Inferring network dynamics from simulated extracellular electrophysiological data

Aran Nayebi anayebi@stanford.edu
Dylan Cable dcable@stanford.edu
Richard Wedeen rwedeen@stanford.edu

Abstract

We compare the performance of two different types of recurrent neural networks (RNNs) for the task of inferring the extracellular voltage of each neuron at a given timestep, when provided with partial information about the simulated extracellular electrophysiological data of a population of 2022 neurons over 6 seconds. In particular, we focus on RNNs that have a sophisticated gating mechanism, namely, the Long Short-Term Memory (LSTM) network and the recently introduced Gated Recurrent Unit (GRU). When trained on 70% of the timesteps (and validated on the remaining 30%), our results indicate that the predicted voltages for each neuron of the GRU network were significantly more plausible than the predictions of the LSTM, when compared to the ground truth voltages. This provides encouraging evidence that RNNs (specifically the GRU) can be considered a promising approach to inferring network dynamics in large populations of neurons. We also discuss some preliminary results towards the goal of inferring synaptic weights from this electrophysiological data as well.

1 Problem Statement and Approach

1.1 Motivation

With the current state of the art, we can expect to get calcium imaging and extracellular electrophysiological data from on the order of hundreds of neurons. One of the more common approaches to inferring the network dynamics of a population of neurons is maximum entropy modeling. Rather than making specific assumptions about the underlying dynamics, we take a relatively small set of measurements on the system as given, and build a model for the distribution over system states that is consistent with these experimental results but otherwise has as little structure as possible. This results in a Boltzmann-like distribution which defines an energy landscape over the states of the system. It is important to note that this energy function has no free parameters, but is completely determined by the experimental measurements. For example, Schneidman et al. [6] use a maximum entropy model to analyze small windows of time where each neuron in a network either generates an action potential (spike) or remains silent, based on multi-electrode recordings from the salamander retina. More specifically, their approach is as follows: If we examine a window of fixed time resolution $\Delta\tau$, then the extracellular responses can be interpreted as binary – either the neuron spikes ('1') or it does not ('0'). Hence, given a population of N neurons, they form an N-dimensional binary vector to describe the instantaneous state of the network. To describe the network as a whole, one would need to know the underlying probability distribution for the 2^N binary vectors, which corresponds to patterns of spiking and silence in the population. Ideally, one wants to find a distribution that is consistent *only* with the measured correlations, and does not implicitly assume the existence of unmeasured higher-order interactions. Since entropy measures the lack of interaction among different variables, then one wants to find the distribution with maximum entropy that is consistent with the measured properties of the individual neurons and pairs of neurons. In fact, they find that the maximum entropy distribution consistent with the mean probability of spiking in each neuron and the correlations among spikes in pairs of neurons is exactly an Ising spin glass.

However, Schneidman et al.'s [6] maximum entropy approach deals with a small population of neurons, around $N = 40$, and to our knowledge the maximum entropy approach has not been used for a population of neurons larger than 200. Moreover, they use the maximum entropy model to infer spiking, which is a discrete problem ('1' or '0' is assigned to whether a neuron spiked or did not spike at a given time). An interesting question to ask is, how well does the maximum entropy approach scale for a much larger population of neurons (2022 in our case), and for a *continuous* problem such as inferring extracellular voltages of each neuron at a given timestep? Vernaza and Bagnell [8] offer preliminary evidence that demonstrates the intractability of maximum entropy modelling in continuous valued spaces of high dimensionality (though in the special case that the features possess a certain kind of low dimensional structure, then they show that maximum entropy modelling is a viable approach).

1.2 Data and Approach

As a result, we wanted to assess the viability of *alternative* approaches to inferring the extracellular voltages of each neuron at a given timestep for a large population of neurons. Costas Anastassiou and his group at the Allen Institute have provided us with data from a 2022 cell simulation (over a duration of 6 seconds), consisting of somatic Vm and [Ca], simulated extracellular depth recordings, and the connectivity matrix of the neurons. We were also provided with the ground truth for voltage outside each neuron.

Using this data (in particular, the extracellular voltages of all 2022 neurons at each timestep, where each timestep corresponds to 1/10 ms for a total of 59999 timesteps ($\sim$ 6 seconds)), we wanted to assess the viability of recurrent neural networks (RNNs) in inferring the extracellular voltage of each neuron at a given timestep. RNNs are an extension of a conventional feedforward neural network, which is able to handle a variable length input, by having a recurrent hidden state whose activation at each time is dependent on that of the previous time. More formally, given a sequence $x = (x_1, \ldots, x_T)$, the RNN updates its recurrent hidden state h_t by

$$h_t = \phi(h_{t-1}, x_t), \tag{1}$$

if $t \neq 0$ (otherwise, at $t = 0$, $h_0 = 0$), where ϕ is a nonlinear function (usually the composition of either a logistic sigmoid function or hyperbolic tangent with an affine transformation). A generative RNN outputs a probability distribution over the next element of the sequence, given its current state h_t, and this generative model can capture a distribution over sequences of variable length.

Thus, if we view the extracellular voltages of all the neurons at a given timestep t as a 2022 dimensional vector x_t, then we can form a seed sequence $x = (x_1, \ldots, x_T)$ of these vectors as input to our RNN and have it output its predicted voltages $\hat{x} = (\hat{x}_2, \ldots, \hat{x}_{T+1})$, where $\hat{x}_i$ is the *predicted* voltage of all the neurons at timestep i, given the previous ground truth voltages $x_{i-1}, \ldots, x_1$ of all the neurons at timesteps $i-1, \ldots, 1$. Given that we have the ground truth voltages of all the neurons at each timestep, we use L2 loss to evaluate performance, given by,

$$\ell(\theta) \equiv \frac{1}{T} \sum_t (x_t - \hat{x}_t)^2, \tag{2}$$

where T is the number of timesteps.

