MA679 Final Project

Lihao Liao, Ruining Jia

May 09, 2022

Contents

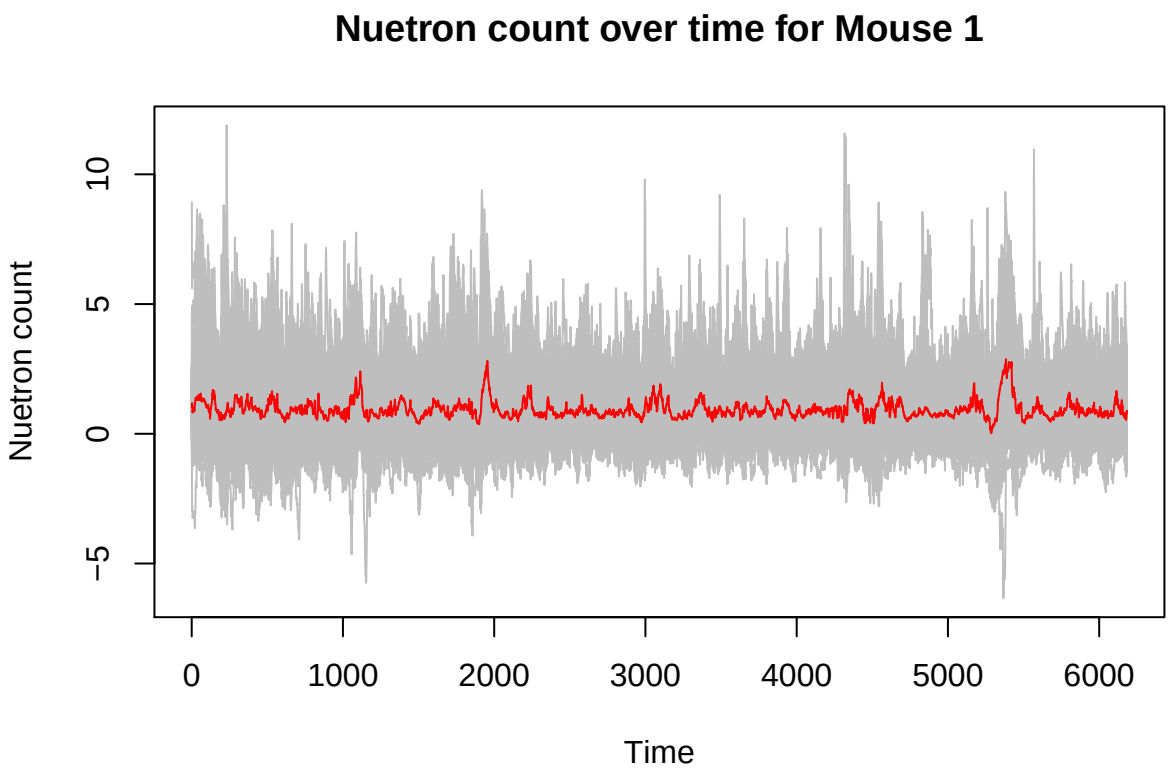

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'tibble'
```

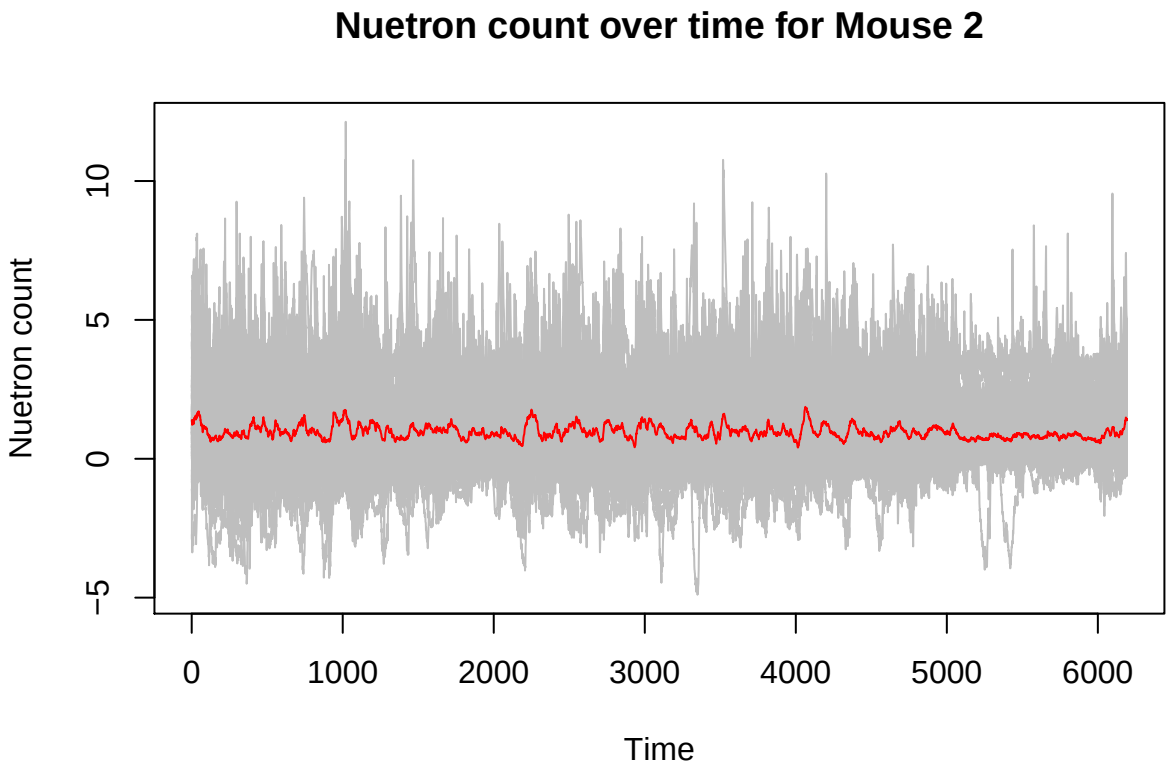
```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'pillar'
```

```
## Warning: replacing previous import 'lifecycle::last_warnings' by
## 'rlang::last_warnings' when loading 'hms'
```

1 Introduction

Here are the plots of neutron counts over time for Mouse 1 and Mouse 2, in which the red line repents the mean neutron counts over time across all the cells. We can clearly see that neutron counts change along time.


Nuetron count over time for Mouse 1
Nuetron count
10
5
0
−5
0
1000
2000
3000
4000
5000
6000
Time


Nuetron count over time for Mouse 2
Nuetron count
10
5
0
−5
0
1000
2000
3000
4000
5000
6000
Time

2 Baseline model

If we do not consider the potential time dependence of the neutron activity and mouse behavior and assume that the each (temporal) sample is independent of each other, we can build of logistic regression model to predict and classify the behavior of each mouse (open arm =1, close arm 0). To do so, we need to only keep the column representing open arm in the behavior data since if it is close arm the value in that column would be 0 and if it is open arm the value in that column would be 1. In addition, we need to combine data for Z score (neutron activity) and behavior into one data set, which can be easily done by merge those two data sets by column since we have already confirmed that ith row