# Using place cells' local field potential to predict a mouse's location

Neural Networks Practical Project

**Farrukh Baratov** s3927083
**Odilon Thioux** s4003322
**Darie Petcu** s3990044
**Maxime Bos** s3953246

**Abstract**

This project aims to use a feed-forward neural network to predict mice movement inside a maze, based on electrophysiological data gathered from electrodes implanted in the mice's hippocampus. A linear model was first used to evaluate the task and to check whether the task was feasible at all. K-folding cross-validation with 9 folds proved that both the linear model and the FNN had loss values that were sometimes higher, and sometimes lower than simply educated guessing. However, the FNN loss values were mostly lower than the linear model's loss values. The loss values of both the linear model and the FNN increased after fold 6.

# Contents

## 1 Introduction

The goal of our project was to train a neural network to predict the position of a mouse in a maze based on the local field potentials (LFPs) of its place cells as accurately as possible, with accuracy being measured by the distance between the predicted position and the actual position. The electrophysiological data used for training and testing was collected by Senzai and Buzsáki (2017), and consists of the LFPs of the mice' place cells. These mice were located in and moving inside mazes when the data was recorded.

Place cells were discovered in 1971 by John O'Keefe and John Dostrovsky. These are pyramidal cells located in the hippocampus of animals. Their defining characteristic is that the firing of these neurons is location-specific, i.e. certain environments trigger the firing of these cells. This location is defined as the "place field". Place fields are specific to each place cell, and the cell's firing is negligible when it is outside of its place field (Muller, 1996).

Senzai and Buzsáki (2017) investigated the properties of granule and mossy cells inside the hippocampal dentate gyrus (DG) for the purpose of creating an electrophysiology-based classification of these cells. To do this, they implanted mice with electrodes and monitored their brain activity while they were exploring various mazes. They found that mossy cells in the DG showed stronger remapping than granule cells did under the same conditions.

Optogenetic validation was used to differentiate between mossy and granule cells, whose LFPs were tracked and stored to create the data set that was used in this research. Figure 1 displays the environments in which the mazes were located. Walls were added to the structures in the figure, in order to create the different mazes used during the experiment. The data set obtained from the (Senzai & Buzsáki, 2017) paper will not be used in this research for differentiating between mossy and granule cells, but instead for the purposes described above and further detailed in the Methods section. Nonetheless, this data set is compatible with the aforementioned research objective.

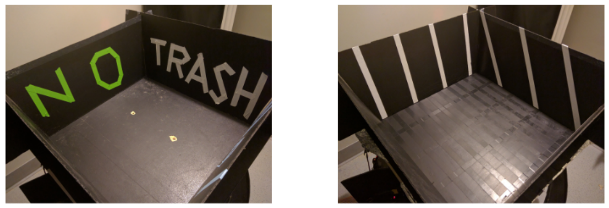


Figure 1: The environments in which the mazes were located during the experiment. (Senzai & Buzsáki, 2017)

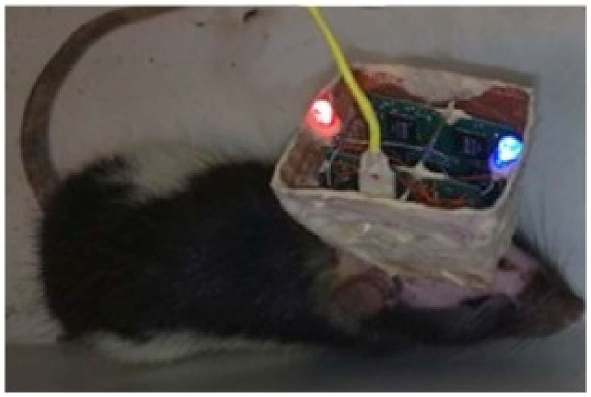


Figure 2: Reference example surgery taken from reference from source paper. (Berényi et al., 2014)

## 2 Data

### 2.1 Source

The data set used for our project was gathered by Senzai and Buzsáki. The data was obtained from `https://gui.dandiarchive.org/#/dandiset/000003`. The source has 2.6 terabytes of data available. This data consists of both position data for both light-emitting diodes (LEDs), in the form of (x,y) coordinates, and the electrophyiological data from the array of electrodes inside the mouse's hippocampus.

