738 Final Project Report

Team: Babak Badnava, Brian McClannahan, Hao Tu, Thomas Woodruff

1. Introduction

In this project our aim is to detect the opening and closing of ion channels given a noisy signal. Ion channels are pore-forming proteins that are present in all cells of plants and animals. They reside in the membrane of the cell and are responsible for the flow of ions across the membrane. Ion transport usually happens very fast, and is stimulated via an electrochemical gradient without the aid of metabolic energy (ie. ATP). The channel itself is only large enough for one or two atoms to pass through at a time. [1] The channel has a feature known as selective permeability. This allows channels to filter which ions are allowed to pass through. Ions are filtered by their species (ie. potassium, sodium) or their charge (ie. cations, anions). The control of ion channels is known as ‘gating’, and can have different sources. These sources can range from electrical or chemical signals to temperature and mechanical forces. [2]

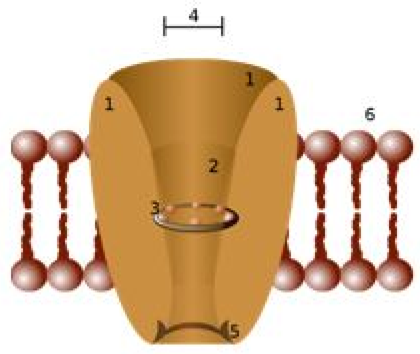


Figure 1. Ion Channel in a cell membrane.

Because ion channels regulate the flow of ions around the cell, they implicitly control the membrane potential, or the electric potential difference between the inside and outside of the cell. This potential is important in a number of bodily functions including the release of neurotransmitters, muscle contraction, growth, and many others. Disruptions or mutations to ion channels are the cause of various disorders, known as channelopathies. [2] Having the ability to measure and understand ion channels plays a critical role in new research. Unfortunately, current methods of detecting events when ion channels are open can be laborious and prone to human error. Therefore we seek to create a machine learning model to automatically detect these events to decrease measuring time and increase accuracy.

The data for this project are electrophysiological signals from cells and the number of open ion channels at a given time instant. There are ten batches of data that each span 50 seconds sampled at 0.0001 sec for a total of 5 million data points. The open channels were simulated and injected with real world noise. [3] Figure 2 shows a representation of the data. The raw signal is displayed and color coded corresponding to the number of open channels for that signal level. The distinction between batches is evident here. It should also be noticed that the signal seems to assume either a flat or parabolic signal type.

