UNDER REVIEW

RESCIENCE C

Replication / ML Reproducibility Challenge 2022

Reproducibility of "Towards Understanding Grokking"

Anonymous[1, ID]

[1]Anonymous Institution

Edited by
(Editor)

Reviewed by
(Reviewer 1)
(Reviewer 2)

Received
03 February 2023

Published
–

DOI
–

Reproducibility Summary

Scope of Reproducibility – In this work, we attempt to reproduce the results of the NeurIPS 2022 paper "Towards Understanding Grokking: An Effective Theory of Representation Learning" [1]. This study shows that the training process can happen in four regimes: memorization, grokking, comprehension and confusion. We first try to reproduce the results on the toy example described in the paper and then switch to the MNIST dataset. Additionally, we investigate the consistency of phases depending on data and weight initialization and propose smooth phase diagrams for better visual perception.

Methodology – There is no open-source code for the paper. Therefore, we re-implemented all described experiments by ourselves. We also used the code provided by the authors to validate training hyperparameters not stated in the paper. As for the computational resources, we spent around 30 CPU and 125 GPU hours on the NVIDIA V100 GPU.

Results – We succeeded in reproducing phase diagrams for the toy example. For the MNIST dataset, we observed a behavior similar to the one from the paper. We used a wider range of hyperparameters leading us to an extra area with the memorization phase. We also argue that the original memorization phase is even more delayed grokking. Therefore, the authors' findings about the MNIST phases are incomplete.

What was easy – After receiving additional instructions from the authors about the details not mentioned in the paper, the reproduction of all results was easy because the authors put significant work into the setup description. Moreover, it was easy to suggest new experiments, as they followed logically from the previous.

What was difficult – Generally, it was difficult to reproduce the results because some critical hyperparameters (the activation function for the toy model and the batch size for MNIST) were not stated in the paper. Considering MNIST, it took too much time to iterate over all hyperparameters' values, as grokking requires about 100k training iterations, which is approximately 30 minutes on the V100 GPU.

Communication with original authors – We contacted the authors two times through e-mail and asked for the validation of the setup. They responded quickly and were very helpful. The authors provided us with the code for the toy example and the MNIST dataset. We did not execute it for our experiments but used it for checking the training details and hyperparameters.

Copyright © 2023 Anonymous, released under a Creative Commons Attribution 4.0 International license.
Correspondence should be addressed to ()
The authors have declared that no competing interests exists.
Code is available at https://github.com/rescience-c/template.

1 Introduction

Grokking is a phenomenon in neural network training when generalization happens well past the point of overfitting, which is contrary to the common intuition of machine learning. It was firstly discovered by [2] and until now was only observed on toy examples.
In this work, we attempt to reproduce the findings of the paper "Towards Understanding Grokking: An Effective Theory of Representation Learning" [1], in which the authors succeeded in capturing grokking on the MNIST dataset constructing a specific setup for this purpose. They also visualize the division of the learning process into four phases by displaying phase diagrams from a grid search of hyperparameters.

2 Scope of reproducibility

The authors consider grokking as one of four learning phases. *Grokking* and *comprehension* lead to generalization, though the former has delayed generalization. *Memorization* is a synonym for overfitting when the model performs well on the training set but can not generalize to the validation set. Finally, *confusion* is an indicator of a training failure when a model can not even memorize the training set. One of the main contributions of the paper is an attempt to describe grokking from the theoretical point of view while we focus on its experimental part. Therefore, we study the following claims from the original paper:

- Claim 1: The model training may happen in four phases depending on the training hyperparameters.
- Claim 2: Grokking is an intermediate phase between comprehension and memorization, being a witness to poorly tuned hyperparameters.
- Claim 3: Grokking is a more general phenomenon which is not restricted to toy algorithmic datasets.

3 Methodology

We conduct our experiments using the PyTorch framework [3]. We re-implemented both the toy setup and the MNIST experiments ourselves.

3.1 Datasets

Toy model — Following the authors, we consider a toy dataset consisting of integer pairs (i, j), where $i, j \in \{0, \ldots, p-1\}$. The target is defined as $i+j$, i. e. non-modular addition of the two features. There are $p(p-1)/2$ unique (i, j) pairs (the pairs $i+j$ and $j+i$ are considered the same). We set $p = 10$ with a total of 55 pairs and use a $45/10$ random train-validation split.

MNIST — The authors use the default version of the MNIST dataset [4], but for training, they randomly sample 1k images out of a possible 60k. The validation set is kept unchanged.