Since using this much data requires more computing power than we have available, we chose to only use the data of one mouse, specifically YutaMouse23. We used the recordings of trials 4 and 5. This means that the model's success largely depends on the quality of the recordings and whether or not they actually record place cell activity.

The data set is encoded as a .nwb file. This is a file type developed by Neuroscience Without Borders, made for use with Python. It essentially functions like a large dictionary (more information can be found at `https://www.nwb.org/about-nwb/`). We developed a Python script to navigate through these files and save relevant data.

From this we chose to save the data containing electrophysiology recordings as well as positional data.

## 2.2 Visualisation

The source data set can be split into two categories: positional data and electrophysiological data. The positional data of the mouse is given on a 2D plane with x and y values in centimeters. Two LEDs are placed 2.5cm rostral and caudal of the mouse's brain (see Figure 2). The mouse's position was recorded based on the position of these LEDs on the mouse's head. Figure 3 is a visual representation of a path that the mouse took in a certain period of time.

The electrophysiological data consists of 65 recordings collected by the electrodes implanted in the mouse. These electrodes recorded electric potentials in the mouse's brain over time. Figure 4 shows ten of these recordings. From this figure, we can see that individual spikes are sufficiently visible. The data collected is in $\mu V$, we see that spikes reach about $25000\mu V = 25mV$ indicating a clear and distinct spike over roughly $0.008s = 8ms$ (10 samples), this is within an order of magnitude of the generally cited duration and strength of action potentials (*Action potential*, 2021).

## 2.3 Data selection

The source data varies greatly from mouse to mouse and trial to trial. This is because the positions of the electrodes was variable. This is not an issue, as the data collected in one trial is quite extensive. We selected data from YutaMouse23 Trial 5, as this trial provided us with the longest sequence of continuous and uninterrupted data. The way this sequence was found will be discussed in more detail in Section 3.1

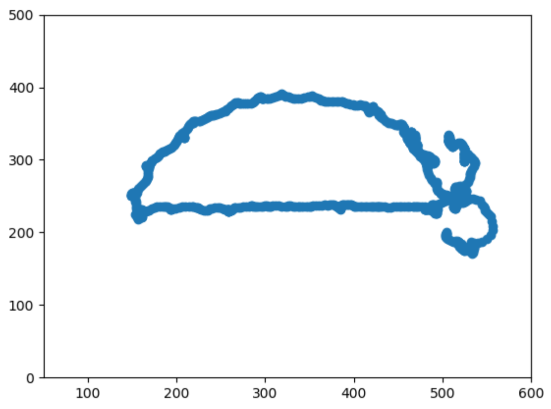


Figure 3: Representation of position data from one position sensor, samples 1000 to 2000. This data is not actually used to train or test the model, but comes from the same mouse and same trial.

# 3 Methods

## 3.1 Data preprocessing

### 3.1.1 Electrophysiology data

The recording frequency of the electrophysiology data and the position data in the source data set is not the same. This was an issue, as the two types of data would need to have the same recording frequency for it to be usable in an MLP. Naively taking the mean of each electrode would not lead to favourable results as it drastically reduces the amount of data that the MLP can be trained on. It also ignores all types of spectral components, specifically neuron spiking frequencies. For these reasons, we opted to use a Fast Fourier transform (FFT). Since we had limited computing power available (specifically, our RAM was limited to 16GB), it was not possible for all of the data to be used. Therefore, we limited ourselves to only the first 12 frequency bins (roughly 0Hz - 212Hz). The FFT is calculated using the interval

