# Low Dimensional Neural Activity Representation

**Dave Lu**
Computational Biology Department
Carnegie Mellon University
Pittsburgh, PA 15213
davelu@andrew.cmu.edu

**Gaurav Molugu**
Department of Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
gmolugu@andrew.cmu.edu

Code Repository: `https://github.com/daveylu/10417-project`

## 1 Introduction

The goal of our project is to better represent time-series high-dimensional neural activity data in lower dimensions. Through this, we hope to learn more about how neural activity is related to our brain function, which is still a topic of great research and importance. We used data collected by the Allen Institute Neuropixels Project [5] to measure mice brain neural activity after presenting a visual stimulus, such as an image or video.

The data mainly consisted of a time-series DataFrame, which recorded the time at which a neuron in the mouse's brain spiked/fired and which Neuropixel probe captured the spike, as well as some meta-data on the visual stimuli being presented at the time of the spike capture. The specific mouse we used (ID 791319847) had 93 Neuropixel probes collecting data in the brain, so our input dimension $n = 93$.

We considered the visual stimulus corresponding one of four classes: spontaneous activity (class 0), flashes (class 1), static gratings (class 2), and natural scenes (class 3). For each separate stimulus shown (each lasting 250ms), we binned the precise time measurements into 5ms bins (literature has established that the average time between two successive neuron firing is 5ms) and created a binary vector for each of the 50 bins, which told us which neuron fired in that 5ms time range. By concatenating these vectors together, we created a time series binary array which accurately represents the neural activity of the mouse for a given stimulus over time. This array will always have dimensions $n \times 50$. We will refer to this data as the time-series binary data for the rest of this paper.

We also collapsed/binned the time series binary array together to get a $n$-length vector containing simple counts of neuron spikes while being shown a stimulus, which we used for our baseline method and autoencoder. This can essentially be thought of as collapsing the time-series binary arrays along their time axis and adding up their values to get a single vector. We will refer to this data as the binned data for the rest of this paper.

## 2 Background

We first implemented a baseline method, which was a support vector machine (SVM) with a RBF kernel. We trained it to classify the simpler binned neural activity data into an stimulus class. The idea is that if we are able to find a high-performance low-dimensional latent space from our more advanced neural networks, we should be able to train a new SVM which uses our low-dimensional embeddings and outperforms the baseline SVM using just the simple binned data. To find this latent space and encode our data in lower dimensions, we decided to use two techniques: an autoencoder and a transformer. The autoencoder will be using the same input as the baseline method (the binned data), while the transformer will be using the time-series binary data. All three methods are attempting to classify the given neural data (encoded or not) into each of the four separate classes. If the autoencoder and transformer are successful in finding a high-performance low-dimensional latent space, we expect to see the SVM using the unencoded data to be worse at classifying it into the four classes than the SVMs using the encoded data.

## 3 Related Work

Our project is inspired by this paper [4], which we referenced a lot while working on this project. The authors of this paper modeled brain neural dynamics during adaptive behaviors via neural probe data, using machine learning to discover a low-dimension, high-performance latent space. Their project used pretty simple neural network architectures such as 1-D convolutional neural networks and the data they used to train the neural networks used very large bins, losing a lot of temporal information as a result. We hoped that through using data with higher temporal resolution and more advanced neural networks, we can learn an even better way to encode brain neural activity in a lower dimension and get better results.

As we came up with ideas on how to train some neural networks to encode our data, we referenced three papers. The first paper we looked at was about contrastive loss of images [2]. From this paper, as well as lectures in class, we started to think about training a model that used contrastive loss as our objective function in combination with a recurrent neural network or transformer. To learn more about how to use contrastive loss with time-series data, we referenced this paper [1], which gave us some ideas on how to apply contrastive loss to our time-series binary data. We also looked at another paper on triplets and contrastive loss [3], and we decided to use the ideas from this paper in our transformer.

## 4 Methods

We implemented a total of three methods: a SVM with a RBF kernel trained on the raw data as our baseline, an autoencoder, and a transformer.

