Leveraging Recurrent Neural Networks for Predicting Motor Movements from Primate Motor Cortex Neural Recordings

Yuanxi Wang, Zuowen Wang, Shih-Chii Liu
Institute of Neuroinformatics
University of Zurich and ETH Zurich
Zurich, Switzerland
yuanxwang@ethz.ch, {zuowen, shih}@ini.uzh.ch

Abstract—This paper presents an efficient deep learning solution for decoding motor movements from neural recordings in non-human primates. An Autoencoder Gated Recurrent Unit (AEGRU) model was adopted as the model architecture for this task. The autoencoder is only used during the training stage to achieve better generalization. Together with the preprocessing techniques, our model achieved 0.71 R^2 score, surpassing the baseline models in Neurobench and is ranked first for R^2 in the IEEE BioCAS 2024 Grand Challenge on Neural Decoding. Model pruning is also applied leading to a reduction of 41.4% of the multiply-accumulate (MAC) operations with little change in the R^2 score compared to the unpruned model.

Index Terms—neural decoding, neural motor decoder network, autoencoder gated recurrent unit

I. INTRODUCTION

Millions of people around the world suffer from a form of body paralysis. For those who suffer injury or degradation of the peripheral nervous system or spinal cord but still retain intact brain function, intracortical brain-machine interfaces (iBMIs) can be a promising solution for partial restoration of the functionality of their limbs. Neural signals from the motor cortex and premotor cortex of patients can be recorded using multi-electrode recording arrays like the Utah micro-electrode array [1]. Decoding these neural signals can help one to uncover the underlying movement intention of the patient, and serves for the downstream tasks such as prostheses controlling [2].

With the availability of more neural recording datasets, researchers can develop new decoding algorithms to determine if they can give accurate predictions of the input stimuli or kinematic states from neural recordings [3]–[6]. Non-linear decoders particularly those based on deep learning algorithms are also widely assessed on these recordings. They consistently report better task accuracy than a linear Kalman filter [7], [8]. For the deep learning models used for neural decoding, features such as multi-unit activity (MUA) [7], [8], entire spiking activity (ESA) [4], and local field potential (LFP) [9] are usually first extracted from the raw neural data and used as input to the network. Data preprocessing steps can help highlight the most relevant neural signals, and improve the signal-to-noise ratio, ultimately enhancing the accuracy and reliability of the decoder.

In this study, we present a neural network-based solution that is both computationally efficient and uses a low memory footprint for decoding the primate reaching motion speed from intracortical neural recordings. The network is trained on recordings of the primate reaching dataset was presented as part of the IEEE BioCAS 2024 Grand Challenge on Neural Decoding. The results of our experiments show that the time duration of the input feature to our model plays a crucial role for the accuracy in the task of finger velocity prediction, emphasizing the importance of data preprocessing of neural decoding. We benchmark our method against baselines using Neurobench [10].

II. METHODS

This section describes the dataset and metrics used in this study, then introduces our AEGRU model architecture, training methods, and methods for reducing the memory footprint and computes for the final model.

A. Dataset

We use the six recordings selected in Neurobench [10] from a bigger dataset released by [11]. This dataset contains intracortical recordings from microelectrode arrays implanted in two non-human primates (NHP) named Indy and Loco, while they performed a reaching task to a target location. These reaches were done by controlling a cursor in the workspace with their fingertips. Once the target was reached, a new target appeared without gaps or pre-movement delay intervals. Spikes in the brain activity were detected by means of the threshold-crossing technique after the baseline drift was removed. Spike data were finally sampled at $250Hz$, the same rate used to sample the finger velocity of the NHP. From the six recordings in the Challenge dataset, 3 recordings came from Indy and the other 3 recordings came from Loco. They are: *indy_20160622_01, indy_20160630_01, indy_20170131_02, loco_20170131_02, loco_20170215_02, loco_20170301_05*. Recordings from Indy and Loco came

TABLE I: Notation list

Notation	Description
N	Number of GRU time steps (sequence length)
WS	Number of sample points in a window for summation
C_i	Number of multi-electrode input channels
C_f	Number of latent layer features
C_h	Number of GRU hidden layer features
C_σ	Number of hidden layer features for latent factor variance prediction
x, x'	Input and preprocessed input
r	Reconstructed firing rates (a.u.)
h, h_0	hidden state and the initial hidden state of GRU layer
f, μ^f, σ^f	Latent factors and their mean and standard deviation

from 96 and 192 input channels of the microelectrode arrays respectively.

