An Investigation Report of the Grokking Phenomenon

Junhe Lou[*] 2100010850

Siqiang Wang [*] 2100010867

Xinye Yang [*] 2100010631

Abstract

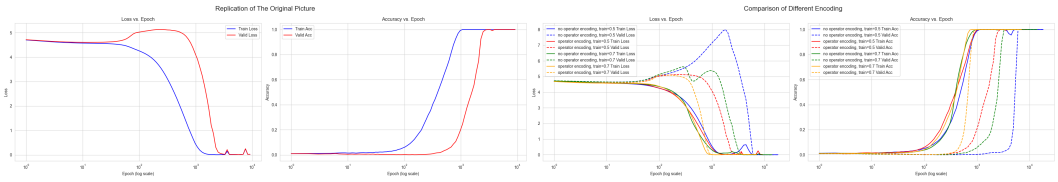
We replicate and analyze the grokking phenomenon of transformers in a modular addition problem from [8]. In a sequence of tasks, we extend grokking to MLP, LSTM and RNNs. We examined the impact of factors such as data size, embedding strategy, training fraction, optimizer, network shape and regularization. Related to core results, we provide broader experiments and discussions about how grokking come into being, and the research horizons of deep learning models. Supplementary materials are available at `https://github.com/stevenWang0825/Mathematics-of-ML-Final-24Fall`.

1 Task 1

1.1 Experimental Background

In this task, we focus on replicating the *grokking* phenomenon presented in the original paper[8] on a small-scale algorithmic dataset: the modular addition problem $(x, y) \mapsto (x + y) \bmod p$. We explored multiple primes p (including 29, 59, and 97), and we also varied both the number of network layers (1-layer vs. 2-layer Transformer) and the training fraction α (0.3, 0.5, 0.7). The goal is to examine how each of these factors affects the time at which grokking occurs.

We largely follow the configuration of the Transformer and the training approach outlined in the original paper [8] and a version of replication `https://github.com/danielmamay/grokking`, using the **AdamW** optimizer. Below is an example of a typical hyperparameter setting (with minor adjustments for different experiments): **Transformer Hyperparameters**:`num_layers`: 2 (also tested 1),`dim_model`: 128, `num_heads`: 4. **Training Hyperparameters**:`batch_size`: 512, `learning_rate`: 5×10^{-5},`weight_decay`: 1.0, `optimizer`: AdamW, `training_fraction` (α): 0.3, 0.5, 0.7,`num_steps`: 100,000. The replication is as follows:



(a) Replication of Grokking (b) Encoding Differences

1.2 Encoding Strategy and Token Indices

Let p be a prime, and consider the function

$$f : \{0, 1, \ldots, p-1\} \times \{0, 1, \ldots, p-1\} \longrightarrow \{0, 1, \ldots, p-1\}, \quad (x, y) \mapsto (x + y) \bmod p.$$

To train a Transformer on this task, we assign each element $x \in \{0, 1, \ldots, p-1\}$ a *token index* also denoted by x. Crucially, the model *does not interpret these token indices as numeric values*; it only

[*]Equal contribution; alphabetical order.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

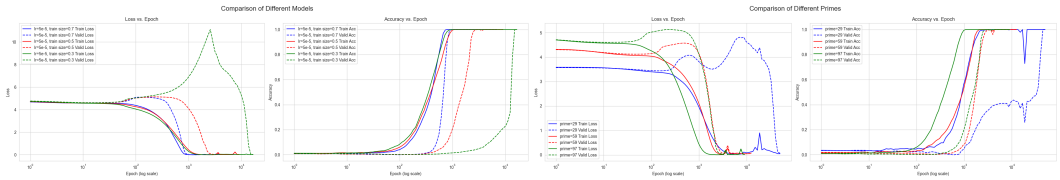
looks up a corresponding embedding vector in $\mathbb{R}^d$. That is, for each token index $j \in \{0, \ldots, p-1\}$, the embedding layer stores a trainable vector $\mathbf{e}_j \in \mathbb{R}^d$, so the model has no direct knowledge that "3" is numerically larger or smaller than "2"—it merely sees distinct vectors $\mathbf{e}_2$ and $\mathbf{e}_3$.