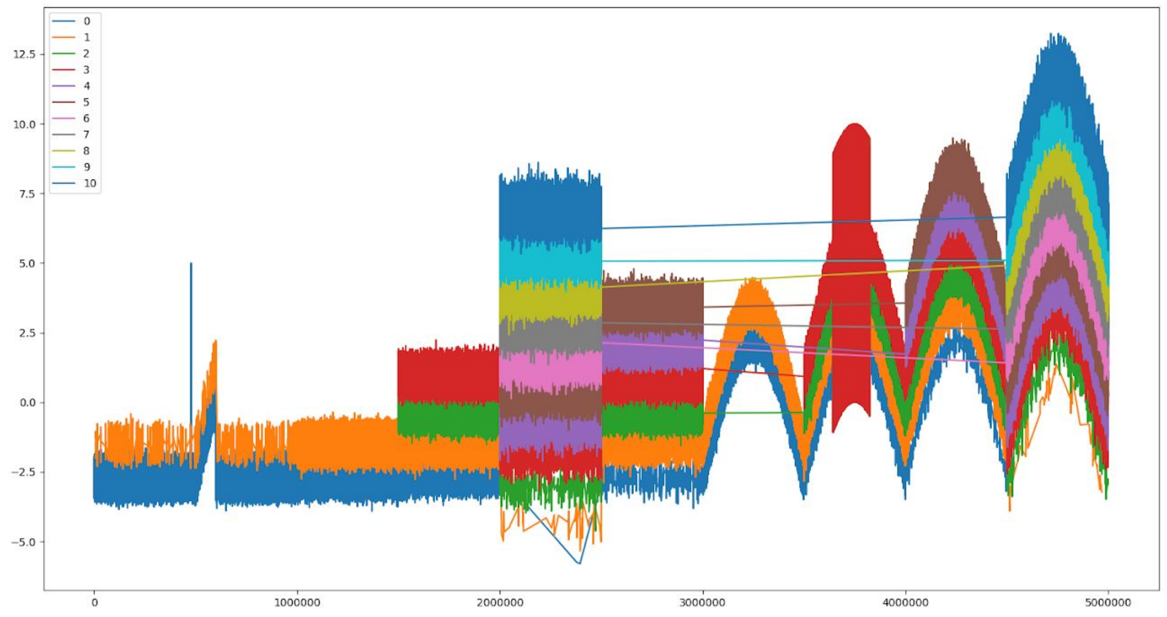


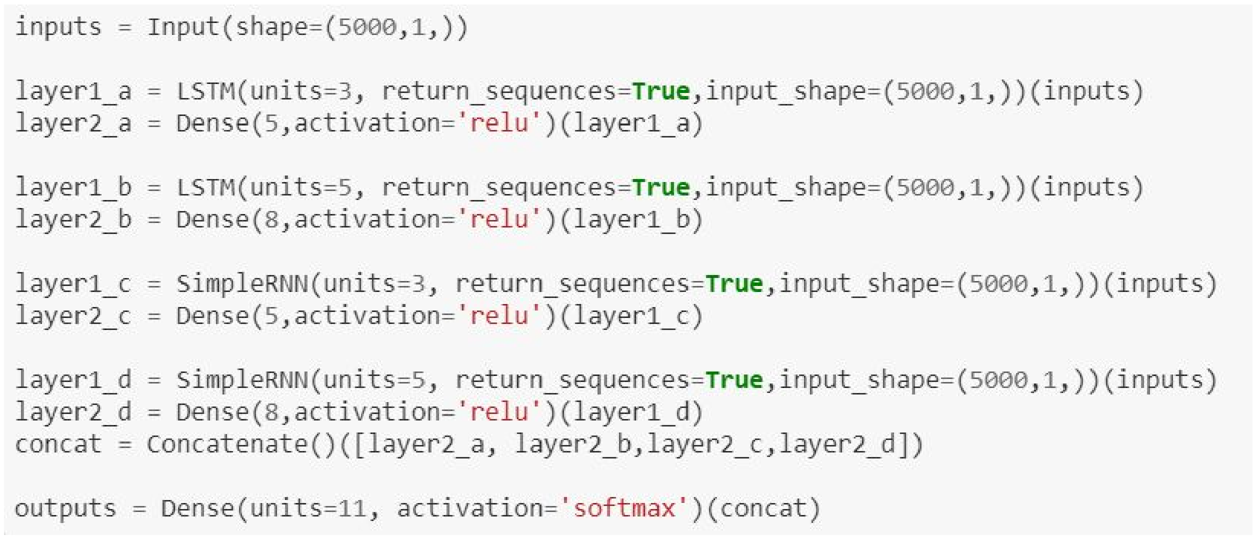
Figure 2. Number of open channels separated by signal value.

2. Model Descriptions

To handle the data, our models made one of two assumptions: the signal/open channels were somehow correlated across time or open channel events were independent of each other. We considered both cases because of our unfamiliarity with the subject and source of the data. The entire source code can be found at https://github.com/Twoodruff/738Project.

2.1 *Multipath RNN*

The best performing model we tested was a multipath recurrent neural network (RNN). This RNN used four separate paths of very small recurrent cells. Each of these paths had some type of recurrent layer connected to a small dense layer. Two of the recurrent layers were SimpleRNN layers and the other two were LSTM layers. One LSTM and SimpleRNN layer had a size of three and a connected dense layer of five while the other LSTM and SimpleRNN had a size of five and a connected dense layer of 10. The model can be seen in Code Block 1 below. These smaller networks showed much more success in calculating the number of ion channels compared to larger networks. The resulting output of each small dense layer was concatenated into a single layer and then connected to a dense layer of size 11 (one for each number of open channels). Originally, the final layer was only one cell, but we found treating this as a classification problem instead of a regression problem achieved better results.