### 4.1 Support Vector Machine

The baseline method is the SVM with a RBF kernel, which we trained to classify binned neural activity data into an image and image class. As previously mentioned, this method used the simpler binned neural activity data rather than the time-series binary data.

### 4.2 Autoencoder

To see if an autoencoder could improve the SVM, we then trained an autoencoder to encode the simpler binned neural activity data into lower dimensions and then used the encoded data to train a new SVM with a RBF kernel. The architecture mainly consisted of linear layers and ELU activation functions stacked on top of each other. The autoencoder was trained over 200 epochs to minimize the mean squared error between the input and the output, and the low-dimensional encodings were taken from the tiny layer in the middle.

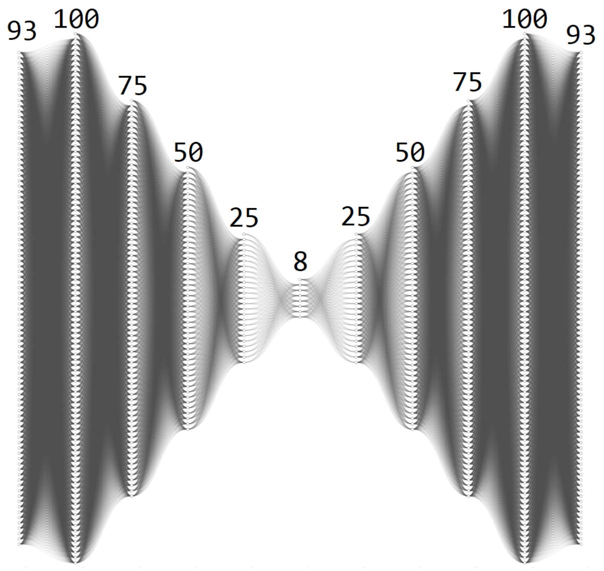


Figure 1: Autoencoder Architecture.

We chose to use an autoencoder because it is perfectly designed for the task of trying to represent the data in a lower dimension. As seen in the figure of the architecture, the autoencoder gradually decreases the number of dimensions of the input until it reaches a bottleneck (the encoder section). At the bottleneck, it now has many less dimensions that the input section. It then tries to reconstruct the original input from that bottleneck in the expanding portion of the autoencoder (the decoder section). Because of the bottleneck, the autoencoder must find a way to capture all of the information necessary to reconstruct the original input from a much smaller space, thus forcing it to discover an information-rich low-dimensional latent space and be able to encode inputs into that latent space. As seen in figure 1, the autoencoder we wrote had a dimension of 8 at the bottleneck.

Because the baseline method and the autoencoder are using the same data, we will be able to directly compare the results between the two models. If the autoencoder is able to discover a low-dimensional latent space that captures useful information that the binned data does not have, we then expect the SVM trained on the low-dimension encoded data from the autoencoder to perform better than the SVM using just the binned data.

### 4.3 Transformer

Next, we trained a transformer encoder model to encode the time-series binary data into lower dimensions as well and used these encodings to train another SVM with a RBF kernel. As the time-series neural activity is primarily a sequence (time-series), we were interested in exploring the seq2seq modeling paradigms. The recent advancement of seq2seq modeling is primarily driven by Transformer models [6].

The time-series neural activity data can be visualised as a binary sequence of neural spikes of length 50 and the neurons can be viewed as a dimensional representation of the visual stimulus. We create a custom dataset which has an anchor (random class), positive (neural activity of the same class), and negative (neural activity of dissimilar class) samples. The samples are randomly generated and each batch has 1024 triplets of anchor, positive and negative. We first linearly project the all three samples at each time bin of dimension $n = 93$ (the number of brain neurons) into a 128 transform dimension. We then pass the three transformed samples through a 4 layer transformer encoder, 4 head attention, 1024 feed forward dimension and 0.4 dropout. The output of the transformer is of dimension (batch, 50, 128), which we then flatten to a dimension of (batch, 50*128). The output is finally passed through a linear layer, which projects the transformer output into a single small vector, where the length of the vector is the (small) number of dimensions desired.

