Decoding handwritten characters from neural activity using a recurrent neural network

Adil Bukhari, Tyler Singer-Clark, Tarun Tiwari, Karamjeet Gulati

29 November 2023

Github: Link

1 Abstract

Brain-computer interfaces (BCI) offer a method of restoring communication capabilities for people with paralysis due to neurological diseases or injuries such as ALS, brainstem stroke, or spinal cord injury. Typically, a user with paralysis controls a BCI by *attempting* movements (despite being unable to perform those movements physically), such as reaching and grasping, or handwriting characters. A prior study collected a BCI handwriting dataset containing one subject's neural data recorded during an explicitly-cued single characters task, and applied k-nearest neighbors (KNN) to the neural data to classify which character the subject was attempting to write. The first goal of our study was to assess whether a recurrent neural network (RNN) could improve decoding accuracy relative to KNN in this explicitly-cued single characters scenario. The second goal of our study was to characterize the impact of certain factors (architecture choices, number of recording electrodes, number of training trials) on the performance of each model when applied to this BCI handwriting data, to guide future efforts to improve classification performance. We applied an RNN to the 31-class classification problem of decoding explicitly-cued single characters (26 letters, 5 special characters), and compared it to our best attempt to replicate the authors' KNN results, as well as to a logistic regression (LR) and feedforward neural network (FFNN). The RNN achieved an accuracy of 85.7%, compared to 80.0% for KNN, 87.8% for LR, and 75.3% for FFNN. Though the RNN outperformed the KNN and FFNN, LR was the best performing model we tried, suggesting that deep learning techniques may not be necessary in the current data paradigm. We also ran each model with a range of numbers of electrodes and numbers of training trials, in order to project how each model may compare in the future with more available data in each dimension. Though it appears qualitatively that the RNN may be better than other models at taking advantage of additional electrodes, it is not yet quantitatively conclusive and remains an open question for future research.

2 Introduction

Brain-computer interfaces (BCI) offer a method of restoring communication capabilities for people with paralysis due to neurological diseases or injuries such as ALS, brainstem stroke, or spinal cord injury[1, 2, 3, 4]. An intracortical BCI (iBCI) consists of a neural recording device surgically implanted into a user's brain, a computer which receives the recorded neural activity and decodes from it the user's intention, and finally an effector which takes the decoder's output and performs a task on behalf of the user. Commonly, the recorded brain area is the motor cortex, which has been shown to provide meaningful information about a user's attempted movements even if their paralysis stops them from physically performing those movements[5].

In a 2021 study[6], researchers demonstrated a handwriting-based control strategy in which an iBCI user attempted arm and hand gestures as if writing on a table in front of them, and the letter or symbol they attempted to draw was decoded and displayed on a computer screen. There are at least two distinct task paradigms under which to train and apply a neural decoder for discrete gesture classification: explicitly-cued (where a single character at a time is cued with an explicit go cue for each one) and self-paced (where the prompt to the user is a sequence of characters and the decoder must infer the exact timing of each attempted gesture as the user goes through the whole sequence). The aforementioned study included both types of tasks (self-paced sentences and explicitly-cued single characters). They achieved 0.94 character classification accuracy on the explicitly-cued single characters data using a k-nearest neighbors (KNN) model. Then they moved on to the self-paced sentences data and achieved a 0.99 character classification accuracy using a recurrent neural network (RNN) model plus a language model rescore step (”autocorrect”) which took advantage of the linguistic structure of the sentence prompt. However, there are potential iBCI applications (drawing, robot arm control) that do not have the benefit of linguistic structure, so there is still value in exploring and improving performance on the single characters data, beyond the demonstrated 0.94.

Many studies have shown deep learning techniques such as RNN to perform well for similar iBCI decoding applications[7, 8, 9, 10], so we wondered if the same would apply to our explicitly-cued single characters scenario. Therefore, the first goal of our study was to assess whether an RNN could improve decoding accuracy relative to KNN in this explicitly-cued single characters scenario. The second goal of our study was to characterize the impact of certain factors (architecture choices, number of recorded electrodes, number of training trials) on the performance of an RNN when applied to this BCI handwriting data, to guide future efforts to improve classification performance.