In essence, this is a sequence modeling task (which is a common use of RNNs in natural language processing) in which we are trying to estimate the next sequence conditioned on the previous sequences. By training the RNN on 70% of the 59999 x_i vectors and validating on the remaining 30% (a common validation split), we can learn useful priors to aid in future generation tasks.

However, the question that arises is what generative RNN model to use. Unfortunately, it has been observed by Bengio et al. [1] that it is difficult to train RNNs to capture long-term dependencies because the gradients tend to either vanish (the common problem of *vanishing gradients*) or explode (rarely, but with severe effects). This makes it difficult to train a traditional RNN using a gradient-based optimization method, primarily because the effect of long-term dependencies is hidden by the effect of short-term dependencies [4].

One approach to solving this problem of vanishing or exploding gradients is to design a more sophisticated activation function by using gating units. The earliest result in this direction to model long-term dependencies is the Long Short-Term Memory (LSTM) network, introduced by Hochreiter and Schmidhuber [5] in 1997. Recently, Gated Recurrent Units (GRU), introduced by Cho et al. [2], have been used to effectively model long-term dependencies in a variety of generic sequence modeling tasks.

Since the jury is still out as to whether the LSTM or GRU is better at sequence generation tasks [7, Slide 24], we wanted to compare the performance of these two different types of RNNs for the task of inferring the extracellular voltage of each neuron at a given timestep, when both models were trained on 70% of the timesteps. Moreover, given that the initial goal of our project was to infer connectivity from the extracellular electrophysiological data provided by Costas (before we found that our models were better suited for the problem of inferring voltage), as an appendix (in §6), we include some of our preliminary results in this direction as well.

2 Results

As mentioned previously, we did a train-dev split of 70% of the 59999 timesteps (the x_i vectors mentioned in §1.2), which formed our training set, and the remaining 30% comprised our dev set. However, prior to this split, we first made the mean of the extracellular voltages of all 2022 neurons across all 59999 timesteps to be 0 and its variance to be 1. This is a common practice in order to ensure that the training and dev losses (in our case, this is the L2 loss defined in equation (1)) are meaningful. Moreover, due to memory constraints, we divided up the 59999 timesteps into 1000 timestep intervals, where each 1000 timestep interval corresponded to a single training example for our models. Thus, unless noted otherwise, every plot assumes that our training examples each consisted of 1000 timestep intervals. We initially trained both the GRU and LSTM models for 105 epochs, and compared their performance, before training for an additional 150 epochs. As can be seen in Figure 1, although the training loss for the LSTM was slightly lower than the training loss for the GRU, their dev losses after 255 epochs were essentially the same. However, their generated ouputs were not essentially the same. For the purpose of illustration, we picked two neurons, namely, neuron 300 and neuron 2000, and plotted their predicted voltages over the first 1000 timesteps by both the LSTM and GRU compared to the ground truth.

As can be seen in Figure 2 and Figure 3, after 105 epochs of training, the predicted voltages for neurons 300 and 2000 by the LSTM and the GRU are rather noisy when compared to the ground truth.

After training both models for an additional 150 epochs, a pattern appears to emerge. As can be seen in Figure 4 and Figure 5, for both neurons, the GRU appears to capture more structure than the LSTM network. For both neuron 300 and neuron 2000, despite the noisiness, the GRU's generated ouputs appear to demonstrate that it can predict *when* the neuron emits an action potential and the duration of the action potential (and when the neuron is not firing, its output voltages also show that it is not firing). On the other hand, the LSTM's generated ouputs appear to have added more sporadic noise, especially when the neuron was resting (not firing). Thus, the GRU appears to capture the general *structure* of each neuron's voltage over time.

Thus, given the GRU's advantageous performance, we wanted to focus more on the GRU and train it longer to see if its generated ouputs would improve over increased training time. We also experimented with dividing up the 59999 timesteps into 3000 timestep intervals (instead of 1000 timestep intervals as we initially had), where each 3000 timestep interval corresponded to a single training example, in order to see if it improved the GRU's performance. As can be seen in Figure 6, however, this change does not improve performance, as the GRU's losses on the dev set are higher with 3000 timesteps per training example, than with 1000 timesteps per training example. Moreover, as can be seen in Figure 7 and Figure 8, for both neurons 300 and 2000, more noise is added in the generated ouputs of the GRU when we train it on 3000 timesteps per training example, instead of 1000 timesteps per training example.

Therefore, it appears that 1000 timestep divisions for the GRU is to be preferred. We also wanted to compare the performance of the LSTM model when it is trained on 3000 timesteps per training examples as opposed to 1000 timesteps per training example. However, the LSTM is less memory efficient to implement (suggesting yet another advantage of the GRU), and as a result, required too much memory than the Stanford Rye clusters could allocate if we trained with 3000 timesteps

