Visualizing Grokking: The Impact of Dynamic Weight Decay on Training and Test Loss Convergence

Anik Biswas
Dept. of Math and Computer Science
Hobart and William Smith Colleges
Geneva, USA
ORCID: 0000-0002-1788-1321
anik.biswas@hws.edu

Khairul Islam
Dept. of Math and Computer Science
Hobart and William Smith Colleges
Geneva, USA
ORCID: 0009-0007-5868-0378
khairul.robotic@gmail.com

Hanqing Hu
Dept. of Math and Computer Science
Hobart and William Smith Colleges
Geneva, USA
hu@hws.edu

***Abstract*—Grokking, the phenomenon that neural networks generalize well after a long time of overfitting, has been actively studied in deep learning . In this work, we investigate the role of weight decay in speeding up grokking dynamics on modular arithmetic tasks. This paper presents and analyzes dynamic weight decay, a new approach that dynamically adjusts the weight regularization strength during training and is able to cut the time to reach generalization by a large margin. We investigate the interplay between sparsity, weight changes, and test loss convergence by a detailed study of input/output embeddings, attention weights, and loss trajectories. Our results show a high initial weight decay with dynamic adjustment strategy leads to faster grokking compared to static weight decay. These findings have broad implications in optimizing neural network training for tasks that require both memorization and generalization.**

***Index Terms*—Transformer model, weight decay, dynamic regularization, attention mechanism, positional embeddings, scaled dot-product attention, neural network training, AdamW optimizer, embedding visualizations, model regularization, heatmap analysis, deep learning.**

I. Introduction

Grokking, a phenomenon where neural networks generalize well only after prolonged over-fitting, has garnered increasing attention within the deep learning community. Initially observed in modular arithmetic tasks, grokking has since been reported across diverse domains, including algorithmic datasets, hierarchical reasoning, and semantic analysis tasks [1, 2]. This delayed generalization is often attributed to the interaction between training dynamics, memorization, and structural simplicity [3, 4]. Weight decay, a form of L2-regularization, plays a pivotal role in grokking dynamics by encouraging the discovery of simpler solutions. Earlier studies suggest that static weight decay can help transition neural networks from memorization-focused solutions to generalized ones [5, 6]. However, static regularization strategies may not optimally support the evolving training dynamics, leaving room for more adaptive approaches. This paper introduces dynamic weight decay, an adaptive regularization technique that adjusts the weight decay coefficient throughout training. Unlike static weight decay, the dynamic approach adapts to the network's progression, accelerating the transition to generalization. Using modular arithmetic tasks such as (a+b) mod(113), we investigate the interplay of sparsity, weight dynamics, and loss trajectories to analyze this method's efficacy. Our contributions include a demonstration of how dynamic weight decay significantly reduces the time to achieve grokking, an exploration of its impact on neural representations, and empirical evidence that links dynamic adjustment with improved test loss convergence.

II. Related Work

The original formulation of grokking emerged from studies on modular arithmetic tasks, where delayed generalization was linked to weight decay and structural regularization [1, 2]. Recent works have extended this phenomenon to more complex datasets and architectures, highlighting its relevance across neural network training paradigms [3, 4]. Weight decay has been identified as crucial for grokking, with studies suggesting it encourages circuit efficiency and facilitates the emergence of sparse subnetworks [5, 6]. However, prior methods rely primarily on static weight decay schedules, which may not fully exploit the evolving dynamics of training [7]. While sparsity and representation quality have been explored in grokking contexts, the role of dynamic regularization remains underexplored. Our study builds on this body of work by proposing and analyzing dynamic weight decay, showing how adaptive regularization enhances generalization speed and model efficiency.

III. Methodology

Here we describe the systematic approach to solving the modular arithmetic task $(a+b) \mod (113)$. The methodology integrates modular arithmetic task formulation, dataset construction, model architecture, training strategies, weight decay regularization, and visualizations.