In this report, we first present the task setup and available dataset. We then detail our preprocessing and decoding pipeline, both for our RNN as well as for our KNN, LR, and FFNN. Finally, we present and compare the results of applying each decoding model to the dataset, including the best performance for each model as well as the performance of each model while varying the amount of training data in terms of number of electrodes and number of trials.

3 Methods

3.1 Dataset

The data in this study were originally collected by Stanford University's Neural Prosthetics Translational Lab (NPTL) and made publicly available at the following link. The data were collected from BrainGate clinical trial participant T5, a man with tetraplegia due to spinal cord injury who cannot move his hands except for some minimal, non-functional twitching. Prior to this study, T5 received two intracortical 96-electrode Utah arrays (192 electrodes total) implanted in the ”hand-knob” region of dorsal precentral gyrus, a brain area known to have neural activity related to hand movements[11].

T5's neural data were recorded while he performed 3600+ trials of an instructed-delay task across 10 different days, each day consisting of multiple 96-trial blocks. In this task, each trial consisted of a character (letter or symbol) appearing on a computer screen next to a red square (the ”delay cue”), followed by the square turning green (the ”go cue”) indicating that T5 should attempt a hand gesture as if handwriting the prompted character (Fig 1A).

From each of the 192 electrodes' voltage signals, threshold crossings were extracted (instances when the signal ”spiked” beyond a certain threshold) and binned into time bins of 10 ms, serving as a proxy for neurons' firing rates. The resulting time series of 192-length vectors was the format given to us, and this was the input to our classification problem. We were also given which character was prompted in each trial, and this was the output of our classification problem.

The list of characters consisted of 26 lowercase Latin alphabet letters, plus the 5 special symbols question mark (?), comma (,), apostrophe (‘), tilde (̃), and greater than (¿), totaling the 31 classes in our classification problem.

3.2 t-SNE visualization

As a starting point, to confirm we saw similar separability between classes as seen in the original study, we used t-SNE to project each trial's neural data into 2D space (Fig 1B). Each input to t-SNE was a single trial's flattened threshold crossings time series clipped to a 1.5 second time window, starting 100 ms after the go cue to allow for the user's reaction time (192 electrodes x 150 time bins = 28800 features). The t-SNE model was fit using the TSNE object from the Python package sk-learn with perplexity = 7. All trials in the t-SNE plot in Fig 1B are from the same day (9 blocks of 96 trials each).