Including + and =: We extend the vocabulary by two additional token indices, one for "+" and one for "=". Each input thus becomes a four-token sequence $[x,\ +,\ y,\ =]$, mapped to $\mathbb{R}^{4\times d}$ before the Transformer processes it. **Excluding + and =:** We remove these operator symbols from the vocabulary, so each pair (x, y) is encoded as just $[x,\ y]$, yielding a $\mathbb{R}^{2\times d}$ input sequence. The model must learn that these two tokens correspond to addends, despite having no explicit "+" token.

Surprisingly, when we embed the operators (+ and =) as additional tokens—thus feeding a input into the Transformer—grokking tends to emerge *earlier* but proceeds more gradually. In contrast, with a input (omitting explicit operators), the model's "insight" arrives *later* but manifests as a sharper jump in test accuracy. Just as 2b shows. One plausible explanation is: the model's representational space is now partially dedicated to embedding these symbols, thus it may spread out its learning steps more evenly, leading to a gentler ascent toward perfect generalization. Conversely, with only two numeric tokens, the model lacks such direct hints; it remains uncertain for longer, then converges more suddenly once it grasps the $(x + y) \bmod p$ pattern. Additionally, having fewer tokens (2 vs. 4) can reduce early "symbolic scaffolding," resulting in a more abrupt transition once the hidden arithmetic rule is finally discovered.

1.3 Delayed Onset of Grokking With Different α and Different Primes

With $\alpha = 0.3$, even though only 30% of the total p^2 data is used, the model still undergoes a prolonged period of overfitting on the training set before abruptly reaching high test accuracy. This closely matches the observations in [8]. When $\alpha = 0.5$ or $\alpha = 0.7$, the "lag" period before test accuracy spikes is shortened; the model reaches high accuracy on the test set earlier. Surprisingly, when p

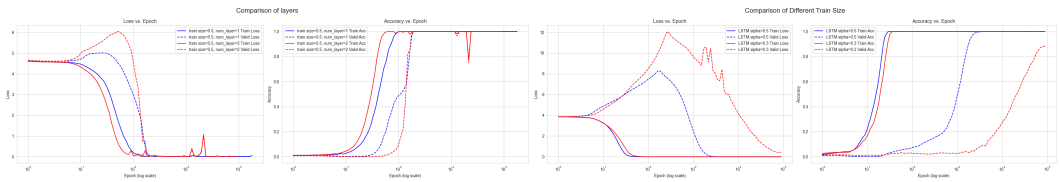


(a) Delayed Onset of Grokking With Different α (b) Scaling with Larger p

= 97, we observe a quicker transition from overfitting to generalization than with p = 29 or p = 59. Intuitively, because p = 97 yields a larger dataset (97^2 samples), a single epoch passes through more unique training pairs, enabling the model to acquire a broader range of examples in fewer total steps. By contrast, for smaller primes p = 29 or p = 59, each epoch contains fewer samples, so the grokking transition often appears more delayed in terms of global training steps. Moreover, training with smaller primes tends to exhibit more oscillations, resulting in a less stable overall training process.

1.4 Effects of Network Depth (1-Layer vs. 2-Layer Transformer)

Counter to initial expectations, a 2-layer Transformer tends to overfit the training set more quickly but experiences a *slower* grokking moment than the 1-layer model. In other words, its "grokking gap" is larger. By contrast, the 1-layer network, though less powerful overall, often achieves the sudden jump in test accuracy earlier, resulting in a smaller gap. This outcome aligns with the intuition that a more parameter-rich architecture has a broader search space and thus requires additional time for the sudden insight that lifts test performance to near-perfection.