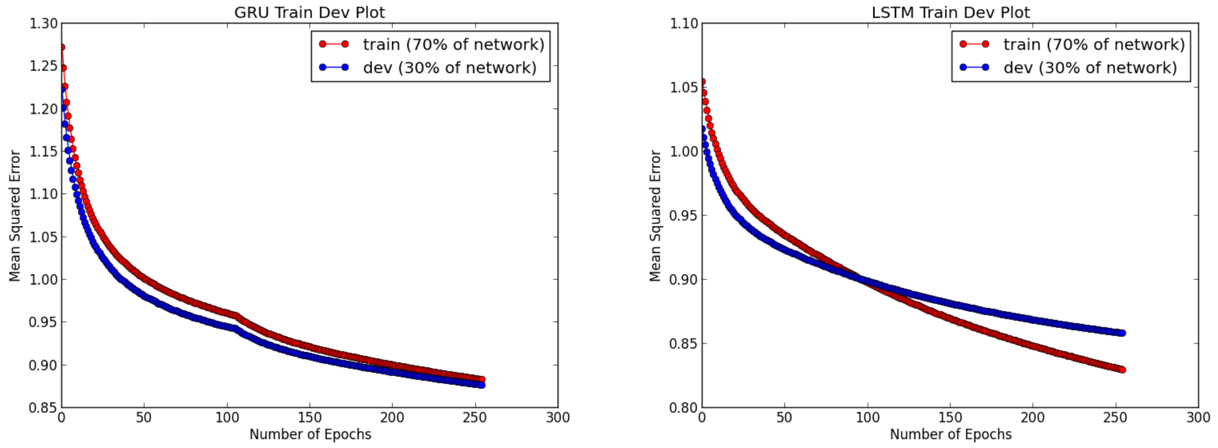


Figure 1: **From left to right:** GRU train-dev plot and LSTM train-dev plot after 255 epochs.

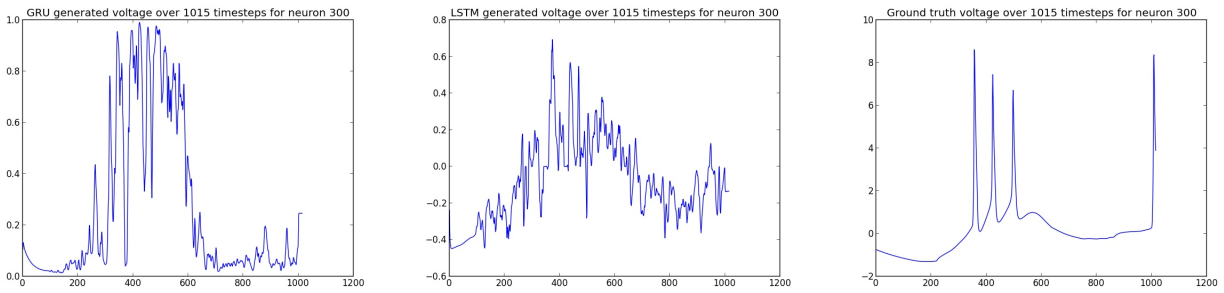


Figure 2: **From left to right:** plot of the predicted voltages of the GRU, LSTM (after both were trained for 105 epochs), and the ground truth voltage of neuron 300 over the first 1015 timesteps.

per training example. Thus, we trained the GRU for a total of 1000 epochs (with the usual 1000 timesteps per training example), and its training and dev loss curves are plotted in Figure 9.

After 605 epochs of training, the GRU's generated ouputs appear to become less noisy (as can be seen in Figure 10), however they still definitely differ from the ground truth generated ouputs. By 1000 epochs of training, the GRU's generated ouputs (plotted for neurons 300 and 2000 in Figure 11) appear much less noisy when compared to the ground truth. Especially for neuron 2000, the GRU's generated output is quite similar to the ground truth voltage for this neuron. For neuron 300, on the other hand, the GRU's output is much less noisy than before and is better modeling the structure of the voltage over time (though it is still different than the ground truth).

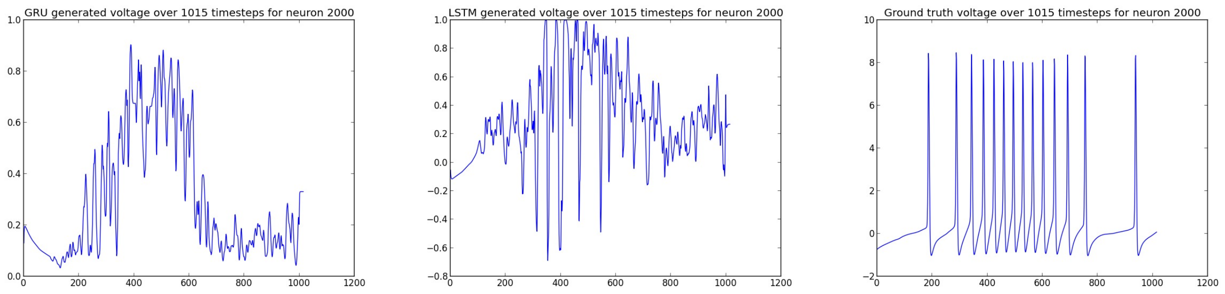


Figure 3: **From left to right:** plot of the predicted voltages of the GRU, LSTM (after both were trained for 105 epochs), and the ground truth voltage of neuron 2000 over the first 1015 timesteps.

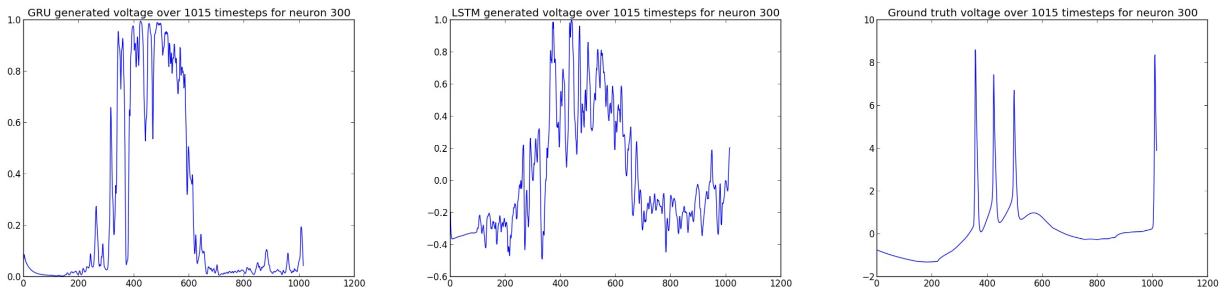


Figure 4: **From left to right:** plot of the predicted voltages of the GRU, LSTM (after both were trained for 255 epochs), and the ground truth voltage of neuron 300 over the first 1015 timesteps.