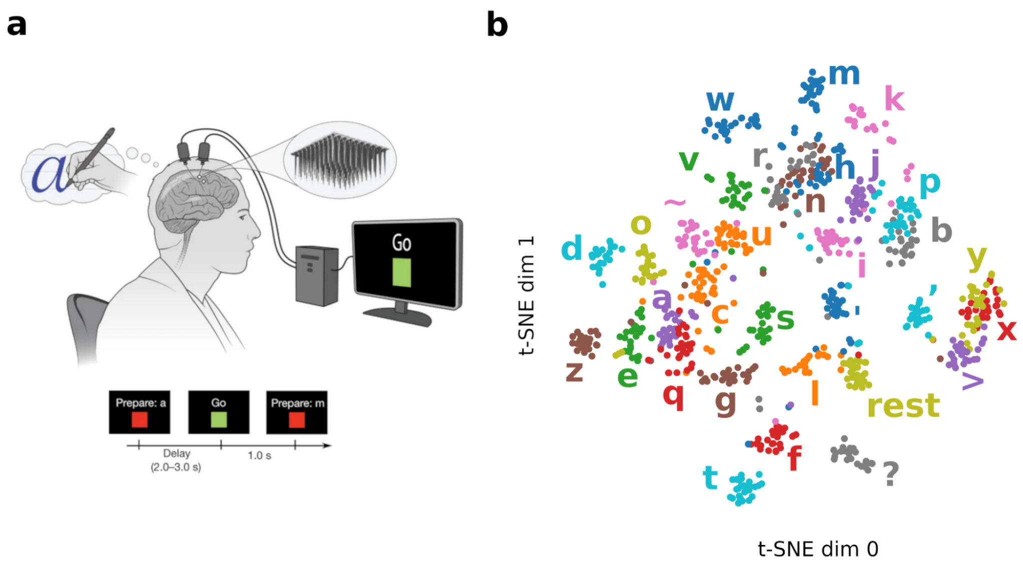


Figure 1: Experimental setup and dataset
a. The experimental explanation figure from Willett et al. 2021 (Figure 1, Panel a), directly copied here only for the reader's convenience. In the experiment, a paralyzed user's neural activity was recorded while they viewed single-character prompts on a computer screen and attempted the corresponding handwriting movements in their mind.
b. Our own replication of Figure 1, Panel e from Willett et al. 2021. t-SNE 2D projections of neural data during the task. Each point represents a single trial. As expected, trials cluster by the character cued, and characters with similar shapes are close together, e.g. the triplet (r, n, h) or the pair (b, p).

3.3 Preparation for training

The following steps were common for all the models. First, trials from all days were randomly split into train and test sets, in proportions of 0.8 and 0.2 respectively. Using only trials in the train set, the mean and standard deviation of the threshold crossings time series were calculated per electrode, and per block (to account for neural activity shifts over time). These means and standard deviations were then used to z-score each electrode's threshold crossings time series to have mean 0 and variance 1. Then each trial was clipped to a 1.5 second time window, starting 100 ms after the go cue to allow for the user's reaction time. Each electrode's threshold crossings time series (or the principal component time series in the case KNN, as described later) was smoothed using a Gaussian kernel with a width of 3.0 bins.

3.4 LR model

To train the LR model, each trial's window of threshold crossings was flattened to a 28800-length vector (192 electrodes x 150 time bins = 28800 features). With these features vectors as the input vectors and the prompted characters as the output labels, a model was fit on the train set using the Logistic Regression object from the Python package sklearn with solver "newton-cg" (other solvers errored out). This model was then applied to the held-out test set to calculate the accuracy.

3.5 KNN model

To train the KNN model, first the dimension of each time bin's z-scored threshold crossings vector was reduced from 192 to 25 using principal components analysis (PCA). Each trial's window of principal component values was flattened to a 3750-length vector (25 principal components x 150 time bins = 3750 features). With these features vectors as the input vectors and the prompted characters as the output labels, a model was fit on