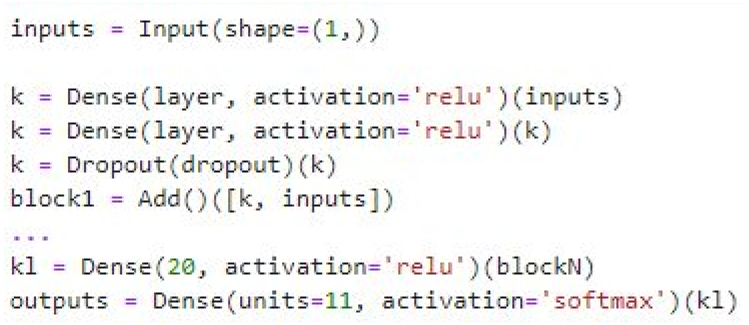


```
inputs = Input(shape=(5000,1,))

layer1_a = LSTM(units=3, return_sequences=True,input_shape=(5000,1,))(inputs)
layer2_a = Dense(5,activation='relu')(layer1_a)

layer1_b = LSTM(units=5, return_sequences=True,input_shape=(5000,1,))(inputs)
layer2_b = Dense(8,activation='relu')(layer1_b)

layer1_c = SimpleRNN(units=3, return_sequences=True,input_shape=(5000,1,))(inputs)
layer2_c = Dense(5,activation='relu')(layer1_c)

layer1_d = SimpleRNN(units=5, return_sequences=True,input_shape=(5000,1,))(inputs)
layer2_d = Dense(8,activation='relu')(layer1_d)
concat = Concatenate()([layer2_a, layer2_b,layer2_c,layer2_d])

outputs = Dense(units=11, activation='softmax')(concat)
```

Code Block 1. Multipath RNN model.

2.2 *ResNet*

Under the assumption that the number of open channels was only dependent on the signal at the current time step, we developed a model with a ResNet architecture. The goal was to create a model complex enough to capture the open channels despite the noise. The input to the network was a single signal value that was to be mapped to a single output. Each residual block consisted of two dense layers and a dropout layer, as shown in Code Block 2. Two models were created to try and capitalize on the two signal types. One model was trained on batches (1-6), which represent the flat signal type. The other model was trained on batches (2,7-10), which represent the inverse parabolic signal type. Note that batch two was included in both models because there are traces of both signal types. These two models were used in the ensemble method described in Section 2.5. A model was also trained over the entire data set but was not used in the ensemble method.



```
inputs = Input(shape=(1,))

k = Dense(layer, activation='relu')(inputs)
k = Dense(layer, activation='relu')(k)
k = Dropout(dropout)(k)
block1 = Add()([k, inputs])
...
kl = Dense(20, activation='relu')(blockN)
outputs = Dense(units=11, activation='softmax')(kl)
```

Code Block 2. Residual block.

The ‘flat’ model was trained using two residual blocks with 20 neurons in each Dense layer and a dropout of 0.1. The ‘parabolic’ model was trained using two residual blocks and 30 neurons in each Dense layer and the same dropout. The full data model was trained with three residual blocks and 20 neurons per Dense layer, also with the same dropout rate. The ‘flat’ and ‘parabolic’ models were only trained and tested on their respective data batches and that is what is reflected in the reported accuracies in Section 3.

2.3 *Bidirectional LSTM*

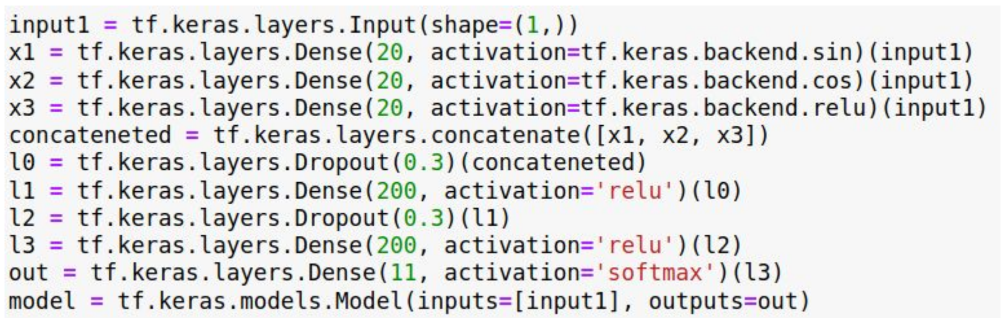
In this model, we constructed an architecture which has a bidirectional LSTM layer followed by a time distributed dense layer, shown in Code Block 3. We thought this might yield a good result since the raw data could be time dependent. In the bidirectional LSTM, the output layer can get information from past (backwards) and future (forward) states simultaneously, which can capture the features of the data set better than RNN.

```
model = Sequential()
model.add(Bidirectional(LSTM(units=20, return_sequences=True,
input_shape=(5000,1))))
model.add(TimeDistributed(Dense(units=1,activation='relu')))
model.compile(loss='mean_squared_error',optimizer='sgd',metrics=['accuracy'])
```