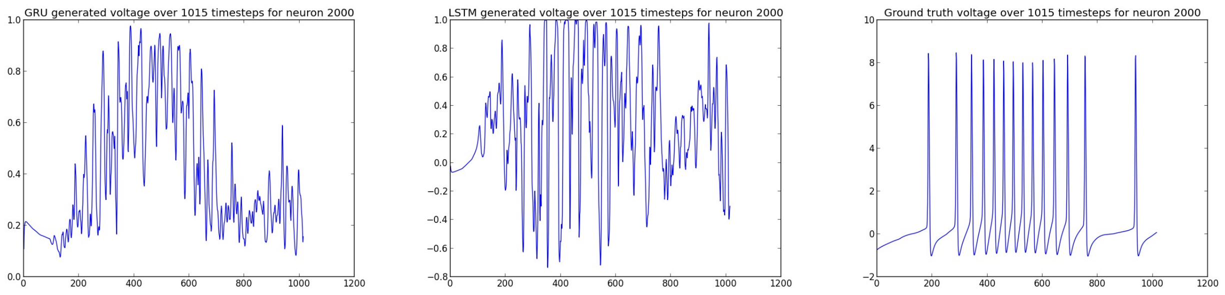
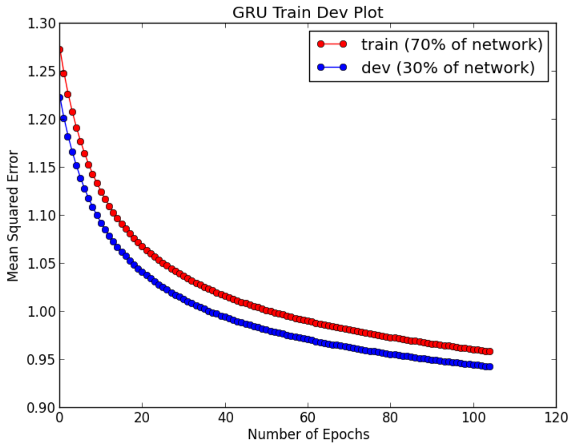


Figure 5: **From left to right:** plot of the predicted voltages of the GRU, LSTM (after both were trained for 255 epochs), and the ground truth voltage of neuron 2000 over the first 1015 timesteps.



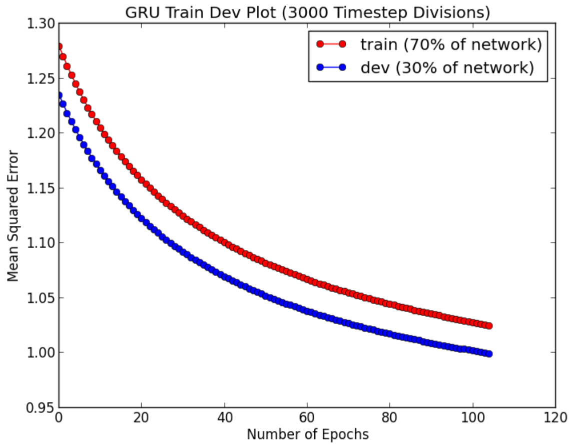


Figure 6: **From left to right:** GRU train-dev plot (where each training example consisted of 1000 timesteps) and GRU train-dev plot (where each training example consisted of 3000 timesteps) after 105 epochs.

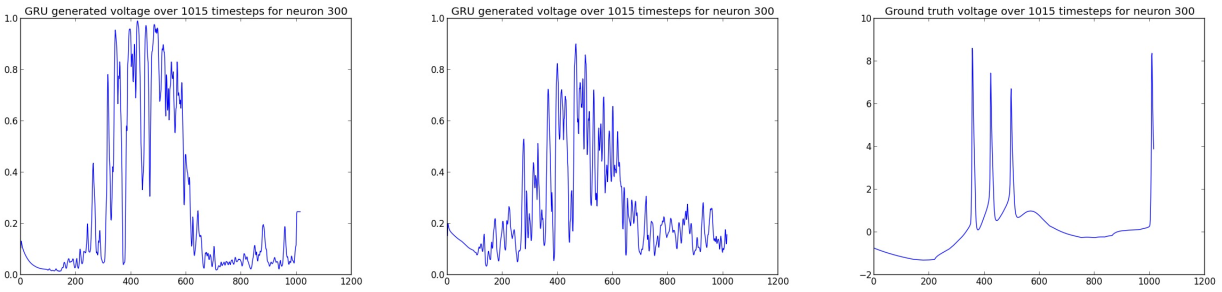


Figure 7: **From left to right:** plot of the predicted voltages of the GRU with each training example consisting of 1000 timesteps, the GRU with each training example consisting of 3000 timesteps (after both were trained for 105 epochs), and the ground truth voltage of neuron 300 over the first 1015 timesteps.