3.2 Model descriptions

Toy model — The toy model consists of two parts: an embedding encoder and an MLP decoder (denoted as Dec). Input integers (i, j) are interpreted as characters and mapped

to 1-D embeddings $\mathbf{E}_i, \mathbf{E}_j$. The model output is $\text{Dec}(\mathbf{E}_i + \mathbf{E}_j)$. Generally, we solve a classification task predicting one of $2p - 1$ possible sums (we call them classes for clarity). Following the authors, we train the model in two setups: classification and regression. The former trivially optimizes cross-entropy loss, while for the latter we need to generate $2p - 1$ fixed random gaussian vectors serving as targets, one for each class, and minimize mean squared error (MSE) between the model output and the target vector for the ground truth class. We set the size of these target vectors equal to 30. The final prediction is a class with the highest probability in classification and a class with a target vector nearest to the model output in regression. The decoder is a 3-layer MLP with a Tanh activation, with $1 - 200 - 200 - 19$ dimensions for classification (as $2p - 1 = 19$ outputs for $p = 10$) and $1 - 200 - 200 - 30$ dimensions for regression. The embeddings are initialized from a uniform distribution $U[-\frac{1}{2}, \frac{1}{2}]$, while the default PyTorch initialization is used for the linear layers in MLP.

MNIST – The authors use the same 3-layer MLP with the hidden dimension of 200 but a ReLU activation. The input image is flattened before feeding to the model, so the overall dimensions are $784 - 200 - 200 - 10$. As the whole theory of grokking in the paper is based on the encoder-decoder idea, the authors call the first two layers an "encoder" and the last one – a "decoder". They initialize the model weights with Kaiming uniform initialization [5] and then multiply them by 9, which means that initialization has an increased scale. Instead of cross-entropy loss, MSE loss with one-hot targets is used. The authors find that it helps the network to fit fast to the training set and much later to the validation set.

3.3 Hyperparameters

Toy model – We train the toy model with AdamW [6] for 10^5 iterations as proposed by the authors. The learning rate $\eta_{\text{emb}} = 10^{-3}$ and weight decay $\lambda_{\text{emb}} = 0$ of embeddings encoder remain fixed, while we sweep over decoder hyperparameters searching for different learning phases. The ranges are taken from the original paper and make up $[10^{-5}, 10^{-2}]$ for the learning rate (logarithmic), $[0, 20]$ for the weight decay in classification, and $[0, 10]$ for the weight decay in regression (both weight decay ranges are linear). The batch size equals 45, meaning that a full sample gradient is calculated at each iteration. The train-validation split seed is fixed among different runs, as well as the model initialization seed.

MNIST – Similar to the toy model, the authors use AdamW optimizer but fix the learning rate $\eta_{\text{enc}} = 10^{-3}$ only for the encoder part of MLP. Weight decay changes for the whole model during sweeping in a logarithmic range of $[10^{-5}, 10^{1}]$, and the learning rate does only for the decoder in a logarithmic range of $[10^{-6}, 10^{0}]$. According to the paper, the model is trained for 10^5 iterations with a batch size equal to 200. Both the data generation seed and the model initialization seed are kept the same among different runs.

3.4 Experimental setup and code

For each pair of learning rate and weight decay, we train a model for 10^5 iterations, as it was proposed by the authors. Then we classify the result of training as one of the four phases. To do so, we count the number of steps to reach 90% accuracy on train and validation sets, T_{train} and T_{val} respectively, for the toy example. For the MNIST dataset, the threshold is 60% accuracy. Then the authors define the phases as in Table 1.

Phase	criteria		
	$T_{train} \leq 10^5$	$T_{val} \leq 10^5$	$T_{val} - T_{train} < 10^3$
Comprehension	+	+	+
Grokking	+	+	–
Memorization	+	–	Not Applicable
Confusion	–	–	Not Applicable

Table 1. The definition of training phases.

3.5 Computational requirements

We carried out the toy model experiments on 16 Intel Xeon Gold 6240R CPU cores in parallel because we were limited in computational resources. We found CPU computations faster than running on a single NVIDIA V100. Every phase diagram has $21 \cdot 21 = 441$ pixels, each representing a separate launch (i. e. a unique pair of weight decay and learning rate values). Full training for 10^5 iterations takes about 5 minutes on a single CPU core, so the budget for a single diagram constitutes $441 \cdot 5 \div 16 \div 60 \approx 2.5$ CPU hours.
In fact, we used an early stopping after both training and validation accuracy had reached a threshold of 90%, so the actual budget can be reduced to a maximum of 5 times depending on the amount of "comprehension" phases on the diagram.
For the MNIST dataset experiments, we used a single NVIDIA V100 GPU. The full training of a 3-layer MLP took approximately 30 minutes. To draw a phase diagram, we have run $14 \cdot 15 = 210$ training processes. Therefore, the total time consumption should have been about $210 \cdot 30 \div 60 = 105$ hours. Using an early stopping with an accuracy threshold of 60% allowed us to reduce time costs twice, leading to approximately 50 GPU hours.
Experiments with the increased number of maximum iterations took proportionally more time for both the toy model and MNIST. Overall, all experiments took roughly 30 CPU and 125 GPU hours.

4 Results

Generally, we succeeded in reproducing the phase diagrams from the original paper (for both the toy model and MNIST). All three claims proposed by the authors agree with our experiments. Our contribution can be summarized as follows:

1. We reproduced the toy model (Addition classification and Addition regression) and obtained the diagrams with all 4 phases present, supporting Claim 1. Besides, we study how phase diagrams change for different random seeds and decoder activation functions in Appendix A, B.