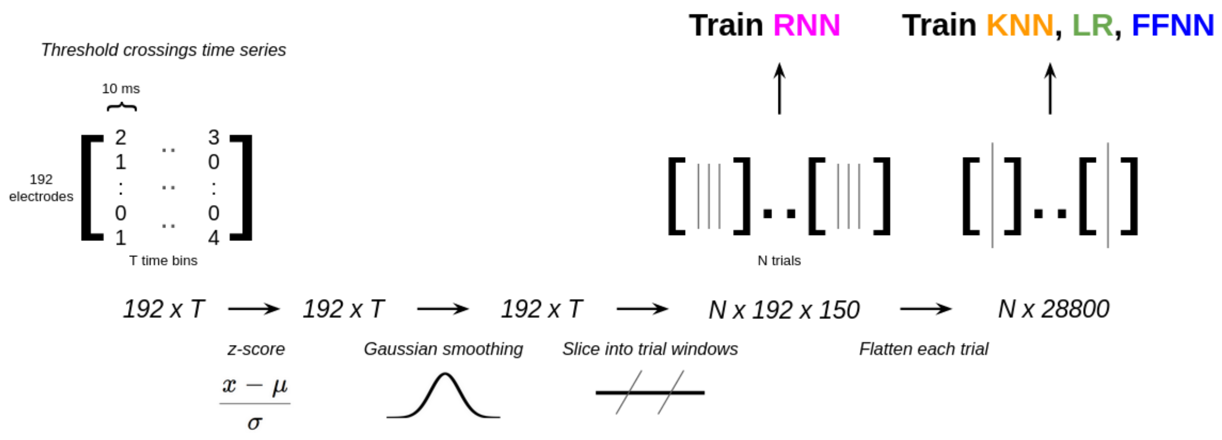


Figure 2: Pre-processing pipeline
The preprocessing steps to get from the given dataset to the inputs to our models. For all models, we start with a time series of threshold crossings for 192 electrodes, then z-score (normalize), smooth, and slice into windows relative to reach trial's go cue. An exception omitted from this figure for simplicity is that for KNN we fit a PCA model and use the top 25 principal components in place of the 192 electrode time series. All models except the RNN flatten each trial's time series because they operate on 1D vectors, whereas the RNN operates on a sequence of vectors so the flattening step is not needed.

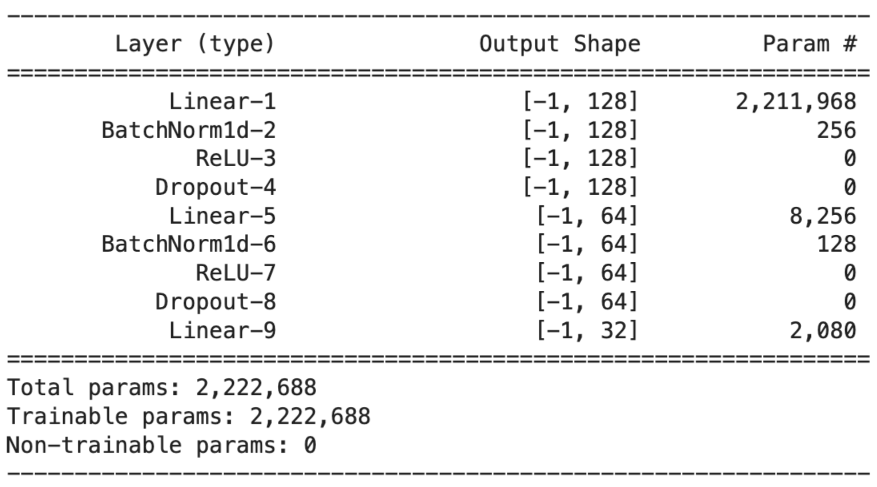
the train set using the KNeighbors Classifier object from the Python package sklearn with the number of neighbors = 5. This model was then applied to the held-out test set to calculate the accuracy.

3.6 FFNN model

To design the FFNN model, we employed a progressive approach, evolving from a Simple Classifier to an Enhanced Neural Network. Initially, the Simple Classifier, a base linear model with a single fully-connected layer, was trained using the Adam optimizer with a learning rate of 0.001. The model was trained over 10 epochs, tracking training and testing accuracy using Tensorboard. Building upon this, we transitioned to training the model in batches, we introduced batch normalization and ReLU activation functions to enhance learning, and incorporated dropout layers to prevent overfitting. The batch processing significantly improved handling of the large dataset. Finally, the Enhanced Neural Network (referred to as FFNN in Fig 5), with a more complex architecture including multiple hidden layers and varying dropout rates, was trained using the SGD optimizer with momentum = 0.9. This model underwent a more extensive training regime of 20 epochs, with accuracy and loss metrics recorded for each epoch to monitor performance improvements.

