

The hyperparameters of the LSTM are set as follows: 2 layers, 128-dimensional model input, a hidden dimension of 4×128, a vocabulary size of $p + 2$, a default input sequence length of 5, and dropout regularization (default is 0).