B. Metrics

The decoding performance of the models on the Grand Challenge dataset is measured by the Neurobench harness [10]. They consist of the following three metrics: R^2 accuracy, memory footprint, and computational complexity measured by the effective multiply-accumulate (MAC) and accumulate (AC) operations.

1) R^2 accuracy: The alignment between the ground truth and predicted velocities is quantified by the R^2 score:

$$R_x^2 = 1 - \frac{\|\bm{v_x} - \hat{\bm{v}}_x\|_2^2}{\|\bm{v_x} - \overline{v}_x\|_2^2}, \quad R_y^2 = 1 - \frac{\|\bm{v_y} - \hat{\bm{v}}_y\|_2^2}{\|\bm{v_y} - \overline{v}_y\|_2^2} \quad (1)$$

where $\|\cdot\|_2^2$ denotes the squared of L_2 norm, $\bm{v_x}, \bm{v_y}$ vectors describe the x and y-components of the actual velocity over time, $\hat{\bm{v}}_x, \hat{\bm{v}}_y$ are the vectors of predicted velocity components over time, and $\overline{v}_x, \overline{v}_y$ are scalars of the average x- and y-components of velocity. Minus $\overline{v}_x$ or $\overline{v}_y$ is broadcasted on every timestep of $\bm{v_x}$ and $\bm{v_y}$ respectively. The final R^2 is averaged over R_x^2 and R_y^2 as described in Eq. 1.

2) Memory footprint: The memory footprint measurement sums up the memory needed for storing the model parameters and activations at their respective numerical precision. In Neurobench memory footprint does not exclude pruned weights.

3) Computational complexity: The computational complexity is calculated from the average number of effective MAC operations and AC operations per inference pass through the model. It excludes zero activations and zero connections.

The number of ACs is used to indicate the computation cost of the spiking neural network (SNN) layer, i.e. in the Neurobench baseline models, SNN3D and SNN_streaming. Although an AC operation uses less computational resources than a MAC operation, for simplicity, we choose to use their sum as a metric because the number of ACs is consistently much lower than the number of MACs.

C. Model architecture

The basic building blocks of the AEGRU model are described next starting with the GRU model and extending to the autoencoder.

1) GRU: The proposed AEGRU model is based on the Gated Recurrent Unit (GRU) [12], a basic building block of gated Recurrent Neural Networks (RNNs) proposed to mitigate the issue of either vanishing or exploding gradients during training and useful previously for processing temporal sequences such as time series biomarker processing [13] due to its capability of streamlining the data processing. The GRU update equations are shown below:

$$r(t) = \sigma(W_{ir}x(t) + W_{hr}h(t-1)) \quad (2)$$

$$u(t) = \sigma(W_{iu}x(t) + W_{hu}h(t-1)) \quad (3)$$

$$c(t) = \tanh(W_{ic}x(t) + r(t) \odot W_{hc}h(t-1)) \quad (4)$$

$$h(t) = (1 - u(t) \odot c(t) + u(t) \odot h(t-1)) \quad (5)$$

where $x(t)$ is the current input, $r(t)$ the current reset gate, $u(t)$ the current update gate, $c(t)$ the candidate hidden state, and $h(t-1), h(t)$ the previous and current updated hidden states. Ws include both weights and biases of the GRU layer, σ is sigmoid activation function, and $\odot$ is Hadamard product.

2) Autoencoder: An autoencoder (AE) is employed during the training process. The AE consists of two components - an encoder and a decoder. The encoder maps the input x to a latent space f and the decoder reconstructs the input $\hat{x}$ based on f. The training loss function of the AE includes a term that compares the difference between the input and the reconstructed output.

