Predicting Text from Intracranial Neural Signals

Abby Seseri
Department of Computer Science
University of California, Los Angeles
UID: [redacted]
abbyseseri@ucla.edu

Abstract

Intracranial neural speech decoding allows people with paralysis to communicate, but current models must be improved upon to be used practically. I methodically optimized a gated recurrent unit (GRU) baseline for phoneme error rate (PER) reduction on the Brain-to-Text 2024 dataset. My best model had an 18.37% PER, which is a 16.4% improvement from the baseline (21.98%). After experimentation, I observed that optimization changes (AdamW + OneCycleLR) provided the largest gain (21.98% $\rightarrow$ 19.60%), and I used regularization to further improve the model. Transformers trained nearly four times faster but underperformed (best: 21.22% PER). In this experiment, I found that optimizing existing architectures yielded better performance than investigating several architectures for this dataset. If research to find strategies that decrease PER continues, neural speech decoding will be much closer to clinical viability.

1 Introduction

Neural speech decoding from intracranial recordings shows promise in restoring communication to people with paralysis [9]. Recent datasets like Brain-to-Text 2024 offer opportunities for improving these models[10]. The baseline gated recurrent unit (GRU) model achieves 21.98% phoneme error rate (PER), but a lower PER is needed for practical communication devices.

In the past, others have explored time-masking transformers [2], diphone decoding [4], and consistency regularization for CTC loss [11]. However, there is a lack of research in discovering which modifications provide the greatest performance increase for this neural dataset.

In this experiment, I explore four techniques: (1) optimizer and learning rate scheduling, (2) data augmentation, (3) different architectures, and (4) regularization techniques. I found that optimizer improvements provide the largest increase in performance, with regularization yielding smaller, additional gains. To contrast, Transformers underperformed compared to optimized GRU architectures. My experiment gives insight into strategies researchers can use to improve neural speech decoding performance.

2 Methods

2.1 Implementation Details

All experiments were conducted on Google Cloud Platform using an `n1-standard-4` instance with 4 vCPUs, 15 GB memory, and one NVIDIA V100 GPU. I used Python 3.9 with the environment that is specified in the baseline repository.

2.1.1 Code Modifications

I modified four files from the baseline:

1. `augmentations.py`**:** I added the TimeMasking and FeatureMasking classes. The TimeMasking class randomly masks contiguous temporal chunks during training. This helps the model stay resilient against large gaps in neural signals.

2. `model.py`**:** I added layer normalization to the GRUDecoder class and implemented the TransformerDecoder class such that it uses positional encodings. I kept the day-specific linear layers preserved so both architectures could use them. The day-specific linear layers are encodings that ensure day-to-day differences when collecting data are accounted for.

3. `neural_decoder_trainer.py`**:** I modified this file to optimizer the model. For instance, I replaced Adam with AdamW and the learning rate schedule LinearLR with OneCycleLR. Furthermore, I also incorporated the data augmentation pipeline to apply time/feature masking from `augmentations.py`. I also added checkpoint resumption functionality for convenience, but this does make the learning rate scheduler restart.

4. `train_model.py`**:** I modified this file to create separate configurations for each experiment I attempted. I also modified hyperparameters to compare similar architectures with different dropout probabilites, batch sizes, etc.

2.2 Training Protocol

In each experiment, I trained for 10,000 batches with batch size 64 (or 128 when attempting to regularize). Validation PER is calculated every 100 batches.

2.3 Baseline and Incremental Improvements

I started by running the baseline GRU model and applied optimizations over time to the best performing models:

1. Optimizer Refinements: I began by replacing Adam with AdamW because of research indicating that AdamW improves generalization because of decopled weight decay ([5]). At the same time, I exchanged the linear learning rate decay with a OneCycleLR scheduler. The aim of this decision was to achieve faster convergence because of the cyclical learning rate.