A. Dynamic Weight Decay

Unlike static weight decay, which maintains a constant λ throughout training, dynamic weight decay modifies λ based on stages of training to better align with the network's learning dynamics.

1) Adjustment Strategy: Our dynamic adjustment strategy involves the following steps:

1) Initial Phase: Apply a high weight decay coefficient to encourage sparsity and prevent early overfitting.
2) Intermediate Phase: Gradually reduce λ to allow the network to fine-tune weights without excessive regularization.
3) Final Phase: Stabilize λ to a lower value to maintain generalization without hindering performance.

The transition between phases can be governed by metrics such as validation loss plateauing or a predefined number of epochs.

Static weight decay applies a constant penalty on the weight norm:

$$\mathcal{L}_{\text{total}} = \mathcal{L} + \lambda \cdot ||w||^2 \tag{1}$$

Dynamic weight decay adjusts λ_t during training:

$$\lambda_t = \begin{cases} \lambda_{\text{initial}}, & t < t_{\text{warmup}} \\ \lambda_{\text{final}} + (\lambda_{\text{initial}} - \lambda_{\text{final}}) \cdot \left(1 - \frac{t - t_{\text{warmup}}}{T - t_{\text{warmup}}}\right), & t \geq t_{\text{warmup}} \end{cases} \tag{2}$$

B. Dataset Construction

We evaluate the effectiveness of dynamic weight decay on modular arithmetic tasks, specifically $(a + b) \mod 113$. The task is to compute:

$$\text{Output} = (a + b) \mod (113) \tag{3}$$

where $a, b \in [0, 112]$. This yields $113^2 = 12,769$ unique data samples. Each input sequence is represented as:

$$\mathbf{x}_i = [a, b, 113], \quad y_i = (a + b) \mod (113) \tag{4}$$

The inclusion of the special token 113 ensures a consistent input length of three tokens for all samples, enabling compatibility with the transformer model's fixed input requirements. The dataset is generated programmatically using the following parameters:

- p: Defines the modulus and range of integers $a, b \in [0, p-1]$.
- frac_train: Fraction of the dataset allocated for training. For us it is 70 percent of the dataset.
- DATA_SEED: Fixed random seed to ensure reproducibility during shuffling and splitting.

The dataset includes all pairs (a, b), resulting in p^2 samples:

$$\text{Total Samples} = p^2 \tag{5}$$

The dataset is shuffled and split into training and testing subsets:

$$\text{Training Set Size} = \lfloor \text{frac\_train} \cdot p^2 \rfloor \tag{6}$$

$$\text{Test Set Size} = p^2 - \text{Training Set Size}] \tag{7}$$

Layer (Type: Depth-Idx)	Output Shape	Param #
HookedTransformer	[1, 3, 113]	–
– Embed: 1-1	[1, 3, 128]	14,592
– HookPoint: 1-2	[1, 3, 128]	–
– PosEmbed: 1-3	[1, 3, 128]	384
– HookPoint: 1-4	[1, 3, 128]	–
– ModuleList: 1-5	–	–
—- TransformerBlock: 2-1	[1, 3, 128]	–
—— HookPoint: 3-1	[1, 3, 128]	–
—— Identity: 3-2	[1, 3, 128]	–
—— Identity: 3-3	[1, 3, 128]	–
—— Identity: 3-4	[1, 3, 128]	–
—— Attention: 3-5	[1, 3, 128]	66,048
—— HookPoint: 3-6	[1, 3, 128]	–
—— HookPoint: 3-7	[1, 3, 128]	–
—— Identity: 3-8	[1, 3, 128]	–
—— MLP: 3-9	[1, 3, 128]	131,712
—— HookPoint: 3-10	[1, 3, 128]	–
—— HookPoint: 3-11	[1, 3, 128]	–
– Unembed: 1-6	[1, 3, 113]	14,577
Total Params		**227,313**
Trainable Params		**227,313**
Non-Trainable Params		**0**
Input Size (MB)		0.00
Forward/Backward Pass Size (MB)		0.01
Params Size (MB)		0.12
Estimated Total Size (MB)		**0.13**

Model

Summary Table. This table summarizes the transformer model architecture, including parameter counts and memory requirements.