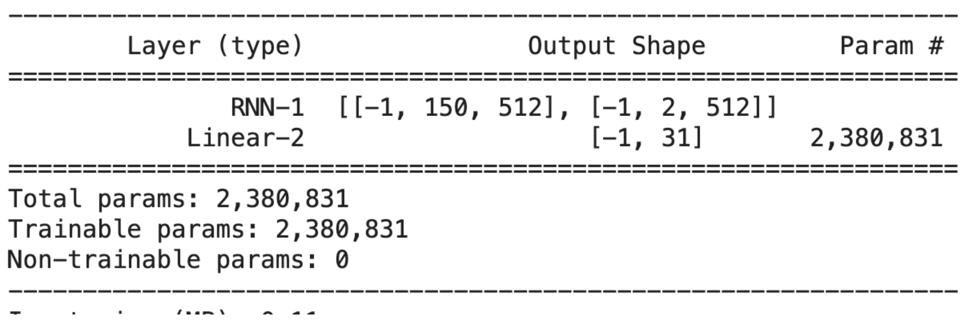
3.7 RNN model

For the RNN model, unlike the other models, we did not flatten each trial's neural data, since RNNs accept sequences of vectors as input. The RNN model, implemented using PyTorch, allowed flexibility in choosing the hidden size, number of layers, and dropout, with specific parameters set as follows: a hidden size of 512, two layers, and a dropout of 0.5. The training process involved stochastic gradient descent (SGD) with a learning rate of 0.0005, momentum set to 0.99, and weight decay at 0.001. The model underwent training for 50 epochs, incorporating data augmentation by adding random noise (with a standard deviation of 0.1) to every z-scored value at every time bin. The augmentation aimed to avoid overfitting while training on the same small dataset for many epochs.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                 [-1, 128]       2,211,968
       BatchNorm1d-2                 [-1, 128]             256
              ReLU-3                 [-1, 128]               0
           Dropout-4                 [-1, 128]               0
            Linear-5                  [-1, 64]           8,256
       BatchNorm1d-6                  [-1, 64]             128
              ReLU-7                  [-1, 64]               0
           Dropout-8                  [-1, 64]               0
            Linear-9                  [-1, 32]           2,080
================================================================
Total params: 2,222,688
Trainable params: 2,222,688
Non-trainable params: 0
----------------------------------------------------------------
```

Figure 3: *Summary of the FFNN model architecture (only the Enhanced Neural Network version)*



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
               RNN-1  [[-1, 150, 512], [-1, 2, 512]]
            Linear-2                  [-1, 31]       2,380,831
================================================================
Total params: 2,380,831
Trainable params: 2,380,831
Non-trainable params: 0
----------------------------------------------------------------
```

Figure 4: *Summary of the RNN model architecture*

3.8 Model evaluation

For each model described above, we ran the entire process (from the random splitting of train and test sets through preprocessing and training) independently and then evaluated the resulting model on the test set to get an accuracy value. This was repeated independently 5 times, and the accuracies from all 5 repetitions were averaged to yield a mean accuracy for each model.

3.9 Varying Data size

In addition to evaluating each model with all electrodes and a full train set, we also systematically reduced the data in both of these dimensions and evaluated the models on these reduced datasets. Due to time constraints, we did not include the FFNN in this analysis. For the other 3 models, we ran the same procedure as the main analysis described earlier, but with a random subset of 48, 96, and 144 electrodes. Similarly, we evaluated each model with a random subset of 300, 600, 900, 1200, 1500, 1800, and 2100 train trials. Though we again ran each model 5 times to get mean accuracies at each of these points, the exception was the RNN which we only ran twice (again due to time constraints).

4 Results

4.1 Model accuracy

A comparison of the models' accuracies is shown in Fig 5. The RNN achieved a test accuracy of 85.7% with a standard deviation of 1.1. Our KNN achieved a test accuracy of 80.0% with a standard deviation of 1.0.

LR achieved the highest test accuracy of 87.8% with a standard deviation of 1.6. The FFNN model achieved a test accuracy of 75.3% with a standard deviation of 0.9. Chance performance (calculated using shuffled labels) came out to 3.1%.

4.2 Varying Data size

To explore the impact of varying data sizes on model performance, we systematically reduced the number of training trials and electrodes. Fig 6A and 6B show the resulting accuracies for each model under these data-limited conditions.