2. Data Augmentation with Time Masking: In Feghhi et. al.'s paper, masking temporal segments during training helped make the model more robust to noise ([2]). Consequently, I also introduced time-masking augmentation to the training of my model. This technique randomly masks contiguous temporal chunks of the neural signal during training (20% probability, max 30 timesteps) to make the model more robust to potential gaps in neural recordings.

3. Incorporating LayerNorm: Adding LayerNorm to hidden-to-hidden layers of a recurrent neural network has been proven to improve performance, especially by making the model insensitive to initial weights ([1]). Therefore, I incorporated a LayerNorm layer to the baseline GRU Decoder model to stabilize training.

2.4 Regularization Parameter Study

I used a GRU with LayerNorm, AdamW, OneCycleLR, and TimeMasking as a base because this was the highest performing model after making incremental improvements. I then studied the effects of increasing three regularization parameters: batch size (64 to 128), white noise standard deviation (0.8 to 1.0), and dropout rate (0.4 to 0.5). I augmented these hyperparameters to reduce overfitting.

2.5 Transformer Architecture Hyperparameter Search

I also explored several Transformer-based architectures because training was significantly faster for 10,000 batches than the GRU architecture. I attempted various combinations of hyperparameters, namely the hidden dimensions (128, 256, 512), layer counts (2, 4, 8, 16), and dropout rates (0.1, 0.3).

3 Results

3.1 Performance Comparison

Table 1 summarizes the progressive improvements to the baseline GRU model and Transformer performance. Table 2 shows the individual effects of regularization parameters on the LayerNorm-GRU model. Table 3 presents the Transformer hyperparameter exploration.

Table 1: Core Architecture Performance Comparison

Architecture	Val. PER	Val. Loss	Time (min)
Baseline GRU	0.2198	0.8155	67
GRU with Progressive Optimizations:			
+ AdamW & OneCycleLR	0.1960	0.9067	67
+ Time Masking	0.1948	0.9069	67
+ Layer Normalization	0.1937	0.9133	67
+ Full Regularization*	**0.1837**	**0.8728**	133
Transformer Variants:			
Best Transformer (8L-128H)	0.2122	1.0518	17

**Full regularization includes increased dropout (0.4→0.5), white noise (0.8→1.0), and batch size (64→128)*

Table 2: LayerNorm-GRU Regularization Parameter Study

Configuration	Batch Size	White Noise	Dropout	Val. PER
Base LayerNorm Model	64	0.8	0.4	0.1937
+ Batch Size 128	128	0.8	0.4	0.1888
+ White Noise 1.0	128	1.0	0.4	0.1853
+ Dropout 0.5	128	1.0	0.5	**0.1837**

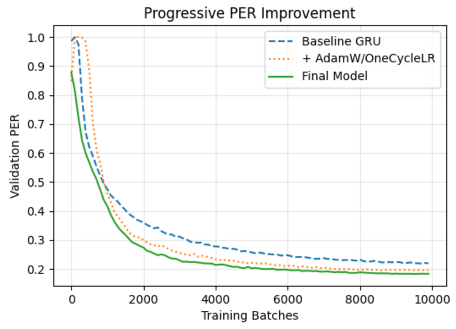
Table 3: Transformer Architecture Hyperparameter Search

Configuration	Hidden	Layers	Dropout	Val. PER
Base Transformer	256	4	0.1	0.2248
Hidden Size Study:				
128 hidden	128	4	0.1	0.2334
512 hidden	512	4	0.1	0.2133
Layer Depth Study:				
2 layers	128	2	0.1	0.2598
8 layers	128	8	0.1	**0.2122**
16 layers	128	16	0.1	0.9637
Dropout Study:				
Dropout 0.3	128	2	0.3	0.2833
Dropout 0.3	256	4	0.3	0.2406