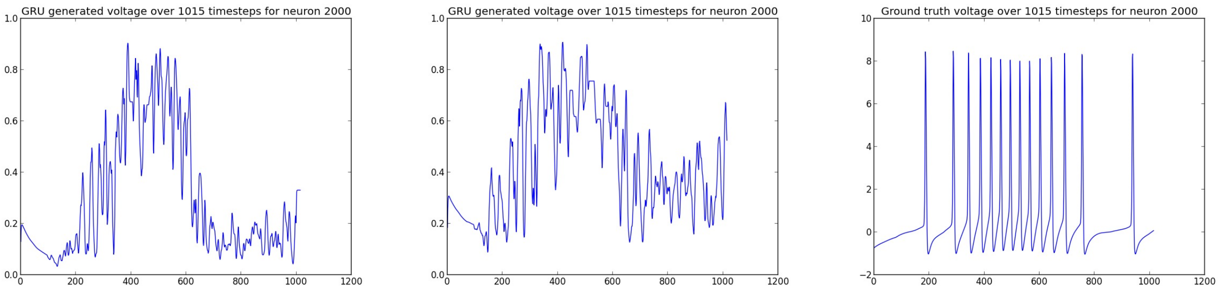


Figure 8: **From left to right:** plot of the predicted voltages of the GRU with each training example consisting of 1000 timesteps, the GRU with each training example consisting of 3000 timesteps (after both were trained for 105 epochs), and the ground truth voltage of neuron 2000 over the first 1015 timesteps.

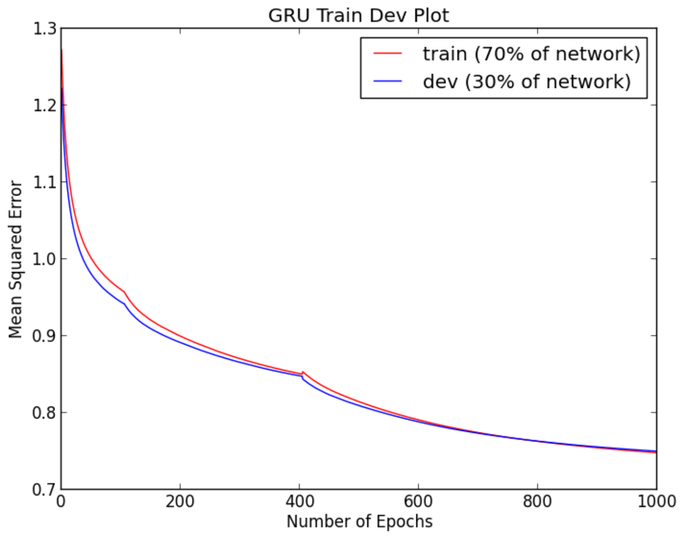


Figure 9: GRU train-dev plot after 1000 epochs.

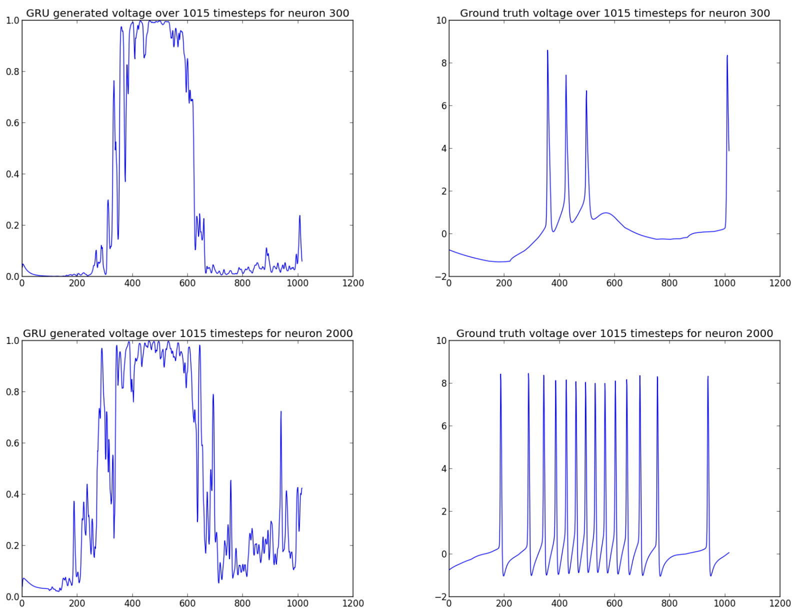


Figure 10: **First row:** plot of the predicted voltage of neuron 300 by the GRU (after it was trained for 605 epochs) and its ground truth voltage over the first 1015 timesteps. **Second row:** plot of the predicted voltage of neuron 2000 by the GRU (after it was trained for 605 epochs) and its ground truth voltage over the first 1015 timesteps.