Code Block 3. Bidirectional LSTM block

2.4 *Feature Extractor*

In this model, we also assumed that the number of open channels in each timestamp only depends on the current value of the measured signal and developed a model that uses different activation functions as feature extractors in the first layer. The activation functions, other than ReLU, that we used are “sine” and “cosine” activations. The basic version of this model overfits easily and has poor performance. It is shown in Code Block 4. We used other techniques to boost the performance of this network such as adding dropout layers and bootstrapping methods. Since this data is imbalanced, we also attempted a “One vs All” method for this dataset. We trained a set of networks on each of the classes separately, and then aggregated the result by voting. The class with the largest number of votes, each vote for a specific class, is nominated as the predicted class. In this ensemble, each of the networks would be trained to do a binary classification on one of the classes. To handle the problem of imbalanced data, we collected all of the samples from that specific class and the same number of samples randomly from other classes for each trained network. Although each model has a good performance, the aggregated result does not perform very well. For bootstrapping methods, we trained a set of networks, each on a subset of data. In this case the label for each of the samples will be determined by voting. The difference between this model and the previous model is that each network in the last setting was a binary classification, but in this method each network is trained on all of the classes.



```
input1 = tf.keras.layers.Input(shape=(1,))
x1 = tf.keras.layers.Dense(20, activation=tf.keras.backend.sin)(input1)
x2 = tf.keras.layers.Dense(20, activation=tf.keras.backend.cos)(input1)
x3 = tf.keras.layers.Dense(20, activation=tf.keras.backend.relu)(input1)
concateneted = tf.keras.layers.concatenate([x1, x2, x3])
l0 = tf.keras.layers.Dropout(0.3)(concateneted)
l1 = tf.keras.layers.Dense(200, activation='relu')(l0)
l2 = tf.keras.layers.Dropout(0.3)(l1)
l3 = tf.keras.layers.Dense(200, activation='relu')(l2)
out = tf.keras.layers.Dense(11, activation='softmax')(l3)
model = tf.keras.models.Model(inputs=[input1], outputs=out)
```

Code Block 4. Basic version of feature extractor network

2.5 *Ensemble*

We also tried to create an ensemble approach using different combinations of these methods. We included four models in our ensemble method: the multipath RNN model, both ResNet type models, and the basic feature extraction model with dropout. Each ensemble would use 2-4 models. The ensemble would run each model on the data independently then add the results from all of them together and pick the maximum value from the combination. The different ensemble combinations can be found in Table 2 below.

3. Results

An overview of the results from each of the different methods is shown in Table 1. Each model is discussed further in the subsequent sections.

Table 1: Model Testing Accuracy

Model	Accuracy
Multipath RNN	71.23%
ResNet - full	58.96%
ResNet - flat	86.26%
ResNet - parabolic	52.35%
Bidirectional LSTM	40.42%
Basic Feature Extraction (overfits easily)	22.00%
Ensemble of Feature Extraction	31.60%

3.1 *Multipath RNN*

The multipath model was the best performing model tested. It is an improvement on the model we presented during our April progress update which achieved ~44% accuracy. It was designed to take advantage of both LSTMs and SimpleRNN cells, while keeping the total model size very small to prevent overfitting on the singular sequence input. It had a training accuracy of 70.96% while maintaining a testing accuracy of 71.23%, indicating it was not overfitting at all. Previously, we presented a binary classifier that only tried to predict if there were any open channels or not, not how many channels were open. This multipath model also showed an improvement in that area, going from 94.3% binary classification accuracy to 96.4%. This shows it maintained its high accuracy when there were zero open ion channels, but greatly increased its accuracy on the specific number of open ion channels. One possible improvement on this model could be adding another path for one-dimensional convolutional filters as they have shown great ability to analyze time series as well.

