Dynamical Systems Reduction in Experimental Brain Modeling

An exploration of machine learning approaches to system identification in neural spike data

A Practicum Report in partial fulfillment of the requirements for a Master of Science in Data Science at Brown University

Christopher J. Rohlicek
B.A. Applied Mathematics, Harvard University, 2020

August 5, 2021

Contents

List of Figures

List of Tables

1 Problem Statement

Physical and simulated neurological systems are generally complex and one cannot easily discern the underlying dynamical structure from measurement data. In the case of having a computational model based on interconnection of vast numbers of basic units (e.g., neurons, or sub-neuron compartments), such a model generally suffers from large computation requirements needed to generate meaningful amounts of data.

Many researchers have considered the problem of forming "simpler" generative models that can produce "realistic" time series data, for example fitting data experimentally obtained from a physical system and/or generated using a complex computational model [1][2][3][4]. However, although perhaps computationally more tractable, such approaches do not necessarily provide insight into the underlying structure of the system being simulated.

There is a need to be able to accept measurement time series data of a complex neurological system (e.g., large number of recording channels) and form a simplified time series that preserves "essential" aspects of the original data in the sense that the original data can be reconstructed, or at least that a complex time series with essential properties equivalent to the original data can be. Such a simplification may, for example, be represented by a mapping from a high-dimensional space to a reduced dimensional space in which behavior of the system may be more evident.

A further need, going beyond merely preserving essential aspects of an original time series, is an ability to describe causal system dynamics in such a simplified space. This goes beyond being able to "invert" an entire simplified time series to produce the entire original series, and may provide insight to the structure of the system.

Moreover, it should be possible to accurately predict responses to perturbations or other conditions different from the original data. Ideally, in both the mapping of data and the inference of a dynamical system, it is desirable for the result to be insensitive to the specific experimental conditions. Under this condition, multiple recording setups would map to the same underlying structure, thereby permitting a very practical level of generalizability to a given neural structure.

An example of a neurological model that has been studied relates to the mammalian visual system [5]. The neural hierarchy of the visual system has been studied extensively in the human and non-human primate, but evidence in mice is much more sparse, as the mouse's smaller brain presents challenges to multi-area recording.

2 Summary of Aims and Findings

The overarching goal of this project was to develop a method for identifying low dimensional dynamical systems from high dimensional neural recordings in a way that preserves the dynamical structure of the data. The purpose of this method is to extract "intrinsic" dynamics that capture the essential behaviors of complex neural systems, whether simulated or identified only by experimental data. This problem draws on many different disciplines, including dynamical systems, machine learning, and classical statistical methods. Further, many applications of data science practices in this project were motivated by the large size of the datasets: to account for both a large number of neurons and the fast timescale of their activity, modern neural recordings involve hundreds of dimensions (electrodes) at kHz sampling rates.

At the outset, the goal of the project was the development of a system that takes high-dimensional neural spike time series data as input and produces either a less complex time series that preserves fundamental qualities of the original data, or a dynamical system that can generate (e.g., extrapolate) that less complex time series. In other words, the output time series will be less complex in that it will exist in a lower-dimensional space, and the essential qualities of the original data will be preserved in the sense that the transformation applied by the system can be undone to recover the input.

The specific forms of the models used to discover the reduced-dimension time series from the input data was determined by successive stages of experiments that built up to the full-scale task. In the final stage of experiments, the model being used (and whose design was supported by both domain knowledge of the data and by the results of earlier experiments) appeared unable to learn any dynamical structure in the input data.

From the model's failure to find a dynamical embedded state space for the input data, the explanation that seemed most plausible was that there was in fact no dynamical structure in the experimental neural spike data being used, for example, due to inadequate spatial sampling of neurons actually measured. In order to test this hypothesis I performed a series of analogous experiments with synthetic datasets which I built with known underlying dynamical structure. These experiments provided evidence that supports my hypothesis, as the experiments using an artificially non-dynamical dataset reached the same results as the ones performed on the real-world data, and likewise the experiments using an artificially dynamical dataset successfully yielded an embedding of the data that preserved the dynamical structure.

3 Data

3.1 Allen Institute Neuropixels Data

The main source of project data was the Allen Institute Brain Atlas [6]. The Brain Atlas includes many experimental datasets recorded using Neuropixels silicon probes, which are very densely-packed sensors that provide data with a high level of spatial resolution. The recordings used specifically were from experiments in which mice were presented with the visual stimulus of a drifting gradient. The data is published online and is publicly available via the Allen Institute software development kit (SDK)[1]. The SDK includes Python modules designed to let users take advantage of specific subsets of the data (in total amounting to 855 GB) in a convenient, memory efficient manner. More specifically, the Allen Institute Neuropixels dataset originates from experiments designed to characterize the functional hierarchy of the mammalian visual system [7].

3.2 Data Processing and Assembly

To assemble the main dataset used throughout this project, I used the AllenSDK to collect all of the spike times from the different trials of drifting gradient presentations. I chose to limit the data to those trials in which the drifting gradient stimulus was used in order to keep the underlying structure of the data as consistent as possible (the rationale being that different stimuli would likely activate different structures in the brain). I chose to collect the data from all available drifting gradient trials in order to have a large amount of data without using artificial data augmentation methods that might have introduced bias into the data. Lastly, I chose to limit all of the trials included in the data to be from the same experimental session. This was because of the fact that the specific recording apparatus is kept constant throughout an experimental session, so in order to prevent any bias from making its way into the data as a result of a change in recording equipment or arrangement of probes I limited the data to a single experimental session.

Within the chosen experimental session there were 628 trials of available data for the drifting gradient stimulus, but of those, 141 included recording errors in which a subset of the recording channels failed to capture data. In order to avoid problems that could be caused by these missing regions of data, I removed the corrupted trials from the dataset and proceeded with 487 trials worth of data. Each trial was 2 seconds long and recorded at a 30 kHz sampling frequency, and included 131 recording channels ("units"). In order to create a more manageable amount

[1] `https://allensdk.readthedocs.io/en/latest/visual\_coding\_neuropixels.html`

of data and reduce measurement bias from the data I binned the spike times to a resolution of 1 millisecond (this is a common preprocessing step when dealing with neural spiking data because of the high variance in neural spike times). To create the binary spike dataset from the AllenSDK spike times, I created a matrix with 131 rows (corresponding to each recording channel) and 2001 columns (corresponding to the distinct milliseconds in the trial) for each trial, and populated the cells in the matrix with a 1 value at the points corresponding to the channel and time in which at least one spike occurred, and a 0 elsewhere.

3.3 Synthetic Datasets

Based on the results found in the later experiments, I also created two additional synthetic datasets so the results yielded by the empirical data from the Allen Institute would be comparable to the results from data with known characteristics. The results that led to the creation of these additional datasets and the their use will be described in the results section.

3.3.1 Dynamical Dataset

The first synthetic dataset was created with explicit dynamical structure that would be simple enough for the embedding model to identify easily. To generate this dataset in a way that included variability somewhat matching the characteristics of the empirical Allen Institute data, I calculated the ensemble-wide average channel interspike time from the empirical data and stored that as the variable τ. Using this, I defined an oscillator with period τ to give the underlying state of the synthetic data at a given time. Each channel was modelled as an inhomogeneous Poisson process where the rate varied periodically with period τ, with the rate for a particular channel being modelled as having a normal distribution with a randomly drawn mean (in phase) and a common variance. Randomly generated spikes were binned into 1-millisecond bins as with the empirical data. For a detailed explanation of how this data was generated, see Appendix A.

3.3.2 Non-dynamical Dataset

To create the matching synthetic dataset built with no dynamical structure linking one channel's state to another, I iterated through a blank dataset of the same shape as the Allen Institute data and in each recording channel placed the same total number of spikes but assigned them to indices drawn from the uniform distribution. This has the effect of maintaining major aspects of the empirical data's characteristics, while ensuring that each channel acts as an independent Poisson process.

3.4 Exploratory Data Analysis

3.4.1 Extracting and Cleaning Data from AllenSDK

The Allen Institute Brain Atlas provides access to large amounts of experimental data through use of the AllenSDK Python API. To extract the data using the AllenSDK we created a local cache that allows for access to specific sessions' data. One session contains the data collected for multiple trials of different experiments that were all conducted under a fixed set of experimental conditions (i.e. a single set of subject specimens, experimental setups, etc.).

Within the choice of a single session (session ID: 715093703) we filtered the data to select only those recording channels with a high signal to noise ratio (SNR). Due to experimental conditions during the recording process, some channels of data were corrupted with more noise than others so the data is labeled with each channel's SNR. In order to remove noisy data from the set that would be used through the project and to make a dataset of more manageable size, only the channels with an SNR > 4 were retained. This filtering step reduced the number of active recording channels in the data from 884 to 131. From the filtered set of high-SNR data, we selected only the recordings associated with the presentation of a drifting gradient visual stimulus. Each of these drifting gradient trials includes two seconds of data, throughout which the mouse was being shown the visual stimulus (that is, the stimulus is active throughout the trial). In analyses of neural systems under stimulus presentations, it is important to note the time intervals in which the stimulus is active and in which it is not. Without knowing this it is possible to include the change from active to inactive stimulus in a set of data and then falsely attribute any change in output to the neural system rather than a change in input stimulus.

The AllenSDK can be used to produce raster plots of spike times for a given stimulus presentation (i.e. a single trial of a stimulus being presented and the neural response being recorded) in the form shown in figure 1.

At this stage of data selection, we had 628 trials' data in the dataset, but some of the trials contained incomplete data due to certain recording channels having failed to collect data. The way that these sensor failures were documented in the AllenSDK was through a session attribute that listed invalid time intervals, and a warning message that would get presented to the user if any of their selected data was drawn from an invalid time interval. To inspect the data to find sensor failures in the full set of trials we had selected we generated a heatmap of active recording channels in each trial, which is shown in figure 2.

To check that the missing data that can be seen in figure 2 corresponded to the invalid time intervals listed for the session, we marked each trial that contained spike times from any of the invalid intervals and found that that accounted for all of the damaged trials. To clean the data

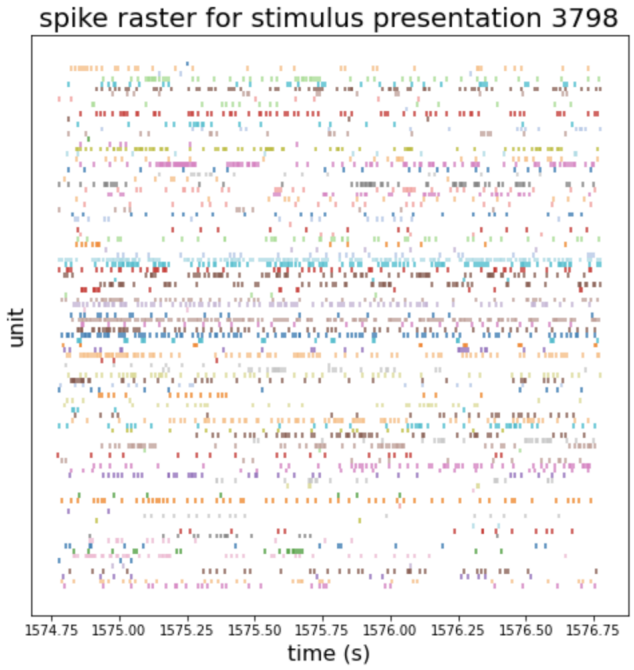


Figure 1: AllenSDK-generated raster plot of spikes for single trial

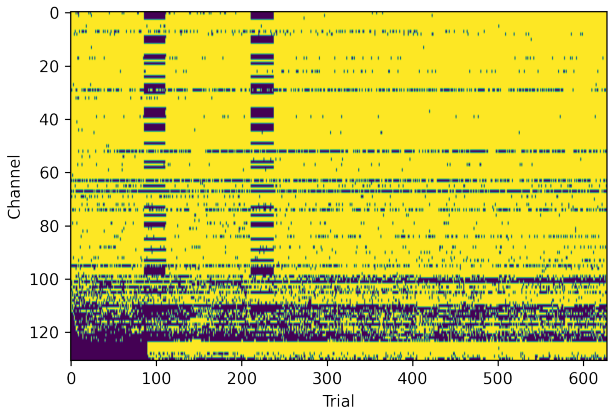


Figure 2: Binary heatmap of active recording channels in trials

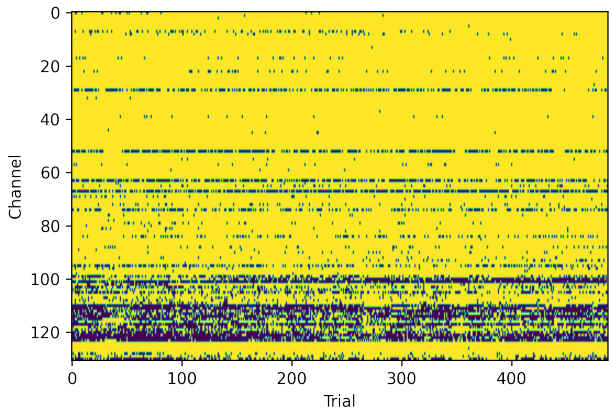


Figure 3: Binary heatmap of active recording channels after cleaning the data

we removed these damaged trials from the dataset rather than implement any kind of imputation because there was no reliable way to impute the missing values based on the surrounding sensor data both because of the complexity of the system and because of the high proportion of missing values. Further, in the AllenSDK documentation regarding these missing values it was clear that the sensor failures were independent from specific neural activity, so we were confident that removing the damages trials would not bias out remaining data.