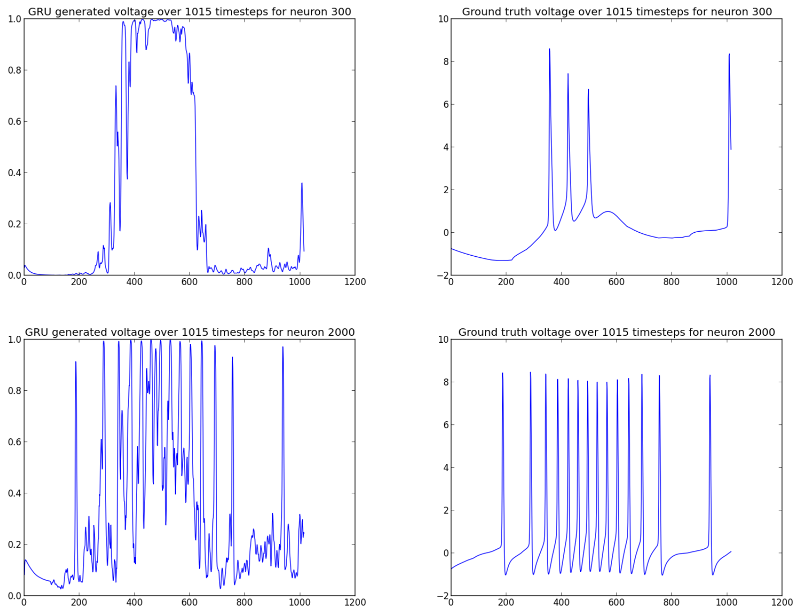


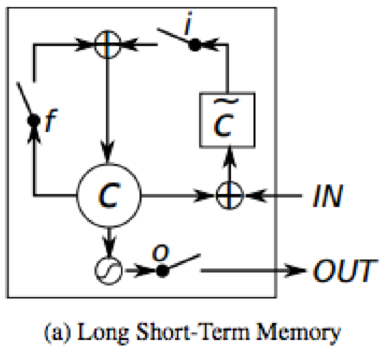
Figure 11: **First row:** plot of the predicted voltage of neuron 300 by the GRU (after it was trained for 1000 epochs) and its ground truth voltage over the first 1015 timesteps. **Second row:** plot of the predicted voltage of neuron 2000 by the GRU (after it was trained for 1000 epochs) and its ground truth voltage over the first 1015 timesteps.

3 Discussion

The LSTM network exhibited slightly lower training loss than the GRU but both exhibited similar validation loss, when trained on 70% of the network for 255 epochs. However, on their generated outputs, the LSTM and the GRU network vastly differed. The generated outputs of the LSTM were unsatisfactory and primarily consisted of noise. On the other, the generated output of the GRU network was visually plausible when compared to the ground truth. Therefore, it appears that the GRU network exhibited superior performance over the LSTM network, after both were trained for 255 epochs. Moreover, while increasing the number of timesteps per training example from 1000 timesteps to 3000 timesteps did not improve the GRU's performance, training the GRU for more epochs did in fact improve its performance. As we saw in Figure 11, for some neurons the generated output was more plausibly closer to the ground truth than for others. Thus, while recurrent networks do appear to be promising in terms of inferring voltages over time, there is still some work to be done.

Thus, to further investigate the viability of recurrent networks for this task, if we had more time, we would have trained *both* the GRU and the LSTM network for more epochs to get an even better gauge on their comparative performance. Furthermore, the number of gated recurrent units and LSTM units in our network should also affect performance. In our implementation, we make use of a single GRU unit and a single LSTM unit to represent transitions between previous hidden states and future hidden states. Given more time, we would have experimented with more gated units to see how much they improve performance. Finally, the number of hidden dimensions in our implementation was 2022 (the number of neurons). This seemed reasonable since the inputs are also of the same dimension. However, an interesting question to ask is whether 2022 hidden dimensions is optimal and if not, what number of hidden dimensions could potentially improve performance.

Unfortunately, due to the time and memory constraints of the Stanford Rye clusters, we were unable to pursue these future directions fully. We are eager to experiment with more complex architectures and larger simulated networks of neurons to see how well our preliminary results generalize and evaluate the effect of recurrent networks in predicting the network dynamics of a large population of neurons.



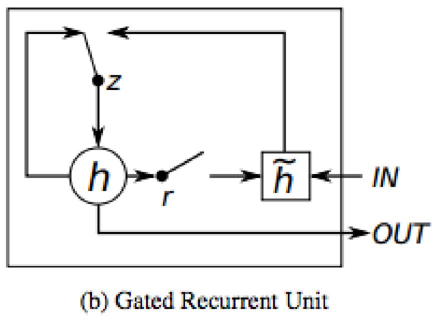


Figure 12: Illustration of (a) LSTM and (b) GRU, adapted from [4], Figure 1. (a) i, f and o are the input, forget and output gates, respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content. (b) r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation.

4 Methods

4.1 Network Architecture

The two networks we use in our task are the GRU and the LSTM. We describe their architectures below (adapted from [4], Sections 3.1-3.2).

GRU architecture

The activation h_t of the GRU at time t is a linear interpolation between the previous activation h_{t-1} and the candidate activation $\tilde{h}_t$:

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t, \tag{3}$$

where the update gate z_t decides how much the unit updates its activation, or content, and $\circ$ denotes element-wise product. The update gate is computed by

$$z_t = \sigma \left(W_z x_t + U_z h_{t-1} \right). \tag{4}$$

The candidate activation is $\tilde{h}_t$ is computed by

$$\tilde{h}_t = \tanh \left(W x_t + r_t \circ (U h_{t-1}) \right), \tag{5}$$

and the reset gate r_t is computed by

$$r_t = \sigma \left(W_r x_t + U_r h_{t-1} \right). \tag{6}$$

LSTM architecture

Each LSTM unit maintains a memory c_t at time t. The output h_t is given by

$$h_t = \sigma_t \tanh \left(c_t \right), \tag{7}$$

where σ_t is an output gate that modulates the amount of memory content exposure. The output gate is computed by,

$$\sigma_t = \sigma \left(W_0 x_t + U_0 h_{t-1} + V_0 c_t \right), \tag{8}$$

where V_0 is a diagonal matrix. The memory cell c_t is updated by partially forgetting the existing memory and adding a new memory content $\tilde{c}_t$:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \tag{9}$$

where the new memory content is

$$\tilde{c}_t = \tanh \left(W_c x_t + U_c h_{t-1} \right). \tag{10}$$

The extent to which the existing memory is forgotten is modulated by a forget gate f_t, and the degree to which the new memory content is added to the memory cell is modulated by an input gate i_t. Gates are computed by

$$f_t = \sigma\left(W_f x_t + U_f h_{t-1} + V_f c_{t-1}\right), \tag{11}$$

$$i_t = \sigma\left(W_i x_t + U_i h_{t-1} + V_i c_{t-1}\right), \tag{12}$$

where V_f and V_i are diagonal matrices.