(a) Effects of Network Depth (b) Grokking in LSTM

2 Task 2

2.1 Grokking on the LSTM Model

In addition to exploring Transformers, we also tested a two-layer LSTM model for the modular addition task. Owing to the slower convergence typical of LSTM architectures (relative to Transformers) under the same hardware resources and parameter settings, we mainly examined the case $p = 47$ and varied the training set fraction α (e.g., 0.3 or 0.5) to observe differences in overfitting and grokking.

We define embeddings for each token index in $\{0, \ldots, 47\}$, mapping each to a trainable vector in $\mathbb{R}^{128}$. The embeddings are processed by a two-layer LSTM ($d_{\text{hidden}} = 256$), which updates states $(\mathbf{h}_t^{(\ell)}, \mathbf{c}_t^{(\ell)})$ at each time step. At the final step, $\mathbf{h}_3^{(2)}$ is transformed via a linear mapping to logits in $\mathbb{R}^{47}$, and predictions are made using a softmax cross-entropy objective.The LSTM is trained with `AdamW` ($\eta = 10^{-3}$, weight decay 10^{-1}, batch size 128) for up to 10^5 epochs. Although slower than the Transformer, the LSTM achieves high accuracy on modular addition tasks, with smoother convergence as the training fraction α increases, as shown in Figure 3b.

2.2 Grokking on the MLP Model

We employed embedding layers combined with a 3-layer MLP model, featuring a maximum bandwidth of 256. To maintain simplicity, we did not encode the operators "+" and "=".

The batch size was set to 512; larger batch sizes hindered the model's ability to grok under certain parameter configurations. Other parameters were kept consistent with those used in Task One: $p = 97$, a learning rate of 5×10^{-4}, and the `AdamW` optimizer with a weight decay of 1.

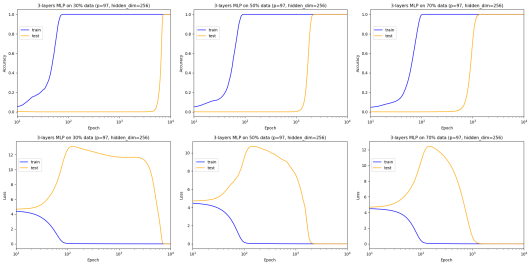


Figure 4: MLP results with different training fraction.

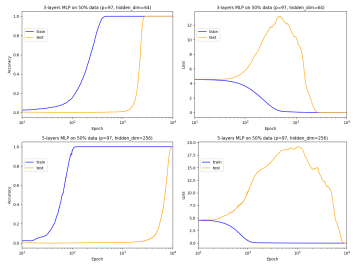


Figure 5: MLP results with different layer numbers and model width.

We investigated the impact of different training fractions α on the MLP's performance, as shown in Figure 4. The curves for training accuracy and loss exhibited similar trends across various α values, suggesting that the training fraction primarily influences the speed of grokking rather than its overall effectiveness.

To determine how structural modifications to the MLP would affect results, we varied both the number of layers and the model width, as shown in Figure 5. Adding extra layers did not lead to faster grokking; rather, it negatively impacted training stability, resulting in slower convergence. Adjustments to the model's width showed minimal effect. A configuration of 3 layers with a width of 256 provided adequate capacity, while increasing the number of parameters beyond this threshold resulted in overfitting.