To remove the impacted trials we traversed through the spike times in each of the trials' data and removed all trials with spike times that lied inside any of the invalid intervals. After removing all of the trials that overlapped with the invalid intervals we found a much cleaner set of data which we can visualize again in terms of active channels versus trial (see figure 3). Removing the corrupted trials eliminated 141 trials, bringing us to a dataset comprising 487 trials' data.

As described in the Data Processing and Assembly section, the AllenSDK spike time data was used to generate the binary data that we later gave as input to the models and used as the main object of the analyses through the project. The main experimental dataset was of shape (487 trials, 131 dimensions, 2001 samples).

4 Models and Methods

The goal of the project was to process the data shown in the previous section and identify the underlying dynamical structure in such a way that an analogous lower-dimensional dataset could be generated with the same structure, or such that the dynamics could be analyzed on their own. Before describing the models used I will establish the notation I will use to talk about the problem.

4.1 Notation and Problem Setup

The input to the models were discrete spikes (binned to 1-ms resolution), denoted $s[n,t] \in \{0,1\}$ for $1 \leq n \leq N$ the number of recording channels and $1 \leq t \leq T$ the time (bin number) of a given sample ($T = 2001$ as results from the 1-ms binning of the two seconds of experimental data given for each trial). For notation, S represents a two-dimensional range of spikes, $\underline{s}[t]$ represents a vector of the spikes across all N channels at a given time t.

Bin-occupancy probabilities, $p[n,t]$, give the continuously-valued probability that channel n will have at least one spike in the t^{th} bin.

An *embedding* of an input, E, is a reduced-order representation of the spikes $e[m,t]$ for $1 \leq m \leq M \leq N$ the reduced number of channels and $1 \leq t \leq T$. The encoders that create the models all use tanh nonlinearity functions, so all elements in an embedding will have values in the interval $(-1,1)$.

Estimated quantities are denoted with hats, e.g., $\hat{p}[n,t]$.

In my approach to this modeling task, the sequence of mappings I hope to create with these models can be summarized by:

$$\mathrm{S} \rightarrow \mathrm{E} \rightarrow \hat{\mathrm{P}}$$

I next describe a loss function between an estimated probability and a set of spikes $\mathcal{L}(\hat{\mathrm{P}}, \mathrm{S})$. Binary cross-entropy loss is of the form:

$$\begin{aligned}\mathcal{L}(\hat{\mathrm{P}}, \mathrm{S}) &= -\sum_{n,t} \left(s[n,t] \log \hat{p}[n,t] + (1 - s[n,t]) \log(1 - \hat{p}[n,t])\right) \\ &= -\sum_{n,t\,:\,s[n,t]=1} \log \hat{p}[n,t] - \sum_{n,t\,:\,s[n,t]=0} \log(1 - \hat{p}[n,t])\end{aligned}$$

In this problem setting, binary cross-entropy provides a measure of how closely a set of spikes S matches a predicted set of firing probabilities $\hat{\mathrm{P}}$[2].

[2]Minimizing binary cross-entropy loss reaches the maximum likelihood solution for the output distribution [**?**]

In order to counteract bias that can result from an imbalance of spike and non-spike samples in the data, a weighting term, w, can also be added to the binary cross-entropy loss function[3]:

$$\mathcal{L}(\hat{\mathrm{P}}, \mathrm{S}, w) = -w \sum_{n,t\,:\,s[n,t]=1} \log \hat{p}[n,t] - \sum_{n,t\,:\,s[n,t]=0} \log(1 - \hat{p}[n,t])$$

In practice, w can be used to counteract a deficit in the number of samples containing a spike relative to the number of samples not containing a spike. For example, in a case where there are 100 times as many negative samples as there are positive samples, setting $w = 100$ would result in the loss acting as if there was an equal number of samples from both classes.

A *memoryless spike-to-probability autoencoder* is a mapping of the following form:

$$\mathcal{M} : \underline{s}[t] \mapsto \underline{e}[t] \mapsto \underline{\hat{p}}[t]$$

which induces a mapping of the entire sequence $\mathrm{S} \mapsto \hat{\mathrm{P}}$. While this mapping is not strictly an autoencoder in the sense of producing the same type of output as the data given as its input, it is most informative to call this model an autoencoder because the same binary spikes are used as its input and target data. The reason $\underline{\hat{p}}[t]$ is reached is because optimization with binary cross-entropy loss will give a continuous output that can be interpreted as firing probabilities after being passed through a sigmoid nonlinearity.

A *windowed spike-to-probability autoencoder* is a mapping of the following form:

$$\mathcal{M} : \mathrm{S}\left[\cdot, t - \frac{\tau}{2} : t + \frac{\tau}{2}\right] \mapsto \underline{e}[t] \mapsto \underline{\hat{p}}[t]$$

which also induces a mapping of the entire sequence $\mathrm{S} \mapsto \hat{\mathrm{P}}$, and takes τ additional input samples into account as "time delays" that provide temporal context for the sample at time t.

Lastly, a *next-step (spike-to-probability) predictor* is a mapping of the form:

$$\mathcal{M} : \underline{s}[t] \mapsto \underline{e}[t] \mapsto \underline{e}_{+}[t+1] \mapsto \underline{\hat{p}}[t+1]$$

which induces a mapping of the sequence $\mathrm{S}[\cdot, 1 : T-1] \mapsto \hat{\mathrm{P}}[\cdot, 2 : T]$. In this expression, $\underline{e}_{+}[t+1]$ represents the *"next-step extrapolation"* of the embedded state at time $t+1$. This extrapolation of the embedding state into the future can be attained by giving the embedding $\underline{e}[t]$ to a model that simulates the state space's autonomous evolution.

[3] `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html`

4.2 Experiments

To give the dynamical systems question context, I first started with a memoryless baseline that tells us how Ill an embedding works that can only see the current time. Next, I allowed the embedding to account for a window of history and future in addition to the current time in order to see the improvement that comes from the added time context. Finally, with the best-performing embedding of the memoryless and windowed models tested, I used the encoder and decoder sub-models in cooperation with a next-step predictor that would map the embedding at time t to the predicted embedding at time $t + 1$.

All of the basic neural network functions and architectures that went into these experiments were implemented in PyTorch using their standard library.

4.3 Model Descriptions

4.3.1 Memoryless Autoencoder

The memoryless autoencoders I tested were fully-connected neural networks with hyperbolic tangent nonlinearities used in each pre-output layer. The final layer's output was passed through a sigmoid nonlinearity in order to bound the model output to the interval $(0, 1)$ so it could be interpreted as neuron firing probabilities (i.e. bin occupancy probabilities). The parameters tested were the number of layers used in the encoder and decoder networks, the bottleneck dimension, and the weighting term applied to the binary cross-entropy loss.

4.3.2 Windowed Autoencoder

The windowed autoencoders I tested followed the same architecture and hyperparameter exploration process as the memoryless models, but with the difference in input being that the windowed models would accept two-dimensional arrays of input from a series of times rather than a sample from a single time. Figures 4 and 5 show the different function of the two types of model in their application to the binary spike data.

4.3.3 Next-step Predictor

The next-step predictor I built places a recurrent neural network in between an encoder and decoder network (such as the ones trained in the two halves of the autoencoder networks described above). The role of the different parts of the model and the dimensionality of the intermediate quantities can be described as follows:

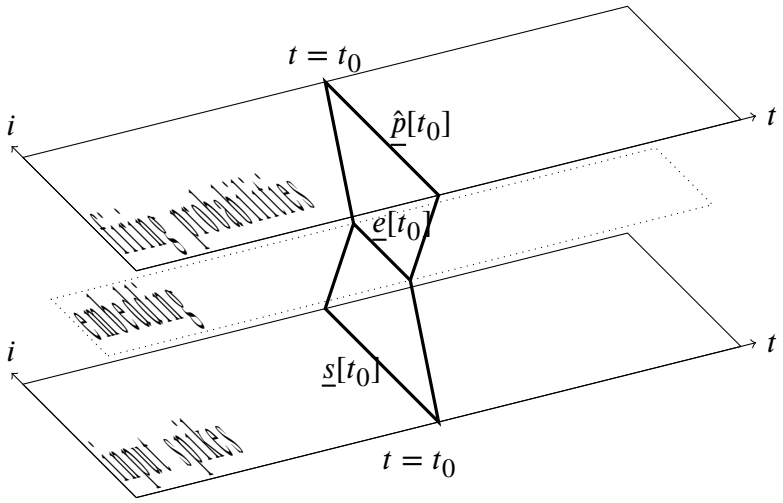


Figure 4: Memoryless spike-to-probability autoencoder

- $\text{Enc}\left(\text{S}[\cdot, 0 : T-1]\right) = \text{E}[\cdot, 0 : T-1] \in \mathbb{R}^{M \times T-1}$ for M the bottleneck dimension of the autoencoder, or equivalently, the dimension of the embedding space. This network is able to take in a sequence of samples because the encoder network induces a sequence-to-sequence mapping through its sample-wise application to all of the samples in a given input sequence.

- $\text{RNN}\left(\text{E}[\cdot, 0 : T-1]\right) = E_+[\cdot, 1 : T] \in \mathbb{R}^{M \times T-1}$. The RNN takes a sequence of samples as input and returns an equally-sized (both in length and dimension) sequence of next-step extrapolations.

- $\text{Dec}\left(E_+[\cdot, 1 : T]\right) = \hat{\text{P}}[\cdot, 1 : T] \in \mathbb{R}^{N \times T-1}$ for N the number of recording channels in the input data. As with the encoder network, a sequence-to-sequence mapping is induced with this feedforward network by applying it to each sample in the sequence.

In the experiments described in this report, I tested and compared models with RNNs of depth equal to 3 layers. Under ideal training behavior, the RNN should be able to learn the next-step update rule for a given embedding $\underline{e}[t]$, which could be described as the RNN successfully learning the dynamics of the embedded data space. Figure 6 shows the full structure of the predictor network.

Because of the relatively complicated coordination that needs to be executed between the different parts of this model, there are certain weight-initialization and training procedures that were used in some of the experiments in an effort to alleviate stress on the optimization process.

