Predicting Mice Behavior from Neural Activity Using Encoder-Decoder Transformer

Jorge Acedo **Jialin (Joyce) He**

Abstract

In the era of rapid development of machine learning techniques, they are being applied in different tasks solving different problems, including in behavior science and neuroscience. The data used was obtained from a study investigating continuous decision-making in mice, recording both timestamped behavioral information and concurrent neuronal activity. Given the nature of both sequences, an encoder-decoder transformer model was implemented to explore the possibility of using the neural activity of mice to predict its lever-held-down duration. Three models were tested, each to perform binary classification, multi-class classification, and sequence-to-sequence translation on neural activity and behavioral data collected from mice performing lever-held-down tasks. The three models' performances indeed exceed our expectations considering the input is noisy neural data. Indicating that this type of model can be used to predict mice activity given neural activity information.

1 Introduction

Machine Learning techniques are now applied in solving various problems people face when trying to understand the brain and behavior. Specifically, neural network models with transformer architecture are being most famously used for the task of recreating human behavior such as text generation. In the present report, we use information from an article that analyzes the continuous behavior of mice in a free-roam environment and its associated neural activity. Precisely, mice are placed into an operant box where if a lever is pressed for more than 1600 milliseconds then the mouse is rewarded with food pellets.

Considering the temporal nature of the data we are using for this project and inspired by the current attention to transformers, we chose to use the encoder-decoder transformer architecture to process the data. In the present report, we implemented three transformer models each with a different transformation of the output sequence. The first model performs binary classification taking in neuronal signals to determine whether the current lever press can reach the threshold of 1600 ms. For the second model, we increased the number of output classes to 10 by dividing the duration variable into 200 ms intervals. The final model uses the same neuronal signals to determine the precise raw duration of the lever press in milliseconds.

2 Method

2.1 Data

To collect data, mice underwent brain injection procedures and performed the lever hold-down task. For the lever hold-down task, each mouse was placed in an operant box. Without any instruction or pre-training, the mouse needs to discover that it has to hold down the lever for at least 1600ms in order to receive the reward. Data is generated including whether the lever is held down during each frame, head-entry movement, whether the reward is received, and correlated neural activity indicated by calcium concentration is recorded. In addition, all data points are timestamped. Each mouse experienced multiple days of a behavioral experiment. The existing data includes 8 mice's data for a total of 33 unique combinations of mice and trial days.

Datasets of both neural activity and behavior for each individual day and subject were joined by the timestamps. The duration of each lever press was calculated and all corresponding data of the mice was ordered. The corresponding neural signal between each lever press was also extracted. The final data frame includes the order of each lever press within the individual day, the subject identifier, the day, the threshold which is 1600 ms, the lever press duration, whether the duration met the threshold, and the neural signal in each lever press interval.

To better capture the temporal relationship of the calcium concentrations between lever presses and to have a format that can be better processed by the transformer model, we applied Wav2Vec to the input data. The Wav2Vec generates a consistent size vector representation of the input by extracting meaningful features from the raw vector, in this case, the calcium signal of the brain activity. As a result, while the transformer model takes in one vector for each lever press, the combination of the elements within each vector still carries the information on the relationship between data points in between lever presses. Both the input and output sequences were zero-padded to the biggest sequence, which consisted of 703 events.

Finally, the 33 subject-day trials are unevenly distributed among the eight mice, the minimum number of trials per mouse was two, and the maximum was eight. Therefore, to ensure that each mouse had at least one trial in both the training and test sets, we randomly selected one trial for each mouse as the test set and the rest consisted of the training set. Furthermore, six trials were selected randomly using the same criteria as the validation set. With this partition methodology, the training set consists of 6,780 lever presses, the validation set of 1,163, and the test set of 1,693.

2.2 Model Architecture

Each model consists of an encoder-decoder structure with a transformer architecture. The encoder part of the model processes the input sequence and creates a representation of the neural activity vectors. The decoder part of the model uses the encodings to generate a prediction of the sequence of lever press durations.

The encoder takes in the transformed input sequence and applies a linear transformation that changes the input dimension from the maximum sequence length to a specified dimension size. Then, the sequence passes through a positional embedding layer. This layer adds a vector of sinusoidal patterns, that encodes the positional representation of each element in the sequence, to the input embedding. Dropout is applied to introduce regularization. This new embedding then goes through the transformer encoder layers, the number of layers is specified as a hyper-parameter. It will first compute multi-headed self-attention with a specified number of attention heads. The outputs of the self-attention layer are fed to a feed-forward neural network. This encoder output will then be fed into the decoder part of the model.