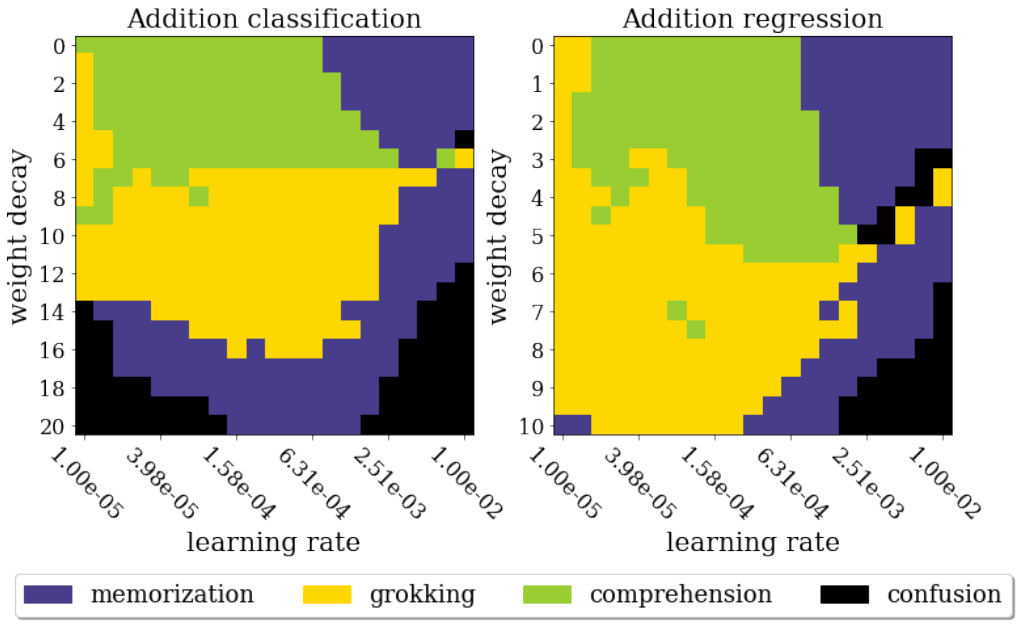
2. We reproduced the MNIST experiments minding the authors' code. We observed all 4 phases, contributing to Claims 1, 3.

3. The Claim 2 describing the intermediate position of grokking also holds in both experimental setups.

4. We set up additional experiments described in Section 4.2, which show smooth transitions between phases, complementing Claim 1. We also doubt the existence of memorization on MNIST, illustrating that this phase is even more delayed grokking.

4.1 Results reproducing original paper

Figure 1. Learning phases over decoder learning rate and weight decay for the toy model classification (left) and regression (right) setups.



Toy model — The phase diagrams for both classification and regression setups are shown in Figure 1. The same diagrams plotted by the authors can be found in Section 4.1 of the original paper (Figure 6). Clearly, all 4 phases are presented in both diagrams, supporting the Claim 1. The phases without generalization (memorization and confusion) occur for the large values of decoder learning rate (in both setups) or for the large values of decoder weight decay (classification setup). The rest values of hyperparameters enable generalization, leading to comprehension (mainly for smaller values of weight decay) or grokking (for larger values of weight decay). The area of grokking in both setups is located between comprehension and memorization, agreeing with the Claim 2. However, the phase borders strongly depend on a data seed and slightly on a model seed, which we discuss in Appendix A.

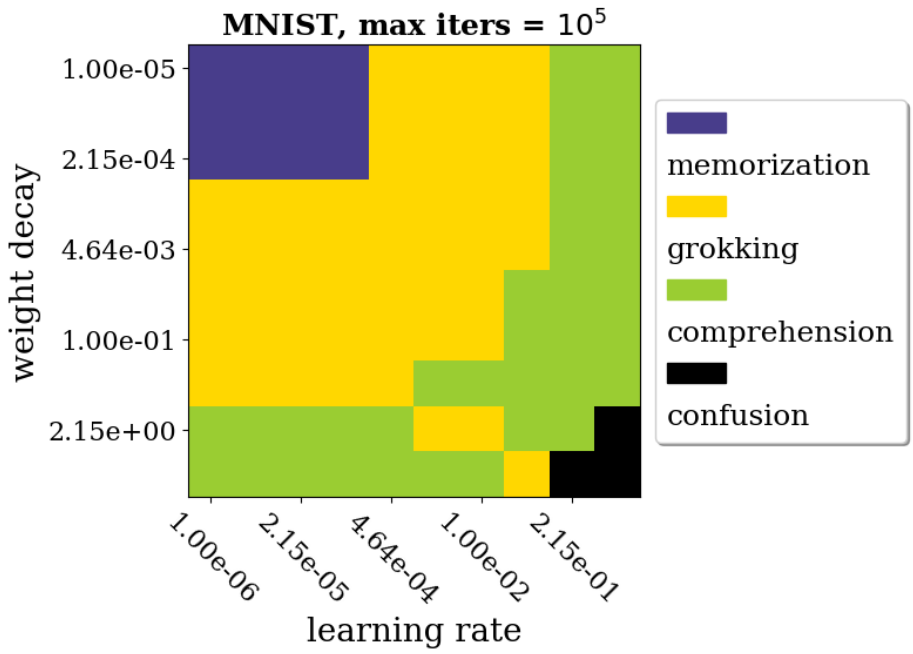
MNIST — In Figure 2, we show the reproduced phase diagram of learning dynamics on MNIST. The corresponding phase diagram from authors' experiments can be seen in Section 4.3 of the original paper (Figure 8). In our diagram we have all 4 phases in the same order as in the diagram from the paper, with grokking sandwiched between memorization and comprehension, which proves all three claims from Section 2. However, the area of confusion phase is much smaller in our results. In Section 4.2, we extend the boundaries of hyperparameters and get a more similar phase diagram. In the same Section 4.2, we argue that, in fact there is no memorization phase for MNIST, so grokking does not necessary stay between comprehension and memorization, which indicate an inaccuracy in Claim 2, but it does not brake any intuition about grokking.