in each data set both represents the ith (temporal) sample during the EDA. After that, we can build a logistic regression model with mouse behavior (open arm or close arm) as outcome and neutron activity (Z scores) of each cell as predictors.

We split the data into two data sets: Training and Test for each mouse. Training data were used to build the logistics regression model and test data were used to check the performance of the logtisitc regression model. For each mouse, we first fit the logistic regression model with all the cells (predictors). We then only keep the cells (predictors) that are statistically significant at 5% confidence level and refit the logistic model with only the significant cells. False Negative rate, False Positive rate, and Precision rate were obtained using test data for each mouse and can be found in the following table

Table 1: Error rate and precision for logistic regression model of the test data

	False Negative rate	False Positive rate	Precision rate
Mouse 1	0.0875764	0.1723238	0.8949353
Mouse 2	0.0474334	0.1222571	0.9397201
Mouse 3	0.2638332	0.3138564	0.7199350
Mouse 4	0.2237569	0.3934837	0.7395777
Mouse 5	0.0366252	0.1269350	0.9476242
Mouse 6	0.0628723	0.1563422	0.9200000
Mouse 7	0.2951699	0.2427184	0.7343750
Mouse 8	0.0398366	0.1032028	0.9460317
Mouse 9	0.0250587	0.0456432	0.9716733
Mouse 10	0.1001410	0.1287356	0.8889860
Mouse 11	0.0537634	0.0811594	0.9388235
Mouse 12	0.1953125	0.2638889	0.7823795
Mouse 13	0.0893372	0.0659459	0.9276730

In the following models, we will be using data from Mouse 1.