The target sequence also goes through a linear transformation to the same dimension as the input sequence. The decoder layer is similar to the encoder layer, it first takes the embedded target sequence and then goes through multi-head attention. Nevertheless, the encoder output is taken to compute encoder-decoder attention that then goes through a feed-forward layer. Such as with the encoder, the whole decoder layer is repeated as many times as specified. In the end, the decoder output goes through a linear transformation to the number of classes that we wish the model to output. The first two classification models then go through softmax which returns the class with the highest score from the total of possible outcomes. The third model applies two linear transformations with a ReLU activation function and this output is added to the original target sequence to implement residual connection.

3 Experiment

To meet the goal of predicting lever press duration in a more and more precise manner, from a binary outcome to a continuous outcome, based on the basic transformer architecture described in the method section, we changed some layers and the dimensions of the inputs and outputs of each model. Hyper-parameter-tuning was also done for each model.

3.1 Model 1: binary classification

In the binary classification model, the output goes through a sigmoid function at the end, instead of a softmax function. The sigmoid function takes the final linear mapping and calculates the probability of the current lever press being greater than 1600 ms, which is the threshold for the mice receiving the reward. Because there are only two classes, using the sigmoid function is more efficient. If the model predicts that the probability is higher than 0.5, it will predict that the current lever press met the reward criteria.

3.2 Model 2: multi-class classification

In the multi-class classification model, an embedding layer is added to the target sequence at the beginning of the decoder section of the model. The embedding layer takes the target sequence of discrete labels of classes and maps them to a vector

representation of the desired dimension.

A linear layer and softmax are applied at the end of the decoder. The linear layer maps the hidden states to the number of possible classes. The softmax takes this vector and returns a vector of the same size with the probability of the current observation belonging to each class. In this case, the model predicts the class with the highest probability.

3.3 Model 3: Sequence to sequence translation

In the sequence-to-sequence prediction model, lazy batch normalization is implemented. This special form of batch normalization does not require the number of features from the input and infers such information from the first forward pass of the current input and then initializes weights, bias, running mean, and running variance. The reason lazy batch normalization is used is that the input of the target sequence can have a high variance from trial to trial and cannot be defined beforehand. Furthermore, since the lever press duration can take a value between something near zero to 35,000 milliseconds, applying normalization helps to stabilize the gradients and prevent issues with the initialization of weight parameters.

A total of two linear layers were added to the end of the decoder of the model mapping the hidden states to the desired outcome size. The linear layers not only reduce the size of the outcome but also allow the model to capture more of the internal pattern of the data by initializing weights and biases, which are then updated during training. In between the linear layers, ReLU activation is added to introduce non-linearity and complexity to the model. With the activation function implemented, the model can also be more generic and more generalizable.

A residual connection is deployed as the final step of the transformer, and its outcome is returned as the final result. The residual connection technique basically adds the original target sequence to the output of the decoder and linear transformations, allowing the original information to skip the process of intended layers and reach the end. By doing so, information loss and distortion throughout multiple layers and backpropagation are avoided. Without residual connection, the model showed great difficulty predicting something other than the mean of the sequence. This was possible because the gradients affected the latter layers more than the first ones.

3.4 Hyper-parameter Tuning

In addition to the change in the fundamental structures of the neural networks, we also experimented with different hyper-parameters to improve performance. For example, the learning rate for the first model was 0.0001. Nevertheless, given the higher complexity of the second and third models, an increase in the learning rate to 0.001 helped the model to decrease its loss more quickly. Higher learning rates were tested but they lead to exploding gradients. Furthermore, a different number of dropout rates were tested. While a higher dropout rate helps the model to be able to generalize better, a large dropout rate may result in an unstable performance in testing. A dropout rate of 0.5 was chosen for the first and the second models, while for the third model, a dropout rate of 0.4 performed better in the test trials. Momentum was also tuned so that it helps decrease the loss function more quickly without leading to unstable or bad local minimums. Momentum helped the most to speed up the converging process in the third model.

