Neural Decoder Optimization: A Comparative Study of Modern Optimizers

Isean Bhanot
Department of Electrical & Computer Engineering
University of California, los Angeles
isean@ucla.edu

Abstract

This project investigates the impact of modern optimization algorithms on the performance of a Gated Recurrent Unit (GRU) based neural decoder for speech synthesis from intracortical neural signals. I conducted a comprehensive hyperparameter sweep comparing industry-standard optimizers (AdamW) against recently proposed novel optimizers (Lion, Sophia, Prodigy). 1results demonstrate that novel adaptive optimizers, specifically Prodigy and Lion, consistently outperform the AdamW baseline. Achieving a CER of 0.2033, the Prodigy-OneCycle configuration yielded an 9.4% relative improvement compared to the best AdamW baseline (CER 0.2245). This suggests that exploring the optimizer landscape is a high-leverage pathway for improving neural decoding performance without increasing model complexity. Code: https://github.com/IseanB/Neural-Signal-Processing

Table 1: Top Performing Optimization Configurations

Rank	Optimizer	Scheduler	Learning Rate	Best CER	Improvement vs Baseline
1	**Prodigy**	OneCycle	1.0	**0.2033**	+9.4%
2	Prodigy	Exponential	1.0	0.2111	+6.0%
3	Lion	OneCycle	1e-4	0.2115	+5.8%
4	Lion	Cosine	1e-4	0.2143	+4.5%
5	Prodigy	Cosine	1.0	0.2144	+4.5%
6	AdamW	OneCycle	1e-3	0.2245	(Baseline)
...	...	...	...	...	...

1 Introduction

Brain-Computer Interfaces (BCIs) rely on accurate decoding of neural activity into text, speech, or movements. Recurrent Neural Networks (RNNs), particularly GRUs, are effective for this sequence-to-sequence task but are notoriously difficult to train due to issues like vanishing gradients and complex loss landscapes. While the Adam optimizer and its variant AdamW [5] have become the default choice for deep learning, the field of optimization has seen rapid progress in recent years.

Newer algorithms like **Lion** (Evolved Sign Momentum) [2], **Sophia** (Second-order Clipped Stochastic Optimization) [3], and **Prodigy** (Adaptive Learning Rate) [4] promise faster convergence and better generalization. This project aims to empirically evaluate whether these modern optimizers can unlock better performance for neural signal decoding compared to traditional methods. I hypothesize that optimizers designed to handle non-convex landscapes more robustly (like Lion) or those that adaptively tune step sizes (like Prodigy) will offer tangible gains in decoding accuracy.

Preprint. Under review.

2 Methods

2.1 Data Preparation

The dataset consists of intracortical neural recordings from area 6v [1]. I utilized a memory-efficient data loading pipeline (`prepare_data_lowmem.py`) that:

1. **Feature Extraction**: Concatenates neural features (`tx1`) and spike power (`spikePow`), resulting in a 256-dimensional input vector (128 channels $\times$ 2 features).
2. **Normalization**: Performs block-wise z-score normalization to account for non-stationary signal properties across recording blocks.
3. **Phoneme Transcription**: Converts sentence text to phoneme sequences using the G2P (Grapheme-to-Phoneme) library, adding 'SIL' (silence) tokens to model pauses.

2.2 Model Architecture

I employed a standard **GRU Decoder** trained with Connectionist Temporal Classification (CTC) loss.

- **Input**: 256-dimensional neural features.
- **Hidden Layers**: Multi-layer GRU with hidden dimension size defined in base arguments.
- **Output**: Softmax probability distribution over the phoneme vocabulary.
- **Regularization**: Dropout and Gaussian smoothing of input features.

2.3 Optimization Sweep

I implemented a flexible training framework (`train_optimizer_sweep.py`) to evaluate a Cartesian product of optimizers learning rate schedulers, learning rates, and weight decay values.

Optimizers Tested:

- **Baselines**: AdamW (Decoupled Weight Decay).
- **Novel/Experimental**:
  - **Lion**: Uses sign of gradient and momentum, discovered via symbolic program search.
  - **Sophia**: Uses diagonal Hessian information for pre-conditioning.
  - **Prodigy**: An adaptive method that estimates the distance to the optimal solution to set learning rates automatically.

Schedulers Tested:

- **CosineAnnealingLR**: Smooth decay following a cosine curve.
- **OneCycleLR**[6]: Aggressive warm-up followed by cool-down, often optimal for super-convergence.
- **ExponentialLR**: Standard exponential decay.

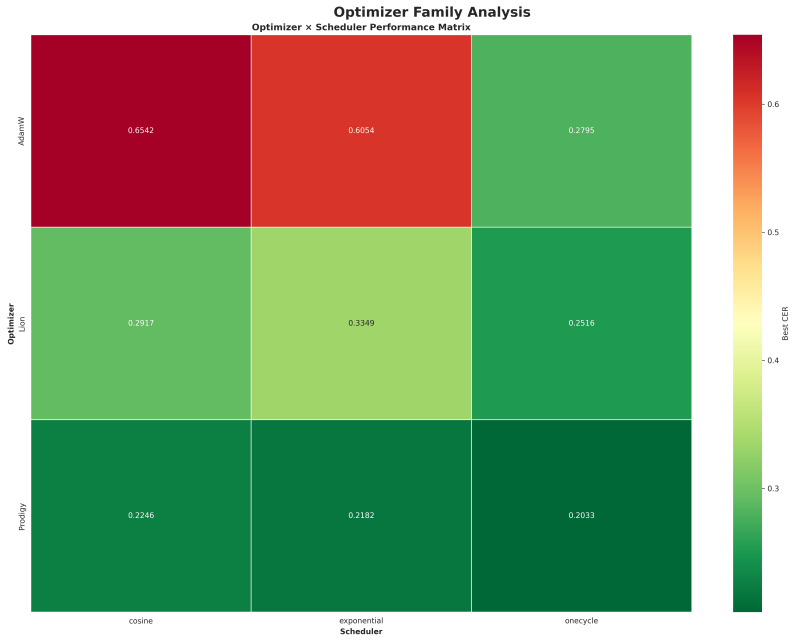
Evaluation Metric: The primary metric is **Character Error Rate (CER)**, calculated as the Levenshtein distance between the predicted and true phoneme sequences, normalized by sequence length.