4.2 Results beyond original paper

As a further investigation of grokking in the scope of the proposed setup, we ask two questions:

1. Do we need to discretize phases?
2. Does memorization exist or is it even more delayed grokking?

Figure 2. Phase diagram for MNIST.



Question 1 – By asking the first question, we argue that the boundaries of phases are blurred and cannot be set by some threshold. In order to prove that, we construct smooth phase diagrams using the following procedure. We train models using the same setup, but for each model we save the optimization steps at which the model has reached the desired train and validation accuracy, T_{train} and T_{val}, respectively. Then we split the diagram in two parts. The first part corresponds to comprehension and grokking, which both generalize to the validation set. For this part we draw the difference between validation and train steps $T_{val} - T_{train}$. The second part corresponds to memorization and confusion. As at these phases the model does not reach the desired validation accuracy. We draw the number of train steps T_{train} for memorization and the maximum number of optimization steps for confusion.

Question 2 – To answer the second question, we train the toy model for $3 \cdot 10^5$ iterations (3x from the original training budget) and the MNIST model for $1.5 \cdot 10^5$ iterations (1.5x from the original training budget). We plot the smooth phase diagrams described in the previous paragraph and compare them to our reproduction of the original diagrams.

Results (Q1) – We have conducted such experiments for both toy model setups and the MNIST dataset. The resulting diagrams (Figures 3, 4, 5) show that for the toy model, there is *no implicit gap* between comprehension and grokking, as well as between memorization and confusion, while for the MNIST dataset, there is *a noticeable gap* between all four phases.

Results (Q2) – Considering the toy model (Figures 3, 4), we observe that after increasing the number of training iterations, the overall phase diagram is preserved. A few pixels on the boundary of memorization and grokking have converted from the former to the latter phase. However, a vast memorization area is still present in the diagram, meaning that this phase actually exists in the toy setup (i. e. memorization *is not more delayed grokking* but rather a separate phase).

Figure 3. Discrete (left) and smooth (right) phase diagrams for the Addition classification toy model. The discrete diagram has 10^5 max iterations of training, while the smooth diagram is extended to $3 \cdot 10^5$ training iterations.

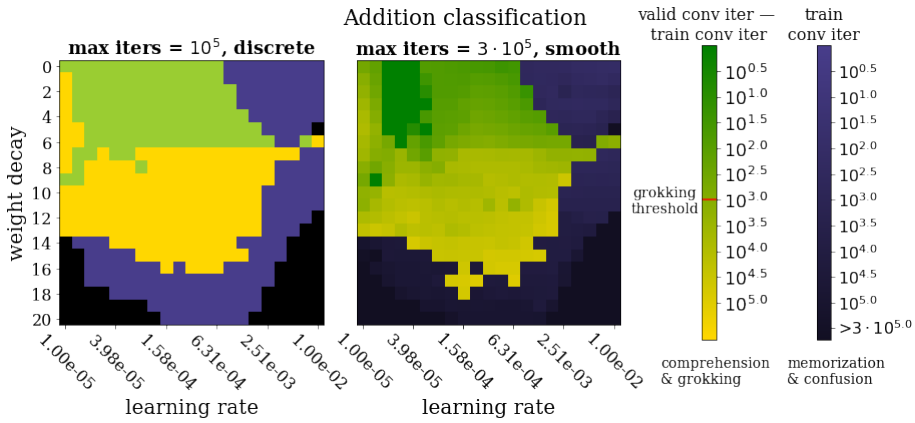
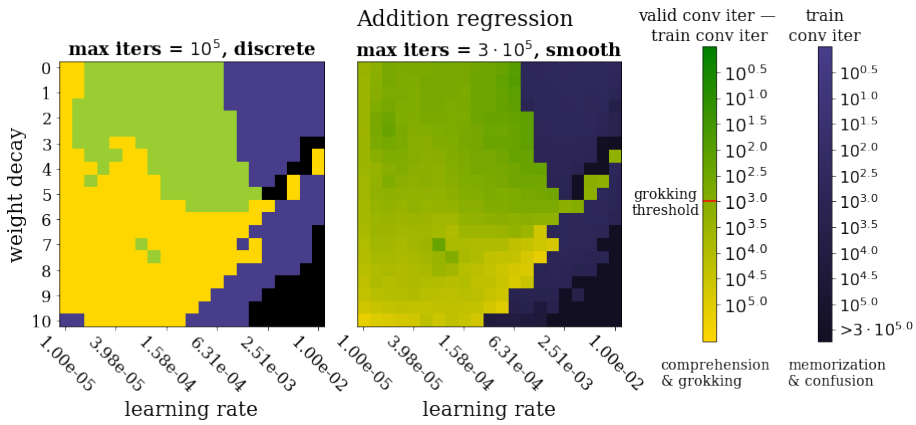


