JOURNAL OF
EMERGING INVESTIGATORS

Article

Comparing transformer and RNN models in BCIs for handwritten text decoding via neural signals

Aashna Hari[1], Joseph Isaacs[2]
[1]Horace Mann School, New York, New York
[2] Computer Science and Technology, University of Cambridge, Cambridge, United Kingdom

SUMMARY

A Brain-Computer Interface (BCI) is a technology that enables the user to operate devices or manipulate objects solely through their brain activity, which is especially useful for paralyzed individuals in regaining communication. The purpose of this study is to explore the use of a custom transformer model in BCIs that translates the neural activity present when an individual with limited verbal and fine-motor skills attempts to handwrite. Previous studies have found that transformers performed better than recurrent neural networks (RNNs) in translation tasks similar to decoding neural signals into intended handwritten text. Due to this known benefit, we hypothesized that the transformer BCI would show promise through the recorded metrics. The neural signals of a tetraplegic individual attempting to handwrite were provided by existing research. We conducted four trials using training data with or without augmentation, as well as training the model to separately minimize training and validation loss. Compared with the original RNN BCI, the transformer model performed less favorably across all trials. Although the results do not indicate that the transformer currently outperforms an RNN BCI, it is important to note that further testing of the model's capabilities is necessary before determining whether transformers can enhance BCI communication. Future testing could include training the model with a larger and more preferable dataset, training for a longer duration, comparing training times between the RNN and transformer, or assessing how the transformer improves with an offline autocorrect feature.

INTRODUCTION

A brain-computer interface, or BCI, is a system that allows the user to control devices or objects using only their brain signals (1). A BCI is especially useful for those with paralysis or limited motor skills, as it can help them regain the ability to communicate. However, these systems can also be used non-medically, such as in gaming (2). Neural signals are generated when a person moves or thinks and can be directly detected and recorded from the brain. BCIs were originally dependent on recorded electroencephalogram (EEG) activity but have since expanded to varying degrees of invasiveness up to the use of an intracranial implant, which provides the best quality of signal recording (1).

BCIs use machine learning (ML) models to translate the recorded neural signals into a generated output, such as text. ML models learn how to translate these signals based on a correlated set of inputs and outputs that are provided in training. These models identify patterns and recognize what types of signals represent certain parts of an output. As it is trained, the model tests its own knowledge using a validation dataset consisting of data it has not yet encountered. The model then checks its predictions of the translation against the real answers and readjusts its weights on different features, which allows the model to decide which features are most important.

There are many types of ML models, and each performs best in certain situations, one of which is recurrent neural networks (RNNs). An RNN feeds its results back into itself, making this model best at working with temporal or sequential data such as stock prices or weather data (3). For example, an RNN model can predict that it is more likely to be raining the day after it rains than after a sunny day.

Another ML model is the transformer. The transformer model is a newer neural network that was proposed in 2017 by Vaswani et al. for sequential data (4). Unlike RNN, the transformer model has no recurrence as it is based on an encoder-decoder configuration connected by an attention mechanism to draw global dependencies between the input and output. Positional encoders are used to tag the data elements that are moving through the network (4). The attention units, located throughout the encoder and decoder layers, compare these tags and use mathematical functions to create a map of the relationships between elements (4). In the decoder, there is a multi-head attention layer that consists of several of these attention layers running in parallel (**Figure 1a**) (4). The inputs to each attention layer are queries, keys, and values (4). The dot product of the query with all keys is computed and scaled before a softmax function (which converts vectors to a probability distribution) is applied to obtain the weights for the values (4). This is essentially a compatibility function between the query and the corresponding key. The output of the layer is the weighted sum of the values (4).

A key difference between transformers and RNNs is their ability to parallelize (4). Parallelization refers to the technique of dividing a task into smaller sub-tasks that can be executed concurrently. In an RNN, parallelization is prevented because of the limitations of sequential computation (4). The RNN passes the time-series data into its layers step-by-step. Since layers remember previous inputs and use them in future predictions, there is a considerable training dependency: the state sequences at position t are a function of the earlier states at the previous position, t-1, as well as the input for position t (**Figure 2**). While including some attention mechanisms in an RNN (akin to those that exist in a transformer) can reduce this dependency, the majority of computing would