3 A basic neural network approach (Flatten layer model)

The aim of this work is to predict the future behavior of a mouse given its current neural activity as well previous behaviors and neural activity. We need to decide how far back we would like to look back and how we would like to sample data during the training process for depicting future behavior. For the basic neural network framework, we choose to look back as far as 40 time points ago and draw one data point every 10 time points for predicting the behavior of a mouse for the next time point.

An essential part of the neural network is that we need to process the raw data into a format that a neural network can ingest. To do that, we first need to make sure that the data are numerical. Secondly, we normalized the raw data to make each time series be on the same scale. We then need to obtain a batch of data containing data in the past 40 time points as well as the "current" time point. Given the size of the data we have, if we feed them all in at one time, it might cause a huge computation burden on the computer that can crash R. Therefore, we decided to "chop" the data into small batches and feed the one batch in instead of the whole data. We set the batch size to be 120, which means there are 120 samples in each batch. This is

done by a custom generator function based on the mouse data, which outputs a list of two elements: Inputs and targets. For each training epoch, we did 500 steps (batches of samples), which means we "searched" the whole data about 60 times for each training epoch ($120 * 500 * 6/6000$).

We first fit a basic neural network model, which is a flattened layer network. A flatten layer will take an array input, as a flatten layer, i.e., an vector. For example, a 2 dimensional 4×4 array will become a one-dimensional vector of 16 in a flattened layer. The following picture illustrates the flatten layer neural network we have applied in this project. A 2 dimensional 120×111 array has been fed in and a one-dimensional vector of 13320 has been taken in the first layer. The flatten layer networks will not account for temporal dependence between observations.

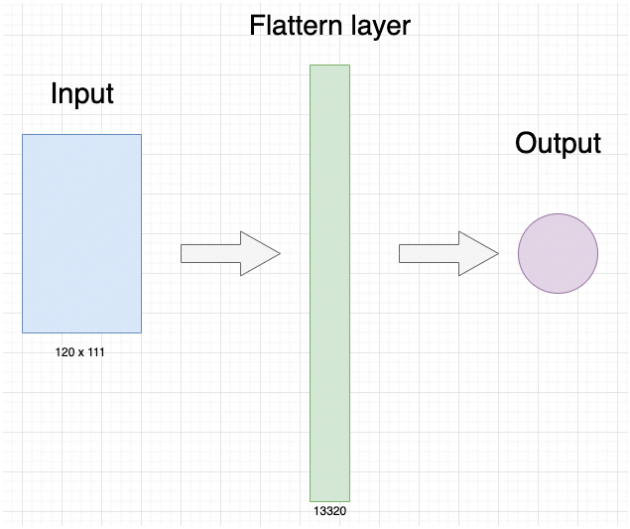
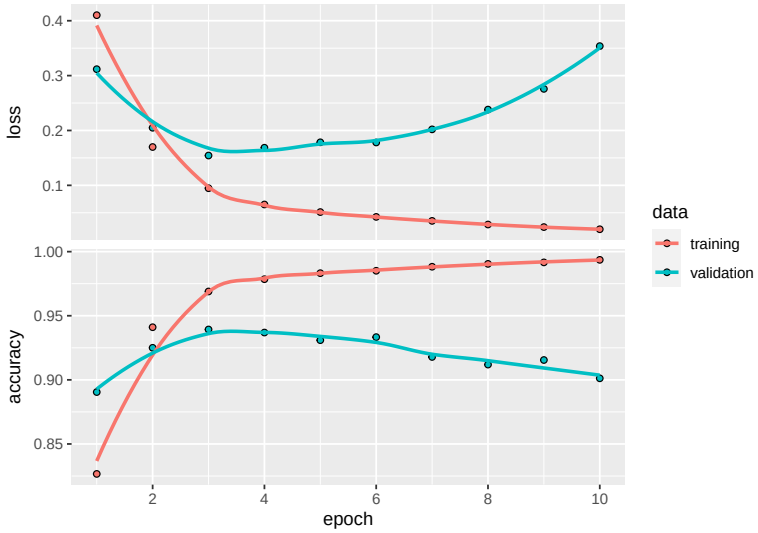


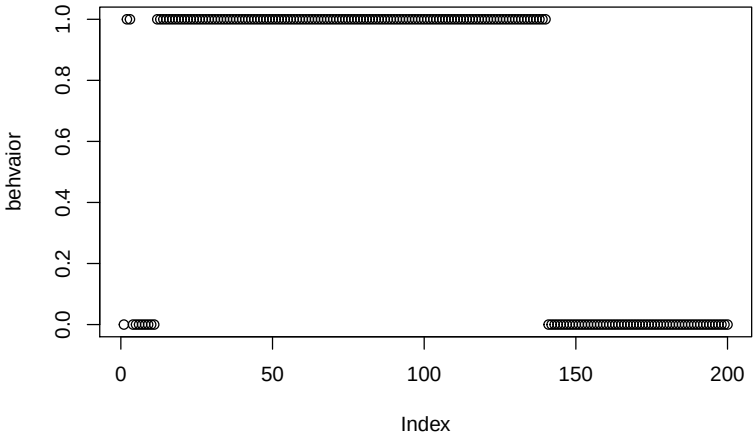
Figure 1: Flatten layer

We split the data into three parts: training, validation, and test data. Data obtained at the first 3000 time points will be used as training data. Data obtained at the next 1000 time points will be used as validation data. The rest data will be used as test data.



4 Recurrent neural network (RNN)

A recurrent neural network (RNN) has often been used to predict an event that does not only depends on the predictors of the current time point but also depends on the previous events. In this project, the behavior of the mouse does not only depend on its current neutral activity but can also be affected by its previous behaviors. From the plot of the behavior of Mouse 1 during the first 200 time points, we can see that its behavior in the next time point tended to be consistent with its previous time points.



The following plot shows the basic idea for an one-layer many-to-one RNN.

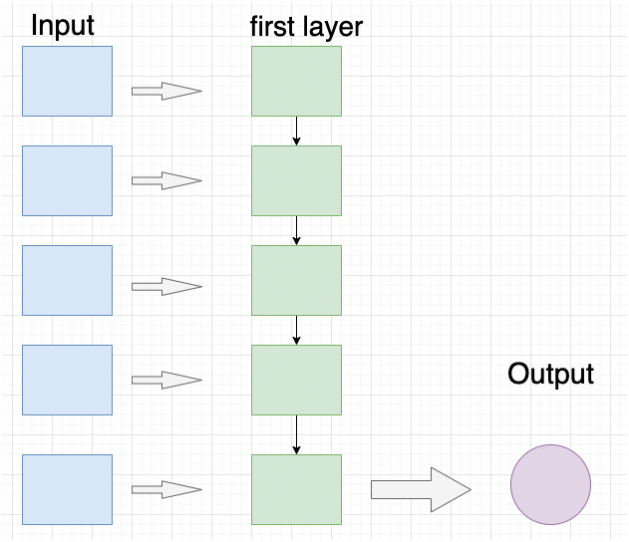
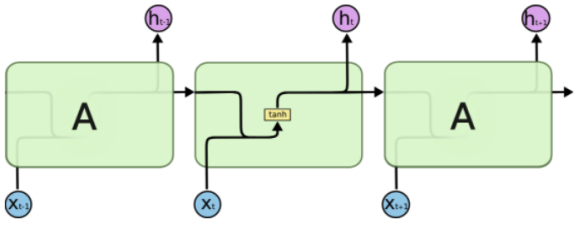


Figure 2: RNN

Here is the unfolded-version of a simple RNN

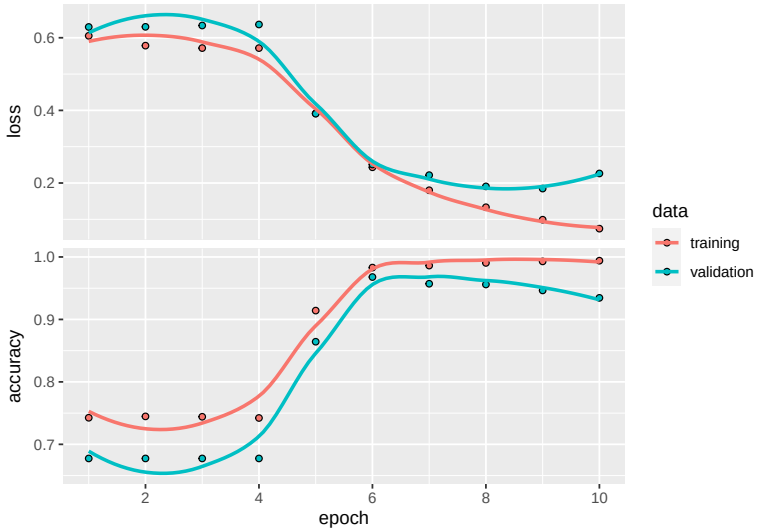


The general steps included in a simple RNN are:

1. Input $X(t_1)$ is fed into RNN and generate an output $h(t_1)$
2. Output $h(t_1)$ and $X(t_2)$ will both serve as input for the second step, which generate output $h(t_2)$.
3. Output $h(t_1)$ and $X(t_3)$ will both serve as input for the second step, which generate output $h(t_3)$.
4. This goes on until hit the final output $h(t_{k+1})$, where k is the number of time points we would like to look back.

4.1 Simple RNN

We first fit a simple RNN that look back as far as 40 time points ago to predict the behavior at the next time point. The accuracy of validation is about 0.9, which is not bad can be better.

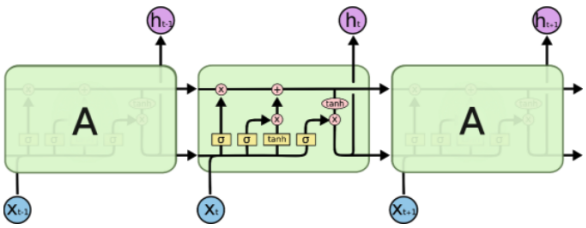


4.2 Long Short Term Memory RNN (LSTM)

Simple RNN is a good baseline RNN. The major advantage of RNN is that we believe or assume that it is able to provide previous information in the current training, i.e., it can take temperate dependency into consideration during model training. However, long-term dependencies are often missed during the optimization process. For example, a simple RNN might perform not as well as we expect it to be, if the prediction of current or future behavior of a mouse is related not only to its recent status but also its status in the very past.

Long Short Term Memory RNN model (LSTM) will remember the previous information (long-term dependency) and use it for processing current input, as it is illustrated in the following picture. LSTM will "carry out" the previous output and feed it into the current training process. In other words, LSTM will not only consider the temporal dependency between observations but it will also consider the temporal dependency between predictions.

The basic idea of LSTM RNN is illustrated in the following diagram. Within each recurrent layer, there are four more interacting layers that controls the information fed in the cell state.



This following plot shows a more detailed framework of LSTM.

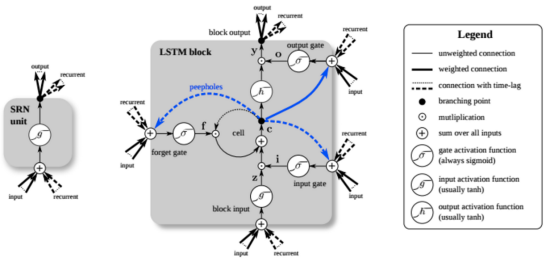
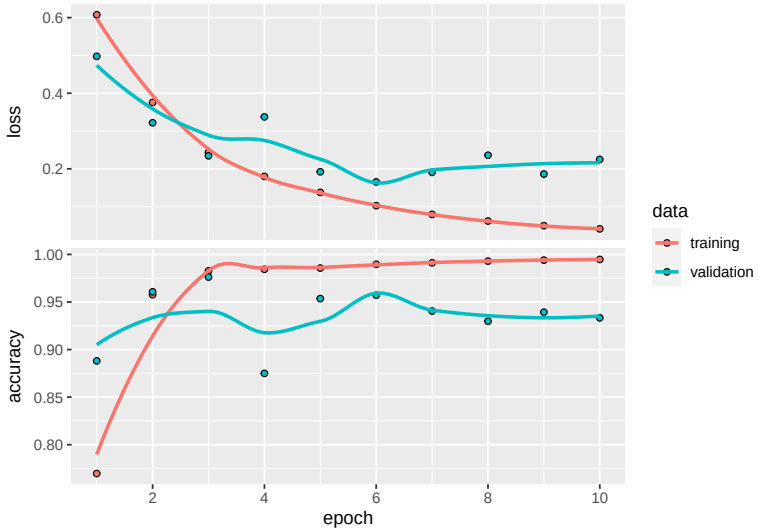


Figure 3: More detailed LSTM