3) AEGRU inference architecture: Fig. 1 shows the architecture consisting of a GRU layer of 32 neurons as the core, flanked by two fully connected (FC) layers. The FC layer before the GRU layer has 32 neurons while the FC layer after the GRU layer has 2 neurons. Our inference architecture has a memory footprint of 45.5 kB and thus could be deployed on edge devices. The network inference formulation is as follows:

$$f = \text{FC}_{\text{upstream}}(x') \quad (6)$$

$$h = \text{GRU}(f, h_0) \quad (7)$$

$$v = \text{FC}_{\text{downstream}}(h) \quad (8)$$

where $x' \in \mathbb{R}^{N \times C_i}$ is the preprocessed input, $f \in \mathbb{R}^{N \times C_f}$ the latent factor, $h, h_0 \in \mathbb{R}^{C_h}$ the GRU hidden state, and $v = (v_x, v_y) \in \mathbb{R}^2$. Other variables are defined in Tab. I.

4) AEGRU training: In addition to the backbone FC-GRU-FC components for inference, an auxiliary branch is included and used specifically only during training. This branch bifurcates from the output of the upstream FC layer. It is used to reconstruct the firing rate of the input spikes computed using the spike count within a window. This auxiliary branch combined with the upstream FC layer, forms an autoencoder (AE) to better align the intermediate latent factors with the underlying features of the input spike counts. The mean of the latent factor f, μ_f, as generated by the upstream FC layer, serves as the input to the GRU layer in the forward phase. The variance of f, σ_f^2, is generated by the 2-layer FC network (FC1 and FC2). The sampled f from the μ_f and σ_f^2, is passed to the GRU layer. On the side of the auxiliary branch, f is
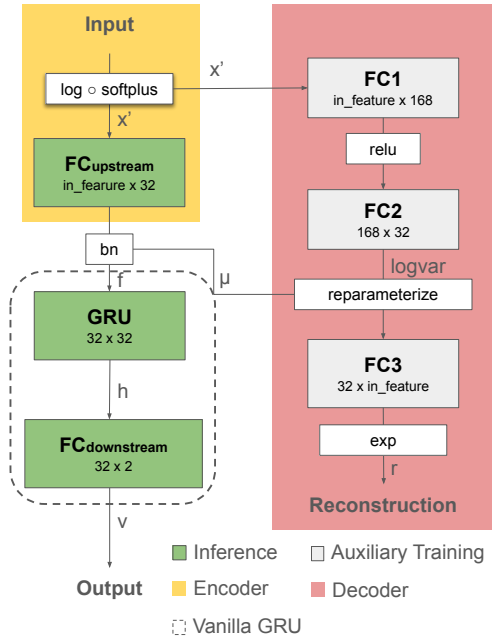
Fig. 1: Our AEGRU network architecture. Green blocks represent the inference pass, and the grey blocks compose the auxiliary branch used during training. The weight dimensions are indicated in the blocks. 'bn'=batch normalization.

used to reconstruct the firing rate r using FC3 followed by an exponential transformation. The forward pass for auxiliary training is formulated as below:

$$\mu_f = \text{FC}_{\text{upstream}}(x') \tag{9}$$

$$\log \sigma_f^2 = \text{FC2}(\text{ReLU}(\text{FC1}(x'))) \tag{10}$$

$$f \sim \text{Gaussian}(f|\mu_f, \sigma_f^2) \tag{11}$$

$$r = \exp(\text{FC3}(f)) \tag{12}$$

The training loss function $L = w_v L_v + w_x L_x$ is comprised of two terms: L_v, the mean square error (MSE) between the velocity prediction $\hat{v}$ and ground truth v; and L_x, the Poisson negative log-likelihood (Poisson NLL) between the reconstructed firing rate, r, and the input x as used for auxiliary training:

$$L_v = \frac{1}{t} \sum_t ||\hat{v}_t - v_t||_2^2 \tag{13}$$

$$L_x = -\frac{1}{t} \sum_t \log(\text{Poisson}(x_t|r_t)) \tag{14}$$

D. Data preprocessing

1) Neural activity representation: Following the raw data processing and sub-window method in [3], we bin the data samples into windows of size WS. Note that each WS represents $WS \times 4$ ms of spike data. As illustrated in Fig. 2, the input feature is defined by first choosing WS number of data samples, then the data samples (firing rates) in each window are summed and N number of steps of such windows are
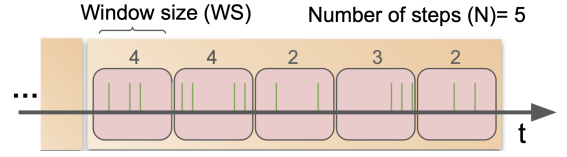