https://doi.org/10.59720/24-027

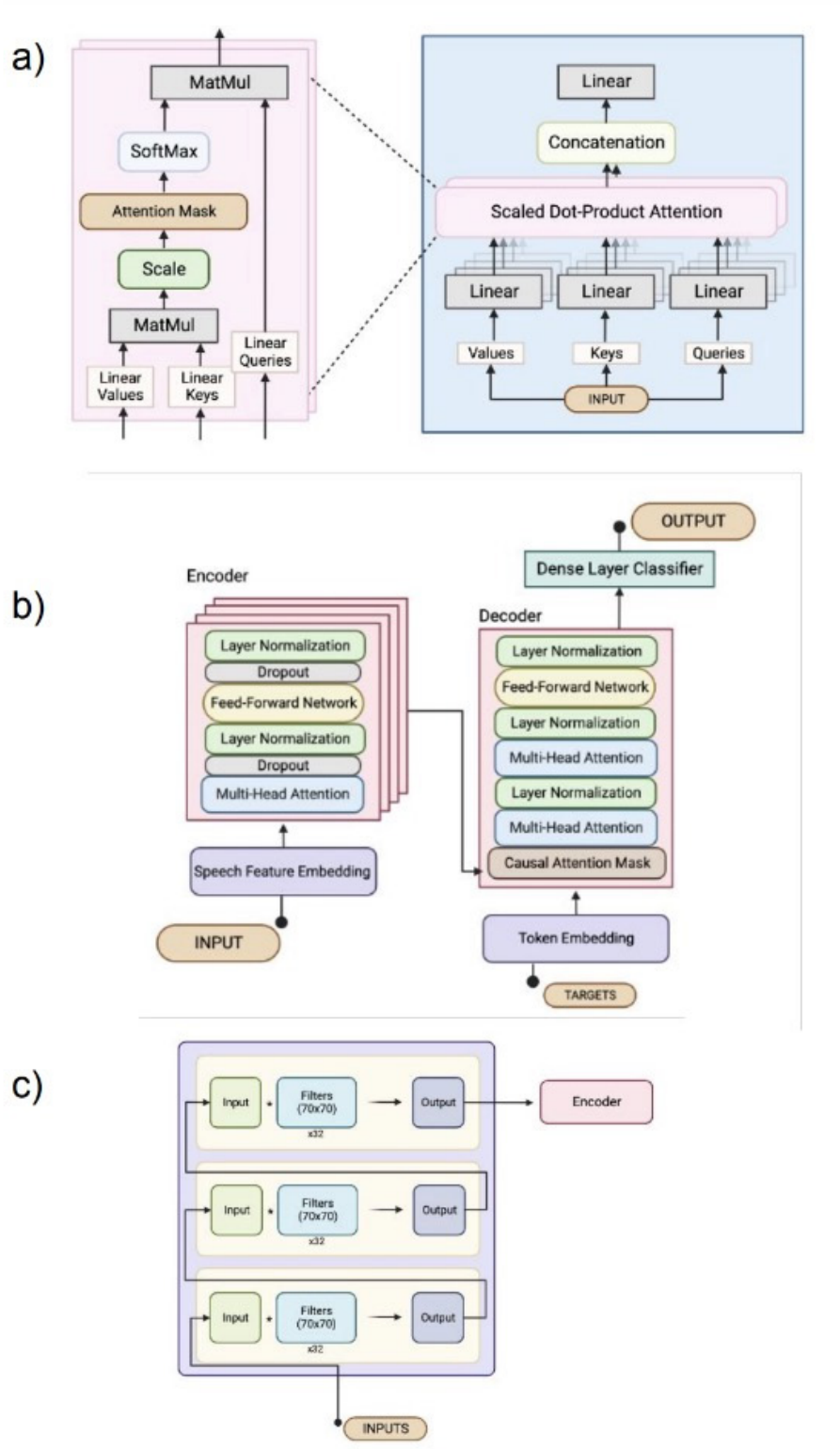

Figure 1: Graphical summary of overall transformer architecture. a) Flowchart representing the sequence of data processing in a multi-head attention layer, including the interior architecture of a scaled dot-product attention layer. The scaled dot-product attention layer included sublayers of matrix multiplication (MatMul), softmax function, and an attention mask in addition to the scaling layer. The key, value, and query tensors were all identical tensors that represented targets. b) Flowchart representing the sequence of data processing in the full transformer model. There are four identical encoder layers with one decoder layer. c) Flowchart representing the sequence of data processing in the speech feature embedding. Made up of three convolutional layers that feed into each other.

https://doi.org/10.59720/24-027

remain sequential (4). By contrast, the attention layers of a transformer can compute the attention function for a set of queries simultaneously, which increases the efficiency when training (4).

The transformer has also been shown to train faster than an RNN while still achieving the same or better accuracy (5). In an English-to-German translation task, the transformer produced the highest BLEU score when compared to the existing best results from other models, such as Convolutional Sequence to Sequence Learning, ByteNet, and Mixture-of-Experts (4). In another comparative study on interpreting EEG data as movements, the transformer also had the highest accuracy compared to other classifiers (EEGNet, DeepConvNet, and Support Vector Machines) (6).

Previous research in the field of intracortical BCIs showed that the current lowest error rate for characters from user-generated sentences in real-time was 8.54%, determined using softmax cross-entropy loss, which calculates how well the model's predictions matched the true characters (7). The participant in that study was a tetraplegic due to a high-level spinal cord injury (7). He was instructed to attempt to write as if his hand were not paralyzed, and the resulting neural activity was recorded by two previously implanted intracortical microelectrode arrays (**Figure 3**) (7). These signals were then fed into an RNN, which converted the neural population time series into a real-time output by thresholding the probabilities of each character at each time step (7). Afterward, the output was combined with a vast vocabulary language model, which reduced the error rate to 2.25% (7).