To initialize the weights in the predictor's encoder and decoder networks to a useful starting

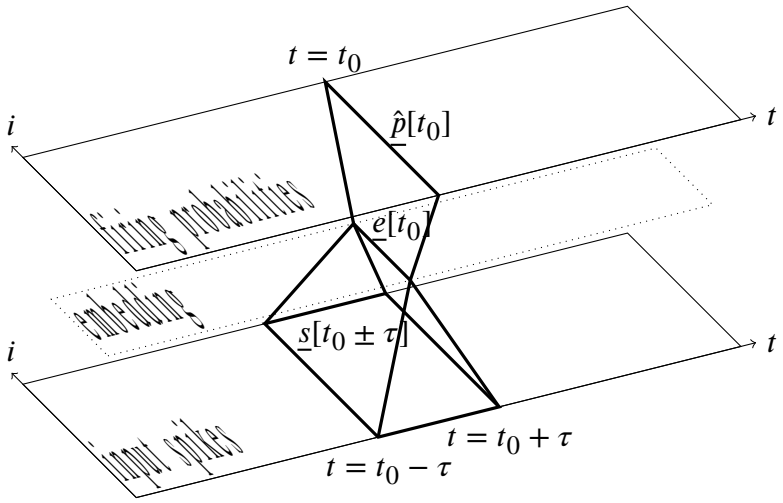


Figure 5: Windowed spike-to-probability autoencoder

point for the training procedure, one can initialize with the weights found in the encoder and decoder networks from one of the trained memoryless or windowed autoencoder models. The point of those models' training procedure is that the encoder and decoder networks are able to map the data into and out of an embedding space that is optimized for reconstruction of the original sample. The intuition behind using these weights to initialize the encoder and decoder of the next-step predictor is that the optimal function to map the data into and out of the RNN may be close to the functions that are optimized for static data reconstruction.

The second initialization step can be applied to the weights in the RNN. The intuition for the initialization procedure is that the true update step would be more easily found by starting from a combination of identity functions than a randomly-initialized set of weights. The update function learned by the predictor's RNN can be expressed as follows:

$$\underline{e}_+[t+1] = \tanh\left(W_{ih}\underline{e}[t] + W_{hh}\underline{e}_+[t] + \underline{b}_h\right)$$

for W_{ih} the input-to-hidden transformation (matrix), W_{hh} the hidden-to-hidden transformation, and $\underline{b}_h$ the bias[4]. By initializing $W_{ih} = W_{hh} = I + \varepsilon$ (for ε small-magnitude noise) the weights were set to start at a point where the next embedded state is a simple, and plausibly useful, function of the previous one.

Based on the intuition behind both of these initialization steps, a two-staged training procedure was also implemented in which the model was first trained with the encoder and decoder

[4]In Pytorch, the RNN layer object is implemented with two bias objects b_{ih} and b_{hh}, but this is functions equivalently to the case of having a separate bias for both matrices.

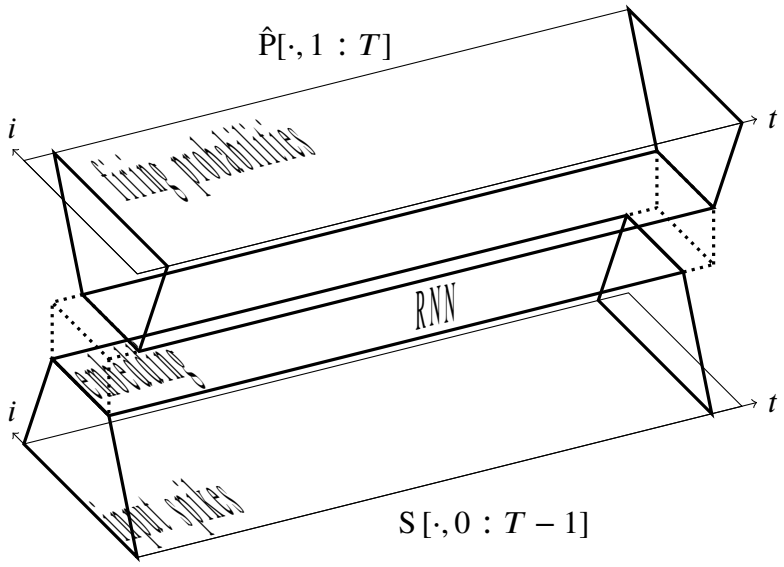


Figure 6: Next-step spike-to-probability predictor

weights frozen, and then training was continued with all weights in the network unfrozen and able to train concurrently. The reason for this approach is that under the initialization procedure it is reasonable to assume that the RNN weights will be farther from optimal than the encoder and decoder weights. Because of the relatively small number of weights in the RNN compared to the encoder and decoder, training all of the weights together could result in the RNN weights' gradient being dwarfed by the encoder and decoder's collective gradient, thus making training unstable and ineffective for the RNN. The encoder and decoder weights, having been trained previously on the static data reconstruction task, are likely to need only fine-tuning for the prediction task.

5 Results

In the following section I will first describe the results of the memoryless and windowed spike-to-probability autoencoder experiments, then the results of the next-step predictor experiments, and finally the ensuing analysis and further experimentation used to interrogate the conclusions drawn from those outcomes.

5.1 Memoryless autoencoder

In order to choose the best memoryless spike-to-probability autoencoder that could serve as the initial state for the encoder and decoder in the predictive model I performed a parameter sweep over various combinations of numbers of layers, binary cross-entropy weighting terms, and bottleneck dimensions. The values used for number of layers were $L \in \{1, 2, 3\}$, for binary-cross entropy weight were $w \in \{25, 50, 100\}$, and for bottleneck dimension were $M \in \{10, 20, 50, 100\}$.

To measure the performance of each model I applied the trained network to the entire dataset and compared the output to the input on an entry-by-entry basis. The output of the transformation is a dataset of equal size with corresponding entries to the original dataset, but each entry gives the model's prediction of neuron spike probability given the encoding of the time sample at the network bottleneck. To get an aggregate measure of model performance over all entries in the dataset, I calculate the average predicted probability of neuron spiking under the input condition of there being a spike and under the condition of there not being a spike (for implementation see Appendix B.

Table 1 shows the results of each model that was tested. Because the main objects of interest in these experiments were the models that could embed the data into a low-dimensional space, I focused on the performance of the models with a bottleneck dimension of 10 or 20. Among these experiments the 2- and 3-layer models showed comparable performance, so the 3-layers models were deemed most suitable for use in the next-step prediction models based on their increased expressivity over the 2-layer models. The binary cross-entropy weight term showed minor impact on the overall model performance, with an increased value raising both the average probability of correctly predicting a spike and incorrectly predicting a spike.

5.2 Windowed autoencoder

After comparing different forms of the memoryless autoencoder I performed a set of experiments to see if the models' performance was improved by the addition of surrounding time samples to a given input (that is, where the memoryless model took a single vector $\underline{s}[t]$ as input, the windowed

Performance of memoryless autoencoders				
Layers	Bottleneck dimension	BCE weight	$\hat{p}_{avg}$(spike|input spike)	$\hat{p}_{avg}$(spike|no input spike)
1	10	25	0.8823	0.0567
1	10	50	0.9321	0.0733
1	10	100	0.9642	0.0960
1	20	25	0.9732	0.0152
1	20	50	0.9848	0.0210
1	20	100	0.9915	0.0259
1	50	25	0.9984	0.0006
1	50	50	0.9986	0.0009
1	50	100	0.9992	0.0014
1	100	25	1.0000	0.0000
1	100	50	1.0000	0.0000
1	100	100	1.0000	0.0000
2	10	25	0.9657	0.0142
2	10	50	0.9829	0.0229
2	10	100	0.9887	0.0305
2	20	25	0.9868	0.0083
2	20	50	0.9876	0.0104
2	20	100	0.9938	0.0163
2	50	25	0.9959	0.0011
2	50	50	0.9975	0.0026
2	50	100	0.9977	0.0049
2	100	25	0.9955	0.0015
2	100	50	0.9967	0.0025
2	100	100	0.9975	0.0049
3	10	25	0.9775	0.0118
3	10	50	0.9840	0.0189
3	10	100	0.9904	0.0266
3	20	25	0.9842	0.0066
3	20	50	0.9903	0.0099
3	20	100	0.9939	0.0142
3	50	25	0.9916	0.0041
3	50	50	0.9939	0.0056
3	50	100	0.9951	0.0129
3	100	25	0.9809	0.0065
3	100	50	0.9841	0.0110
3	100	100	0.9862	0.0169

Table 1: Memoryless autoencoder results

Performance of windowed autoencoders					
Layers	τ	Bottleneck dimension	BCE weight	$\hat{p}_{avg}$(spike|input spike)	$\hat{p}_{avg}$(spike|no input spike)
3	10	10	25	0.7202	0.0885
3	10	10	50	0.8330	0.1305
3	10	10	100	0.8908	0.1788
3	10	20	25	0.7949	0.0631
3	10	20	50	0.8762	0.0924
3	10	20	100	0.9253	0.1341
3	20	10	25	0.2519	0.1840
3	20	10	50	0.6940	0.1725
3	20	10	100	0.7904	0.2563
3	20	20	25	0.5573	0.1199
3	20	20	50	0.7612	0.1547
3	20	20	100	0.8397	0.2352
3	50	10	50	0.4410	0.2754
3	50	10	100	0.5571	0.4524
3	50	20	25	0.2570	0.1878
3	50	20	50	0.4754	0.2661
3	50	20	100	0.6291	0.3937

Table 2: Windowed autoencoder results

model takes a series of vectors $S[\cdot, t-\tau : t+\tau]$ as input). Because the memoryless autoencoder experiments led us to the conclusion that the 3-layer models would be the best choice to use in the prediction models, only 3-layer models were tested in the windowed experiments. Additionally, the only models that were tested were those with a bottleneck dimension of 10 or 20 because the models with larger bottleneck size were proven to be unnecessary in the memoryless experiments, as the 10- and 20-dimensional bottleneck models performed well. The results of the windowed autoencoder experiments can be found in table 2, and their performance was calculated using the same model evaluation function described for the previous experiments.

The windowed autoencoder results show that the widening of the model input to include neighboring time samples reduced the model's ability to predict neuron spikes. Because this task involves strictly more information given as input than was given in the memoryless task, the diminished performance is likely the result of the increased training burden of optimizing a larger neural network. In these experiments, the windowed models are much larger than the memoryless models in order to account for 20, 40, or 100 more samples per input than in the memoryless case (those being the input size increases for the $\tau = 10, 20, 50$ models respectively).

Given these results from adding windowed input to the model, I moved onto building the next-step predictor using the 3-layer memoryless autoencoder as the starting point for the encoder and decoder sections of that model.

Performance metrics for predictors applied to empirical data			
M	Max. channel var.	$\hat{p}_{avg}$(spike|input spike)	$\hat{p}_{avg}$(spike|no input spike)
10	0.0006	0.4093	0.3163
20	0.0209	0.4152	0.3078

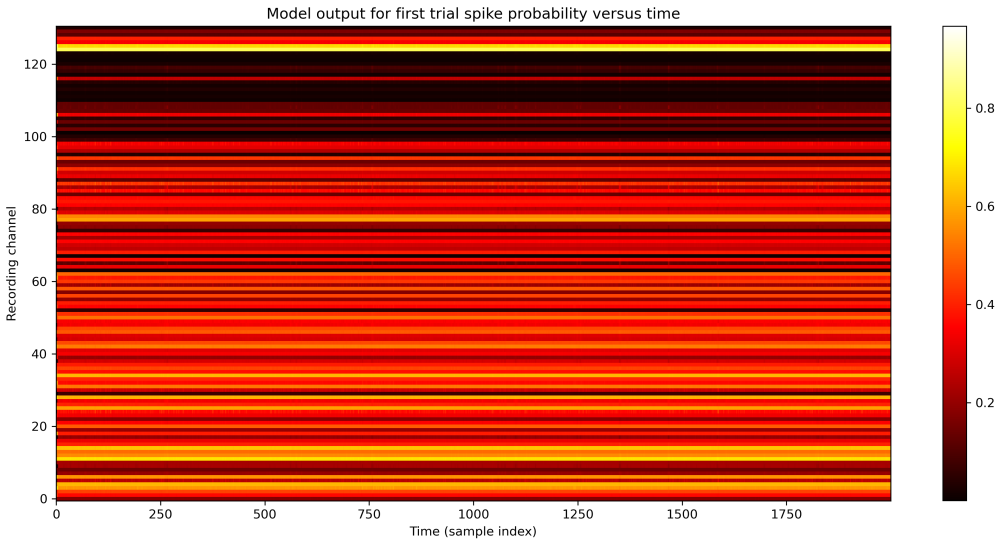
Table 3: Performance metrics for reduced-dimension predictors