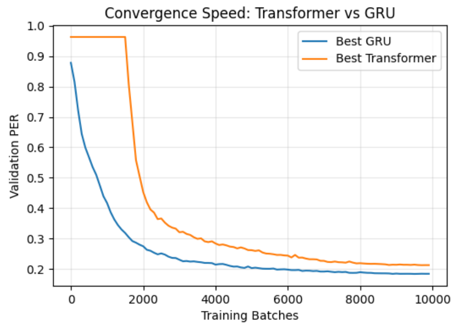
3.2 Training Dynamics

Figure 1a shows the validation PER across training for key models. The baseline (blue) converges near 22% PER. The model with the AdamW and OneCycleLR optimizations (orange) reaches a lower plateau near 19.6% PER. The final model with LayerNorm and regularization (green) achieves the lowest final PER with a smoother training curve.

Figure 1b compares the best GRU (blue) and best Transformer (orange). The Transformer's PER remains near 100% for approximately the first 1,500 batches before decreasing. It converges to a higher final PER than the GRU.



(a) PER reduction with progressive optimizations



(b) Convergence comparison: GRU vs Transformer

Figure 1: Training dynamics showing (a) PER improvements from the baseline to the final, regularized LayerNorm-GRU model and (b) Transformer convergence compared to GRU

3.3 Summary of Key Outcomes

The optimizations produced the following key results:

- The final, regularized GRU model achieved a validation PER of 18.37%, representing a 16.4% relative reduction from the baseline PER of 21.98% (Table 1).
- Incrementally increasing batch size (64 $\rightarrow$ 128), white noise (0.8 $\rightarrow$ 1.0), and dropout (0.4 $\rightarrow$ 0.5) each caused a lower PER for the LayerNorm-GRU model (Table 2).
- The best-performing Transformer configuration (8 layers, 128 hidden units, 0.1 dropout) achieved a PER of 21.22%. To contrast, a 16-layer variant reached a PER of 96.37% (Table 3).
- The best Transformer trained in 17 minutes, compared to 67 minutes for the GRU baseline.

4 Discussion

4.1 Optimizer Choices

My exploration has revealed key strategies for improving neural speech decoding on this dataset. Although I hypothesized that architectural novelty would produce the largest gains, I found that training methodology like optimizer selection and scheduling offered the most impactful improvement (21.98% $\rightarrow$ 19.60% PER).

This insight agrees with recent findings in deep learning optimization. Specifically, AdamW's superiority over the baseline's Adam optimizer is largely due to AdamW's decoupled weight decay mechanism ([5]). In standard Adam, weight decay (L2 regularization) interacts with the adaptive learning rate, which can render the regularization less effective. On the other hand, AdamW keeps these components separate, ensuring weight decay acts directly on the weights and is not scaled by the running gradient estimates. Because input features can span different scales and distributions in neural decoding, AdamW provides more consistent regularization for all parameters.

Using the OneCycleLR scheduler also greatly benefited the model. Linear decay monotonically decreases learning rates. Alternately, OneCycleLR implements a warm-up phase (30% of training in this configuration) followed by annealing ([6]). This approach helps handle saddle points in the loss function more effectively during early training. The combined effect was a lower final PER and accelerated convergence, as seen in Figure 1a where the orange curve (AdamW+OneCycleLR) decreases much faster than the baseline. This acceleration is especially valuable when applying neural decoding because quick model calibration makes it more useful in clinical settings.

4.2 Regularization

While applying each regularization technique, the PER continued to decrease (Table 1), which confirms that the baseline must be overfitting the training data. Each method touched upon a different facet of regularization:

- **Time Masking** is inspired by (2). I applied time masking to neural recordings by masking contiguous segments. This encourages the decoder to rely on broader temporal context thus improving the model's robustness to gaps in neural signals.
- **LayerNorm** stabilized training of GRU hidden states (1). As a result, I was able to use stronger, subsequent regularization techniques like increased dropout without destabilizing training.
- The regularization parameter study (Table 2) showed that increasing batch size and white noise provided the largest combined gain. A greater batch size indicates better gradient estimates and greater white noise prevents overfitting to specific values. While the final dropout probability increase had a smaller performance gain, its role in preventing neuron co-adaptation (7) forced the model to learn more generalizable features of the 256 neural input features.