As earlier reviewed, transformers have demonstrated superior accuracies compared to other classifiers, ML models which categorize data points, which implies that a transformer may be able to improve on the error rates of RNN in handwritten text decoding (4, 5). However, there are no existing studies that compare RNNs and transformers for this task. Knowledge of this could result in improved devices for paralyzed individuals to communicate with. Therefore, this current study aimed to determine how a transformer model, when used in a BCI, compared to the RNN BCI. We hypothesized that there would be an improvement in the accuracy and loss metrics when using transformer models over RNNs. It is important to test the hypothesis in this context to determine this. However, the hypothesis was refuted by the results.

RESULTS

To test the model's accuracy and loss with increasing variability, four trials were performed with different goals during the training: minimizing training loss without data augmentation, minimizing validation loss without augmentation, minimizing training loss with augmentation, and minimizing validation loss with augmentation. The data augmentation consisted of minor transformations applied to the recorded neural signals for each intended handwritten sentence to increase variability. The training loss represents how well the model is able to memorize the provided training data and match neural signals to text, while the validation loss tests its ability to generalize its learning by predicting the text for a given set of neural signals.

The epochs for each trial were run with a training batch size (how many pieces of training data were looked at during one iteration) of 3 and a validation batch size of 2 due to limits on how much data could be stored in memory concurrently. During each epoch, the model iterated through all of the training data and tested itself on the validation data. After each epoch, the model automatically changed the weights and its settings based on the results from the epoch. Since Adam, an adaptive optimizer, was used, the model makes fewer adjustments when the results are better (lower loss and higher accuracy (8). When we told the model to minimize training loss, the model only looked at the results from the training portion of the epoch when updating itself. Similarly, when we trained it to minimize validation loss, the model only looked at how it performed on the validation data. The same built-in TensorFlow model checkpoint callback was used in each trial to monitor whichever loss (training or accuracy) was being minimized in that trial and save checkpoints when that loss decreased. The subsections that follow will show the loss and accuracy (both training and validation) per epoch of the model (a custom transformer model as described in Methods) in each of the four trials performed.

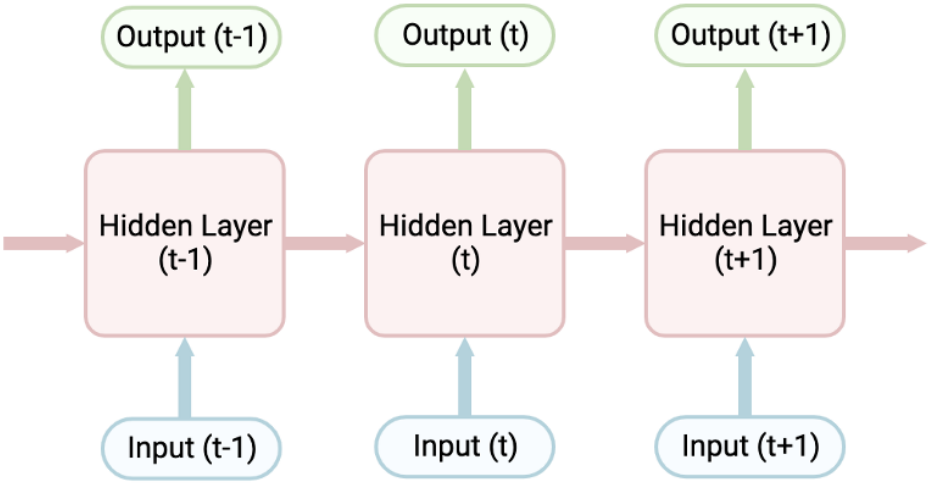

Figure 2: Graphical summary of overall recurrent neural network (RNN) architecture. Flowchart representing the sequence of data processing in an RNN for three arbitrary time (t) steps.

https://doi.org/10.59720/24-027

a)

	0	1	2
1101	-0.6596868116212892	-0.40077329102962733	-0.3944841681664573
1102	0.12899480524611162	-0.40077329102962733	-0.3944841681664573
1103	-0.6596868116212892	-0.40077329102962733	0.7699995409000268
1104	0.12899480524611162	-0.40077329102962733	0.7699995409000268
1105	0.12899480524611162	-0.40077329102962733	1.934483249966511
1106	-0.6596868116212892	-0.40077329102962733	-0.3944841681664573
1107	-0.6596868116212892	-0.40077329102962733	0.7699995409000268
1108	0.9176764221135125	-0.40077329102962733	-0.3944841681664573
1109	-0.6596868116212892	2.389190775752595	1.934483249966511
1110	2.495039655848314	-0.40077329102962733	1.9344832499665108
1111	0.12899480524611162	2.389190775752595	4.263450668099479

b)

1216	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1217	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1218	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1219	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1220	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1221	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1222	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1223	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1224	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.5
1225	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1226	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1227	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1228	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1229	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
1230	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

Figure 3: Samples of neural signals (inputs) and targets. a) Columns represent different electrodes, while rows represent time steps; each cell represents the signal sent by that electrode for the time step. b) Each column represents a different character (ex. 2 = “a”, 29 = “>“). Each row represents a different time step. When a cell is red (value of 1.0), the respective character was being written during that time step. A value of 0.5 represents transitions between characters.