4.2 Implementation

We initially wrote our own GRU and LSTM implementations. However, we soon found that these implementations were insufficient for training over our dataset, mainly due to the fact that they did not leverage GPU acceleration. As a result, we implemented the GRU and LSTM using the Keras library [3], which uses Theano for fast tensor manipulation and CUDA-based GPU acceleration. Moreover, we used Hinton's RMSprop for training, as it is an effective adaptive learning rate method[1]. We trained both our models on the Stanford Rye clusters, which have powerful GPUs. In particular, the Rye 1 cluster has a Tesla C2070 GPU, and the Rye 2 cluster has a GTX 480 GPU. Please refer to our code for our implementations, in particular, the Readme file explains in detail the libraries needed (which were the standard Numpy, Scipy, and Theano) and what each file in our codebase does.

5 Attribution

Aran implemented and trained the LSTM and GRU, and Richard and Dylan primarily worked on functional connectivity (their results are summarized in the appendix in §6). We thank Costas Anastassiou and his team at the Allen Institute for providing us with the simulated network data for our project. We thank Amy Christensen for providing us with code to read the .bin files provided by Costas. Finally, we thank Tom Dean for teaching CS 379C and putting us in contact with Costas.

6 Appendix: Functional Connectivity

Our original goal for this project was to infer synaptic weights from electrophysiological data. We found this to be more difficult than anticipated, but it was still a fun ride! For each of the 2022 neurons in the network, we had somatic voltage time series recordings. The recordings were for six seconds, and had resolution one tenth of a millisecond. As a result, we had 59999 recordings per neuron.

We assumed that the network was implemented according to a leaky integrate and fire model. Furthermore, we assumed that the input $I_i(t) = \sum_j w_{ij} \sum_f \alpha(t - t_j^{(f)})$, where w_{ij} represents the synaptic strength between neuron j and neuron i, and α is some kernel function (we assumed exponential).

6.1 Post Spike Voltage Average

Our philosophy was that if neuron i is presynaptic to neuron j, we should see the voltage of neuron j rise when neuron i fires.

To make this explicit, let $T = 59999$ be the total number of timesteps, set $V^{(i)} \in \mathbb{R}^T$ to be the somatic voltage of neuron i, and let $S^{(i)} = \{s_1^{(i)}, ..., s_n^{(i)}\}$ be the firing times of neuron i, where n is the total number of spikes of neuron i. Thus in our notation, $V_t^{(i)}$ will denote the voltage at timestep t in neuron i.

[1]RMSprop is currently unpublished, but Hinton describes this method on Slide 29: `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. It uses a moving average of squared gradients in order to adjust the Adagrad method in a very simple way in an attempt to reduce its monotonically decreasing learning rate.

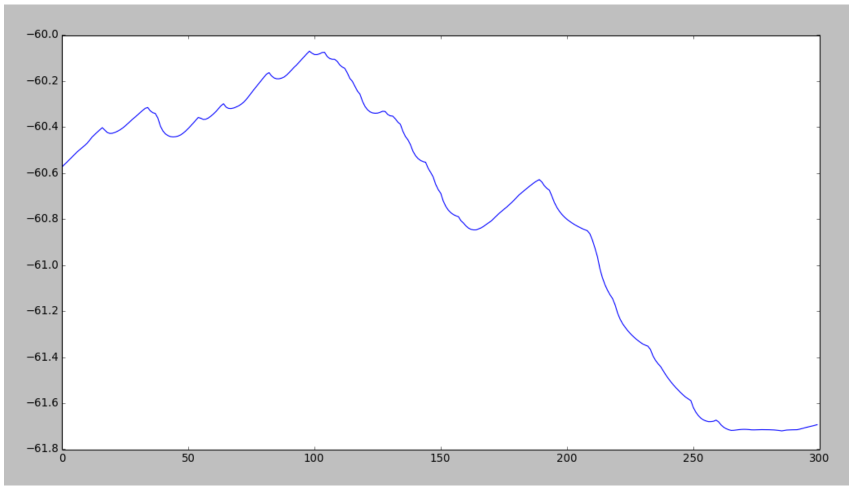


Figure 13: Plot of W_t for two unrelated neurons. Average Voltage (mV) vs. Time (10^{-4} sec)

In order to answer this question, we calculated the average voltage of neuron j, t timesteps after neuron i fires as t ranges from 1 to 300. Concretely, we found $W^{(ij)} \in \mathbb{R}^{300}$, where

$$W_t^{(ij)} := \frac{1}{|S^{(i)}|} \sum_{s_k^{(i)} \in S^{(i)}} V_{s_k^{(i)}+t}^{(j)}$$

for each $1 \leq t \leq 300$. In other words, $W_t^{(ij)}$ is the average voltage of neuron j, t time steps after a spike in neuron i, where the average is taken over each spiking instance.

As a first step, we tried plotting $W^{(ij)}$ for two neurons that did not have a direct connection (see Figure 13). However as can be seen in the plot, the voltage might significantly increase for a number of reasons unrelated to a direct connection between neuron i and neuron j.

This complex activity could be explained by a number of reasons. One hypothesis is that there is a neuron in between these two neurons so that there is no direct connection, but neuron i repeatedly causes a voltage rise in neuron j. Another explanation could be that these are not accurate estimations of the true effect, and we have too much noise. Or perhaps neuron j experienced a few drastic changes in voltage due to spikes which ended up dominating these calculations and skewing the average.