2.3 Grokking on the Bi-directional RNN Model

We implemented a 3-layer Bi-directional RNN (BiRNN) model with `hidden dimension` 128, `batch size` 512, `learning rate` $5e-5$. Encoding strategies are the same as previous models. Test results 6 show that BiRNN can learn the problem smoothly without a significant grokking process, using the Adam optimizer without any weight decay. Overfitting occurs after ≈ 2000 epochs. Adding the number of RNN layers does not improve learning largely. Instead, stacking more than 4 layers will introduce chaos into the plateau period and cause instability (this will also be showed in later tasks). We also point out that smoothing the accuracy curve only requires reducing the learning rate; batch size does not play an important part in stability, but rather affects the time and memory

usage efficiency. Switching to the AdamW optimizer with weight decay $= 1$ does not make a real difference in this case; the training will be slower. Accordingly, for actual performance, a warm-up process with $\approx 1e-4$ and learning rate decay towards $\approx 1e-5$ is recommended. **BiRNN presents a regular learning curve with no grokking gap under standard parameter settings.**

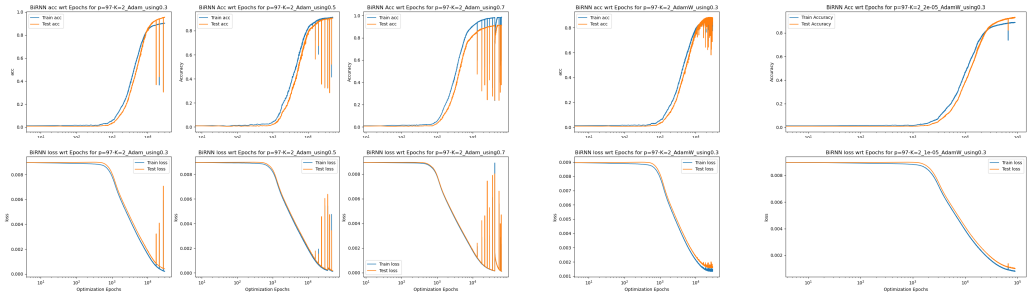


Figure 6: Results for BiRNN; $p = 97$. The left half is the comparison of different fractions of total data used as training data, namely $0.3, 0.5, 0.7$. The middle part showcases the AdamW optimizer's performance when using 0.3 fraction training data. The right part is an example of how a small learning rate as $2e-5$ can smoothen the learning process.

3 Task 3

3.1 Issues on `torch.optim`'s Weight Decay

According to documents `https://pytorch.org/docs/stable/optim.html`, Adam and AdamW differ at the first two upgrade steps (other dampening steps after these two are the same):

$$\text{Adam}: \begin{cases} g_t = \nabla_\theta f_t(\theta_{t-1}) \\ g_t = g_t + \lambda\theta_{t-1} \end{cases} ; \text{AdamW}: \begin{cases} g_t = \nabla_\theta f_t(\theta_{t-1}) \\ \theta_t = \theta_{t-1} - \lambda\gamma\theta_{t-1} \end{cases}$$

where the decay is λ and learning rate η. In Adam, the decay is added to the gradient; AdamW adds the decay separate from the gradient updates. The original motivation of AdamW [7] was to add regularization directly to the parameter, not the gradients. Experiments in the paper showed the approach's stability. We justified these discussions in figure 7. Intuitively, the Adam decay should be set as learning rate $\times$ AdamW decay. To investigate this further, we preformed a grid search on `learning rate` and `weight decay` pairs 8.