5.3 Next-step predictor

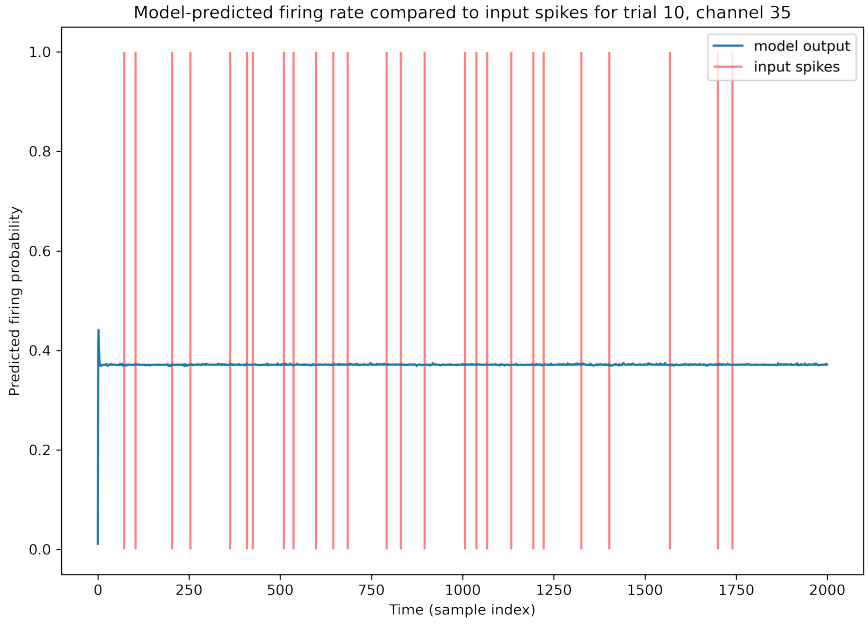
Given the results of the experiments involving the static autoencoders, I initialize the encoder and decoder sections of the next-step predictor with the weights taken from the memoryless autoencoder with a binary cross-entropy weight of $w = 100$ and bottleneck/embedding dimension of whatever is desired for a given experiment.

The first experiments I ran were those using an embedding dimension of $M = 10$ and $M = 20$. Both of these experiments showed model outputs that were only minimally variable in time. Figures 7 and 8 show the outputs of the $M = 10$ and $M = 20$ models, respectively. In addition to those plots, model performance was also measured using an adapted form of the model evaluation method given in Appendix B to calculate the same average spike probabilities. To quantify the degree to which the model's output was constant through time, maximum variance in any single channel's output through time was also calculated. Those results are given in table 3.

After finding that the next-step predictor failed to identify dynamics in the empirical AllenSDK data (indicated by the lack of time-varying output given by the models) a number of diagnostic experiments were performed to investigate the possible causes of the models' shortcomings.

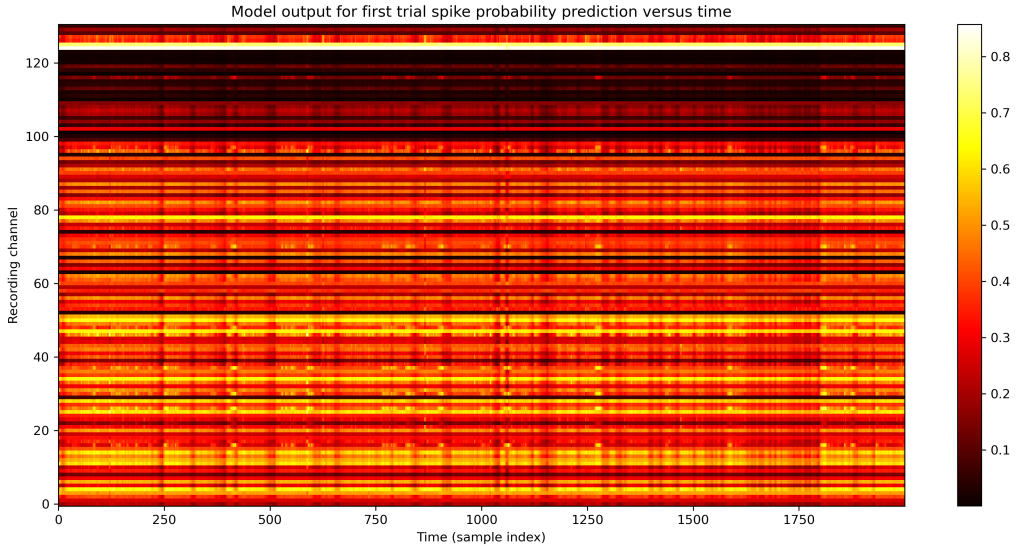


(a)

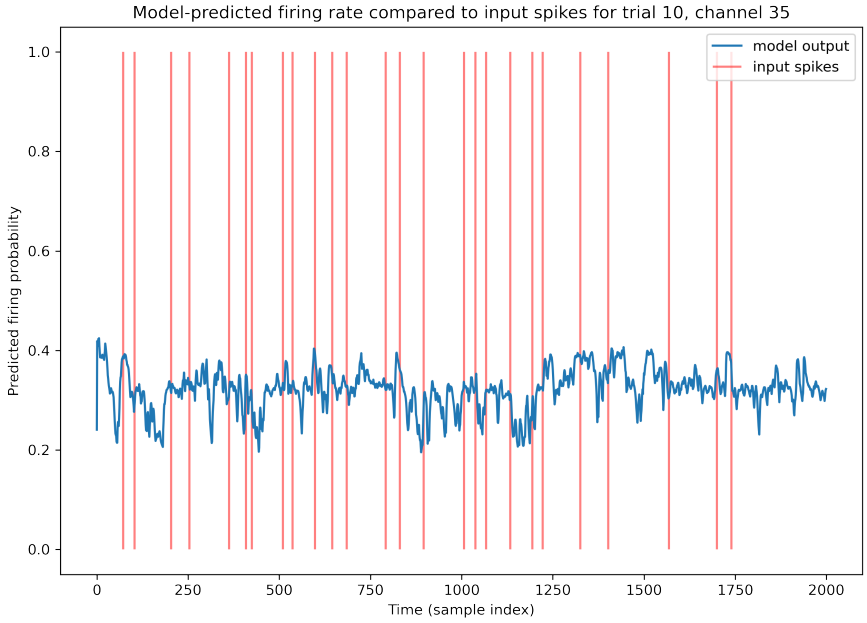


(b)

Figure 7: Model output for $M = 10$ condition: (a) is a heatmap of all channels for a single trial, and (b) is a single channel and reference input spikes.



(a)



(b)

Figure 8: Model output for $M = 20$ condition: (a) is a heatmap of all channels for a single trial, and (b) is a single channel and reference input spikes; while this output does clearly change with time, it fluctuates with a very small variance.

6 Diagnostic Experiments

In the next-step prediction experiments, the models appeared to learn to give an output that remained roughly constant through time (in other words, the trained models gave an output that largely seemed to ignore its input). This behavior was inconsistent my expectation that the data should have an underlying dynamical structure. This expectation was the result of domain knowledge of the biophysical system that generated the data, but given the models' lack of ability to identify that structure I ran a series of experiments meant to check for possible sources of error.

6.1 Synthetic Dynamical Data

The first set of diagnostic experiments I ran were to check whether the next-step predictor could identify and learn a very simple dynamical structure built into a dataset. To create two datasets a causal structure between channels we used the approach described in Appendix A with standard deviations of $\sigma = 5$ and $\sigma = 20$ bins on each channel's firing probability function. The resulting structure of this dataset was such that the different channels would reach their peak firing probabilities in a cascading manner, such that channel i's peak would come before channel $i + 1$'s peak, etc. This pattern would repeat (i.e. each channel's firing probability function would repeat) with a period equal to the average interspike time calculated over the ensemble of channels in the empirical dataset. The purpose of creating datasets with $\sigma = 5$ and $\sigma = 20$ was to provide a way of observing the behavior that would result from adding more noise to the system.

Using these datasets we conducted experiments in which next-step predictors were trained under the following conditions:

- an embedding dimension $M = 10$, using the special initialization and two-staged training procedures described in the previous experiments, and using the $\sigma = 5$ dataset
- $M = 131$ using random initialization and single-staged training, and using the $\sigma = 5$ dataset
- $M = 10$ using the special initialization and two-staged training procedures, and using the $\sigma = 20$ dataset

The inputs and results of the experiments are shown in Figures 9 and 10, and the numerical performance metrics are given in Table 4.

From the performance of the next-step predictor on the diagnostic task of being presented a dataset with simple dynamical structure (with varying levels of noise), one can conclude that the model itself is able to identify simple dynamical structure.

Performance metrics for predictor experiments					
M	σ	Max. channel var.	$\hat{p}_{avg}$(spike|input spike)	$\hat{p}_{avg}$(spike|no input spike)	Dataset
10	5	0.0671	0.6737	0.1515	dyn.
10	20	0.0454	0.3579	0.2184	dyn.
131	5	0.0623	0.6796	0.1486	dyn.
10	N/A	0.0003	0.3985	0.3069	non-dyn.
131	N/A	$2.8133 \cdot 10^{-08}$	0.4006	0.3080	non-dyn.
10	N/A	0.0006	0.4093	0.3163	empirical
131	N/A	0.0223	0.3746	0.2849	empirical

Table 4: Performance metrics achieved by predictors with various datasets

In addition to the above experiments, I also inspected the two dynamical datasets and measured different characteristics that provide insight into the independence of the different data channels. The point of these analyses is to investigate the possibility that the empirical data lacks correlational structure between its different channels of data. The most informative characteristics that were measured were the number of distinct 131-vectors present in the data (i.e. the number of distinct input samples given to the models in training), the distribution of the average interspike times found in each channel, and the eigenspectrum of the empirical transition matrix observed in the embedded data.

The number of distinct vectors is useful to give us an idea of how the inter-channel structure in the data affects the number of distinct elements found in the dataset. The interspike time distribution is useful because it provides a straightforward way to compare the behavior of the different channels to independent Poisson processes, which would give no interdependence to the channels and would show exponentially-distributed interspike times in all channels. Finally, the eigenspectrum of the empirical transition matrix shows us the distribution of cycles between groups of states in the embedded data. It is useful to measure this quality of the embedded data space that is created by a given model because this allows us to see whether there are subnetworks of states that exhibit cycles that may be indicative of overall system dynamics.

In order to find the eigenspectrum for the embedded state space created by one of the models, I first created that state space by applying the model's encoder to all of the input data samples, and then using a k-means clustering algorithm to vector quantize the space. This quantization would divide the space into regions centered around a set number (I used $k = 100$ in all of these analyses) of centroids that define the distinct "states" of the embedding space. To then construct the state transition matrix I traversed through each of the vector sequences in the dataset and calculated the overall empirical transition probabilities based on each vector's corresponding state. This quantization strategy provided a tractable way of computing an empirical state transition

matrix given the fact that the embedded space is 10-dimensional (i.e., too large to use scalar quantization of the space to achieve any sort of useful data resolution).

The number of distinct vectors found in the dynamical dataset with $\sigma = 5$ bins is equal to 26,433 and the distribution of channel interspike times and transition matrix eigenspectrum are shown in Figure 11. It is clear from the distribution of interspike times that the channels do not show exponential distributions, which is expected given the known structure built into the data. The eigenspectrum also reflects the inter-channel structure in the data in the high number of transition matrix eigenvalues that are between 0 and 1.

The number of distinct vectors found in the dynamical dataset with $\sigma = 20$ bins is equal to 58,451 and the interspike time distribution and eigenspectrum are given in Figure 12. As can be seen in comparing the characteristics of the $\sigma = 20$ dynamical dataset to the $\sigma = 5$ dataset, the increased variance in each channel's firing probability function does a great deal to obfuscate the underlying dynamical structure.

6.2 Synthetic Non-dynamical Data

The next set of diagnostic experiments that were run were to find out how the next-step predictor would perform when given an input dataset whose individual channels were manifestations of independent Poisson processes. The motivation behind this set of experiments was to see whether or not the model output found using the empirical AllenSDK data would resemble the output found using a dataset with known independence between channels. If the outputs were similar then would support the hypothesis that the the AllenSDK dataset might by constructed in such a way that removes any dependence between recording channels.