Increasing the number of electrodes continued to improve accuracy in all models all the way through the maximum number of electrodes available in this study (192), i.e. no models plateaued. Though only qualitative and not quantitative, LR appeared to exhibit a sharper elbow and flatter tail than RNN and KNN.

Increasing the number of train trials also continued to improve accuracy in all modes all the way through the maximum number of train trials used (2100), i.e. no models plateaued.

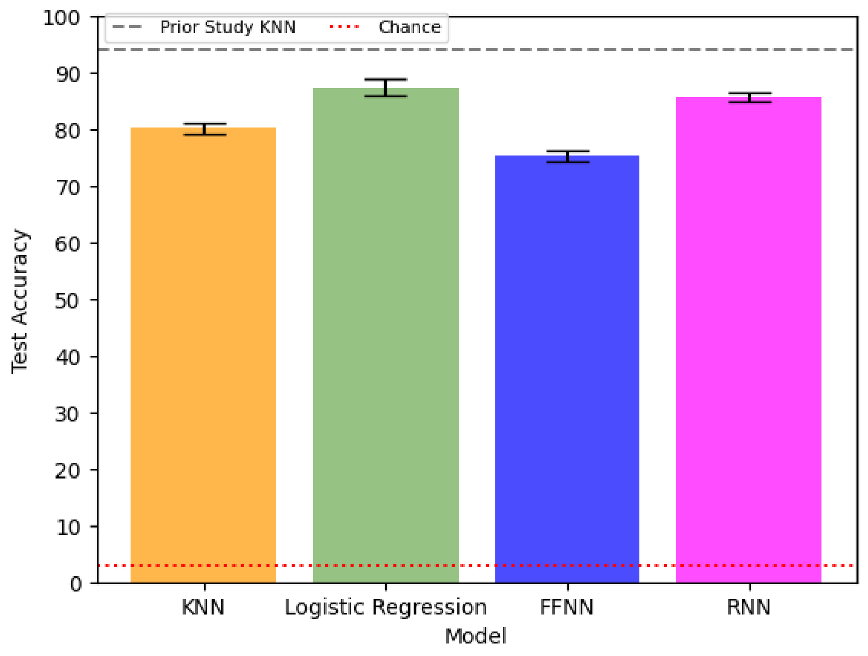
5 Discussion

None of our models reached the 0.94 mark set by the original study's KNN. However, the original study provides minimal detail on the methods used and the evaluation process that yielded that 0.94 mark because the explicitly-cued single characters task was not the primary focus of that study. This made it difficult to ensure a robust comparison between our results and the original study's results, highlighted by the fact that we were not able to match their mark with our own model of the same type (KNN). Even our t-SNE visualization does not separate as definitively as that of the original study, so there may be an impactful preprocessing step they are performing that we are not. Further discussion focuses on comparing our own models.

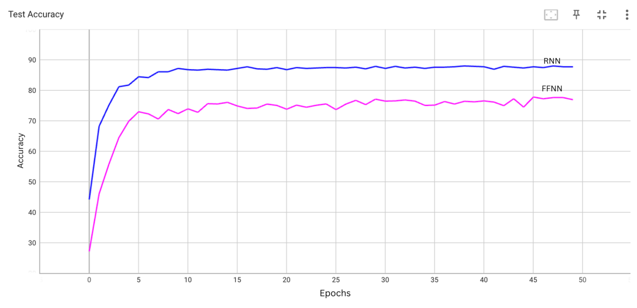
As seen in Fig 5A, the RNN outperformed our KNN and FFNN models, but LR was our best performing model. This suggests that deep learning techniques may not be necessary in the current data paradigm, as the linear baseline did just as well as anything else we tried.

We also aimed to get at the question of how each model's performance would scale with more data, both in terms of number of recording electrodes as well as training trials. As seen in Fig 6A, none of our tested models plateaued before 192 electrodes (all the electrodes we had available in this study), suggesting that neural recording hardware advances that increase the number of recording electrodes will enable better performing handwriting BCI, but deep learning techniques may not be required to take advantage of the additional data. However, qualitatively it appears the RNN was increasing faster as a function of number of electrodes than the other models, suggesting it would surpass them with more electrodes available, but a future direction for us is to verify this claim quantitatively by sampling the number of electrodes more granularly (i.e. more than just 4 data points per model) and fitting a function per model to project beyond 192 electrodes.