3.2 Hyperparameters of MLP

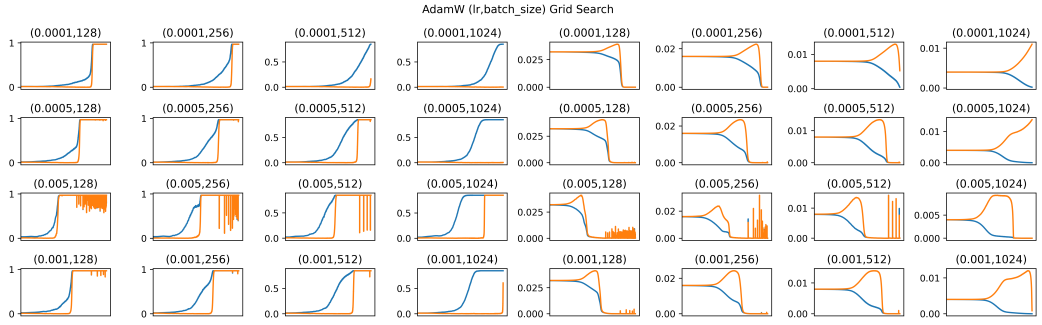


Figure 7: Results of grid search on learning rate and batch size using AdamW optimizer; batch size fixed as 256; all epochs 10^4 i.e. $\approx 1.4 \times 10^5$ steps; weight decay fixed as 1.The left are accuracies and losses are on the right.

Other than Adam and AdamW, we used 3 other optimizers for learning the algorithmic dataset. We also changed model dimensions and other regularization methods 9. Due to the reasons above, we did not use weight decay in the adaptive gradient optimizers besides AdamW, though NAdam had the interface to the AdamW decay version. Fixing AdamW with weight decay 1, we also changed dropout, weight decay, training fraction and model dimensions for a more thorough comparison than

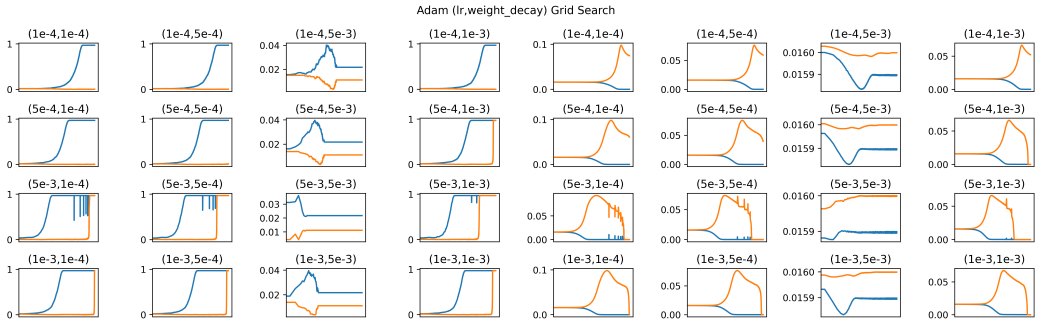


Figure 8: Results of grid search on learning rate and weight decay, using Adam optimizer; batch size fixed as 256; all epochs 10^4 i.e. $\approx 1.4 \times 10^5$ steps. Accuracies are on the left and losses are on the right.

part 2.2. The results demonstrate the crucial part of $L2$ regularization (weight decay) and the trivial part of dropout in learning the dataset. Using half of the whole set as training is empirically best. Increasing the dimension and depth of the model does not make a significant improvement. Our problem's rather small data amount. may be the reason. A smooth scaling-law like improvement cannot be reproduced under a less than $100k$ parameter scale.

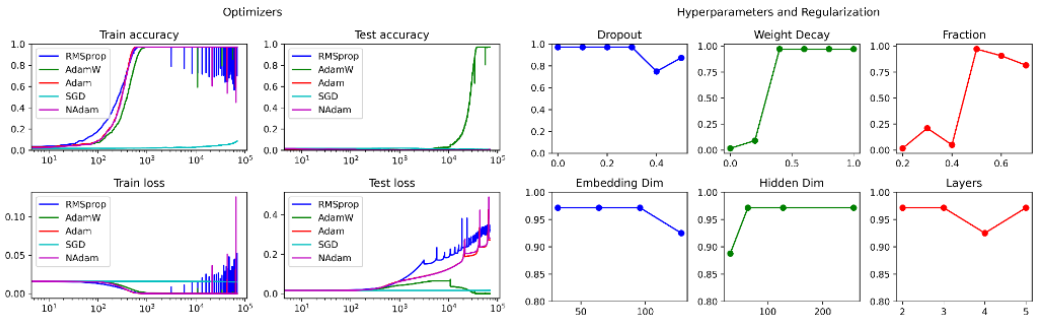


Figure 9: Effect of altering optimizers and hyperparameters in $p = 59$ MLP. Notice the y-axis range in the dimension and layers part is $[0.8, 1]$.