The dataset used for these experiments is a slight adaptation of the empirical AllenSDK dataset. To create the non-dynamical dataset I made a copy of the empirical spike dataset, and in each channel of data I shuffled the spikes such that the channel in the new dataset contained the same number of spike but located at times sampled randomly from the uniform distribution over the set of possible sampling indices. I chose this strategy as a way of changing the data as little as possible while severing all possible inter-channel dependence.

As with the dynamical datasets, the non-dynamical dataset was used to train next-step predictors using embedding dimensions $M = 10$ and $M = 131$. The performance metrics for the two experiments are given in Table 4 and the input and output plots for the $M = 10$ condition are shown in Figure 13. The $M = 131$ condition yielded outputs that have the same appearance as the outputs for the $M = 10$ condition.

Performing the same dataset-wide analyses that were performed with the dynamical dataset,

I found the number of distinct vectors in this non-dynamical dataset to be equal to 109,495 and show the distribution of channel interspike times and the eigenspectrum of the empirical transition matrix in Figure 14.

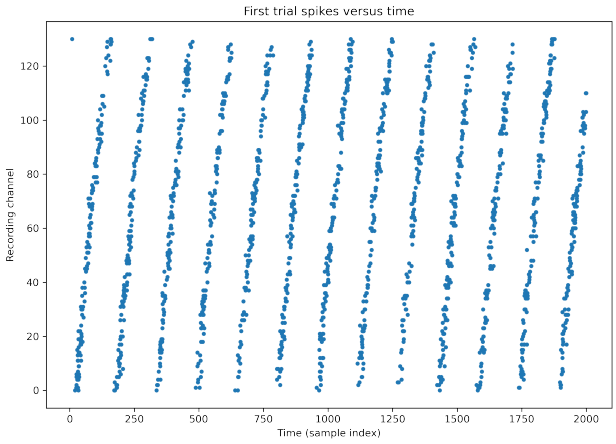
6.3 Comparison to Empirical Data

To compare the results found with the dynamical and non-dynamical synthetic datasets, the empirical dataset was given to a next-step predictor model with $M = 131$ and performed the same dataset-level analyses as described above. The performance metrics of the next-step predictors which took the empirical data as input with $M = 131$ are given in Table 4 and the output plots are given in Figure 15 (and the plots for the $M = 10$ condition were given in Figure 7 in the Results section).

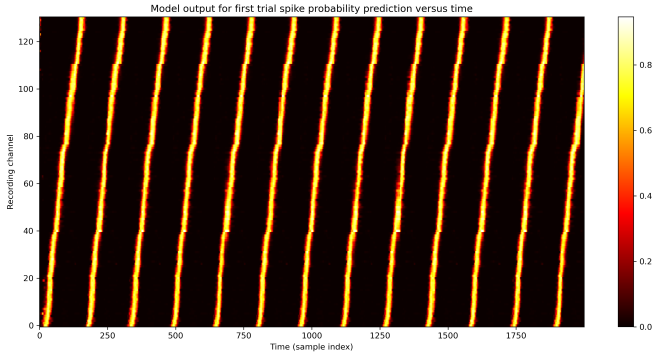
The dataset-level analyses showed us that the empirical data contains 113,883 distinct vectors and has a distribution of average interspike times and eigenspectrum that look very similar to those of the non-dynamical dataset. Figure 16 shows the interspike time distribution and the eigenspectrum of the state transition matrix.

7 Conclusion

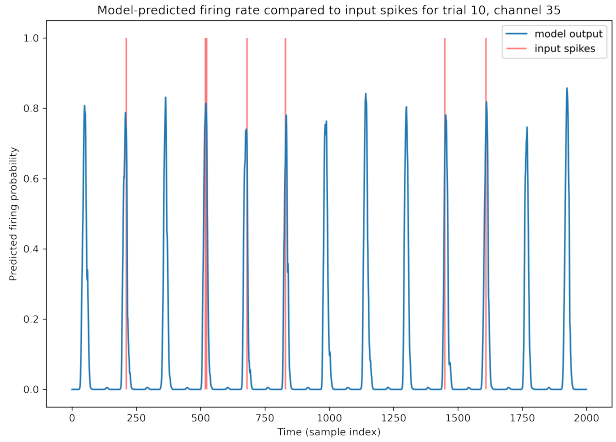
This project is meant to serve as a preliminary exploration into the use of neural networks to model dynamics in experimental neural spike data. The conceptual impetus behind the algorithm design described in this report was the idea that an optimal set of mappings to identify a useful embedding space for the data and perform the dynamical system identification could be learned through gradient descent given a sufficiently well-organized and well-initialized set of models. Through these experiments, however, the predictive model failed to identify any clearly dynamical structure in the empirical spike data. Through my diagnostic experiments and dataset analyses I investigated preliminary explanations for the negative result that was found. My diagnostic experiments and synthetic datasets give us two extreme points of reference that tell us that the predictive model can in fact identify dynamical structure in simple cases and is therefore not broken, and that a dataset with no dynamical structure yields results similar to the ones were found from the empirical dataset. This however does not allow for the definitive conclusion to be made that the empirical dataset does not contain dynamics, but this remains a possibility. In the Pitfalls and Future Work section I describe a number of next steps one could take to further investigate this hypothesis.



(a)

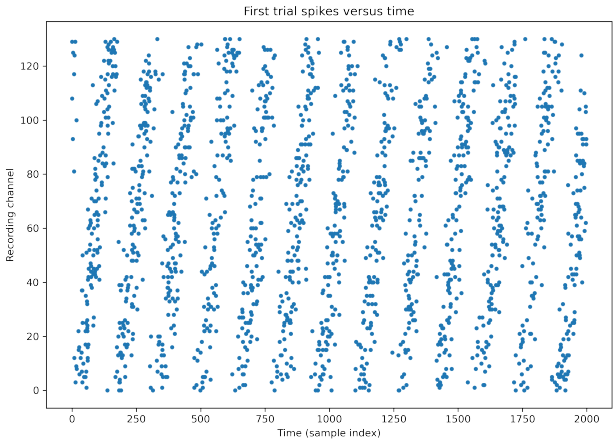


(b)

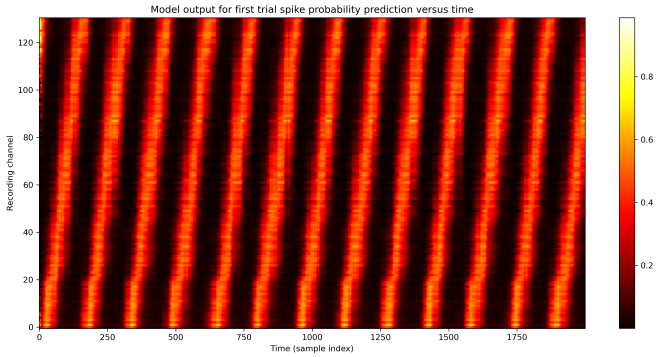


(c)

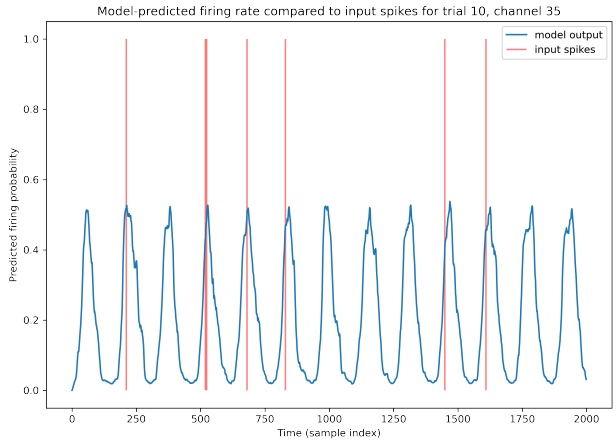
Figure 9: Model input/output for $M = 10$ condition for a dataset with $\sigma = 5$ bins: (a) is a plot of a single trial's spikes across all channels versus time, (b) is the corresponding heatmap of the model's output, and (c) is a single channel's output and reference input spikes.



(a)

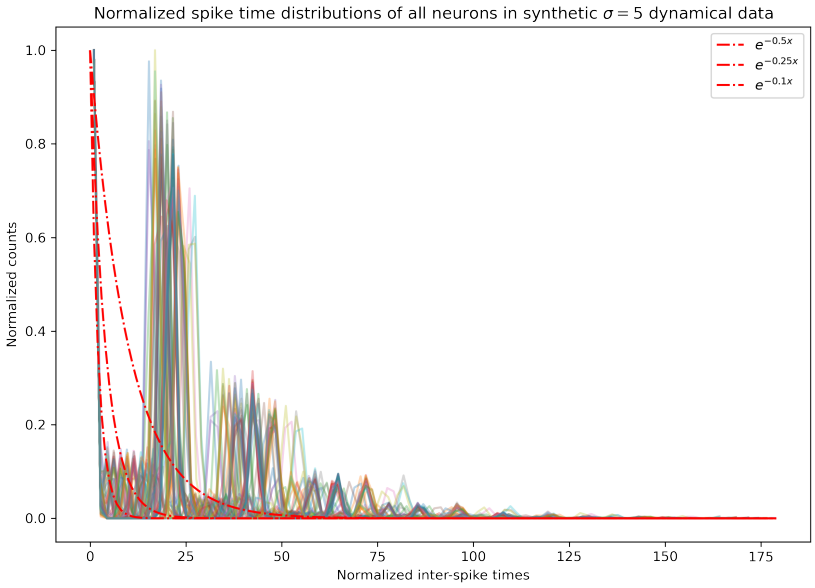


(b)

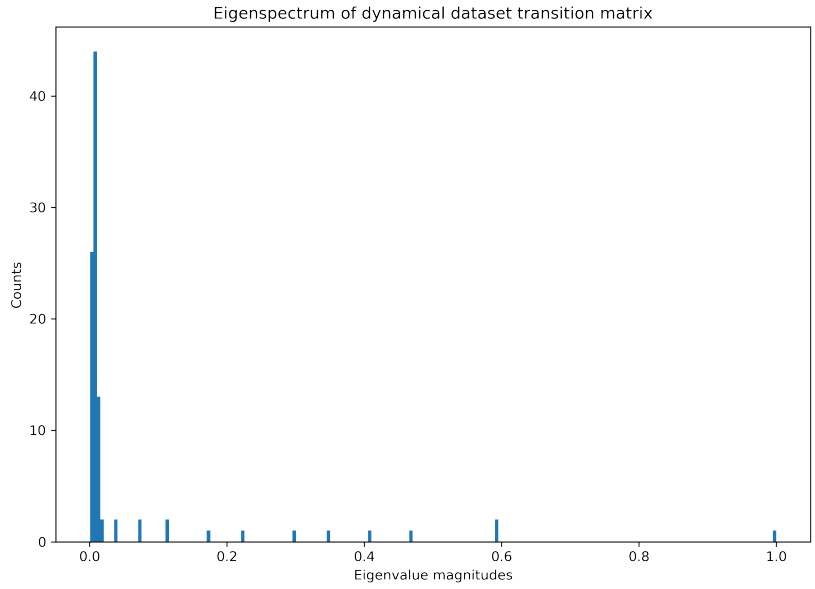


(c)

Figure 10: Model input/output for $M = 10$ condition for a dataset with $\sigma = 20$ bins: (a) is a plot of a single trial's spikes across all channels versus time, (b) is the corresponding heatmap of the model's output, and (b) is a single channel's output and reference input spikes.

