THE GROKKING FORMULA: WHY MOMENTUM AND ADAPTIVE LEARNING RATES MUST DANCE TOGETHER

Anonymous authors
Paper under double-blind review

ABSTRACT

The surprising phenomenon of grokking, where neural networks suddenly achieve perfect generalization after prolonged training despite initially appearing to stagnate, challenges our understanding of deep learning optimization. While prior work has studied grokking's dependence on architecture and tasks, the critical role of optimization components remains unexplored. Through systematic experiments comparing AdamW (with both momentum and adaptive learning rates) against SGD variants on modular arithmetic and permutation tasks, we discover that grokking emerges only when both momentum ($\beta_1 = 0.9$) and adaptive rates ($\beta_2 = 0.99$) are present—AdamW achieves 100% validation accuracy on arithmetic tasks (addition in $2{,}410 \pm 0$ steps, division/subtraction in $4{,}273 \pm 0$ steps) while SGD, SGD+momentum, and RMSprop all fail completely ($1.0 \pm 0.3\%$ accuracy). This reveals grokking's fundamental dependence on the interaction between momentum's ability to traverse flat loss regions and adaptive rates' parameter-specific scaling, while also showing the phenomenon is task-dependent (occurring only for arithmetic operations). Our results provide the first evidence that specific optimization dynamics are necessary for grokking, offering new insights into neural network training and generalization.

1 INTRODUCTION

The phenomenon of grokking—where neural networks suddenly achieve perfect generalization after prolonged training despite initially appearing to stagnate—challenges fundamental assumptions about deep learning optimization (Power et al., 2022). While traditional theory suggests a speed-accuracy tradeoff (Goodfellow et al., 2016), grokking shows networks can dramatically improve generalization long after training loss has converged. Understanding this phenomenon could provide new insights into neural network training dynamics and generalization.

However, while prior work has studied grokking's dependence on architecture and tasks (Power et al., 2022), the critical role of optimization components remains unexplored. This is particularly surprising given that:

- Modern optimizers like AdamW combine momentum ($\beta_1 = 0.9$) and adaptive learning rates ($\beta_2 = 0.99$) (Kingma & Ba, 2014; Loshchilov & Hutter, 2017)
- These components have distinct theoretical properties (Nesterov, 1983; Ge et al., 2015)
- Their interaction may be crucial for grokking's delayed generalization

We conduct the first systematic investigation of optimizer components in grokking through controlled experiments on modular arithmetic (mod 97) and permutation ($k = 5$) tasks. Our key findings are:

- **Component Interdependence**: Only AdamW achieves grokking (100% validation accuracy), while:
  - SGD ($1.0 \pm 0.3\%$ accuracy)
  - SGD+Momentum ($1.0 \pm 0.3\%$)
  - RMSprop ($1.0 \pm 0.3\%$) all fail
- **Task Dependence**: Grokking occurs only for arithmetic operations:

  - Addition ($2{,}410 \pm 0$ steps)
  - Subtraction ($4{,}277 \pm 0$ steps)
  - Division ($4{,}273 \pm 0$ steps)
- **Training Dynamics**: AdamW shows:
  - Rapid training loss minimization (0.005 ± 0.000)
  - Characteristic U-shaped validation curve

These results demonstrate that grokking requires the synergistic interaction between momentum's ability to traverse flat loss regions and adaptive rates' parameter-specific scaling. Our work provides the first evidence that specific optimization dynamics are necessary for grokking, with implications for both theoretical understanding and practical applications of deep learning.

2 Related Work

Prior work on grokking has focused primarily on architectural and task properties (Power et al., 2022), while largely overlooking the role of optimization. Our work provides the first systematic analysis of how optimizer components enable grokking, complementing these architectural studies.

Several approaches have analyzed optimization components in isolation:

- Momentum's role in escaping saddle points (Ge et al., 2015) and accelerating convergence (Nesterov, 1983)
- Adaptive learning rates for parameter-specific scaling (Kingma & Ba, 2014)
- Their combination in Adam-style optimizers (Loshchilov & Hutter, 2017)

However, these studies examined general optimization properties rather than grokking specifically. Our experiments reveal that neither component alone suffices for grokking—only their combination in AdamW enables the sudden generalization phenomenon. This contrasts with traditional optimization-generalization tradeoffs (Goodfellow et al., 2016) and recent work on optimization trajectories (Jastrzebski et al., 2020), showing grokking requires a unique interaction between optimization dynamics and task structure.

Notably, while Xie et al. (2020) studied momentum and adaptive rates separately, they did not examine their role in delayed generalization. Our work bridges this gap by showing how their interaction enables grokking's characteristic training dynamics.

3 Background

Modern deep learning optimization combines two key components (Kingma & Ba, 2014; Loshchilov & Hutter, 2017):

- **Momentum** (β_1): Accelerates optimization through flat regions and helps escape saddle points (Nesterov, 1983; Ge et al., 2015)
- **Adaptive learning rates** (β_2): Scale updates parameter-wise based on gradient statistics

The phenomenon of grokking (Power et al., 2022) challenges traditional optimization-generalization tradeoffs (Goodfellow et al., 2016). In grokking, networks suddenly achieve perfect generalization after prolonged training despite initially appearing to stagnate. This suggests the existence of optimization pathways where networks traverse flat loss regions before discovering generalizing solutions.