4.3 Transformers

Transformers trained four times faster than the GRU architecture, but they underperformed in accuracy because of a fundamental data efficiency mismatch. Transformers do not have the GRU's innate temporal inductive bias and instead use positional encodings to learn sequence order (8), so they require much more data to learn effectively. This is evident because of the roughly 1500-batch warm-up period (Figure 1b). Moreover, transformer's global self-attention mechanism may also not align well with the sequential, day-specific linear layers. Deep Transformers (16-layer) failed much worse (96.37% PER), revealing that transformers are more sensitive to depth and initialization than the more robust GRU.

4.4 Clinical Implications and Future Work

The results have a clear clinical trade-off. The transformer's speed (17 min) benefits fast calibration, but the optimized GRU's superior performance (18.37% vs. 21.22% PER) is crucial for intelligible communication. In the future, systems could potentially use Transformers for initial calibration before relying on GRUs for daily use.

Because of computational limits, my experiments suggest several future directions. Hybrid architectures could combine GRU sequence modeling with efficient attention. Test-time adaptation may help address neural non-stationarity. Finally, integrating language models can provide a much richer linguistic context beyond phonemes, as shown in recent brain-to-text decoding work (4).

Ultimately, my experiments suggest optimization of a GRU architecture yielded better performance than architectural novelty alone, offering a realistic path toward clinical utility targeted by the Brain-to-Text benchmark (10).

5 Conclusion

Overall, I improved neural speech decoding performance through incremental optimization of the baseline GRU model. My best model achieved 18.37% PER, which is a 16.4% improvement over the baseline's performance. The increase in performance is largely attributed to a combination of optimizer refinements (AdamW, OneCycleLR) and regularization (time masking, layer normalization, increased dropout, batch size, and white noise).

To summarize, some important findings included that optimizer improvements causes the biggest increase in performance for this task. Applying several different regularization techniques proved to decrease the PER more. Furthermore, LayerNorm helped with regularization and stabilizing training. Transformers were much quicker while training, but this had a tradeoff of worse performance for the same number of batches. Ultimately, optimization of the existing GRU architecture was more successful than utilizing new architectures like Transformers. These improvements suggest that incremental optimization may be able to meaningfully advance neural speech decoding toward clinical use.

References

[1] Ba, J.L., Kiros, J.R., & Hinton, G.E. (2016). Layer normalization. *arXiv preprint* arXiv:1607.06450.

[2] Feghhi, E., Kaasyap, S., Hadidi, N., & Kao, J. C. (2025). Time-masked transformers with lightweight test-time adaptation for neural speech decoding. *arXiv [cs.HC].*

[3] Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning* (pp. 369–376).

[4] Li, J., Le, T., Fan, C., Chen, M., & Shlizerman, E. (2024). Brain-to-text decoding with context-aware neural representations and large language models. *arXiv [eess.SP].*

[5] Loshchilov, I., & Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint* arXiv:1711.05101.

[6] Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820.*

[7] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), 1929–1958.

[8] Vaswani, A., Shazeer, N., Parmar, N. et al. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998–6008).

[9] Willett, F.R., Kunz, E.M., Fan, C. et al. (2023). A high-performance speech neuroprosthesis. *Nature*, 620, 1031–1036.

[10] Willett, F.R., Li, J., Le, T. et al. (2024). Brain-to-text benchmark '24: Lessons learned. *arXiv [cs.CL].*

[11] Yao, Z., Kang, W., Yang, X. et al. (2024). CR-CTC: Consistency regularization on CTC for improved speech recognition. *arXiv [eess.AS].*