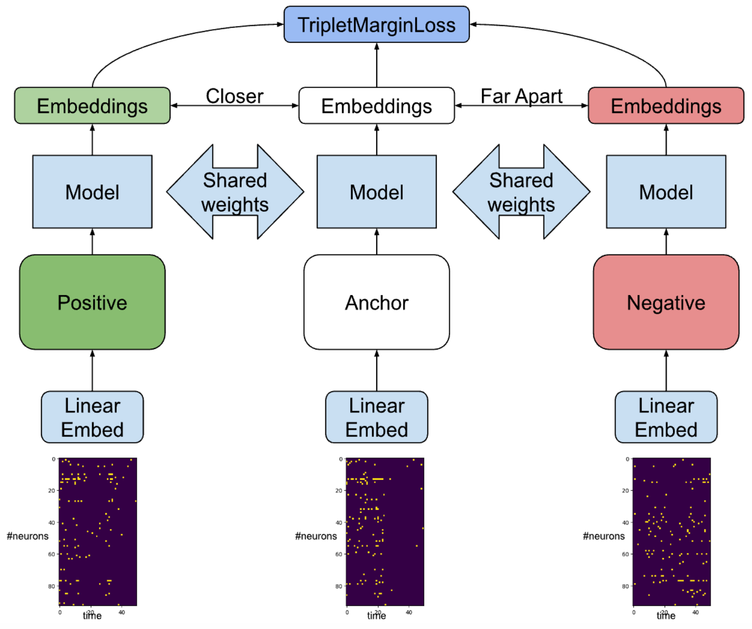


Figure 2: Transformer Architecture.

A TripletMarginLoss [3] is calculated using the three embeddings of anchor, positive, and negative samples. The loss is backpropagated through the network and weights are updated using Adam Optimizer with learning rate of 0.001 for a total of 50 epochs.
We hypothesize that the use of attention based sequence modeling of the neural activity preserves the temporal dependence of the neural activity and helps in providing richer context compared to the binned data. Just like with the autoencoder, the transformer embeddings are validated using a SVM classifier with a RBF kernel. To maintain comparability across models, we maintain the same embedding dimension as the autoencoder (8 dimensions) and the same SVM configuration across all three models.

## 5 Results

### 5.1 Support Vector Machine

The confusion matrix is a visual representation of what labels the SVM predicted and what the true labels were. The x-axis corresponds to what label was predicted, and the y-axis corresponds to the true label. The color of each block represents how many times a true label was predicted as another label. The diagonal is where correct predictions reside since the predicted and true labels match. The SVM trained on the raw data has an average accuracy over five folds of cross-validation of 64.97%.

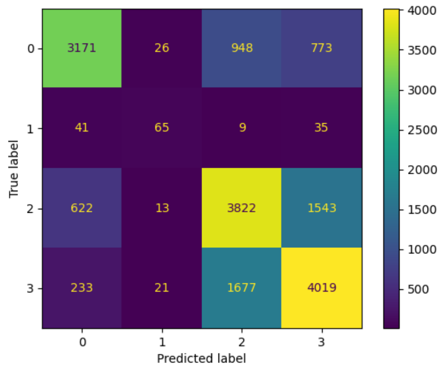
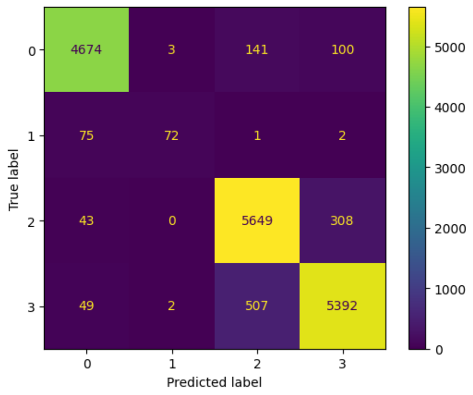