4 Task 4

4.1 $K \geq 2$ for Transformer and MLP

Due to time constraints, we conducted experiments at $p = 23$ with larger batch sizes for greater values of K. While there is a clear grokking phenomenon observed at $K = 2$ in both the Transformer and MLP models, the grokking gap diminishes as K increases in both architectures.

Initially, we questioned whether changes in batch size influenced the grokking gap. However, our observations indicate that this is not the case. The second column (for $K = 3$, batch size = 256) and the third column (for $K = 3$, batch size = 1024) in Figure 10 suggest that batch size is not the dominant factor; rather, it is the value of K that significantly impacts the grokking phenomenon.

4.2 $K \geq 2$ for the BiRNN Model

BiRNN can learn well without grokking when $K > 2$, but requires a smaller learning rate to keep the process stable 11. We also conducted a partial search 12. In contrast to others, BiRNN does *not* rely on weight decay to generalize (zoom to see the curves are not the same). It suggests that BiRNN matches with the symmetry and sequential structure of the algorithmic data. For example, it can achieve $> 90\%$ accuracy in < 200 epochs with large lr warm-up, but transformers cannot be as fast.

5 Task 5

Previous experiments show several irregular behaviors when a standard model learns the algorithmic datasets of two numbers' modular addition. Specifically, this phenomenon is the most obvious when the numbers added are only 2. It is also known that adaptive gradient methods with decay (AdamW)

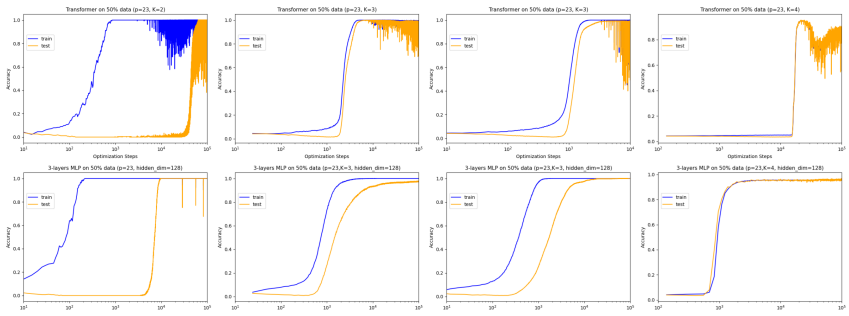


Figure 10: Transformer and MLP results with $p = 23$. The first column shows the result for $K = 2$ with batch size 64. The second and third column show the result for $K = 3$ with batch size 256 and 1024. The last column shows the result for $K = 4$ with batch size 1024.

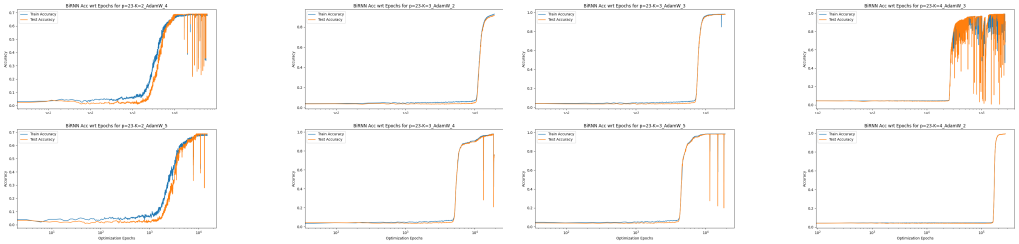


Figure 11: Exploratory BiRNN results with $p = 23$. The first column shows the performance on the $K = 2$ dataset, using 4 (upper) or 5 (lower) RNN layers; The middle part shows results using $2, 3, 4, 5$ RNN layers for $K = 3$. The right part shows $K = 4$, using 3 (upper) or 2 (lower) layers; AdamW weight decay fixed as 1; learning rate fixed 0.0005; batch sizes are $64, 256, 1024$ for $K = 2, 3, 4$.**Note that the accuracy plateau was ≈ 0.7 for $K = 2$.**

preforms better than direct SGD on this dataset 9. We present our explanations below and supporting experiments in 13.