Figure 4. Discrete (left) and smooth (right) phase diagrams for the Addition regression toy model. The discrete diagram has 10^5 max iterations of training, while the smooth diagram is extended to $3 \cdot 10^5$ training iterations.



The results for the MNIST dataset are inpicted on Figure 5 and guide us to the following observations. First, memorization in the upper left corner *is actually an even more delayed grokking* because training the model for more additional steps has led to convergence on the validation set. Second, there is a notable gap in convergence steps on the validation set between grokking and comprehension phases. Therefore, the phases are indeed well-separated. Third, we obtained a new unexpected memorization phase in the upper right corner. We investigated how train and validation accuracy change over the optimization process for this area of hyperparameters. The results can be seen in Figure 6 (left). It seems like the model falls into an unstable local minimum and then gets kicked out of it during further optimization steps because of large learning rate. Hence, while such behaviour is memorization by definition, in fact, it is not what we used to call by that name. We argue that this configuration is closer to confusion, as we finish iterations with a model having a random accuracy on both training and validation sets, in spite of visiting a local minimum during the optimization.

Looking at Figure 5, one might think that every pair of hyperparameters at the comprehension phase is eligible, as they lead to a low difference between validation and training convergence iterations. However, it is critical to keep in mind the convergence iteration on the training set alone (i. e. in terms of iterations, it is better to have

Figure 5. Discrete (left) and smooth (right) phase diagrams for the MNIST dataset. We highlight the range of hyperparameters from the original paper with a red rectangle. The discrete diagram has 10^5 max iterations of training, while the smooth diagram is extended to $1.5 \cdot 10^5$ training iterations.

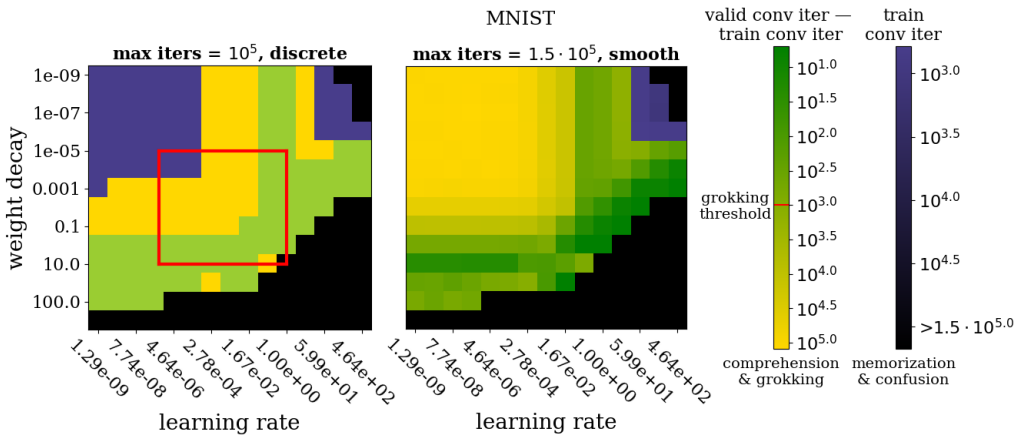
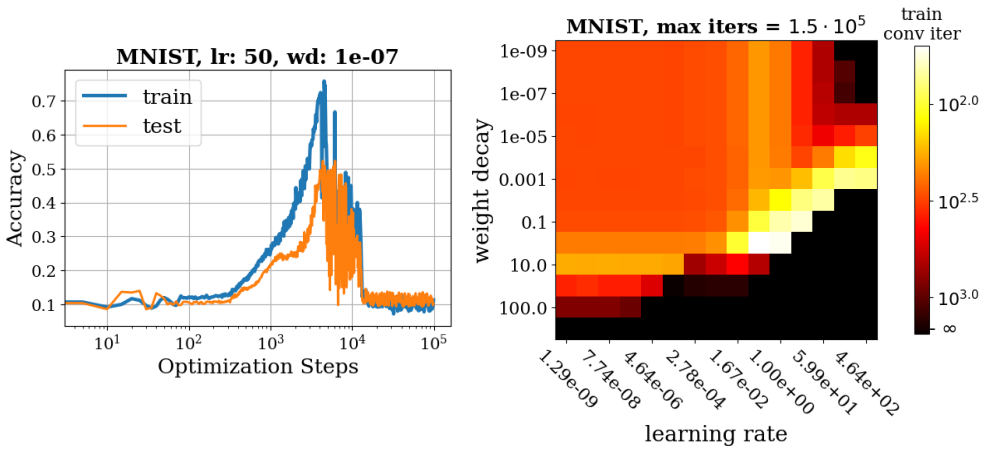


Figure 6. Train and validation accuracy during the optimization process with the learning rate equal to 50 and the weight decay equal to 10^{-7} (left). Convergence iterations for different pairs of weight decay and learning rate on the MNIST *training* set (right).