C. Model Architecture

The transformer model is designed specifically for this task. It consists of the following components:

1) Input Embedding Layer (W_E):

- Shape: $[p+1, d_{\text{model}}]$, where $p+1$ is the vocabulary size including the special token 113.
- Purpose: Maps input tokens into d_{model}-dimensional embeddings:

$$\mathbf{E}_{\text{input}} = \mathbf{W}_E \cdot \mathbf{x} \tag{8}$$

2) Positional Embedding:

- Shape: $[1, n_{\text{ctx}}, d_{\text{model}}]$, where n_{ctx} is the context length.
- Purpose: Adds positional information to preserve token order:

$$\mathbf{E}_{\text{pos}} = \mathbf{E}_{\text{input}} + \mathbf{P} \tag{9}$$

3) Transformer Block: The transformer block is the core computational unit and consists of:

- Attention Mechanism: Computes relationships between tokens using scaled dot-product attention:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^\top}{\sqrt{d_{\text{head}}}}\right) \cdot \mathbf{V} \tag{10}$$

- Feed-forward Layer: Applies a two-layer dense network with ReLU activation:

$$\text{MLP}(\mathbf{h}) = \text{ReLU}(\mathbf{W}_1 \cdot \mathbf{h}) \cdot \mathbf{W}_2 \tag{11}$$

4) Output Projection (W_U):

- Shape: $[d_{\text{model}}, p]$.
- Purpose: Projects final hidden states to the output space:

$$\mathbf{y}_{\text{pred}} = \text{Softmax}(\mathbf{W}_U \cdot \mathbf{h}) \tag{12}$$

D. Training Procedure

The model is trained using optimizer, AdamW with the following hyper-parameters:

- Learning rate: 0.001
- Checkpoint : 1000 epochs
- Epochs: 25000
- Weight decay schedule: drops by 0.04 per epoch

1) Loss Function: The task is modeled as a classification problem using cross-entropy loss:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} \log P(y_i|\mathbf{x}_i) \tag{13}$$

2) Optimizer: The AdamW optimizer updates weights using:

$$w_{t+1} = w_t - \eta\left(\nabla_w \mathcal{L} + \lambda \cdot w_t\right) \tag{14}$$

where η is the learning rate and λ is the weight decay coefficient.

E. Analysis Metrics

To assess the impact of dynamic weight decay, we analyze:

- Training and Test Loss Convergence: Monitoring how quickly and effectively the model converges on training and test loss.
- Sparsity Levels: Measuring the sparsity of weight matrices throughout training.
- Weight Dynamics: Tracking changes in weights, including magnitude and direction.
- Neural Representations: Examining input/output embeddings and attention weights as heat maps.

IV. Results

A. Heat-maps:

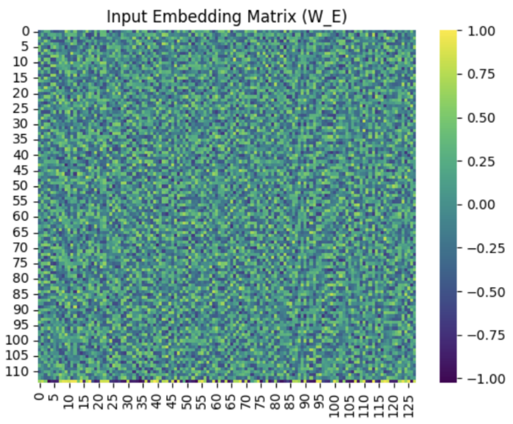

Fig. 1. Input Embedding Matrix (W_E). Shape: (114, 128). This Heat-map visualizes the normalized input embedding values.Sparsity:0.47%

This section analyzes the sparsity and density patterns in embedding and attention weight matrices, along with their implications for interpretability and performance.