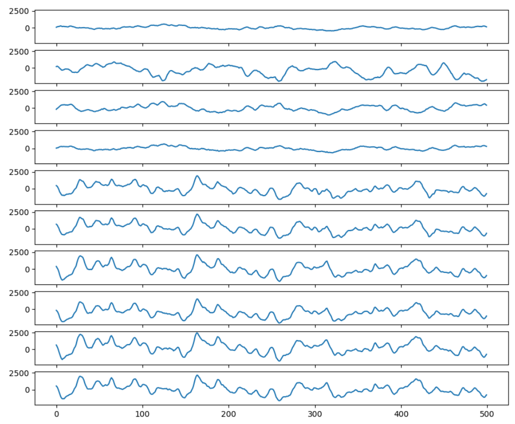


Figure 4: Representation of 0.4 seconds of electrophysiology recordings of 10 electrodes. There are a number of very obvious spikes, indicating that we should be able to properly record individual spikes.

from the last position data point to the current position data point. This results in 12 points for each electrode, of which there are either 64 or 65 (see Figure 6). In order to have an acceptable frequency bin resolution (the difference in frequency between the bins), more data was needed, namely 65 data points for each electrode. We could then start dealing with null values. The data set had a lot of points where the position of the mouse was missing, therefore we chose the largest subset of the data where there was little to no interruption, meaning there were little to no null values in that subset. This gave us a data set with roughly 270,000 points - which is still an enormous amount of data - to use for training the model. This data set represents an electrophysiology recording that has a duration of roughly 3 hours and 48 minutes.

In order to allow the MLP to view electrophysiology data from the past, the data set needed to be extended using chunking. This was done by adding the current data point and the two preceding data points in one chunk. Therefore, some of the points needed to be copied in order to get a chunk for every single point, except for the first and second data point, as those do not have enough preceding data points.

We chose a hindsight of 2 mostly because of constraints in our resources. This means that the model had $65 * 12 * 3 = 2340$ natural number values to interpret for each chunk (see Figure 5). With the chosen hindsight, the size of the data set increases by a factor of 3. From now on we will be referring to each of those chunks as one data point. This means that our total data set included around $270,000 * 2340 = 631,800,000$ values resulting in about 4.7 GiB of data. A copy of the data set is kept when splitting using k-fold. This way, we have access to both the data set, and the testing and training data. The resulting data set is 9.4 GiB and contains FFT data points based on electrophysiology data. It does not yet including the actual mouse position coordinate data points.

### 3.1.2 Positional data

The position data was contained in two separate files, one for the positions of each of the LEDs mounted on the mouses head. We combined the two files by averaging the points at any given time, so as to get the position of the center of the mouse's head. Half of the points were discarded due to down-sampling from 39 Hz to 19.5 Hz, which was done to make sure that the positional data has the same sampling frequency as electrophysiological data. This was done by discarding all even-numbered data points.

The resulting positional data set has a size of roughly 4 MiB. Each data point contains only 2 values, the x-coordinate and the y-coordinate. The positional data set has exactly the same amount of points as the electrophysiology data set, meaning there is a 1 to 1 correspondence between the two.

## 3.2 MLP architecture

For this task, we used the Keras package, a subpackage of Tensorflow. This allows us to "easily" take advantage of the parallel nature of our GPU while also keeping the code simple.

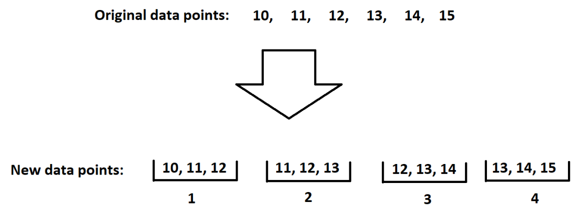


Figure 5: A simple diagram to illustrate how our data points are transformed to include the hindsight.

The final MLP had an architecture consisting of 1 input layer (which was "flattened"), 2 hidden (dense) layers (25 and 10 neurons respectively), and finally an output layer with only 2 neurons (see Figure 7). The following subsections briefly explain the mathematical background of our model and the hyperparameters that are used.