$T_{train} = 1000, T_{val} = 5000$ rather than $T_{train} = 50000, T_{val} = 50001$). It follows from Figure 6 (right) that the convergence is fast only for a small portion of hyperparameters, and, surprisingly, this area lies close to the border of confusion.

5 Discussion

As discussed in prior works [7, 8], poor model generalization (i. e. memorization) is related to localization in a sharp local minimum. Then, due to a small shift between train and validation datasets, the model reaches high accuracy on the training dataset and nearly random accuracy on the validation dataset. Thus, we argue that for more complex datasets (compared to the toy examples), which have a negligible distribution shift, it is almost impossible to fall into such sharp narrow local minima, so memorization is not observed in practice.

5.1 What was easy

The authors did an excellent job describing all the setups for each experiment. However, some crucial details were missing, such as the activation function for the toy model and the batch size for the MNIST dataset. We contacted the authors and asked for these values. After that, it was easy to reproduce the results.

5.2 What was difficult

Reproduction and additional research on the toy model were difficult because the authors did not specify the activation function for the MLP, and we assumed it to be *ReLU*, as for the MNIST model. In practice, it turned out to be *Tanh*, which completely changed the training phases. The impact of an activation function is described in Appendix B. Also, the resulting phase diagrams strongly depend on the data generation seed, which is shown in Appendix A. We spent significant time trying to achieve the authors' results before discovering that this was the source of the problem.

5.3 Communication with original authors

We have contacted the authors twice: the first time to ask for the MNIST dataset code and the second time to ask about the toy model, as we thought we misinterpreted the setup. In both times, the authors responded very quickly and provided all the information we needed to fill in the blanks, which helped us a lot. We thank the authors for their concern and involvement.

References

1. Z. Liu, O. Kitouni, N. Nolte, E. J. Michaud, M. Tegmark, and M. Williams. "Towards Understanding Grokking: An Effective Theory of Representation Learning." In: **Advances in Neural Information Processing Systems**. Ed. by A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho. 2022.
2. A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra. "Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets." In: **CoRR** abs/2201.02177 (2022). arXiv: 2201.02177.
3. A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. "Automatic differentiation in PyTorch." In: **NIPS-W**. 2017.
4. Y. LeCun, C. Cortes, and C. J. Burges. **The MNIST database of handwritten digits**.
5. K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: **2015 IEEE International Conference on Computer Vision (ICCV)**. 2015, pp. 1026–1034.
6. I. Loshchilov and F. Hutter. "Decoupled Weight Decay Regularization." In: **International Conference on Learning Representations**. 2019.
7. N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima." In: **International Conference on Learning Representations**. 2017.
8. P. Chaudhari, A. Choromanska, S. Soatto, Y. LeCun, C. Baldassi, C. Borgs, J. Chayes, L. Sagun, and R. Zecchina. "Entropy-SGD: Biasing Gradient Descent Into Wide Valleys." In: **International Conference on Learning Representations**. 2017.

Appendix

A. Phase diagrams for different random seeds

In this section, we study the importance of random seeds for reproducing phase diagrams from the paper for the toy model. We sweep over 3 data splitting random seeds (**data seed**) and 3 model initialization random seeds (**model seed**). The results are presented in Figures 7, 8. Turns out that the presence and proportion of the 4 phases is

heavily affected by the data seed. For some setups it becomes impossible to observe grokking for any considered set of hyperparameters (Addition regression, data seed = 43, model seed = 101, 102), while other setups make confusion much less common (Addition classification, data seed = 41). The impact of model seed is much smaller but still considerable. Variation of model initialization changes the shape of phase borders, while the localization of the phases is mainly preserved.

Figure 7. Addition classification phase diagrams for different pairs of data seed and model seed.