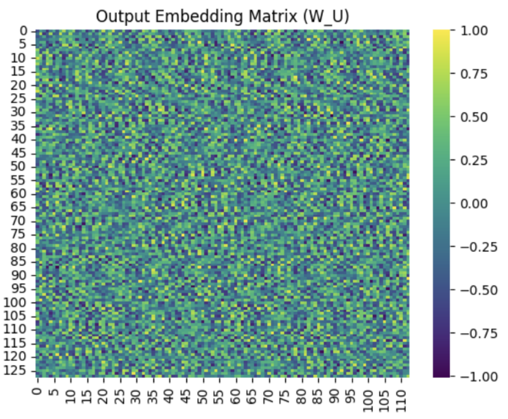

Fig. 2. Output Embedding Matrix (W_U). Shape: (128, 113). This Heat-map visualizes the normalized output embedding values. Sparsity: 0.57%.

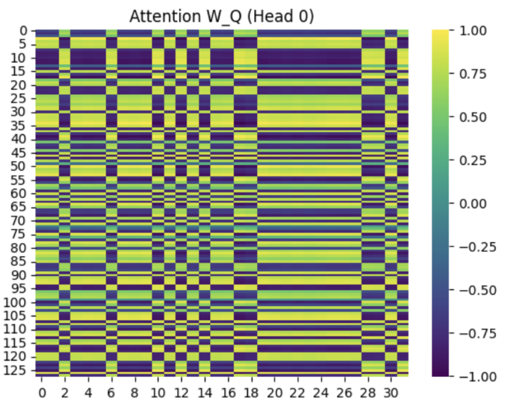

Fig. 3. Attention Weight Matrix (W_Q) for Head 0. Shape: (4, 128, 32). This Heat-map visualizes the attention weights for Head 0.

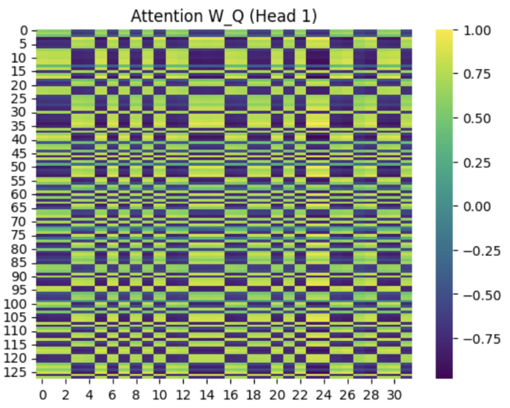

Fig. 4. Attention Weight Matrix (W_Q) for Head 1. Shape: (4, 128, 32). This Heat-map visualizes the attention weights for Head 1.

B. Input Embedding Matrix (W_E)

- Fig. 1 shows a sparsity of 47%, enabling task-specific representations while enhancing interpretability.
- Distributed activations efficiently encode input features without compromising performance.
- Output Embedding Matrix (W_U):

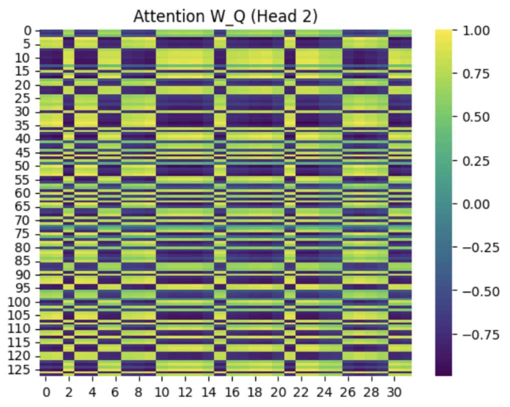

Fig. 5. Attention Weight Matrix (W_Q) for Head 2. Shape: (4, 128, 32). This Heat-map visualizes the attention weights for Head 2.