### 3.2.1 Mathematical structure

The output layer uses the identity function as an activation function, the hidden layers use the hyperbolic tangent. Both hidden layers have regularizers with the same value for $\alpha = \sqrt{5 * 10^{-7}}$. We chose to use the euclidean distance as a loss function, simply because this is what we aim to reduce and it is easily interpretable. Despite the downside of it only punishing linearly, we chose to stick with it since it is more transparent and requires less guesswork. Our exact loss function is:

$$L(h(u_i), y_i) = ||h(u_i) - y_i||$$

Where $L$ is our loss function, $h(u_i)$ is the coordinate output of the MLP, and $y_i$ is the true position of the mouse. This is also the square root of the mean square error loss function. We use the L1 regulariser, with the aforementioned $\alpha$ value of $5 * 10^{-7}$. The optimisation problem could then be defined as follows:

$$h_{opt} = \underset{h \in \mathcal{H}}{argmin} \frac{1}{N} \sum_{i=1}^{N} L(h(u_i), y_i) + \alpha^2 \sum_{\omega \in \theta_h} |\omega|$$

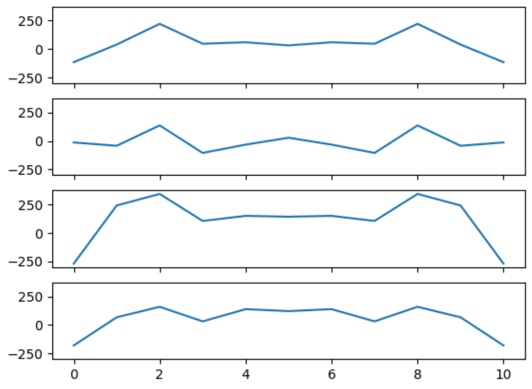
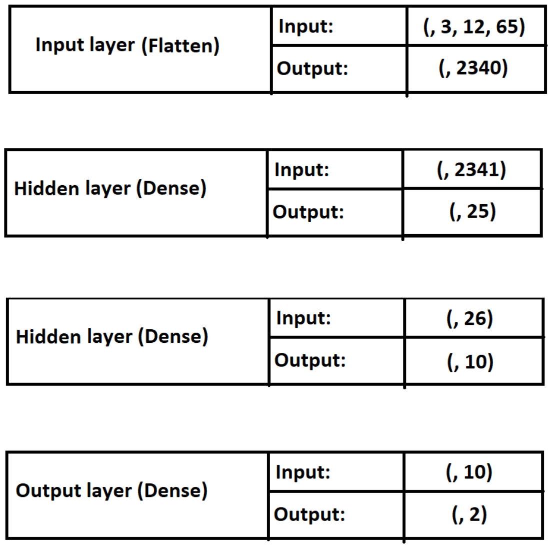


Figure 6: Example of data recorded by 4 electrodes after the FFT has been calculated. This plot shows the frequency bin on the x-axis and the magnitude of frequency on the y-axis. The DC component has been removed for clarity.

Where $h_{opt}$ is the optimal model, $h$ is a model in the model space $\mathcal{H}$, $N$ the total amount of data points, $L(h(u_i), y_i)$ is the loss function described above, $\alpha^2$ is the regularization factor, $\omega$ a weight between two neurons, and $\theta_h$ is the collection of parameter values in the candidate model $h$.

### 3.2.2 Hyperparameters

**Solver:** We chose to use the built-in SGD solver from Keras. The SGD solver makes use of a regular gradient descent to make sure the model converges towards a local minimum. We could have also chosen the ADAM solver, even though our rudimentary testing would slightly indicate otherwise, but the only real advantage it would have given us is that it increases the speed of convergence. We went with the SGD solver because we found it better understandable and easier to work with. With the SGD solver, our model still converges to a local minimum within roughly 3 epochs and after that the model starts overfitting. This overfitting would also have happened if we had used ADAM instead. Using ADAM, therefore, seemed to have no real advantage to us.



| | | |
|---|---|---|
| Input layer (Flatten) | Input: | (, 3, 12, 65) |
| | Output: | (, 2340) |

| | | |
|---|---|---|
| Hidden layer (Dense) | Input: | (, 2341) |
| | Output: | (, 25) |

| | | |
|---|---|---|
| Hidden layer (Dense) | Input: | (, 26) |
| | Output: | (, 10) |

| | | |
|---|---|---|
| Output layer (Dense) | Input: | (, 10) |
| | Output: | (, 2) |

Figure 7: Visualization of how our MLP was organized. The rightmost column indicates the shape of the data.