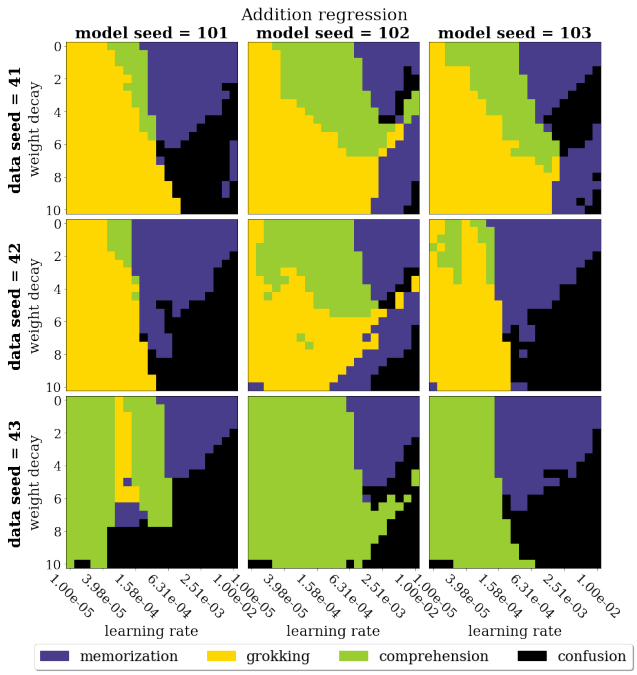
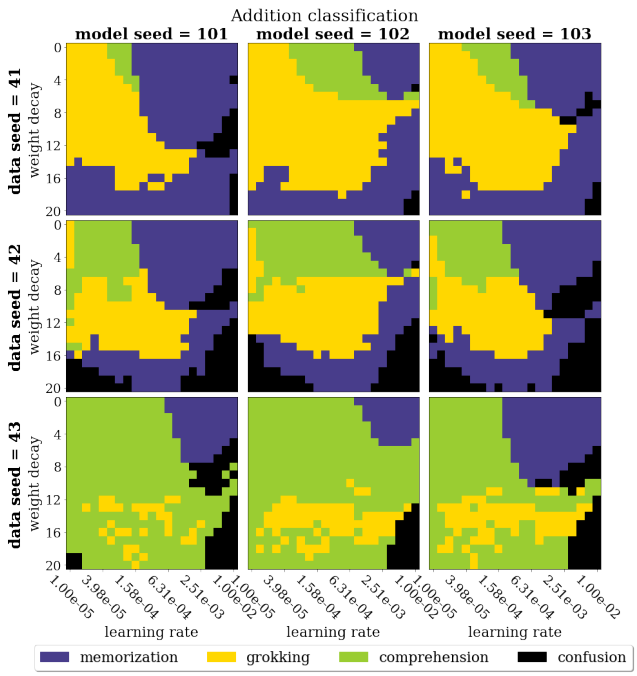


Figure 8. Addition regression phase diagrams for different pairs of data seed and model seed.



B. Phase diagrams for different activation functions

Initially, we failed to reproduce the toy model because we used an activation function (ReLU) different from the one used by the authors (Tanh). After all, it is not specified in the paper. Inspired by this mismatch, we plot the phase diagrams for different MLP activations with fixed data and model seeds. We consider **ReLU**, **LeakyReLU** with negative slope = 0.1 and **Tanh**. The resulting diagrams are shown in Figures 9, 10. Piecewise linear functions (ReLU and LeakyReLU) significantly decrease the area of generalization, turning large values of weight decay into confusion. ReLU function makes the comprehension phase impossible in the Addition regression setup. At first, we hypothesized that the problem of ReLU is its non-injectivity. However, LeakyReLU is injective, but large areas of confusion are still present in the diagrams. So, we leave the question: "What is wrong with these activations?" for future studies. But clearly, the use of Tanh is crucial to reproducing the diagrams from the original paper.

Figure 9. Addition classification phase diagrams for different MLP activation functions.

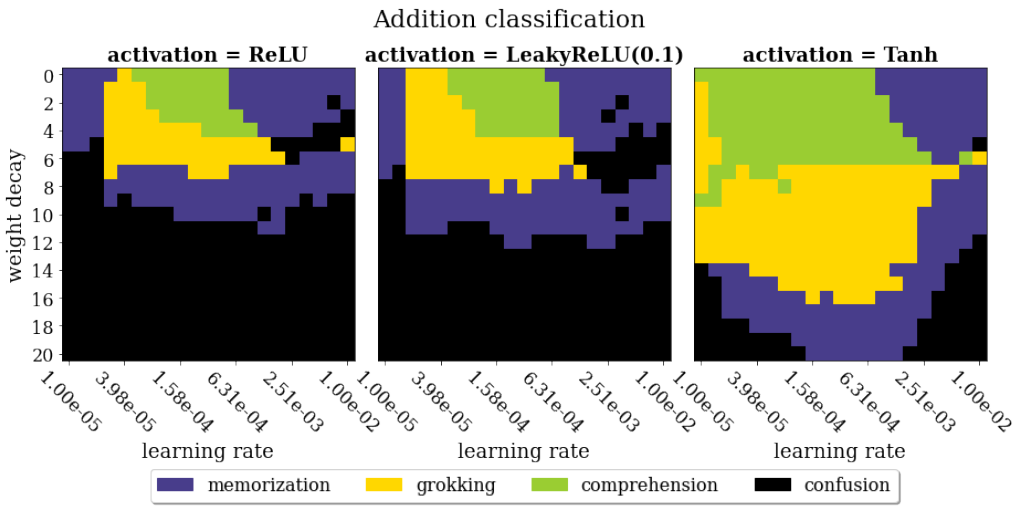
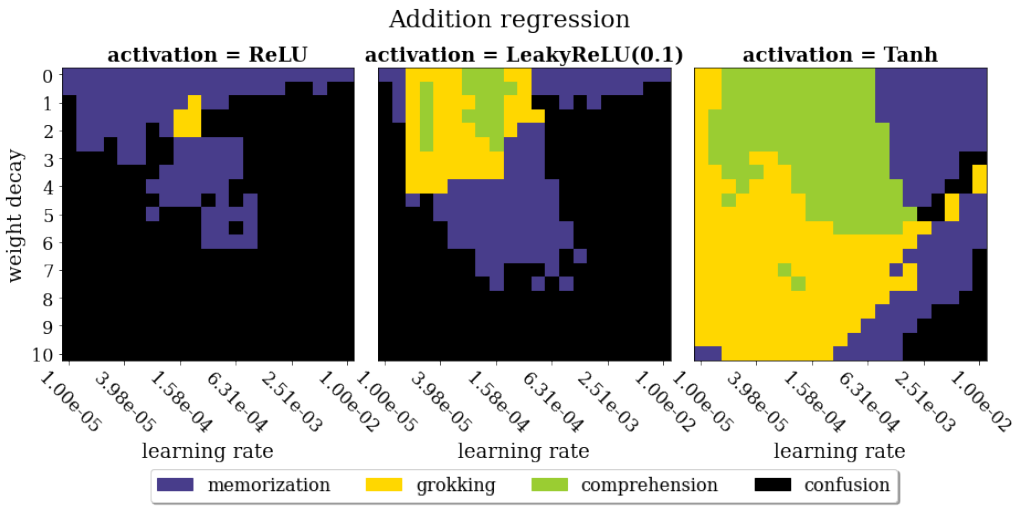