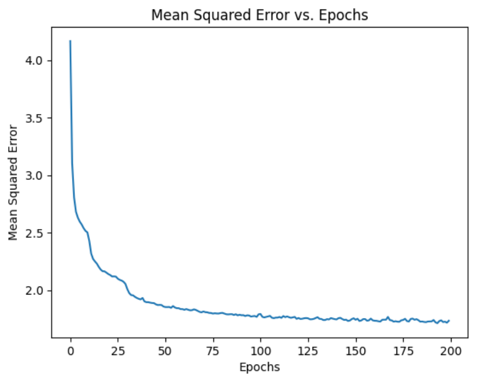
Figure 3: Confusion matrix of the predicted sets of stimuli and true sets of stimuli from the SVM.

### 5.2 Autoencoder

When using the encoded data from the autoencoder, the SVM was able to achieve an average accuracy over five folds of cross-validation of 92.53%. As we hoped, the accuracy of the SVM using the autoencoder encodings did improve over the SVM using the binned data, which is encouraging.



(a) Confusion matrix from the SVM using encoded data from the autoencoder.



(b) Mean squared error decreasing as the autoencoder trained itself.

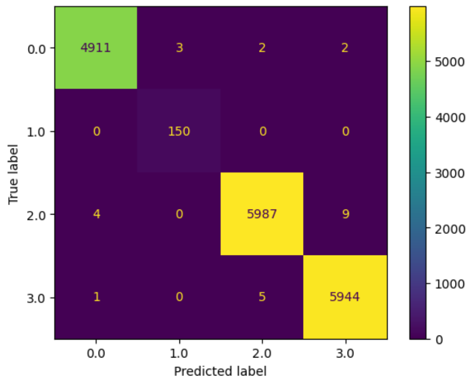
Figure 4: Confusion matrix and losses from the autoencoder.

The graph of the mean squared error dropping over the epochs indicates that the autoencoder was able to learned how to encode the binned neural activity data such that it could recover the original inputs from the low-dimensional encoding.

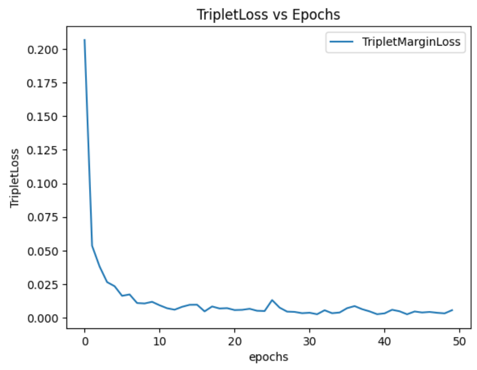
### 5.3 Transformer

When using the encoded data from the transformer, the SVM was able to achieve an average accuracy over five folds of cross-validation of 99.82%. As hypothesized, the SVM using the transformer encodings outperformed both the SVM using the autoencoder encodings and the simple binned data. The transformer hyperparameters were manually fine-tuned to reach the above mentioned performance.

The graph of the TripletLoss dropping indicates that the transformer was able to find a way to encode each $n \times 50$ time-series binary array into a vector of size 8 such that it could differentiate between two encodings from different stimuli and tell that two encodings were from the same stimulus.



(a) Confusion matrix from the SVM using encoded data from the transformer.



(b) Triplet loss decreasing as the transformer trained itself.

Figure 5: Confusion matrix and losses from the transformer.

## 6 Discussion and Analysis

### 6.1 Support Vector Machine

Our baseline, the SVM using the binned data, performed decently well. As stated above, the SVM was around 65% accurate at figuring out what set of stimuli were being shown by looking at the neural activity. For reference, if we were just randomly guessing, we would expect to only be about 25% accurate when predicting the set of stimuli.

### 6.2 Autoencoder

The autoencoder had a dramatic improvement over our baseline. As previously stated, the SVM using encoded data from the autoencoder was around 92.5% accurate at figuring out what set of stimuli were being shown. The dramatic improvement in performance over our baseline indicates that the autoencoder was successful at finding a information-rich low-dimensional latent space, which was exactly what we wanted.