The optimal parameter area of this dataset lies in a *sharp conic area* (see **e,f** in 13), with ridges around it. This makes it hard for optimizers with low regularization to find the area. From a natural aspect, bad generalization using basic optimizers indicate regions in the parameter space that are too steep or irregular. Neural networks often possess this feature. As for the rather low importance of dropout, it can be the result of over-parameterization. Unlike models of the real world (ImageNet...), the algorithmic dataset is poor in features, but the models we use have at least $20k$ parameters.

The steepness and heterogeneity of loss landscapes are caused by the highly non-i.i.d. structure of the modular addition dataset. If we have a uniformly distributed dataset, say MNIST, where the numbers of each class are regularly 'spread' in a cluster, the loss surface of train and test datasets should not differ largely. Thus, the embedding of the train and test set are not far away. We speculate a bad generalization process if we break the random split rule and select the train and test samples adversarially for other datasets. Results for task 4 supports the idea, where the grokking diminishes

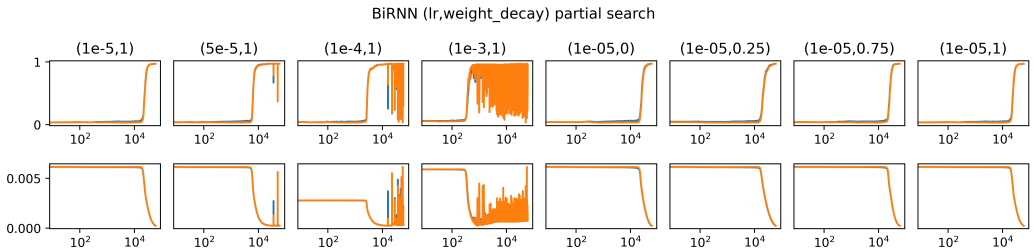


Figure 12: Example of hyperparameter search for BiRNN, $K = 3$. Using AdamW optimizer, 3 RNN layers, batch size 512, hidden dimension 64.

when we learn $K = 4$. In fact, [6] suggested the key to grokking is the input's representation, i.e. embedding. We highly agree with this idea.