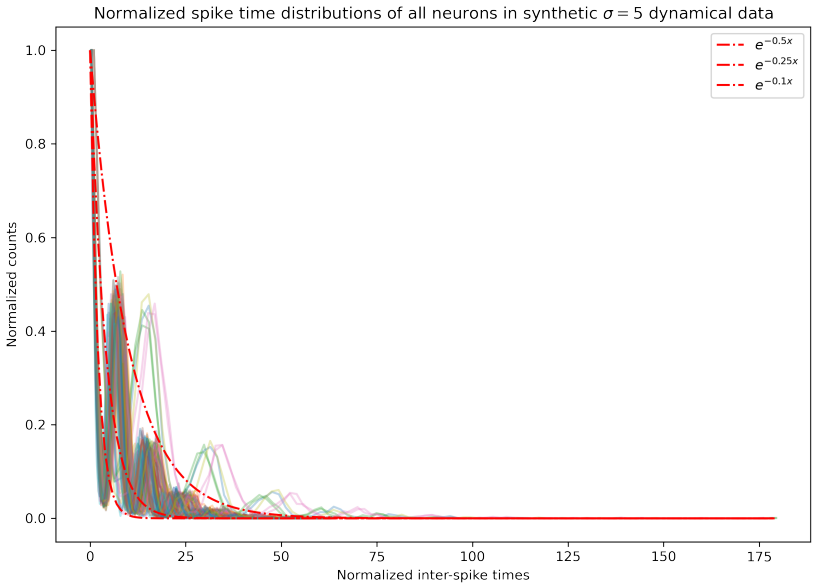


(a)

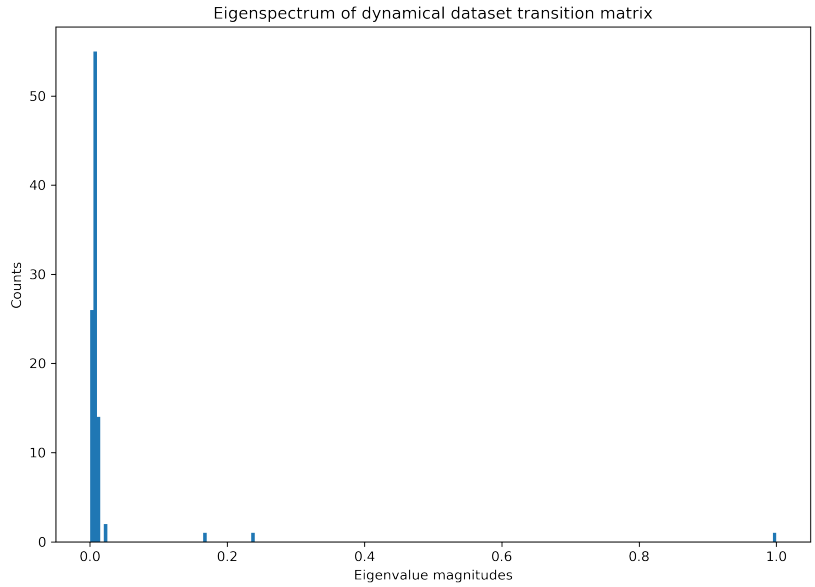


(b)

Figure 11: Data characteristics for dynamical dataset with $\sigma = 5$ bins: (a) is a plot of the different channel/"neuron" average interspike times which are not exponentially distributed, and (b) is the eigenspectrum of the next-state transition matrix associated with the quantized embedding space that arises from using k-means clustering, which shows many eigenvalues in the range (0,1).

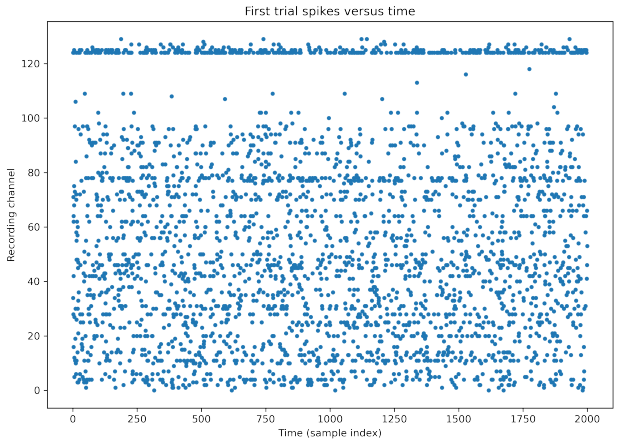


(a)

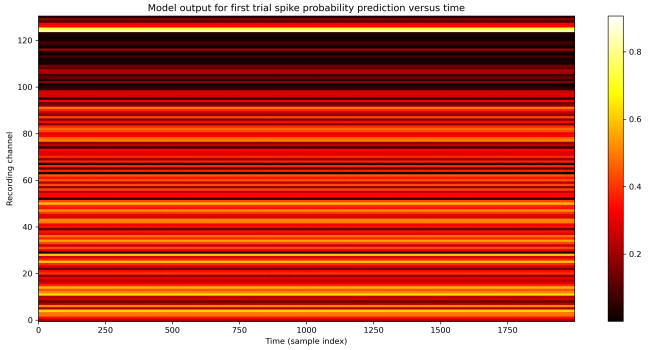


(b)

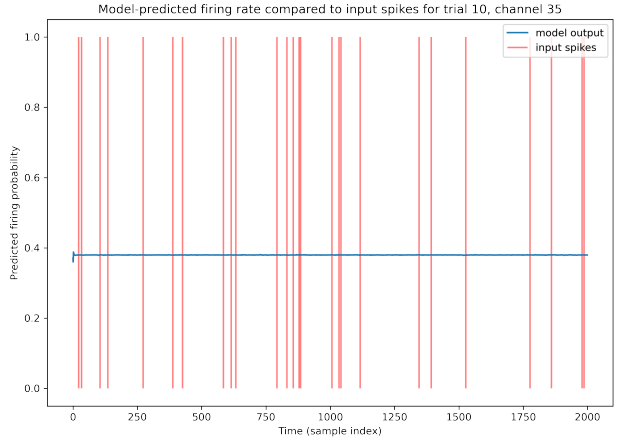
Figure 12: Data characteristics for dynamical dataset with $\sigma = 20$ bins: (a) is a plot of the different channel/"neuron" average interspike times, and (b) is the eigenspectrum of the next-state transition matrix for the embedding space. Both of these plots show the same characteristics as those for the $\sigma = 5$ case but to a notably lesser degree.



(a)

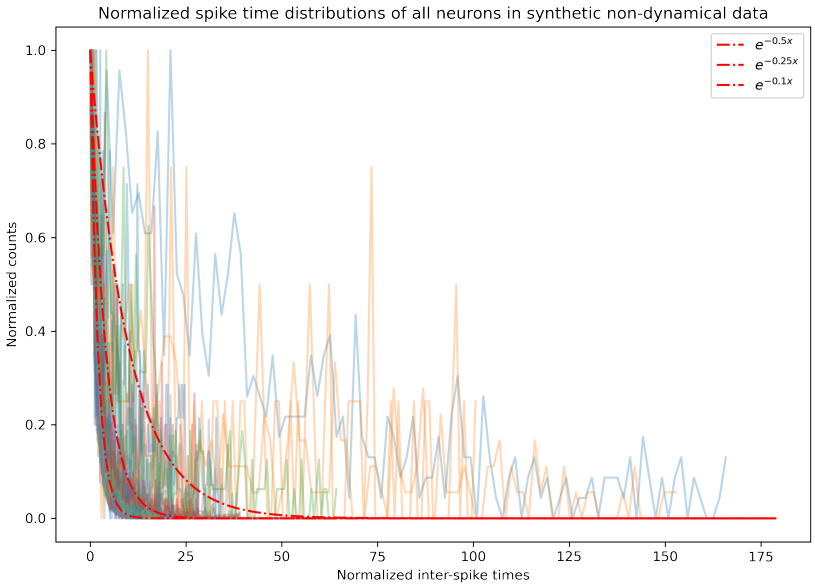


(b)

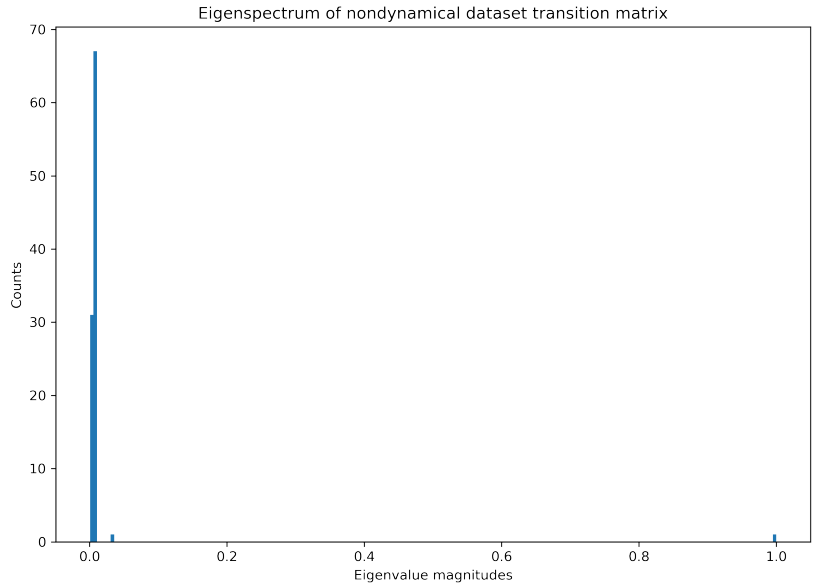


(c)

Figure 13: Model input/output for $M = 10$ condition for a dataset without any inter-channel dependence: (a) is a plot of a single trial's spikes across all channels versus time, (b) is the corresponding heatmap of the model's output, and (c) is a single channel's output and reference input spikes.

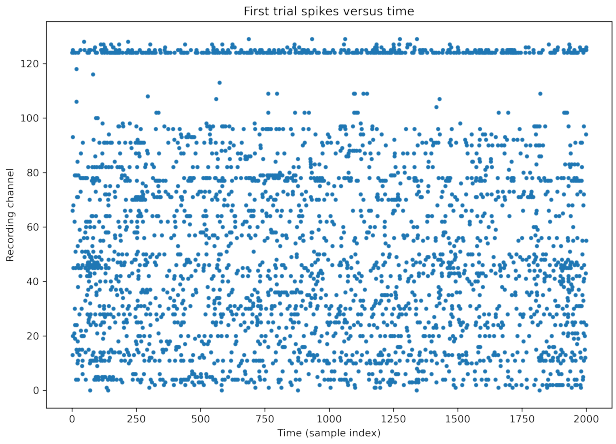


(a)

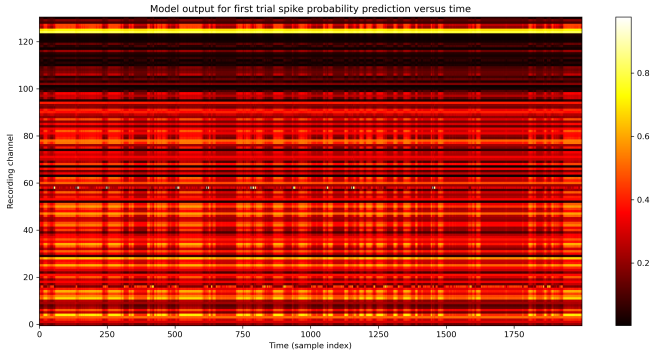


(b)

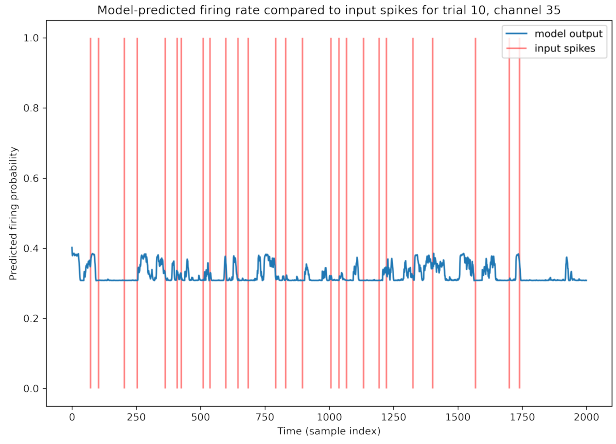
Figure 14: Data characteristics for non-dynamical dataset: (a) is a plot of the different channel/"neuron" average interspike times which appear exponentially distributed, and (b) is the eigenspectrum of the next-state transition matrix associated with the quantized embedding space that arises from using k-means clustering, which shows almost no eigenvalues in the range (0,1).