Trial 1: Minimizing training loss without data augmentation

The data used in this trial was the initial data with only a few basic preprocessing steps (no noise or translations added). The goal of this trial was to show the model’s ability to connect the input data with the output data and find relationships/patterns in the neural signals, which the model was successfully able to do. The trend found demonstrates the model’s ability to overfit as the loss was minimized to 0.247, and the training accuracy, the percentage of correct predictions made by the model on the training dataset, reached 96.11% (**Figure 4**). The validation loss, which evaluates how incorrectly the model performed on the validation data, was 1.5358, and the validation accuracy was 19.33% (**Figure 4**).

Trial 2: Minimizing validation loss without data augmentation

With the same initial data as Trial 1, this trial aimed to see if the same model could generalize the identified patterns to new data, the validation data. As shown in the identified trend, the model was unable to fully generalize, as the validation loss was 0.35, while the validation accuracy was 32.51% (**Figure 5**).

Trial 3: Minimizing training loss with noise and transformations

https://doi.org/10.59720/24-027

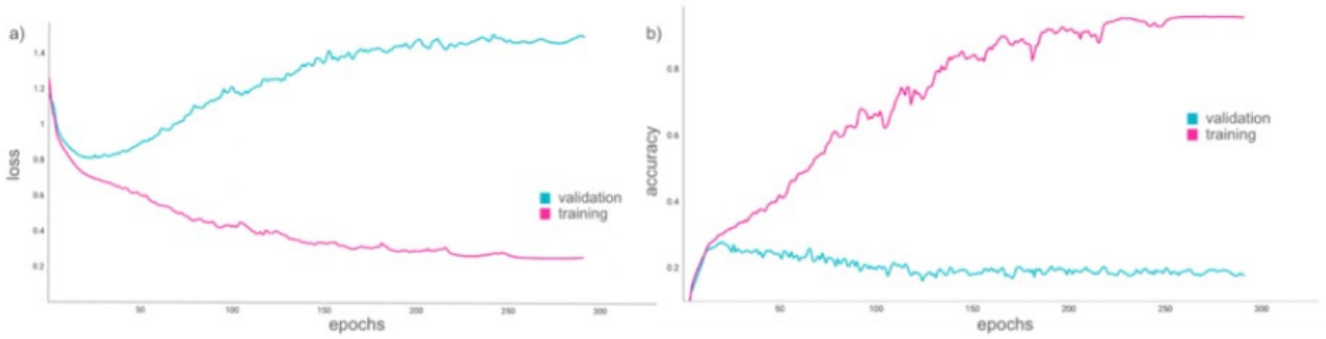

Figure 4: Trial 1 loss and accuracy by epoch. The loss and accuracy metrics were determined during Trial 1 as the transformer model trained for 291 epochs to minimize training loss. The train-validation dataset split was 90%-10% with a training batch size of 3 and a validation batch size of 2. a) Epoch training loss (pink) and epoch validation loss (blue). b) Epoch training accuracy (pink) and epoch validation accuracy (blue). As the program had to be rerun from checkpoints (as there was a software runtime limitation), the graphs are discontinuous at the epochs from which the program restarted.

The data in this trial had white noise added to the neural signals. Similarly, these signals also had a random normal mean drift noise, random walk noise, and cumulative random walk added to them, which slightly offset the data. These augmentations mirror those made in the data when the original RNN BCI was being tested. The results show a final training loss of 0.427 and a training accuracy of 62.81% (**Figure 6**). The validation loss was 0.4071, while the validation accuracy reached 24.60% at the end of the last epoch (**Figure 6**).

Trial 4: Minimizing validation loss with noise and transformations

For the last trial, the (same) model checkpoint callback was used to minimize validation loss. In the trial, the model was trained using the same data as in Trial 3. This trial showed the least favorable results, with a training loss of 0.3869 and an accuracy of 69.06% (**Figure 7**). The validation loss ended at 0.388, and the validation accuracy at 31.58% (**Figure 7**).

DISCUSSION

This study looked at the ability of a transformer model version of a BCI to translate the neural signals of a paralyzed individual. We conducted four separate trials to assess the model's proficiency in discerning patterns in input and output data, as well as its capacity to translate the previously unseen data into output sentences during validation. The first two trials were carried out with minimal preprocessing of the input signals, which aimed to evaluate the model's ability to identify the simplest relationships between inputs and outputs. In contrast, the latter two trials used data augmentation techniques such as introducing noise and translational offsets. These augmentations were introduced to gauge whether the model could still identify these connections despite the additional variability introduced by these processing steps.