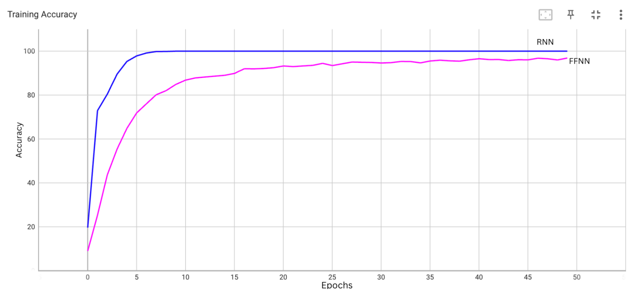
As for the number of train trials (Fig 6B), again no models we tested plateaued, suggesting they all would benefit from more than 2100 train trials. It is difficult to say qualitatively if the RNN is indeed increasing faster than the other models, in part due to the fact that each point in the RNN plot line is a single run instead of 5 runs (due to time constraints) so the precise shape seen in the plot may be due to random variation. Similar to the number of electrodes analysis, a future direction is to make this quantitative with more runs of the RNN and fitting a function per model to project beyond 2100 trials.



a.

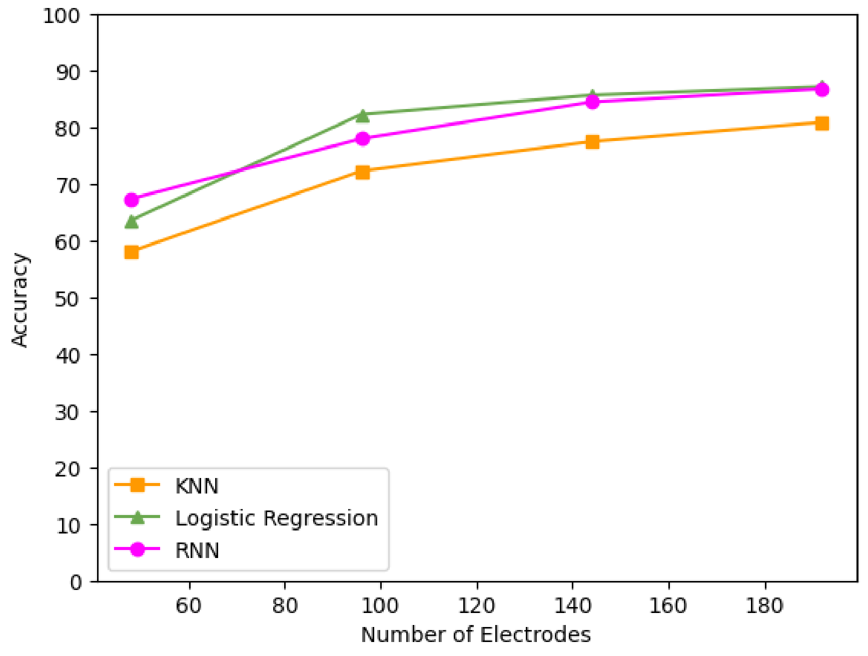


b.