(a)

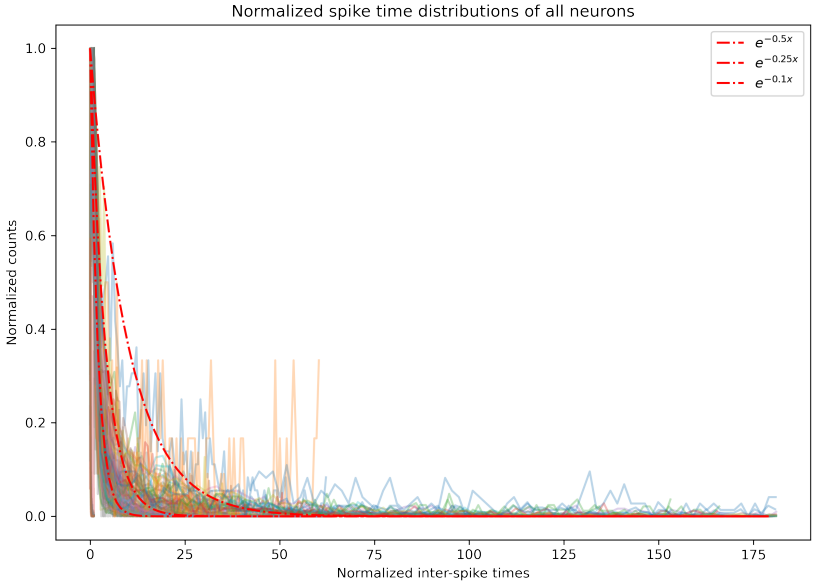


(b)

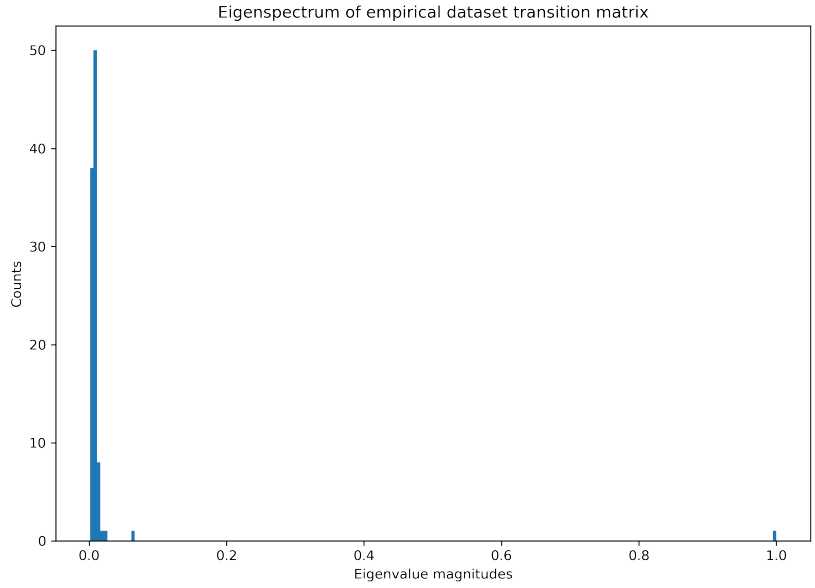


(c)

Figure 15: Model input/output for $M = 131$ condition for the empirical AllenSDK dataset: (a) is a plot of a single trial's spikes across all channels versus time, (b) is the corresponding heatmap of the model's output, and (c) is a single channel's output and reference input spikes.



(a)



(b)

Figure 16: Data characteristics for empirical dataset: (a) is a plot of the different channel/"neuron" average interspike times which also appear exponentially distributed, and (b) is the eigenspectrum of the next-state transition matrix associated with the quantized embedding space that arises from using k-means clustering, which again shows almost no eigenvalues in the range (0,1).

8 Pitfalls and Future Work

My exploration of this problem was limited by time constraints but there are a number of related avenues of inquiry that one could pursue in order to investigate the findings of this project more deeply. In this section I will describe five possible directions of further research that expand on the intuition presented in this report and borrow ideas from elsewhere in the contributing bodies of literature.

8.1 Using interspike time as input data

While my approach focused on the use of binary event indicator data as input to these models, there are alternative approaches to high-dimensional dynamics-learning tasks that instead use input data in the form of a sequence of event times. One particularly similar example to this problem is the Recurrent Marked Temporal Point Process modeling strategy described in [8]. The fundamental difference between this modeling approach and the one described in this report is the formulation of the input sequence as a list of discrete events $\{t_j, y_j\}$ with timing $t_j \in \mathbb{R}^+$ and event marker y_j with $j \in \mathbb{Z}^+$ and $y_j \in [1, N]$ being the unit number, which could be a one-hot vector. Also, rather than inputting time t_j, $\Delta t_j = t_j - t_{j-1}$ could be used. Under this scheme there is more immediate expressive power in the input data to create useful embeddings of event timing and history, and in this paper specifically the modeling approach calls for similar use of an RNN to create feature embeddings as I used in my prediction models.

Related to such an encoding, a temporal map

$$T[t, n] = \min_{t_j < t \,:\, y_j = n} (t - t_j)$$

may encode the same information and may be amenable to modeling of a form motivated by [3].

8.2 Using rate approximations as input data

A difficulty presented by using binary input data is the fact that the sparse nature of the input makes an entropy-based loss metric susceptible to even minor misalignment of the input and output predictions. To make the optimization process more resilient to misaligned input and output there are methods by which one could transform binary input data to be a continuous firing probability approximation of the same form as the models' output. Possible ways to do this range from simplistic methods of convolution with a fixed linear filter of some form, or using pretrained systems such as LFADS [9] that are designed to effectively identify the underlying

rate functions of the system that generated a sequence of binary spikes. The main difficulty with the first approach is that the practice of convolving spike data with a linear filter can be do a great deal to change the nature of the input data by imposing of a parametric form induced by the specific filter used. The drawback to the latter approach is that applying a complex system as a preprocessing step runs the risk of duplicating work that needs to be done by the functions that will create the embeddings from the input rates.

8.3 Different modeling strategy: sequence-to-sequence recurrent autoencoders

There are different modeling approaches that could be used to formulate the prediction task at the center of this problem in a different way. A method that takes a generative approach to the prediction task is motivated by encoder-decoder sequence-to-sequence architectures [10][11]. This architecture of recurrent neural network is marked by its use of distinct encoder and decoder networks which respectively process an input sequence, and generate an output sequence after having been initialized with a summarizing state (referred to as the "context vector") given by the encoder network. This type of method has been applied to problems related to learning dynamical structure in differential equations [12]. In these approaches, the latent data space becomes the space of context vectors that represent full embedded input sequences rather than individual samples. In exploring experiments that were not described in this report I built a framework to use and test these models, but never found performance that was able to generate any sort of useful prediction sequence that would resemble the target sequence. I abandoned this modeling effort in favor of the next-step predictor discussed in this report because that is a simpler modeling approach that places a less onerous prediction task on the RNN.

8.4 Different modeling strategy: neural data transformers

Another modeling paradigm comes from the language modeling literature and the popular use of transformers to create effective feature embeddings of input data. Problems in language modeling and neural spike analysis share a similar data format in the shared use of sparse binary vectors (in the language case that being through the one-hot-encoded word representation, and in the neurophysiological case that being with the binned spike vector) so similar attention-based masking techniques have been used to great effect to learn effective representation of binary spike sequences [13].

8.5 More synthetic datasets with different underlying structure

In order to expand on the preliminary diagnostic experiments I had time to perform in this project, one could generate a number of additional synthetic dynamical datasets in order get a more thorough characterization of the family of functions that are expressible by the next-step prediction model. This would be an important avenue of future work because at the point reached by the end of the experimentation outlined in this report, it was not clear whether the structure I had hoped to find in the data was in fact present or if it was simply too complex to be captured by the model. The intention behind using a multi-layer RNN was to allow for the expression of highly non-linear state-update functions, but I was only able to get a limited view of what structures the model was empirical able to identify. Useful synthetic datasets to run through these prediction experiments would be those in which there are multiple oscillating states that govern the firing probabilities of different subsets of recording channels, those in which there are channels of noise scattered through the dataset, or those in which a given input state maps to multiple embedded states.

9 Related Work

The problem at the center of this project is very multifaceted and draws on work from different areas of computational neuroscience, machine learning, and dynamical systems analysis. From the neuroscience end of the spectrum, there are many areas of research focused on questions of capturing neural population activity in reduced-dimension manifolds of latent factors [9][14][15][16] while others focus on the application of data-reduction techniques more specifically as a way uncovering low-order dynamics in high-dimensional neural systems [17][18]. Some of the approaches such as those described in [14] draw heavily on abstract geometric measurements of neural activity using tools like persistent homology to compare different structures that may be active in the brain during related tasks. Other approaches such as [9] call upon tools from machine learning such as variational inference in order to create generative models that can identify and mimic behavior shown in a set of empirical data.

The machine learning literature is rich with research into many different approaches to this type of problem. Taking a similar route into the problem at [9] are [4][19] and [20] which all use variational-autoencoder-based methods to identify and learn nonlinear dynamics from data in different ways. However, these approaches all present different approaches to the machine learning problems posed by these models. For example, [9] and [4] both focus on the discovery of latent factors that explain the dynamics present in a set of input data, while [19] tackles the problem of not being able to impute high-dimensional time series data in a computationally efficient way, and [20] presents a more efficient way of performing the actual inference step involved in learning from sequence data.

Straying from the generative approaches that rely on variational inference there is another corner of the literature that is concerned with ways of using RNNs as a tool for representing dynamical systems [12][21][22][23][24]. This research draws on fundamental practices from the dynamical systems such as the use of delay-embeddings and fixed-point analysis and bridges the gap between classical methods and how they can be implemented using modern machine learning tools.

Finally, as was mentioned in the Future Work section, there are many methods of modeling neural activity as point processes [3]. These approaches came up in Future Work because they, despite being formulated decades before many of the machine learning approaches given above, are amenable to combined use with neural network methods (such as in [8]). The idea behind the point process modeling frameworks is to articulate the behavior of neurons or groups of neurons using "conditional intensity functions" that account for the impact of a neuron's history on its probability of firing at any point in time. These modeling approaches can be powerful

because they provide frameworks for relating a neuron's probability of firing to its own history, the broader activity of an ensemble of neurons, and on external stimuli or behavior.

10 Appendix A: Description of Synthetic Dynamical Dataset

Each channel in the synthetic data was defined with a firing probability function equal to a Gaussian density function that was parameterized by a mean and variance given by ϕ_i and σ^2 respectively, and varied with the state of the oscillator ϕ. Each channel's value of ϕ_i was drawn from the uniform distribution over the interval $[0, \tau)$, and the list of ϕ_i was then sorted such that $\phi_i < \phi_{i+1} < \phi_{i+2} < ....$ This ordering of the peaks of the different channels' firing probability functions was done to allow for especially powerful visualization of the structure of the dataset when put in use with the models described in this report. The models would not be effected by the ordering of the channels in the data so this was a purely cosmetic specification. The Poisson rate of firing for channel i at a state ϕ is given by:

$$\lambda_i(\phi; \phi_i, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\phi-\phi_i}{\sigma}\right)^2}$$

With the oscillator defined to have a period of τ, the states were discretized along the oscillator to align with the integer bins $c \in \{0, 1, 2, ..., \tau - 1\}$. With the average interspike time τ calculated in milliseconds, this discretization makes sure that the synthetic data will be "sampled" at the same resolution as the empirical data.

Across the i synthetic recording channels, each has a probability of firing at all points c given by $\lambda_i(\phi_c)$ evaluated at that point. $\phi_c = c \cdot \frac{2\pi}{\tau}$ gives the angle corresponding to the c^{th} state on the oscillator. With this I calculate the firing probability of channel i in state c to be

$$p_{c,i} = \int_{\phi_c}^{\phi_{c+1}} \lambda_i(\phi; \phi_i, \sigma^2) d\phi = \Phi\left(\frac{\phi_{c+1} - \phi_i}{\sigma}\right) - \Phi\left(\frac{\phi_c - \phi_i}{\sigma}\right)$$

... for $\Phi(\cdot)$ the standard normal cumulative density function.