Fig. 2: Data pre-processing pipeline. Each input represents $WS \times N \times 4$ ms of neural activity.

taken to represent the neural activity within the time interval of $WS \times N \times 4$ ms.

For every prediction, we use the latest $WS \times N$ consecutive data samples which are then collapsed in N-dimensions for each channel after summation within the sub-windows spanned WS samples, as shown in Fig. 2. For the next prediction, we take a stride of 4ms, i.e. 1 data sample. The final input is of dimension $\mathbb{R}^{N \times C_i}$.

2) Softplus and logarithm: A softplus function is applied to the spike count data so that all values are positive. Assuming that the data is Poisson distributed, a logarithm is used to transform the data into a Gaussian-like distribution. This introduced bias helps to uncover the underlying Gaussian distributed latent factors through a simple linear transformation.

$$\text{Softplus}(x) = \log(1 + \exp(x)) \tag{15}$$

$$x' = \log(\text{Softplus}(x)) \tag{16}$$

E. Training details

Separate models are trained for each of the six recordings. The data split is 50% for the training set; the middle 25% and the last 25% form the validation and test sets respectively. Models are trained over 50 epochs using the training set. The Adam optimizer is used with a learning rate of 0.001 and a weight decay of 0.001. The cosine annealing learning rate scheduler is used to adjust the learning rate dynamically over the epochs. Fine-tuning is conducted over 10 epochs after weight pruning (see Sec. II-F1) using both training and validation sets.

F. Post-training weight sparsification

1) Pruning: To reduce the computational complexity, the model weights were sparsified using the L1 unstructured post-training pruning technique. A target sparsity rate of 50% was specified for both the FC and GRU weights. Following pruning, the model was fine tuned over several epochs to restore task accuracy.

2) Post-rounding weight quantization: The weights were further quantized into an 8-bit fixed-point format, where 1 bit was allocated for the integer part and the remaining 7 bits for the fractional part. For each weight, W, the quantized weight was achieved using : $W_q = \text{round}(2^{\text{qf}} \cdot W)/2^{\text{qf}}$ where qf is the number of fractional bits.

III. RESULTS

A. Hyperparameter optimization

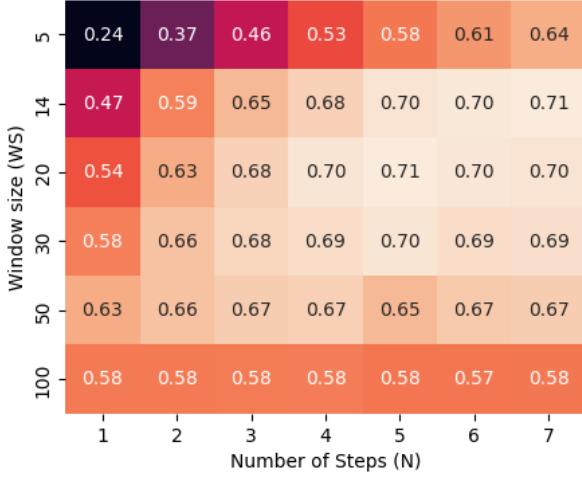*1) Windowing method:* To complete one model execution, the number of data samples for each channel is $WS \times N$

Fig. 3: Heat map of mean R^2 on the test sets of the six recordings, evaluated across window sizes (WS) and number of steps (N).

(see Table I and Fig. 2). Further, N affects the computational complexity, as it sets the number of recurrent steps within the GRU layers. A smaller $WS \times N$ (less data) may cause the model to predict without context of the general kinematics, but a larger $WS \times N$ (longer history of neural activity) may challenge our small model fail to learn. Thus, there is a trade-off to be optimized between the prediction accuracy and the requirements for memory storage and computation complexity.