We also tried different values for hyper-parameters such as the number of heads, number of encoder and decoder layers, and embedding decoder size. Increasing the number of each of these hyper-parameters resulted in a slower training of the models with no increase in the models' performances.

Regardless of the effect learning rate and dropout has, different optimizer algorithms affect the accuracy and loss significantly. We compared stochastic gradient descent optimization and the Adam optimization. Current result shows that the Adam optimization algorithm significantly outperforms the SGD optimization. Example results are shown in Table 1. Theoretically, Adam normally outperforms SGD for several reasons. First of all, it has the ability to perform adaptive learning, meaning that the learning step for each parameters is different because they are separately influenced by the next moment which is decided by the exponentially moving average. Adam also incorporated RMSpropagation, allowing it to deal with more noisy data by decreasing the oscillation when trying to reach the local minimum. Both features also enable models deploying Adam optimization to converge more quickly and become more computa-

tional efficient.

4 Results

With the increasing granularity of the output result, more is demanded from the model's performance, while the transformer architecture again shows its powerfulness and robustness with high performance. The ROC-AUC score for the test set of the first transformer model performing binary classification reaches 1.0. Furthermore, the test accuracy of the second transformer model performing 10-class classification is 100%. Finally, the last transformer model has a test RMSE of 12.89 with the data ranging between 0 to 35,000 milliseconds. The result of the third model is especially meaningful because this can be one of the very first initiatives to directly take in neural signals to predict the precise result of an action.

The following plot shows the close relationship between the target and the predicted sequences of a specific trial. It is worth noticing that the model is able to predict the noisy structure of the data, even predicting the very highest point near 20,000 ms. This specific plot was not hand-picked since all of the other seven trials' plots show the same predictive power of the model.

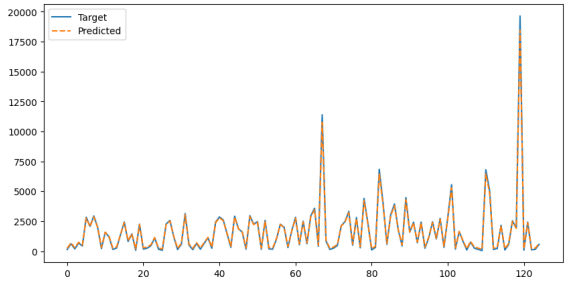

Figure 1: Target and Predicted Sequences of a Test Trial

5 Discussion

It has always been intuitive that neural signals are noisy and that it would be hard for researchers to extract any useful pattern from the huge and complex data that can be directly associated with action regardless of experimental manipulations. However, with the employment of transformer architecture, neural network models are able to extract meaningful features and predict the duration of action down to milliseconds. Such possibilities may open new doors for future neuroscience research, for example, serving as an initial tool for researchers to identify certain actions associated with neural circuits or related brain regions, and even facilitate further human-machine interaction research and development.

References

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853.

Galen Andrew and Jianfeng Gao. 2007. Scalable training of L1-regularized log-linear models. In *Proceedings of the 24th International Conference on Machine Learning*, pages 33–40.

Riyam Arabo, Jorge Acedo, Jiaqi Liu, and Jiaohaer Taolan. 2022. Pinky and the brain. Course Project.

Vitaly Bushaev. 2018. Adam — latest trends in deep learning optimization. Accessed on March 25, 2023.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Harsh Khandewal. 2020. Gradient descent with momentum, rmsprop and adam optimizer. Accessed on March 25, 2023.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

Mohammad Sadegh Rasooli and Joel R. Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *Computing Research Repository*, arXiv:1503.06733. Version 2.

Drew C. Schreiner, Christian Cazares, Rafael Renteria, and Christina M. Gremel. 2022. Information normally considered task-irrelevant drives decision-making and affects premotor circuit recruitment. *Nature Communications*, 13(1).

Distribution of responsibility:

Jorge Acedo: implementing the models, hyperparameter tuning, writing the report

Jialin (Joyce) He: writing the report, hyperparameter tuning

Optimizer	Loss	Validation Accuracy
torch.optim.Adam	1031.78	100%
torch.optim.SGD	1553.03	22%

Table 1: Comparing loss and validation accuracy of the multi-class classification model. Learning rate and dropout is constant for both trials (0.001 and 0.5). Momentum for the SGD optimizer is 0.9. Since more combinations of the hyper-parameters were utilized, only 5 epochs were ran for each combination.