1. The Forget gate layer decides which information will be tossed away from the cell state
2. The Input gate layer decides decides which new information will be fed into the cell state.
3. This information has then been passed into a tanh layer to create a vector of new candidate information, which will be fed into the new cell state
4. The output gate decides which information will be output

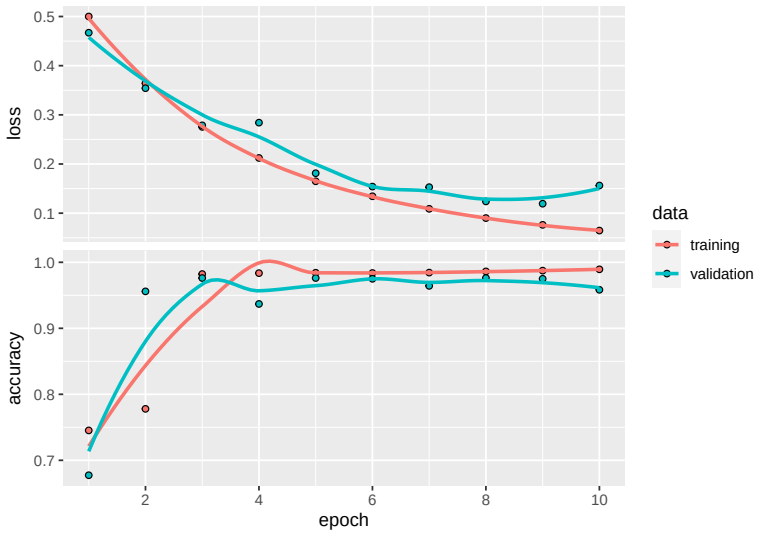
The accuracy of validation for the LSTM RNN is about 0.75, which is quite poor. However, this might be caused by overfitting issues, which will be addressed later



4.3 Gated Recurrent Unit RNN (GRU)

Gated recurrent unit is very similar to LSTM as they share the same principle. However GRU usually requires fewer parameters than LSTM, which makes it easier to run but also makes it have less representational power than LSTM.

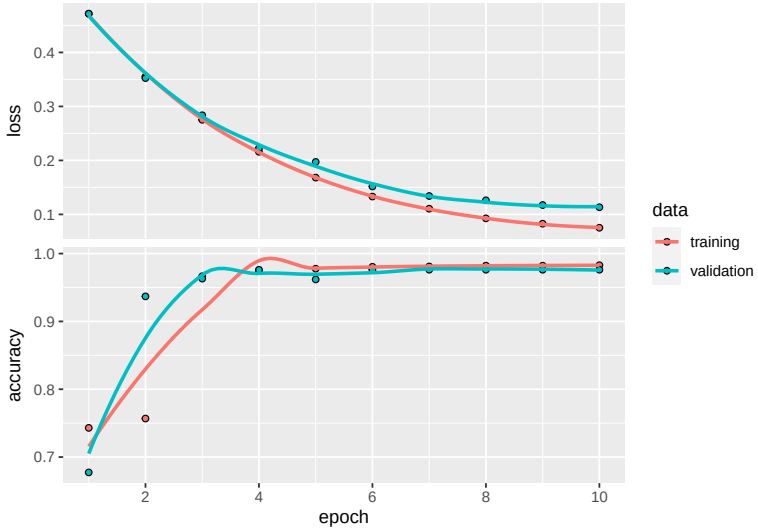
The accuracy of validation for the GRU RNN is about the same as LSTM, which is quite poor. However, this might be caused by overfitting issues, which will be addressed later



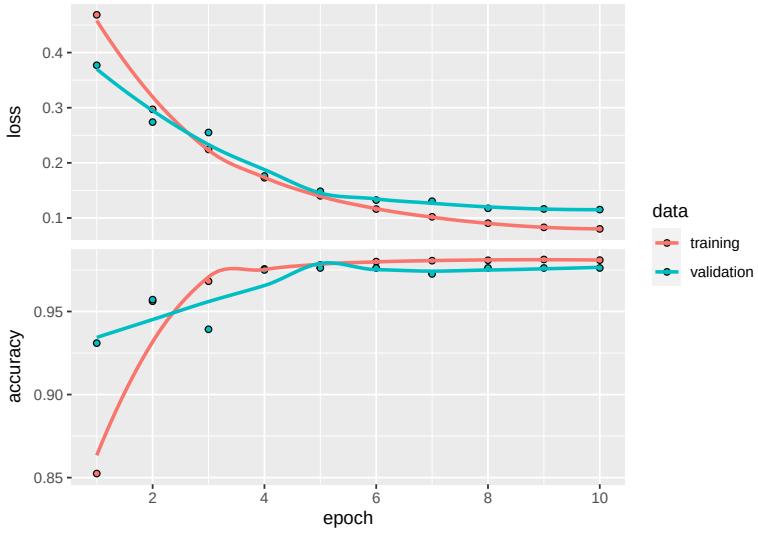
4.4 Address overfitting issues

It is clear that there is overfitting issues for simple, LSTM, and GRU RNN models as the training and validation curves start to diverge from each other after a few training epochs. For the simple RNN model, we can randomly drop out a certain percentage of connections between layers to prevent overfitting issues. For LSTM and GRU RNN, such dropout should be happen at each time step. In addition, a temporally constant dropout was applied to the inner recurrent activation, which will help to regularize he representation of the recurrent gates. In this project, we decided to drop out 20% of the connections between the input and first hidden layer and drop out 20% of the inner recurrent activation for the LSTM and GRU RNN.

4.4.1 LSTM



4.4.2 GRU