6.2 Estimating Current

Spiking of the presynaptic neuron causes the postsynaptic current to rise, which in turn causes the postsynaptic voltage to rise. Thus, we figured that we could more directly calculate the synaptic weights if we estimated the current of the neurons from the voltage. Using a leaky integrate-and-fire model, we note that voltage is always exponentially decaying towards IR, which allows us to infer current (up to multiplication by a resistive constant R) from voltage. More precisely, the model predicts that

$$R \cdot I(t) = V(t) + C\frac{dV(t)}{dt}$$

where R is the membrane resistance, $V(t)$ is the voltage, and $I(t)$ is the current. Since C depends on the time constant of the exponential decay, which is unknown to us, we made an educated guess that $C = 35$.

In the following plot (Figure 14), we picked a neuron (neuron 0) and plotted voltage over time (green) as well as the estimated current over time (blue).

According to the leaky integrate and fire model that we believe underlies Costas's model, the current should always be decaying exponentially towards the baseline current. However, this current does

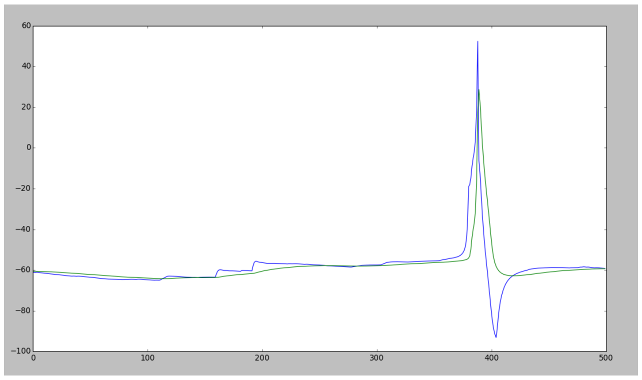


Figure 14: $V(t)$ in green, $R \cdot I(t)$ in blue. Voltage in 10^{-3} Volts and time in 10^{-4} seconds

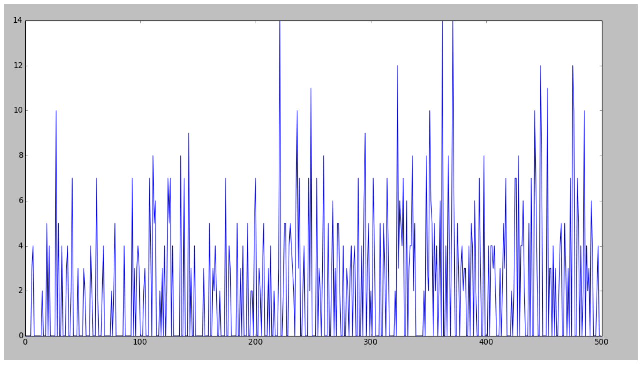


Figure 15: Weighted input spikes over time

not appear to be doing so. Is our estimate for current way off? Do we not understand Costas's model? We are curious to see how Costas's model actually works because we have not been able to reverse engineer the model.

We also calculated the total weighted input of spikes over time for neuron 0, which is plotted in Figure 15. We predicted that the total input would directly cause the movement of current; however, comparing the graphs, it does not seem to be possible to predict current as a function of the spike input. Because we are unable to see how the weighted input drives current/voltage, we are not able to infer functional connectivity.

Interestingly, we find in Figure 16 what appears to be a weak positive correlation between the moving average of the weighted synaptic input, and the voltage of the neuron.

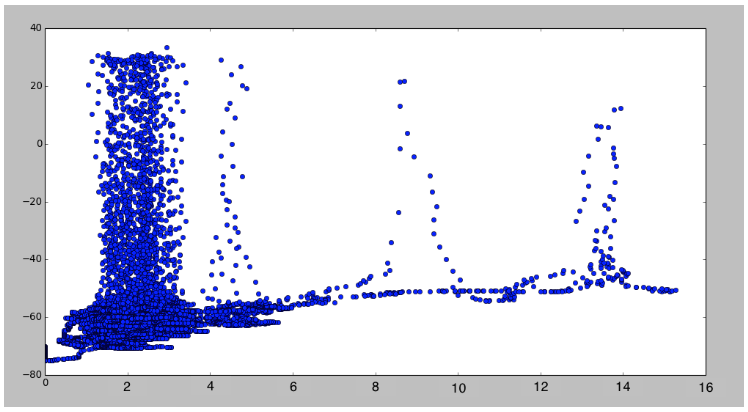


Figure 16: Voltage vs moving average of weighted synaptic input

References

[1] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". *IEEE Transactions on Neural Networks*, **5** (1994):157166.

[2] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. "On the properties of neural machine translation: encoder-decoder approaches". arXiv preprint arXiv:1409.1259, 2014. Presented at the Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8).

[3] F. Chollet. "Keras: Theano-based Deep Learning library". Code: `https://github.com/fchollet`. Documentation: `http://keras.io/`.

[4] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling". arXiv preprint arXiv:1412.3555, 2014. Presented in NIPS 2014 Deep Learning and Representation Learning Workshop.

[5] S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural Computation* **9** (1997): 1735-1780.

[6] E. Schneidman, M. J. Berry II, R. Segev, and W. Bialek. "Weak pairwise correlations imply strongly correlated network states in a neural population". *Nature* **440** (2006): 1007-1012.

[7] R. Socher. "CS 224D Lecture 7: Fancy Recurrent Networks for Machine Translation". April 22, 2015. `http://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf`.

[8] P. Vernaza and J. Bagnell. "Efficient high-dimensional maximum entropy modeling via symmetric partition functions". Advances in Neural Information Processing Systems 25 (NIPS 2012).