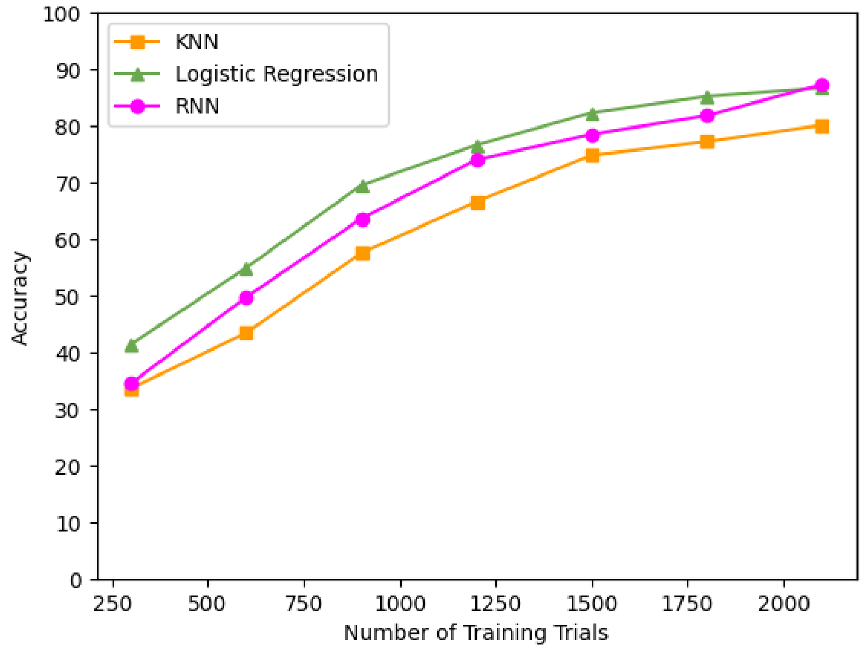


c.

Figure 5: Comparison of Model Classification Performance
a. Comparison of various models' test accuracies across 5 runs each. Bars represent the mean test accuracies, while error bars denote the standard deviation. The dashed line at 94% corresponds to the test accuracy reported in a prior study for KNN, providing a benchmark for comparison. The red dotted line near 3% represents chance-level performance (calculated using shuffled labels).
b. Accuracy of each neural network model on test sets during example training processes.
c. Accuracy of each neural network model on train sets during example training processes



a.



b.

Figure 6: Comparison of Model Performance on Varying Data Sizes
a. Comparison of various models' test accuracies using varying number of electrodes. For KNN and LR, each point represents a mean accuracy across 5 runs, while RNN was only 2 runs.
b. Comparison of various models' test accuracies using varying number of training trials. For KNN and LR, each point represents a mean accuracy across 5 runs, while RNN was only 1 run.

References

[1] Hochberg, L. R. et al. *Neuronal ensemble control of prosthetic devices by a human with tetraplegia.* Nature 442, 164–171 (2006).

[2] Collinger, J. L. et al. *High-performance neuroprosthetic control by an individual with tetraplegia.* The Lancet 381, 557–564 (2013).

[3] Pandarinath, C. et al. *High performance communication by people with paralysis using an intracortical brain-computer interface.* eLife 6, e18554 (2017).

[4] Nuyujukian, P. et al. *Cortical control of a tablet computer by people with paralysis.* PLOS ONE 13, e0204566 (2018).

[5] James C. Bennett, Quan J. Wang, David E. Robertson, Andrew Willett, F. R. et al. *Hand Knob Area of Premotor Cortex Represents the Whole Body in a Compositional Way.* Cell 181, 396-409.e26 (2020).

[6] Willett, F. R., Avansino, D. T., Hochberg, L. R., Henderson, J. M. & Shenoy, K. V. *High-performance brain-to-text communication via handwriting.* Nature 593, 249–254 (2021).

[7] Metzger, S. L. et al.*A high-performance neuroprosthesis for speech decoding and avatar control.* Nature 1–10 (2023) doi:10.1038/s41586-023-06443-4.

[8] Willett, F. R. et al.*A high-performance speech neuroprosthesis.* Nature 620, 1031–1036 (2023).

[9] Hosman, T. et al. *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)* 1066–1071 (2019).doi:10.1109/NER.2019.8717140.

[10] Pandarinath, C. et al.*Inferring single-trial neural population dynamics using sequential auto-encoders.* Nat. Methods 15, 805–815 (2018).

[11] Irwin, Z. T. et al.*Neural control of finger movement via intracortical brain–machine interface.* J. Neural Eng. 14, 066004 (2017).