A hyperparameter search was carried out for a range of WS and N settings. The mean R^2 results in Fig. 3 show that the $WS = 14, N = 7$ configuration achieved the highest mean R^2 score of 0.709 ± 0.04. The default configuration, $WS = 20, N = 5$, had a close score of 0.707 ± 0.05 but used 29.4% fewer MACs than the $WS = 14, N = 7$ configuration.

2) Pruning rate: Following the pruning method described in Sec. II-F1, Fig. 4 shows that R^2 is maintained even with increasing pruning rate until a target pruning rate (TPR) of 0.6. For the reported numbers in Table II, the TPR=0.5. The MACs are reduced by 41.4% at a TPR=0.5 compared to TPR=0. The footprint calculation in Neurobench does not rule out pruned weights thus the footprint size remained the same.

B. Comparison to baseline models

Neurobench [10] provides four baseline models: ANN2D, ANN3D, SNN3D and SNN_streaming on this dataset. These 4 baseline model architectures are further studied in [6]. Table II
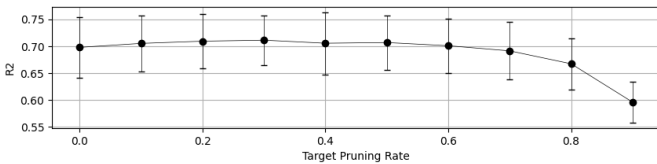

Fig. 4: Change in R^2 when the target pruning rate (TPR) is increased. The mean R^2 starts to decrease when TPR> 0.6.

TABLE II: Performance of baseline models and AEGRU. Mean and standard deviation are calculated for all recordings.

Model	R2	Footprint (kB)	MACs + ACs (k)
ANN2D	0.58 ± 0.06	27.2 ± 6.3	5.0 ± 1.2
ANN3D	0.62 ± 0.05	137.8 ± 43.2	11.5 ± 3.9
SNN3D	0.63 ± 0.05	34.0 ± 9.0	38.1 ± 11.6
SNN Streaming	0.58 ± 0.06	29.2 ± 9.6	0.4 ± 0.2
AEGRU w/o pruning	0.70 ± 0.06	45.5 ± 6.3	42.8 ± 4.1
AEGRU 50% pruning	0.71 ± 0.05	45.5 ± 6.3	25.1 ± 3.3

TABLE III: Comparison of R^2 scores between AEGRU and vanilla GRU. The means and standard deviations are calculated on 5 runs for each recording.

Recording	vanilla GRU	AEGRU
indy_20160622_01	0.74 ± 0.006	0.76 ± 0.014
indy_20160630_01	0.64 ± 0.011	0.65 ± 0.008
indy_20170131_02	0.75 ± 0.004	0.76 ± 0.006
loco_20170131_02	0.61 ± 0.017	0.68 ± 0.013
loco_20170215_02	0.60 ± 0.009	0.62 ± 0.014
loco_20170301_05	0.66 ± 0.013	0.70 ± 0.015
Average	0.67 ± 0.005	0.70 ± 0.008

shows the trade-off between the R^2 accuracy and the other 2 metrics for our AEGRU model and the baseline models. The R^2 scores of these baseline models come from the Neurobench code harness https://github.com/NeuroBench/neurobench with the following WS and N combination: $WS = 50, N = 1$ for ANN2D, $WS = 7, N = 7$ for ANN3D and SNN3D, and $WS = 1, N = 1$ for SNN_streaming. The AEGRU model uses the optimized hyperparameter values as described in Sec. III-A, i.e., $WS = 20, N = 5$ and post-training TPR of 0.5. The mean and standard dedication are calculated for 1 run per recording. Overall, our model achieves the highest mean R^2 of 0.71.

In comparison to the vanilla GRU architecture, the AEGRU model showed a higher R^2 score for all 6 recordings. The mean and std deviation are computed over 5 runs with different random seeds (see Table III). The vanilla GRU model comprises only the GRU layer and the downstream FC layer (see Fig. 1). The upstream FC layer helps to reduce the memory footprint because of the reduced feature dimension to the GRU layer. A possible reason for the higher prediction accuracy is that the model generalizes better and is less prone to overfitting.