Using $p_{c,i}$ I defined the joint firing probability at state c to be $\overline{p}_c$ with the i^{th} element of $\overline{p}_c = p_{c,i}$. This gives us firing probabilities over the 131 channels at all sampling times $t \in \{0, ..., 2000\}$ by iterating around the oscillator, finding the corresponding state for each time as $c \equiv t \mod \tau$. To finally construct the synthetic dataset, for each time t in a given trial I sampled the vector of firings across all 131 channels as the draw

$$\overline{s}[t] \sim Bernoulli\left(\overline{p}_{t \bmod \tau}\right)$$

11 Appendix B: Model Evalulation Function

```
def calculate_avg_pred_prob(model, data, tau):
    '''
    Function to take in a trained model, data tensor, and time-window radius
        tau, and calculate the average predicted probability the model gives
        under the input conditions of spike and no spike.
    This is calculated by going through all data and preparing our predictions
        over all samples, and then separating out the positive (spike) samples
        (i.e. individual entries in the spike matrices) and negative (no spike)
        samples, and calculating the BCE over these two populations.
    '''
    num_trials = data.shape[0]
    num_units = data.shape[1]
    num_recon_samples = data.shape[2] - 2*tau
    reconstructed_output = torch.from_numpy((np.zeros((num_trials, num_units,
        num_recon_samples)))).float()

    for trial in range(num_trials):
        for ind in range(num_recon_samples):
            input_sample = data[trial,:,ind:ind+2*tau+1]
            single_sample_batch = torch.unsqueeze(input_sample, 0)

            with torch.no_grad():
                sample_recon = model.forward(single_sample_batch)

            reconstructed_output[trial,:,ind] = sample_recon

    if tau > 0:
        trimmed_input = np.delete(data, np.s_[:tau],axis=2)
        trimmed_input = np.delete(trimmed_input, np.s_[-tau:],axis=2)
    else:
        trimmed_input = data

    trial_zero_coords = []
    trial_one_coords = []
```

```
for trial in range(num_trials):
    us1, cs1 = np.argwhere(trimmed_input[trial])
    trial_one_coords.append([us1.tolist(), cs1.tolist()])

    us0, cs0 = np.argwhere(trimmed_input[trial] == 0)
    trial_zero_coords.append([us0.tolist(), cs0.tolist()])

bce_wlogit = torch.nn.BCEWithLogitsLoss(reduction='sum')

total_bce_nospike = 0
total_bce_spike = 0
nospike_samples = 0
spike_samples = 0

for trial in range(num_trials):
    total_bce_nospike +=
        bce_wlogit(reconstructed_output[trial][trial_zero_coords[trial]],
        trimmed_input[trial][trial_zero_coords[trial]])
    total_bce_spike +=
        bce_wlogit(reconstructed_output[trial][trial_one_coords[trial]],
        trimmed_input[trial][trial_one_coords[trial]])
    nospike_samples += len(trial_zero_coords[trial][0])
    spike_samples += len(trial_one_coords[trial][0])

avg_bce_nospike = total_bce_nospike/nospike_samples
avg_bce_spike = total_bce_spike/spike_samples

nospike_pred_prob = 1 - np.e**(-1*avg_bce_nospike)
spike_pred_prob = np.e**(-1*avg_bce_spike)

return nospike_pred_prob.item(), spike_pred_prob.item()
```

12 Appendix C: Competencies and Timeline

12.1 Intersection with the four DSI target competencies

12.1.1 *Ability to apply competency in Probability, Statistics and Machine Learning, Data and Computational Science to real-world data science problems*

The different aspects of this project presented problems that align with many different disciplines ranging from dynamical systems analysis to machine learning and statistics, as well as to data science more generally. Many problems relating to algorithm design and model assessment borrowed heavily from probability and statistics, while many practical problems arose from the size and nature of the data itself. Brain activity data is most typically high-dimensional in order to record information from concurrent activity present in a vast number of neurons, and the recording must be sampled at a high frequency in order to achieve a data resolution commensurate with the fine time scale on which neurons operate.

This large-scale data format presents many real-world data science and statistics problems regarding the best ways of preprocessing and using the data. The majority of time spent working on the actual fitting algorithms dealt heavily with problems in statistics and machine learning, as the effort there was to build models that would be best suited to identify the dynamical structure in the data by means of gradient-based optimization of neural network models.

12.1.2 *Formulate appropriate questions, perform analyses, and draw appropriate conclusions from large datasets*

The workflow throughout this project included a great deal of experimentation that informed subsequent decisions and influenced the path of the inquiry. The ultimate path and set of objectives changed from what was originally outlined in the research proposal, and the changes in direction were the result of conclusions that were drawn from large datasets of intermediate results. Specifically, the most obvious stage of the project in which conclusions were drawn from large datasets was during the diagnostic stage in which the original dataset and many additional ones were created and analyzed under varying conditions. This was an important part of the exploration of these results and allowed us to reach the main conclusion present in this report.

12.1.3 *Connect limitations of data and data analysis to bias in results*

Limitations and quality of data was a present factor in the analysis of my results. The fundamental aspect of this analysis was reconciling the experimental results with my expectations based on my

intuition about what structure should be present in a biophysical dataset. As is mentioned in the documentation for the Allen Institute Brain Atlas [7], quality of data is documented clearly for the Neuropixels dataset, and the authors strongly emphasize their quality control efforts. Practically, I was able to take advantage of this documentation of data limitations in such preprocessing steps as filtering recording channels that had a low signal-to-noise-ratio. More broadly, my main conclusion from my experimental results is a statement about ways in which possible aspects of the data I used may introduced bias into my results.

12.1.4 *Demonstrate understanding of the societal impacts of data and its analysis developed through case studies of relevant topics*

An important impact of the tool produced by this project would be to lower the computational expense to run large scale neurophysiological simulations with a minimal sacrifice of biophysical realism (in contrast, say, to the highly simplified individual neurons in the "whole brain" simulations of [25]). Further, insight into the underlying dynamical systems that generated the input data is of central theoretical interest on its own.

12.2 Time Estimates for Stages of Project

The process of becoming familiar with and using the AllenSDK to create the main dataset that was used throughout this project took approximately a week. The process of building the preliminary infrastructure for my first suite of models along with writing all of the data management (particularly that which was involved in getting the code working on the Brown HPC cluster) and performance testing code took approximately two weeks. The next two weeks were spent experimenting with different designs of recurrent neural network models that would be used as possible alternative data embedding strategies and as alternative predictive models. These models also required their own testing framework and data management code. Over the following week and a half the predictive models that were actually used in the parts of the project discussed in this report were written, along with their testing and evaluation framework. The final week and a half were spent performing the diagnostic analyses described in the latter portion of this report. These involved the construction of different synthetic datasets and the re-execution of a number of experiments using these datasets, as well as a number of analyses to uncover dataset-level characteristics that would provide evidence of the dynamical structure present in each one.

13 Appendix D: System Description

The code used to execute the experiments and analyses described in this report can be found in the GitHub repository: `https://github.com/crohlicek/brain_modeling`. The README file contains descriptions of each file's contents.

References

[1] A. S. Benjamin, H. L. Fernandes, T. Tomlinson, P. Ramkumar, C. VerSteeg, R. H. Chowdhury, L. E. Miller, and K. P. Kording, "Modern machine learning as a benchmark for fitting neural responses," *Frontiers in Computational Neuroscience*, vol. 12, p. 56, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fncom.2018.00056

[2] D. Zhou and X.-X. Wei, "Learning identifiable and interpretable latent models of high-dimensional neural activity using pi-vae," *arXiv preprint arXiv:2011.04798*, 2020.

[3] W. Truccolo, U. T. Eden, M. R. Fellows, J. P. Donoghue, and E. N. Brown, "A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects," *Journal of neurophysiology*, vol. 93, no. 2, pp. 1074–1089, 2005.

[4] Y. Zhao and I. M. Park, "Variational online learning of neural dynamics," *Frontiers in computational neuroscience*, vol. 14, 2020.

[5] A. Iqbal, P. Dong, C. M. Kim, and H. Jang, "Decoding neural responses in mouse visual cortex through a deep neural network," *2019 International Joint Conference on Neural Networks (IJCNN)*, Jul 2019. [Online]. Available: http://dx.doi.org/10.1109/IJCNN.2019.8852121

[6] S. M. Sunkin, L. Ng, C. Lau, T. Dolbeare, T. L. Gilbert, C. L. Thompson, M. Hawrylycz, and C. Dang, "Allen Brain Atlas: an integrated spatio-temporal portal for exploring the central nervous system," *Nucleic Acids Research*, vol. 41, no. D1, pp. D996–D1008, Nov. 2012, _eprint: https://academic.oup.com/nar/article-pdf/41/D1/D996/3594088/gks1042.pdf. [Online]. Available: https://doi.org/10.1093/nar/gks1042

[7] J. H. Siegle, X. Jia, S. Durand, S. Gale, C. Bennett, N. Graddis, G. Heller, T. K. Ramirez, H. Choi, J. A. Luviano *et al.*, "A survey of spiking activity reveals a functional hierarchy of mouse corticothalamic visual areas," *Biorxiv*, p. 805010, 2019.

[8] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song, "Recurrent marked temporal point processes: Embedding event history to vector," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1555–1564.

[9] C. Pandarinath, D. J. O'Shea, J. Collins, R. Jozefowicz, S. D. Stavisky, J. C. Kao, E. M. Trautmann, M. T. Kaufman, S. I. Ryu, L. R. Hochberg *et al.*, "Inferring single-trial neural population dynamics using sequential auto-encoders," *Nature methods*, vol. 15, no. 10, pp. 805–815, 2018.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *10.4, 5.5*. MIT Press, 2016.

[11] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, 2014.

[12] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *arXiv preprint arXiv:1806.07366*, 2018.

[13] J. Ye and C. Pandarinath, "Representation learning for neural population activity with neural data transformers," *bioRxiv*, 2021.

[14] R. Chaudhuri, B. Gerçek, B. Pandey, A. Peyrache, and I. Fiete, "The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep," *Nature neuroscience*, vol. 22, no. 9, pp. 1512–1520, 2019.

[15] J. A. Gallego, M. G. Perich, L. E. Miller, and S. A. Solla, "Neural manifolds for the control of movement," *Neuron*, vol. 94, no. 5, pp. 978–984, 2017.

[16] A. D. Degenhart, W. E. Bishop, E. R. Oby, E. C. Tyler-Kabara, S. M. Chase, A. P. Batista, and M. Y. Byron, "Stabilization of a brain–computer interface via the alignment of low-dimensional spaces of neural activity," *Nature biomedical engineering*, vol. 4, no. 7, pp. 672–685, 2020.

[17] M. Advani, S. Lahiri, and S. Ganguli, "Statistical mechanics of complex neural systems and high dimensional data," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2013, no. 03, p. P03014, 2013.

[18] A. H. Williams, T. H. Kim, F. Wang, S. Vyas, S. I. Ryu, K. V. Shenoy, M. Schnitzer, T. G. Kolda, and S. Ganguli, "Unsupervised discovery of demixed, low-dimensional neural dynamics across multiple timescales through tensor component analysis," *Neuron*, vol. 98, no. 6, pp. 1099–1115, 2018.

[19] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther, "A disentangled recognition and nonlinear dynamics model for unsupervised learning," *arXiv preprint arXiv:1710.05741*, 2017.

[20] J.-S. Ha, Y.-J. Park, H.-J. Chae, S.-S. Park, and H.-L. Choi, "Adaptive path-integral autoencoders: Representation learning and planning for dynamical systems," *Advances in Neural Information Processing Systems*, vol. 31, pp. 8927–8938, 2018.

[21] Z. Y. Wan, P. Vlachas, P. Koumoutsakos, and T. Sapsis, "Data-assisted reduced-order modeling of extreme events in complex dynamical systems," *PloS one*, vol. 13, no. 5, p. e0197704, 2018.

[22] E. Pollock and M. Jazayeri, "Engineering recurrent neural networks from task-relevant manifolds and dynamics," *PLoS computational biology*, vol. 16, no. 8, p. e1008128, 2020.

[23] W. Gilpin, "Deep reconstruction of strange attractors from time series," 2020.

[24] D. Sussillo and O. Barak, "Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks," *Neural computation*, vol. 25, no. 3, pp. 626–649, 2013.

[25] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.