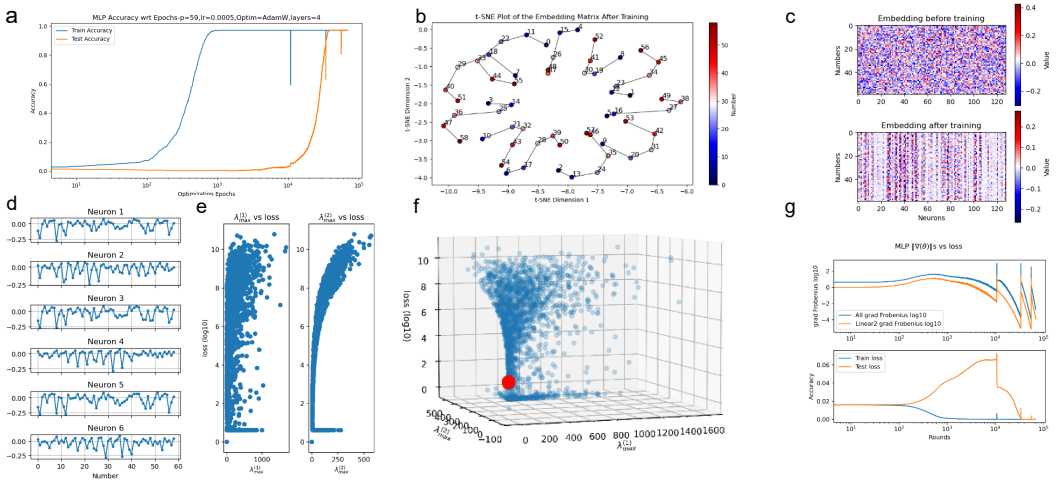


Figure 13: Loss landscape-related studies. Figure **a** presents the training process of this case. **b** is the t-SNE plot of learned embeddings. Lines link the numbers that differ by 11. **c** shows the original random and learned embedding matrices. **d** shows the cyclic and location-aware neuron weights. **e** and **f** shows the loss landscape w.r.t the largest eigenvalues of two linear layer weights, $\lambda^{(1)}_{max}, \lambda^{(2)}_{max}$ (loss is $\log 10$ scaled). **g** plots the change of $\|\nabla_\theta\|$ and $\|\nabla_{\text{linear2}}\|$ during training.

In the experiments, we focus on a typical and easy-to-train grokking case with $K = 2, P = 59, lr = 0.0005$, fraction $= 0.5$ using MLP. The structure consists of a 128 dimension embedding layer, 4 linear-and-ReLU hidden layers and an output layer. The loss function remains as cross entropy. Under these circumstances, we reproduced the cyclic structures discovered from t-SNE analysis in [8]; the embedding matrix possessed sparse structures, indicating only a portion of neurons participated in activation. This also explains the results in parts 2.2 and 3.2, where a low embedding dimension and narrow nets can still learn the dataset well.

Inspired by [3], we tried to approach the phenomenon by replicating a loss landscape analysis. First, to derive sample points in the parameter space, we used Gaussian perturbation on two hidden layers' weights. When the scaling factor increases, we can obtain a set of parameters "surrounding" the learned point, gradually getting further. Next, the projection of the parameter matrix space onto a dimension of 2 was implemented as the largest eigenvalues of the weights. In figure **f**, one can observe a steep loss surface, and some better choices missed . This supports the explanation that "an irregular loss surface dampens generalization" [3]. Next, we intend to analyze second-order gradients (curvatures). However, Hessian computation was deprecated because of limited CUDA memory on our available devices. As an approximation, **g** records the 2-norm of overall parameters and a hidden linear layer weight during training. The norm increases at the initial bad over-fitting stage, and decreases as the grokking take place. The optimal parameter area has drastically small weights, matching the studies of [6].

Future work may include the application of sharpness-aware methods, second-order method families, (quasi-newton, LM, etc.) and the limited representation ability of attention mechanisms. Though the works of Liu et al. [5, 6] provide a self-contained explanation of grokking dynamics, current studies still cannot capture the topics related to the limited generalization power of deep learning models. [2] is a study strongly related to learning arithmetic equations using the transformer. Last year, there has been progress on using transformers to replicate classic learning algorithms like nearest neighbors [4]. More recently, the limited generalization of multi-layer transformers have been shown in pure theoretical contexts [1]. Deep learning architectures, especially attention mechanisms, will continue to draw the community's interest and we hope that future research will shed light on its surrounding mysteries.

References

[1] Lijie Chen, Binghui Peng, and Hongxun Wu. Theoretical limitations of multi-layer transformer, 2024.

[2] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[3] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[4] Zihao Li, Yuan Cao, Cheng Gao, Yihan He, Han Liu, Jason M. Klusowski, Jianqing Fan, and Mengdi Wang. One-layer transformer provably learns one-nearest neighbor in context, 2024.

[5] Ziming Liu, Ouail Kitouni, Niklas S Nolte, Eric Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning. *Advances in Neural Information Processing Systems*, 35:34651–34663, 2022.

[6] Ziming Liu, Eric J Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *The Eleventh International Conference on Learning Representations*, 2022.

[7] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[8] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022.