3.2 *ResNet*

The goal of the ResNet model was to learn the relationship between the signal and open channels with no correlation between data. During training the data was completely mixed. It did well in the classification of the flat models at 86% and put up a comparable performance for the parabolic signals at 52%. This decrease in performance makes sense as the parabolic data is much more complex than the flat data. Each model does better in its respective domain than the full model over both domains. This is why the two smaller models were included in the ensemble and not the full model. Despite the noise and variation of signals, the ResNet models achieved their final accuracies with relatively small, or simple, architectures. Larger models would produce similar or worse results. Within the field of models presented in this paper, the ResNet did surprisingly well for not using sequential data. However, it couldn’t outperform the Multipath RNN when used in the ensemble method. This suggests that the data may have some dependence unaccounted for in the ResNet. Similar to the Multipath RNN, one approach that could be implemented is the use of 1D convolution layers instead of Dense layers.

3.3 *Bidirectional LSTM*

Bidirectional recurrent neural networks are especially useful when the context of the input is needed. For example, in handwriting recognition, the performance can be enhanced by knowledge of the letters located before and after the current letter. There are also articles which mention that Bidirectional LSTM gives good results on time series forecasting [4], which motivated us to try this method on our data. We believe that the low accuracy we got is due to the complexity of our data, since the noise has a great influence on feature extraction.

3.4 *Feature Extractor*

We have tested three different versions of this idea: the basic version, “one vs all”, and ensemble of feature extractors. The basic version saturates easily and has poor performance both on the training set and test set. We believe it is due to the complexity of the data. The ensemble method performs better, but still overfits, and gives an accuracy around 31.6% with 5 networks, each one trained on 60% of randomly sampled data points from the training set. Meanwhile, the training accuracy of this ensemble is 71.6%.

3.5 *Ensemble*

None of the ensemble approaches worked better than using just the multipath model exclusively. This is most likely because all the models were good at finding the number of open channels for the easier-to-classify inputs, but struggled on the harder-to-classify inputs. The multipath model showed ~70% accuracy while the others did not perform as well. For inputs where the multipath model could classify it but the others could not, the ensemble result becomes distorted causing a lower performance. Typically, when working with ensemble approaches the goal of the ensemble is to have the models cover for each other's weaknesses. However, based on these results it is likely the models were all good at predicting on the same inputs and unable to improve each other when working together. One note is that the multipath model had a 92.8% accuracy when it had 70% or greater confidence in its guess. We could possibly find an ensemble approach that improves on the multipath model by always taking the multipath model's result when it has high confidence and only using the other models when the multipath model has low confidence. It would also be worth trying different voting schemes, such as taking the most votes or using the highest confidence vote among all models.

Table 2: Ensemble Testing Accuracy

Combination of Models	Accuracy
All	58.64%
Multipath RNN/ Feature Extractor	59.81%
Multipath RNN/ ResNet (both)	63.59%
Multipath RNN/ ResNet (flat)/ Feature Extractor	55.93%
Multipath RNN/ ResNet (parabolic)/ Feature Extractor	62.42%
Multipath RNN only	70.33%

Note: The test set used during ensemble testing was slightly different from the test set models used during training, which accounts for slightly different accuracies of some models.

4. Conclusions

This dataset is a challenging dataset since it is imbalanced, the number of features are very limited, and wave forms in each of the batches are different from each other. As seen in Fig. 2, a single signal value can represent two different classes in different batches. The most likely cause for this is the time dependent nature of the data. In order to classify on just the current signal, we would need more features to predict the correct number of open channels. Our results confirm the time dependency as our recurrent models gave the best results. The other models that did not have any kind of time dependency like the ResNet model and feature extractor struggled much more and had overfitting problems. Performance might be further improved if we implement some kind of preprocessing techniques, such as filtering the noise.

5. References

[1] Nature Education, "Ion Channel," 2014. [Online]. Available: https://www.nature.com/scitable /topicpage/ion-channel-14047658/. [Accessed 7 May 2020].

[2] Ratan-NM, "Importance of Ion Channels in the Body," News-Medical, 25 October 2018. [Online]. Available: https://www.news-medical.net/health/Importance-of-Ion-Channels-in-the-Body.aspx. [Accessed 7 May 2020].

[3] "University of Liverpool - Ion Switching," 2020. [Online]. Available: https://www.kaggle.com/c/ liverpool-ion-switching/overview.

[4] J. Brownlee, "How to Develop LSTM Models for Time Series Forecasting," Machine Learning Mastery, 6 January 2020. [Online]. Available: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/. [Accessed 7 May 2020].