IV. DISCUSSION

Our AEGRU model achieved a higher R^2 compared to the baseline models through a combination of model design, auxiliary training, and data preprocessing hyperparameters. Furthermore, we found the data preprocessing hyperparameters were vital to the performance. A self-adapting preprocessing hyperparameter method will be studied in future works. Moreover, the sparsified AEGRU could be deployed on customized hardware designed for exploiting weight sparsity [14].

V. ACKNOWLEDGEMENT

REFERENCES

[1] E. M. Maynard, C. T. Nordhausen, and R. A. Normann, "The Utah intracortical electrode array: a recording structure for potential brain-computer interfaces," *Electroencephalography and clinical neurophysiology*, vol. 102, no. 3, pp. 228–239, 1997.

[2] C. Gao, R. Gehlhar, A. D. Ames, S.-C. Liu, and T. Delbruck, "Recurrent neural network control of a hybrid dynamical transfemoral prosthesis with edgedrnn accelerator," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 5460–5466.

[3] B. Zhou, P.-S. V. Sun, and A. Basu, "ANN vs SNN: A case study for neural decoding in implantable brain-machine interfaces," 2023. [Online]. Available: https://arxiv.org/abs/2312.15889

[4] N. Ahmadi, T. G. Constandinou, and C.-S. Bouganis, "Robust and accurate decoding of hand kinematics from entire spiking activity using deep learning," *Journal of Neural Engineering*, vol. 18, no. 2, p. 026011, 2021.

[5] J. Hadorn, Z. Wang, B. Rueckauer, X. Chen, P. R. Roelfsema, and S.-C. Liu, "Fast temporal decoding from large-scale neural recordings in monkey visual cortex," in *4th Shared Visual Representations in Human and Machine Intelligence (SVRHM) NeurIPS Workshop*, 2022.

[6] B. Zhou, P.-S. V. Sun, and A. Basu, "ANN vs SNN: A case study for neural decoding in implantable brain-machine interfaces," *arXiv preprint arXiv:2312.15889*, 2023.

[7] S. Shaikh, R. So, T. Sibindi, C. Libedinsky, and A. Basu, "Towards intelligent intracortical BMI (i^2 BMI): Low-power neuromorphic decoders that outperform Kalman filters," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 6, pp. 1615–1624, 2019.

[8] T. Hosman, M. Vilela, D. Milstein, J. N. Kelemen, D. M. Brandman, L. R. Hochberg, and J. D. Simeral, "BCI decoder performance comparison of an LSTM recurrent neural network and a Kalman filter in retrospective simulation," in *2019 9th International IEEE/EMBS conference on neural engineering (NER)*, 2019, pp. 1066–1071.

[9] N. Ahmadi, T. G. Constandinou, and C.-S. Bouganis, "Decoding hand kinematics from local field potentials using long short-term memory (LSTM) network," in *2019 9th international IEEE/EMBS conference on neural engineering (NER)*, 2019, pp. 415–419.

[10] J. Yik, K. V. d. Berghe, D. d. Blanken, Y. Bouhadjar, M. Fabre, P. Hueber, D. Kleyko, N. Pacik-Nelson, P.-S. V. Sun, G. Tang *et al.*, "Neurobench: A framework for benchmarking neuromorphic computing algorithms and systems," *arXiv preprint arXiv:2304.04640*, 2023.

[11] J. E. O'Doherty, M. M. Cardoso, J. G. Makin, and P. N. Sabes, "Nonhuman primate reaching with multichannel sensorimotor cortex electrophysiology," *Zenodo http://doi. org/10.5281/zenodo*, vol. 583331, 2017.

[12] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[13] Z. Wang, S. Wang, C. Lafaye, M. Saubade, V. Gremeaux, and S.-C. Liu, "Person identification using deep neural networks on physiological biomarkers during exercise," in *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2022, pp. 193–197.

[14] C. Gao, T. Delbruck, and S.-C. Liu, "Spartus: A 9.4 TOp/s FPGA-based LSTM accelerator exploiting spatio-temporal sparsity," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 1, pp. 1098–1112, 2022.