Overall, Trials 1 and 2, which were trained using the original data, showed more favorable results when compared to Trials 3 and 4, which used data augmented with noise. The addition of the random shifts and white noise allowed the data to differ from the target by a small amount, which would theoretically help the model generalize across slight inconsistencies in the neural signals. However, this addition to the input data did not help the model generalize and even hindered its ability to learn patterns from the training stage, specifically in Trial 3.

While the model produced favorable results for the training data in Trial 1, it was unable to generalize the patterns it found

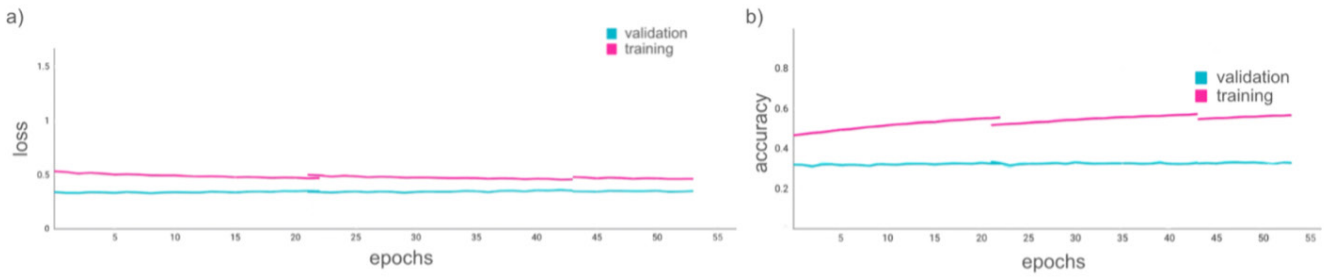

Figure 5: Trial 2 loss and accuracy by epoch. The loss and accuracy metrics were determined during Trial 2 as the transformer model trained for 53 epochs to minimize validation loss. The train-validation dataset split was 90%-10% with a training batch size of 3 and a validation batch size of 2. a) Epoch training loss (pink) and epoch validation loss (blue). b) Epoch training accuracy (pink) and epoch validation accuracy (blue). As the program had to be rerun from checkpoints (as there was a software runtime limitation), the graphs are discontinuous at the epochs from which the program restarted.

https://doi.org/10.59720/24-027

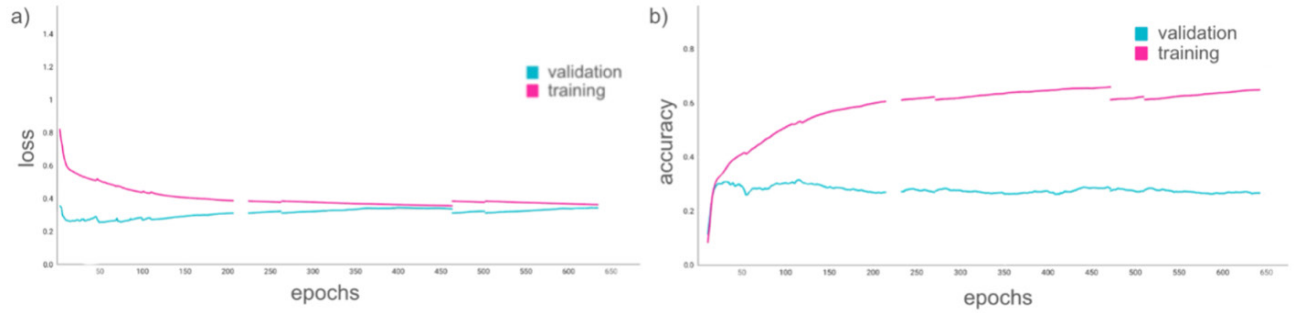

Figure 6: Trial 3 loss and accuracy by epoch. The loss and accuracy metrics were determined during Trial 3 as the transformer model trained for 632 epochs to minimize training loss using the augmented data. The train-validation dataset split was 90%-10% with a training batch size of 3 and a validation batch size of 2. a) Epoch training loss (pink) and epoch validation loss (blue). b) Epoch training accuracy (pink) and epoch validation accuracy (blue). As the program had to be rerun from checkpoints (as there was a software runtime limitation), the graphs are discontinuous at the epochs from which the program restarted.

to the new data the model was presented with, resulting in undesirable metrics for validation data. The final training loss was 0.247, with a training accuracy of 96.11%, while the validation loss was 1.5358, and the validation accuracy was 19.33%. This was, however, the expected outcome for the trial as the model was set to only minimize the training loss. This results in an overfit model that had learned the training examples but not attempted to determine the more generalizable patterns that could apply to the validation data. Since the model had overfit, it was capable of learning patterns in the data, albeit only for the training data in this case.