Figure 10. Addition regression phase diagrams for different MLP activation functions.



C. Embedding optimization trajectories

Finally, we compare the training phases by optimization trajectories of embeddings. This experiment is inspired by Figure 4 from the original paper. We plot the embedding trajectories for different phases of the Addition regression setup. Figure 11 shows the disentanglement of embeddings for comprehension, grokking, and confusion. Memorization is the only phase, which does not have embeddings sorted after 2000 iterations. Interestingly, we do not observe any difference between *grokking* and *confusion*. The discrepancy between these two phases is revealed during further training (Figure 12) when the magnitude of embeddings starts growing faster for *confusion*. It happens because large values of learning rate and weight decay, which correspond to confusion (see Figure 1, right), make it impossible for the decoder to fit, even though the embeddings evolved to the correct order. Note also that the magnitude of converged *grokking* embeddings is remarkably larger than the one of the *comprehension* embeddings. Thus, the magnitude of embeddings *may be a witness* to different training phases.

Figure 11. Optimization trajectories of embeddings over first 2000 training iterations.

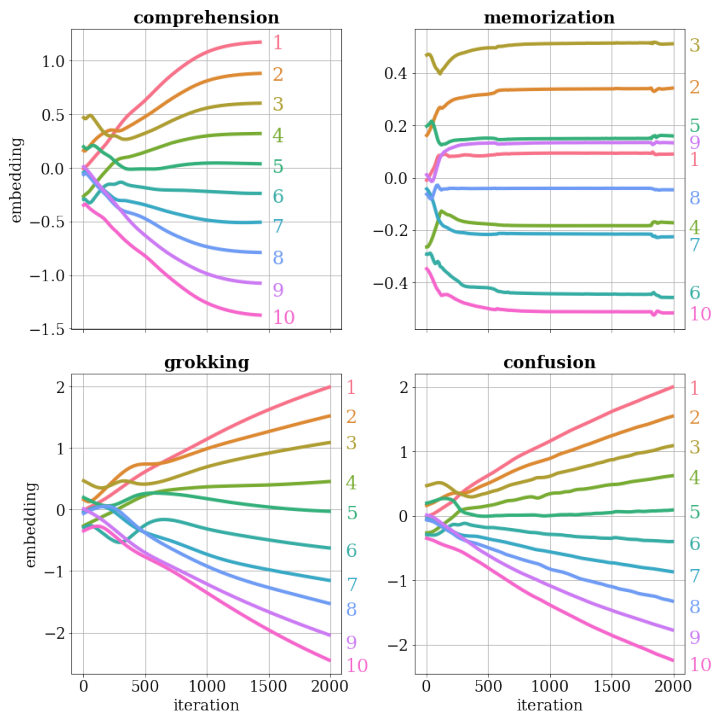
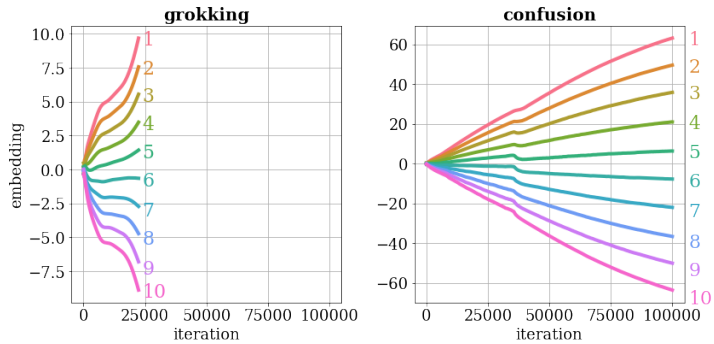



Figure 12. Optimization trajectories of embeddings over whole training process.