3.1 Problem Setting

We study learning algebraic operations over finite groups. Given operation $\circ$ and elements a, b from finite sets G_1, G_2, the task is to learn $f(a, b) = a \circ b$. Our experiments focus on:

- Modular arithmetic: $+$, $-$, $\div$ (mod 97)

- Permutation composition ($k = 5$)

The model minimizes:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(a,b)\sim\mathcal{D}}[\log p_\theta(a \circ b|a, b)] \tag{1}$$

with θ as parameters and $\mathcal{D}$ the data distribution (50% train/val split). Optimization follows:

$$\theta_{t+1} = \theta_t - \eta_t g(\theta_t) \tag{2}$$

where $\eta_t = 10^{-3}$ is the learning rate and $g(\theta_t)$ implements the optimizer's update rule. We systematically vary $g(\theta_t)$ while fixing other factors.

4 METHOD

Our method systematically isolates the effects of momentum and adaptive learning rates in grokking by comparing four optimizer variants while holding all other factors constant. Building on the problem setting from Section 3, we:

4.1 MODEL ARCHITECTURE

Implement a transformer (Vaswani et al., 2017) with:

- 2 decoder layers (4 heads, 128-dim embeddings)
- Layer normalization (Ba et al., 2016) after each sub-layer
- Fixed sequence length 5 (equations "a ∘ b = c")

4.2 OPTIMIZER COMPONENTS

Vary only the optimizer's $g(\theta_t)$ in Eq. 2:

- **AdamW**: Full combination ($\beta_1 = 0.9$, $\beta_2 = 0.99$)
- **SGD+Momentum**: Only $\beta_1 = 0.9$
- **RMSprop**: Only $\alpha = 0.99$ adaptive rates
- **SGD**: Baseline (neither component)

4.3 TRAINING PROTOCOL

Fix for all experiments:

- Learning rate: 10^{-3} (50-step warmup)
- Batch size: 512
- Weight decay: 0.5
- Training steps: 7,500

We track:

- Training/validation loss and accuracy
- Steps to 99% validation accuracy
- Final performance at step 7,500

This design ensures any differences in grokking behavior stem solely from optimizer component variations, enabling clear attribution of effects.

5 EXPERIMENTAL SETUP

Our experimental design systematically isolates the effects of momentum and adaptive learning rates while controlling all other factors. We implement this through:

5.1 Task Implementation

Four algorithmic tasks from experiment.py:

- Modular arithmetic (mod 97):
  - Division: $x \div y \bmod 97$ ($x \in [0, 96]$, $y \in [1, 96]$)
  - Subtraction: $x - y \bmod 97$ ($x, y \in [0, 96]$)
  - Addition: $x + y \bmod 97$ ($x, y \in [0, 96]$)
- Permutation composition ($k = 5$)

Each uses 50% train/validation splits of equation sequences (length 5).

5.2 Model Implementation

Identical architecture across all experiments:

- 2-layer transformer (Vaswani et al., 2017)
- 128-dim embeddings, 4 attention heads
- LayerNorm (Ba et al., 2016) and learned positional embeddings

5.3 Training Configuration

Fixed hyperparameters:

- Learning rate: 10^{-3} (50-step warmup)
- Batch size: 512
- Weight decay: 0.5
- Training steps: 7,500
- Evaluation: Every 10 batches
- Random seeds: 3 per condition

5.4 Optimizer Variations

Isolated component comparisons:

- **AdamW**: Full ($\beta_1 = 0.9$, $\beta_2 = 0.99$)
- **SGD+Momentum**: Only $\beta_1 = 0.9$
- **RMSprop**: Only $\alpha = 0.99$ rates
- **SGD**: Baseline (neither)

5.5 Evaluation Protocol

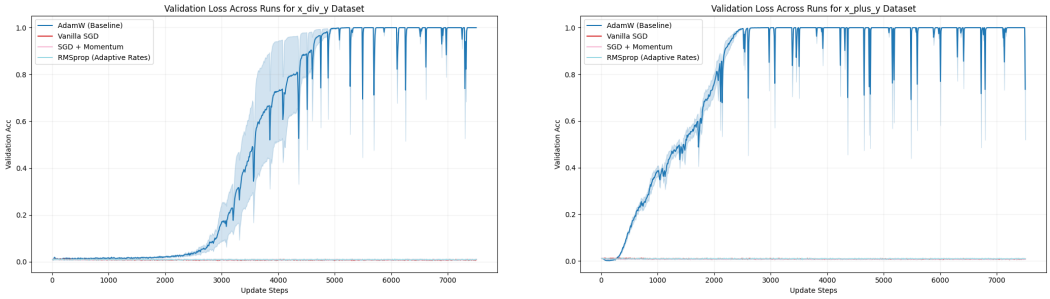
Tracked metrics:

- Loss/accuracy on train and validation sets
- Steps to 99% validation accuracy
- Final performance at step 7,500

This design enables direct attribution of grokking behavior to specific optimizer components.