In Trial 2, the validation loss was noticeably greater than the training loss from the overfitting trial (Trial 1), 0.35 compared to 0.247, while the training accuracy in the overfitting trial was 96.11% compared to the validation accuracy of 32.51% in this trial. The model was unsuccessful at minimizing the validation loss in this trial as the model was unable to generalize its knowledge to the validation data. Unlike in Trial 1, the model was set to minimize validation loss, so we expected the model to have learned patterns instead of specific examples. However, these more unfavorable results in this trial indicate the model's relative inability to find generalizable patterns between the neural signals and the characters in the intended sentence text.

In comparison to Trial 1 (overfitting with unprocessed data), Trial 3 had a significantly greater training loss (0.4273 vs. 0.2467) and a significantly lower training accuracy (62.81% vs. 96.11%). While the model was able to learn the correlation between the inputs and outputs in the training dataset in Trial 1, the model was unable to do the same thing this time when variability was included with noise and transformations to the dataset in Trial 3. This is further evidence of the model's poor ability to generalize.

In Trial 4, the model performed the worst when compared to all previous trials, specifically Trial 2 (minimizing validation loss without noise or transformations. In this trial, the model was not able to recognize patterns in the augmented data to minimize validation loss. As the variability of the data increased from Trial 2 to Trial 4 through the additions of transformations, the model performed poorly (high validation loss and low validation accuracy). This trial most closely reproduced the research that used an RNN for their BCI (7). The original RNN BCI developed by Willet et al. calculated the loss of the character probabilities and the start signal (when a new character began) separately (7). The character probability loss was calculated using softmax cross-entropy and was then summed with the start signal loss (a sigmoid computation) to produce the total loss (7). The reported character loss was 0.0854, with an accuracy of 94.1% when a k-nearest-neighbor classifier was applied to the character probabilities outputted by the model (7). In our transformer model, we calculated character probability loss by comparing the predicted characters to the correct characters; no starting character loss was implemented. The character validation loss in the trial that most closely reproduced the RNN BCI, Trial 4, was 0.3881, with a validation accuracy of 31.58%. The lower error and higher accuracy of the RNN BCI suggest that the model was more favorable than this transformer BCI for the given translation task.

A benefit of the transformer model over an RNN is its ability to translate more accurately (4, 5). When compared to the results of the RNN BCI, however, this transformer model performed less favorably in all trials. A reason for this seemingly contradictory result could be the usage of synthetic data for training in the previous study (7). Synthetic data consisted of a compilation of random combinations of collected neural signals and the corresponding intended handwritten text (7). While there was a limited amount of data collected from the actual participants, the inclusion of synthetic data increased the amount of training and validation data significantly, which allows for the model to allow for some variability when categorizing into letters. Using a greater amount of data may improve the model's ability to generalize, as the model would receive multiple examples to expose it to more potential variation that is allowed for each character (9). However, the significantly reduced amount of data used to train this model could have led to this transformer model being unable to generalize despite the data augmentation and predicted, from previous studies, superior architecture (10).

Due to the device we used for training this transformer model having limited graphics processing unit (GPU) storage, we were unable to add synthetic data to the dataset due to

https://doi.org/10.59720/24-027

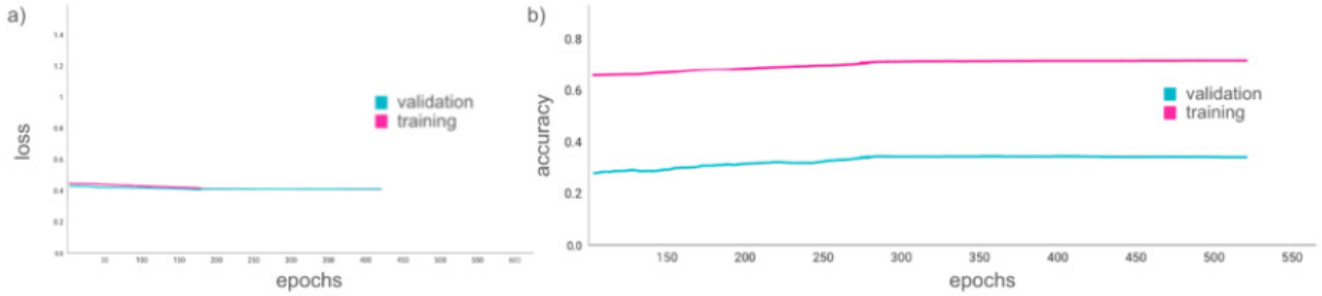

Figure 7: Trial 4 loss and accuracy by epoch. The loss and accuracy metrics were determined during Trial 4 as the transformer model trained for 529 epochs to minimize validation loss using the augmented data. The train-validation dataset split was 90%-10% with a training batch size of 3 and a validation batch size of 2. a) Epoch training loss (pink) and epoch validation loss (blue). b) Epoch training accuracy (pink) and epoch validation accuracy (blue). As the program had to be rerun from checkpoints (as there was a software runtime limitation), the graphs are discontinuous at the epochs from which the program restarted.

its load on the GPU. Due to time constraints, we were unable to find and implement another solution. Using a GPU is a critical part of model training, and the size of its storage has a crucial impact on the model's performance as it allows for more training data, which we had to reduce, and complex model architecture (11-13). Another potential explanation for our unexpected unfavorable results may be that we stopped the training too early, as the model may have reached a more favorable outcome had it been allowed to continue training for longer, especially in Trials 2 and 4, which did not produce an overfit model.