One of the constraints of the autoencoder is that the input dimensions of the data is fixed. This poses an issue if a mouse has extra Neuropixel probes (data has more dimensions) or has missing Neuropixel probes (data has less dimensions). We decided to only use data from one mouse to simplify working with the dataset.

Picking the number of dimensions to encode the data into was also an arbitrary decision we had to make. We chose to use 8 dimensions because the paper that inspired this project [4] also used 8 dimensions.

### 6.3 Transformer

The transformer model significantly outperforms both the Autoencoder and SVM models, showcasing a notable margin in performance (99.82% compared to 92.53% and 64.97% respectively). This substantial improvement in performance likely stems from specific factors unique to the transformer model.

The use of an attention mechanism within the transformer allows it to effectively focus on different time steps within the time-series binary data. Additionally, employing triplet loss, as opposed to Mean Squared Error (MSE) loss, encourages the model to generate representations that cluster more efficiently within the embedding space.

Moreover, utilizing time-series binary data, as opposed to simpler binned data, provides the transformer with richer information. This richer dataset maintains the temporal relationships between each neuron, a crucial aspect that gets lost when using binned data. Preserving these temporal relationships enhances the model's ability to capture intricate patterns in the data. However, despite these advantages, it's important to note that the transformer encounters similar limitations regarding input dimensions as the autoencoder does.

### 6.4 Problems

One challenge our models encounter relates to their limited transferability across different mice within our dataset. The input data comprises neuron spike counts recorded while presenting a stimulus. The problem arises from the lack of standardization in the placement of Neuropixel probes used to measure neuron spikes across mice. Each mouse has a varying number and arrangement of these probes, essentially resulting in different "features" within the dataset for each mouse.

One potential solution to standardize these "features" represented by Neuropixel probes involves considering only those probes located in very similar or identical positions across all mice. Each probe in the dataset contains metadata detailing its location, depth, and other relevant information. By restricting the use of probes situated in consistent positions among all mice, we anticipate the possibility of employing our models effectively across any mouse in the dataset.

### 6.5 Future Work/Improvements

In order to enhance the applicability of our models, we aim to create specialized features that don't depend on the specific number of neurons involved, yet effectively capture the changes in neuron states over time.

Our hypothesized approach involves generating a feature vector for each time interval (e.g., every 25 milliseconds, comprising 5 binary values). This vector uses binary encoding to retain temporal information, incorporating the count of unique encodings observed within a time bin (32 combinations), and the variation in encoding compared to the preceding time bin (resulting in 1024 combinations). This amalgamation results in a 1056-dimensional feature vector.

Crucially, this approach is agnostic to the order of neurons and maintains the temporal nuances of each neuron's activity. Since this vector disregards neuron order and operates algorithmically, it can effectively represent neural activity across multiple mice. Models trained using this feature vector have the potential to generate generalized embeddings that possess both transferability and resilience when applied to various mice's neural activities.

Furthermore, these embeddings can be tested to decipher the stimulus based on the observed neural activity, effectively serving as an inverse interpretation of brain function.

## References

[1] Dmitrii Babaev, Nikita Ovsov, Ivan Kireev, Maria Ivanova, Gleb Gusev, Ivan Nazarov, and Alexander Tuzhilin. Coles: Contrastive learning for event sequences with self-supervision. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD/PODS '22. ACM, June 2022.

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[3] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network, 2018.

[4] Steffen Schneider, Jin Hwa Lee, and Mackenzie Weygandt Mathis. Learnable latent embeddings for joint behavioural and neural analysis. *Nature*, 617(7960):360–368, May 2023.

[5] Joshua H Siegle, Xiaoxuan Jia, Séverine Durand, and et al. Survey of spiking in the mouse visual system reveals functional hierarchy. *Nature*, 592(7852):86–92, April 2021.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.