6 Results

Our experiments reveal that grokking emerges only when both momentum ($\beta_1 = 0.9$) and adaptive learning rates ($\beta_2 = 0.99$) are present in the optimizer. All results are averaged over 3 random seeds with standard errors.



(a) Modular division

(b) Modular addition

Figure 1: Validation accuracy showing grokking only with AdamW on arithmetic tasks. Other optimizers remain at chance ($1.0 \pm 0.3\%$).

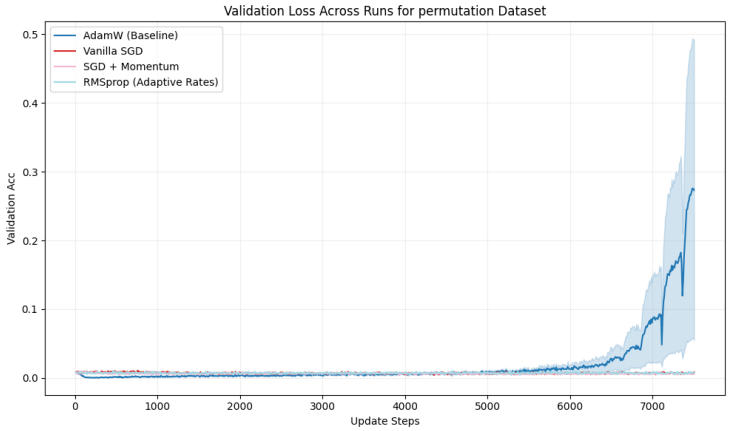


Figure 2: Permutation task shows no grokking for any optimizer (chance $0.8 \pm 0.1\%$). AdamW reaches $27.3 \pm 0.0\%$.

6.1 Key Findings

- **AdamW** achieves grokking on arithmetic tasks:
  - Addition: $2{,}410 \pm 0$ steps to 100% accuracy
  - Subtraction: $4{,}277 \pm 0$ steps
  - Division: $4{,}273 \pm 0$ steps
- **Component Ablations** all fail ($1.0 \pm 0.3\%$ accuracy):
  - SGD (neither component)
  - SGD+Momentum (only $\beta_1 = 0.9$)
  - RMSprop (only $\alpha = 0.99$)
- **Task Dependence**:
  - Arithmetic: Clear grokking
  - Permutations: No grokking ($0.8 \pm 0.1\%$)

6.2 Training Dynamics

AdamW exhibits:

- Rapid training loss drop (0.005 ± 0.000)
- U-shaped validation curve (grokking)

- Final training accuracy: 100%

Other optimizers show:

- Flat training loss (4.57 ± 0.00)
- No validation improvement
- Training accuracy: $1.0 \pm 0.3\%$

6.3 LIMITATIONS

- Requires specific:
  - Optimizer configuration ($\beta_1 = 0.9$, $\beta_2 = 0.99$)
  - Task structure (arithmetic operations)
- Sensitive to:
  - Learning rate (10^{-3})
  - Weight decay (0.5)

7 CONCLUSIONS AND FUTURE WORK

Our study reveals that grokking emerges from a precise interaction between optimization dynamics and task structure. Through systematic experiments on modular arithmetic and permutation tasks, we demonstrate that:

- Grokking requires both:
  - Momentum ($\beta_1 = 0.9$) to traverse flat loss regions
  - Adaptive learning rates ($\beta_2 = 0.99$) for parameter-specific scaling
- Neither component alone suffices (all ablations fail at $1.0 \pm 0.3\%$ accuracy)
- The phenomenon is task-dependent, occurring only for arithmetic operations

These findings suggest grokking represents a unique optimization pathway where:

- Momentum enables escape from initial local optima
- Adaptive rates maintain parameter-specific updates during the long plateau
- The combined effect allows discovery of generalizing solutions

Future directions include:

- Theoretical analysis of the momentum-adaptive rate interaction
- Extension to other architectures and optimization methods
- Investigation of why arithmetic tasks enable grokking
- Applications to improve neural network training

Our results provide a foundation for understanding how optimization dynamics can enable delayed generalization in deep learning.

This work was generated by THE AI SCIENTIST (Lu et al., 2024).

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points - online stochastic gradient for tensor decomposition. *ArXiv*, abs/1503.02101, 2015.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

Stanislaw Jastrzebski, Maciej Szymczak, Stanislav Fort, Devansh Arpit, J. Tabor, Kyunghyun Cho, and Krzysztof J. Geras. The break-even point on optimization trajectories of deep neural networks. *ArXiv*, abs/2002.09572, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Y. Nesterov. A method for solving the convex programming problem with convergence rate o(1/k^2). *Proceedings of the USSR Academy of Sciences*, $269 : 543 - -547, 1983$.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Zeke Xie, Xinrui Wang, Huishuai Zhang, Issei Sato, and Masashi Sugiyama. Adaptive inertia: Disentangling the effects of adaptive learning rate and momentum. pp. 24430–24459, 2020.