The current scope of the project did not explore the difference in training speed between the RNN BCI and this transformer BCI, which would be an interesting topic for further investigation. A transformer is generally able to be trained more efficiently due to its increased parallelization than an RNN, and this difference could allow a transformer BCI to offer a noteworthy improvement over its RNN predecessor (4, 6). This could be achieved by recording the training time for multiple batches of data per model, finding the median, and dividing by the number of examples in a batch. This value could then be compared for the RNN and the transformer models. Another potential extension is including synthetic data in the transformer model and assessing how that affects the results. An improvement in the metrics would then be expected, specifically in the validation ones. This would require a GPU with a greater storage capacity or multiple GPUs, or alternatively, the code could be set to load in individual batches one at a time. The RNN BCI also implemented an offline autocorrect process by sending the outputs from the RNN through a large-vocabulary language model. Comparing the results of the autocorrection on the transformer output versus the RNN output could determine the benefit provided by an offline autocorrection.

The aim of this study was to understand how a transformer BCI compared to a similar RNN BCI when decoding the intended handwritten text of a paralyzed individual from their neural signals in the motor cortex region of the brain. We found that this specific implementation of a transformer BCI did not perform as well as the RNN BCI when comparing the training and validation metrics. Since this outcome may have been impacted by time constraints and device limitations, further exploration is needed to see if a transformer model can play a role in BCIs in the future.

MATERIALS AND METHODS

Dataset

Electrode data was collected from a previously compiled dataset (7). The dataset included 10 sessions over 28 days, during which data was recorded using two microelectrode arrays (with 96 electrodes each) implanted in the premotor area of the brain of a single participant (7). This participant had a complex spinal cord injury and was paralyzed from the neck down, with hand movements limited to twitching and micromotion (7). The participant was tasked with writing as if his hand were not paralyzed and as if he were holding a pen on a piece of ruled paper (7). The dataset included all recorded neural activity (1,000 sentences over 10.7 hours) (7). The data was in the form of binned spike counts, which are integer values corresponding to how many times the voltage on a given electrode crossed a specified threshold during that time bin of 10 milliseconds (7). In a portion of the data, the participant attempted to write full sentences, and in another portion, he attempted single letters (7). Preprocessing of the data included normalizing the data, separating the data into training and validation sets, and further splitting the data into batches (7). The training batch size was 3, and the validation batch size was 2. Since the greater the batch size, the quicker the model was trained, the highest batch size without overloading the GPU was used.

Model

The transformer had a custom architecture but still had the same key layers typical of a transformer model (**Figure 1b)** (4). Its encoder was composed of a stack of four identical layers. Each had a multi-head self-attention mechanism, two dropout sub-layers, a feed-forward network, and two normalization sub-layers (**Figure 1a**).

The decoder was composed of six sub-layers: a masked (casual attention mask) multi-head attention over the output of the encoder stack and the targets, three normalization layers, another multi-head self-attention mechanism that was modified to prevent positions from attending to subsequent positions, and a position-wise fully connected feed-forward network. The output embeddings were offset by one position, which ensured that the predictions for each position only depended on the known outputs at prior positions. Before the data was passed to the decoder layers, the data was

https://doi.org/10.59720/24-027

embedded. This is where the transformer differed from the one proposed in the original transformer architecture paper (4). The transformer in this paper was modeled after Apoorv Nandan's transformer and, as such, contained two embedding classes: one for tokens (before the encoder) and one for speech features (before the decoder) (14).

The token embedding class contained the token and positional embedding. The token embedding mapped each individual character to a representative vector of its meaning. The positional embedding was then added to the tokenization, which described information about the location of the character in the sentence as each position was assigned a unique representation. This allowed the model to know which letters were further from or closer to others and determine relationships based on this.

The speech feature embedding class was made up of three one-dimensional convolutional layers (**Figure 1c**). The output of each was then fed into the next, and the output of the last was returned as the result. The window size (how many time steps were looked at each time) was more than the average number of time steps per letter with the goal of fitting a letter into each window. The stride (how many time steps the window shifted by) was less than that same average number of time steps to limit shifting by more than a letter each time.