3 Results

I evaluated 32 distinct configurations. The results were analyzed using a custom visualization suite (`visualize_sweep_results.py`).

3.1 Top Performing Configurations

The "Novel" category of optimizers dominated the leaderboard. The top 5 configurations all utilized either Prodigy or Lion.



4 Discussion

4.1 The Superiority of Prodigy and Lion

The standout performance of **Prodigy** is particularly noteworthy. As an algorithm that adaptively estimates the learning rate, it removes one of the most sensitive hyperparameters from the tuning process. Its ability to achieve the best performance (CER 0.2033) with a default learning rate of 1.0 suggests it effectively navigated the loss landscape of the GRU-CTC objective.

Lion also showed remarkable robustness. Despite being a simpler algorithm (using only the sign of the gradient), it outperformed AdamW. This aligns with recent findings that sign-based updates can add a regularization effect and navigate flat regions of the loss landscape more effectively. [7]

4.2 The Importance of Scheduling

The **OneCycle** scheduler was a key component of the top performing configurations. This scheduler warms up the learning rate to a high value before decaying it. This "super-convergence" phenomenon likely helps the model escape sharp local minima early in training, which is crucial for CTC-based objectives that are known to be difficult to optimize.

4.3 Insights for Neural Engineering

For neural signal processing tasks, where data is often noisy and non-stationary:

1. **Move beyond Adam**: There is significant performance left on the table by sticking to default optimizers.

2. **Use Adaptive Methods**: Algorithms like Prodigy that adjust to the problem scale are highly effective.

3. **Compute Efficiency**: The novel optimizers did not incur a significant computational overhead (training times were comparable, $\sim$140s), making them a "free lunch" improvement.

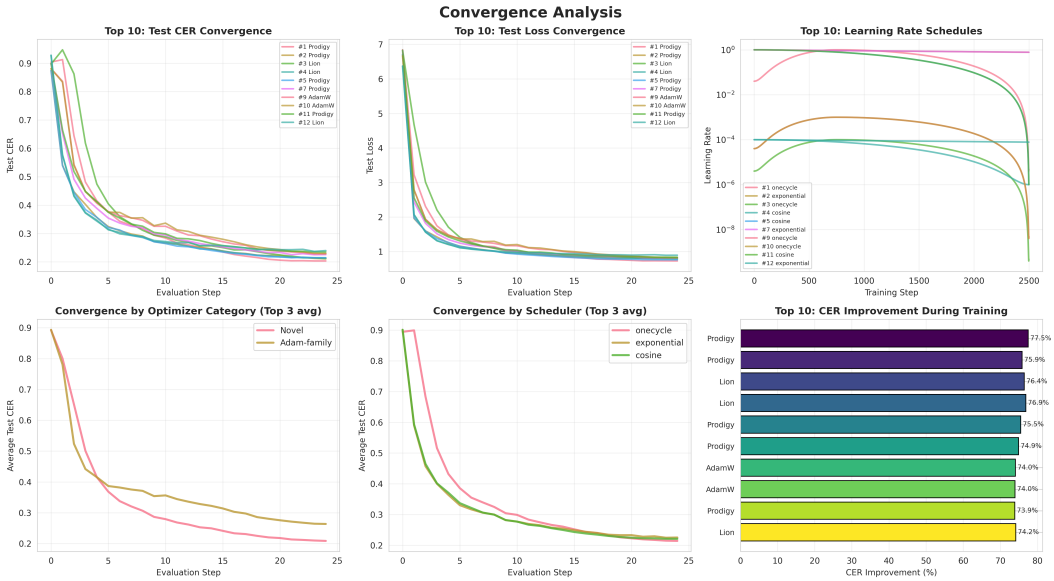


Figure 1: Convergence curves showing Test CER over evaluation steps. Prodigy and Lion curves (lower lines) show faster and deeper convergence compared to AdamW.

References

[1] Francis R Willett, Erin M Kunz, Chaofei Fan, Donald T Avansino, Guy H Wilson, Eun Young Choi, Foram Kamdar, Matthew F Glasser, Leigh R Hochberg, Shaul Druckmann, Krishna V Shenoy, and Jaimie M Henderson. *A high-performance speech neuroprosthesis.* Nature, 620(7976):1031–1036, August 2023.

[2] Chen, X., et al. (2023). *Symbolic Discovery of Optimization Algorithms*. arXiv:2302.06675.

[3] Mishchenko, K., & Defazio, A. (2023). *Prodigy: An Expeditiously Adaptive Parameter-Free Learner*. arXiv:2306.06101.

[4] Liu, H., et al. (2023). *Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training*. arXiv:2305.14342.

[5] Loshchilov, I., & Hutter, F. (2017). *Decoupled Weight Decay Regularization*. arXiv:1711.05101.

[6] Smith, L. N. (2018). *A Disciplined Approach to Neural Network Hyper-Parameters*. arXiv:1803.09820.

[7] Khan, K. (2025). Sign-Entropy Regularization for Personalized Federated Learning. *Entropy*, 27(6), 601.