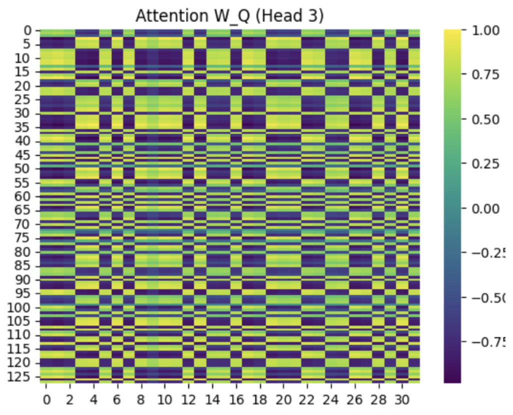

Fig. 6. Attention Weight Matrix (W_Q) for Head 3. Shape: (4, 128, 32). This Heat-map visualizes the attention weights for Head 3.

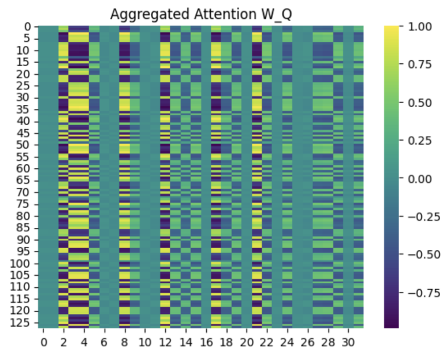

Fig. 7. Aggregated Attention Weight Matrix (W_Q) . Shape: (4, 128, 32). Sparsity of (W_Q): 0.00%