Training

The model was trained to reduce the loss calculated during the testing stage (when the model attempted to translate neural data it had not seen before based on trends noticed during the training phase). The loss function was Categorical Crossentropy, and it was calculated between the one hot encodings of the target sentences and the prediction sentences made by the transformer. The accuracy was calculated between the predictions and the target sentences using the Categorical Accuracy metric. The learning rate was a custom learning rate schedule that increased for the first initial epochs before slowly decreasing. During the training phase, the optimizer (Adam) applied gradients, computed by GradientTape, of the loss and the trainable variables of the transformer. Checkpoints were used to save the weights of the model when the testing loss had improved. The number of epochs used was not intentionally chosen, as the training was stopped after the results seemed to plateau. Trial 1 used 291 epochs, Trial 2 used 53 epochs, Trial 3 used 632 epochs, and Trial 4 used 529 epochs.

Trials

Four trials were run in total. All trials followed a 90/10 training/validation split. The first two trials were done without augmentation of the input signals in an attempt to test the model's ability to find the least complex relationships. The last two trials included data augmentation in the preprocessing to see if the model would still be able to find connections, even with the variability introduced by these processing steps. The purpose of these trials was to test the transformer model's ability to generalize with increasing variability.

The augmentations of the data in trials 3 and 4 included white noise that was added to the neural signals, which was created using a map to a TensorFlow random normal function. Similarly, the input signals were mapped again to a function that applied a random normal mean drift noise, random walk noise, and cumulative random walk, which offset the data by a few time steps. These augmentations were chosen to allow for a more direct comparison to the RNN BCI by Willet et al., which added these same augmentations to the data (7).

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to Lumiere Education for giving me the opportunity to perform this research. I would also like to acknowledge McKinsey & Company's Partner for their sponsorship, without which this research would not have been possible.

Received: April 29, 2023
Accepted: August 23, 2023
Published: January 24, 2025

REFERENCES

1. Kumar, M. Keerthi, et al. "Comparative analysis to identify efficient technique for interfacing BCI system." *IOP Conference Series: Materials Science and Engineering*, vol. 925, no. 1, 2020, p. 012062. https://doi.org/10.1088/1757-899X/925/1/012062.
2. Coin, Allen, et al. "Ethical aspects of BCI technology: what is the state of the art?." *Philosophies*, vol.5, no. 4, Oct. 2020, p. 31. https://doi.org/10.3390/philosophies5040031.
3. Keren, Gil, and Björn Schuller. "Convolutional RNN: an enhanced model for extracting features from sequential data." *2016 International Joint Conference on Neural Networks (IJCNN)*, IEEE, Nov. 2016, pp. 3412-3419. https://doi.org/10.1109/IJCNN.2016.7727636.
4. Vaswani, Ashish, et al. "Attention is all you need." *Advances in Neural Information Processing Systems*, June 2017, p. 30. https://doi.org/10.48550/arXiv.1706.03762.
5. Karita, Shigeki, et al. "A comparative study on transformer vs rnn in speech applications." *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Dec. 2019, pp. 449-456. https://doi.org/10.1109/ASRU46091.2019.9003750.
6. Lee, Po-Lei, et al. "Continual learning of a transformer-based deep learning classifier using an initial model from action observation EEG data to online motor imagery classification." *Bioengineering*, vol.10, no.2, Feb. 2023, p. 186. https://doi.org/10.3390/bioengineering10020186.
7. Willett, Francis R., et al. "High-performance brain-to-text communication via handwriting." *Nature,* vol. 593, May 2021, pp. 249-254. https://doi.org/10.1038/s41586-021-03506-2.
8. Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint*, Dec. 2014. https://doi.org/10.48550/arXiv.1412.6980.
9. Cai, Wenjie, and Danqin Hu. "QRS complex detection using novel deep learning neural networks." *IEEE Access*, vol. 8, May 2020, pp. 97082-97089. https://doi.org/10.1109/ACCESS.2020.2997473.
10. Wang, H., et al. "Machine learning basics." *Weihong Deng*, 05 Jan. 2015, www.whdeng.cn/Teaching/PPT_01_Machine%20learninTg%20Basics.pdf. Accessed 02 Dec. 2023.
11. Baji, Toru. "GPU: the biggest key processor for AI and parallel processing." *Photomask Japan 2017: XXIV Symposium on Photomask and Next-Generation Lithography Mask Technology,* vol. 10454, July 2017, pp.

https://doi.org/10.59720/24-027

24-29. https://doi.org/10.1117/12.2279088.
12. Jeon, Won, et al. "Deep learning with GPUs." *Advances in Computers*, vol. 122, 2021, pp. 167-215. https://doi.org/10.1016/bs.adcom.2020.11.003.
13. Li, Youjie, et al. "Harmony: Overcoming the hurdles of GPU memory capacity to train massive DNN models on commodity servers." *arXiv preprint*, Feb. 2022. https://doi.org/10.14778/3551793.3551828.
14. "Keras Documentation: Automatic Speech Recognition with Transformer." *Keras*. www.keras.io/examples/audio/transformer_asr/. Accessed 02 Dec. 2023.