The training and validation curves did not diverge from each other for the first training epochs for both LSTM and GRU RNN, which suggested that we have successfully preventing the overfitting issues. In addition, the accuracy of validation for the GRU RNN is about the same as LSTM, which is close to 1. The accuracy test data for the LSTM is about 83.33% and is about 1 for the GRU RNN.

5 models for all mice

We decided to perform both LSTM and GRU RNN on all the mouse since GRU RNN requires less computation power and showed very similar or even better results than LSTM RNN.

5.1 LSTM RNN

5.2 GRU RNN

5.3 Compare to baseline model

Table 2: Accuracy of LSTM RNN, GRU RNN, and logistic regression model of the test data

LSTM	GRU	Logistic
0.1000000	1.0000000	0.8949353
1.0000000	1.0000000	0.9397201
0.4833333	1.0000000	0.7199350
0.5916667	1.0000000	0.7395777
1.0000000	1.0000000	0.9476242
1.0000000	1.0000000	0.9200000
1.0000000	0.5750000	0.7343750
0.1000000	1.0000000	0.9460317
0.1750000	0.8750000	0.9716733
1.0000000	0.9083333	0.8889860
1.0000000	1.0000000	0.9388235
1.0000000	1.0000000	0.7823795
1.0000000	1.0000000	0.9276730

6 Take Away Message

1. RNN is generally performs better than simple flatten-layer NN and logistic regression since it is able to address temporal dependency between observations
2. LSTM and GRU RNN performs better than simple RNN since it is able to address long-term dependency, which is evidently required for predicting future behavior of a mouse in this work.
3. The performance of LSTM and GRU RNN can be affected by overfitting issues, which is quite normal when we have high-dimensional data.
4. Overfitting issues can generally be prevented by dropout a certain percentage of connections between layers. For LSTM and GRU RNN, a temporally constant dropout was applied to the inner recurrent activation would also help to prevent overfitting.
5. When the performance of GRU and LSTM RNN are very similar, we may choose to use GRU RNN since it requires less computation power.
6. Approporiate RNN models should be selected based on the data and aim of the study.