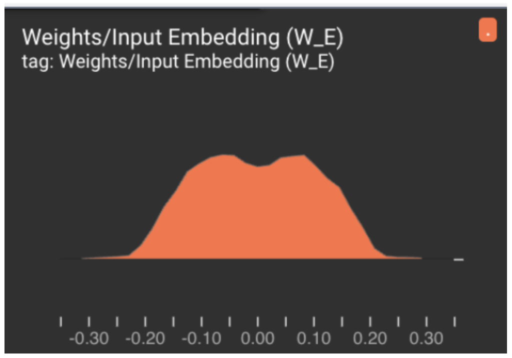

Fig. 8. Input Embedding Weight Distribution

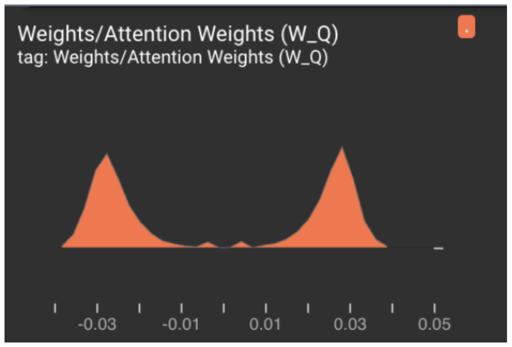

Fig. 9. Attention Weight Layer Weight Distribution

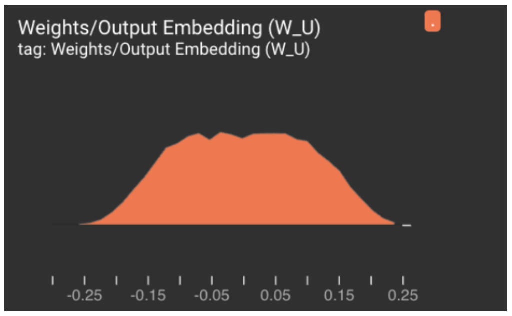

Fig. 10. Output Embedding Weight Distribution

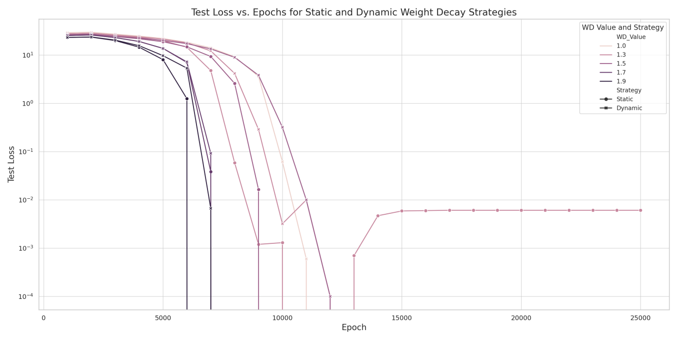

Fig. 11. Logarithmic loss

- Fig. 2 exhibits 57% sparsity, suggesting compact output representations that mitigate overfitting while supporting robust decoding.
- Attention Weight Matrices (W_Q):
- Figures 3–7 highlight uniformly dense attention weights with 0% sparsity. This dense structure ensures comprehensive feature extraction and effective global dependency modeling.
- Aggregated attention weights (Figure 7) confirm the uniformity across heads, critical for multi-head interpretabil-

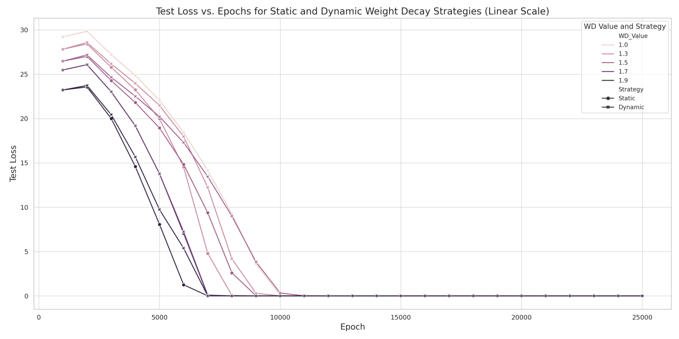

Fig. 12. Linear loss

ity and interaction modeling.

C. *Weight Distributions*

Input Embedding Distribution (W_E):

- Fig. 8 shows a bimodal distribution, enhancing interpretability through adaptive weight sparsity that isolates feature significance.
- Attention Weight Distribution (W_Q):
- Fig. 9 depicts a narrow, compact distribution, reinforcing the role of dense attention for robust feature interaction, consistent with transformer-based research.
- Output Embedding Distribution (W_U):
- Fig. 10 presents a broader spread, reflecting diverse roles in capturing latent output representations, aligning with prior findings on adaptive sparsity.

D. *Sparsity and Weight Decay Insights*

- Sparsity in Embeddings: W_E and W_U challenge the assumption of uniform density, supporting sparsity-driven interpretability while maintaining strong performance.
- Dense Attention Mechanisms: W_Q weights are critical for capturing global dependencies, consistent with prior studies.

E. *Training Dynamics*

Figures 11 and 12: Dynamic weight decay accelerates convergence and enhances generalization by reducing overfitting. Sparse embeddings and dense attention ensure efficient learning and robust performance.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[8] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, arXiv:1312.6114. [Online]. Available: https://arxiv.org/abs/1312.6114

[9] S. Liu, "Wi-Fi Energy Detection Testbed (12MTC)," 2023, gitHub repository. [Online]. Available: https://github.com/liustone99/Wi-Fi-Energy-Detection-Testbed-12MTC

[10] "Treatment episode data set: discharges (TEDS-D): concatenated, 2006 to 2009." U.S. Department of Health and Human Services, Substance Abuse and Mental Health Services Administration, Office of Applied Studies, August, 2013, DOI:10.3886/ICPSR30122.v2

[11] K. Eves and J. Valasek, "Adaptive control for singularly perturbed systems examples," Code Ocean, Aug. 2023. [Online]. Available: https://codeocean.com/capsule/4989235/tree