**Activation:** Both of our hidden layers use the hyperbolic tangent as an activation function. $f(x) = tanh(x)$
**Hidden layers:** Our neural network contains two hidden layers (see Figure 7), the first contains 26 neurons and the second contains 10 neurons. They both make use of bias, which is why their inputs are 1 higher than the output of the previous layer. Both hidden layers include their weights as part of the regulariser. We tried different sizes of models and different numbers of layers, this combination seemed to be the best if we stayed true to Occam's razor.
**Alpha:** The alpha parameter is a value whose square ($\alpha^2$) is a product of all the other components of the regulariser. This $\alpha^2$ is sometimes also referred to as $\lambda$. We chose an $\alpha$ value of $\sqrt{5 * 10^{-7}}$, in other words $\lambda = 5 * 10^{-7}$. Higher or lower alpha values seemed to affect the behaviour of our model negatively.
**Epochs:** We chose to have only 3 epochs, our model seems to converge almost immediately, so more was not necessary.

**Random state:** We chose to stay classic by using the glorot uniform initialiser. This initializes the weights randomly from a uniform distribution between $limit$ and $-limit$. Where

$$limit = \sqrt{\frac{6}{no.\ input\ weights + no.\ output\ weights}}$$

**Learning rate:** The learning rate did not remain constant throughout all the epochs. The first epoch the learning rate was 1.0, which is quite large. We did some tests and it turned out that it doesn't have much benefit to have a lower learning rate at first for our model. The second and third epochs were trained with a learning rate of 0.1, allowing the model to improve without directly overfitting.

## 3.3 Evaluation of performance

### 3.3.1 Linear model

It is quite hard to measure how well our model did without knowing the difficulty of the task. We trained a linear regression to be our baseline "naive" approach which allowed us to Figure out a rough idea of the complexity of the task as well as justify the use of a neural network for this. In order to make our model as naive as possible we did not "chunk" the data like we did for the neural network (meaning no hindsight was used here). We used the sci-kit learn package in python (Pedregosa et al., 2011) in order to train our linear regression. It was allowed to calculate the intercept of the line as our data is not centered and it was not normalized.

### 3.3.2 Educated guess

Even with the linear model as a baseline we needed to make sure our actual model was better than just guessing. In order to do this we calculated an educated guess by taking the mean position of all the points:

$$m(y) = \frac{1}{N}\sum_{i=1}^{N} y_i$$

Where $m(y)$ is the mean of the points, $y$ is the set of training data positions which are also given to the

MLP and the linear regression, $N$ is the number of data points in the training sample, and $y_i$ is the point $[x, y]$ for index $i$. This shows what the model would output if it chose to completely ignore the electrophysiology data (i.e. guessing).

#### 3.3.3 k-fold cross-validation

In order to validate our model we used k-fold cross validation. This not only allowed us to more reliably fine-tune parameters, but also to make sure that the model doesn't overfit too much. There are a couple problems with k-fold validation and the type of data that we are dealing with, but first we shall go over how we implement k-fold validation.

The k-fold cross validation consisted of dividing the data into 9 equal parts without shuffling the data first. One of these parts was taken as testing data and the model was trained on the rest, then the next part is taken as testing and training happens on the rest. This continues until all parts have been used as testing data.

## 4 Results

As was explained in Section 3.3.3, our model was validated using k-fold cross validation consisting of 9 folds in total.

Figure 8 shows that for the first fold (fold 0), both the FNN and the linear model perform worse than the educated guess. This is due to drastic overfitting. However, our FNN model outperforms the linear model for folds 1-6. This indicates that it is not just 'guessing'. The most successful fold is fold 4, where the FNN had a loss of approximately 1.5 cm, meaning we were only 1.5 cm off from the correct location of the mouse. For the same fold, the linear model had a loss of approximately 4.2 cm, and the educated guess had a loss of 8.5 cm.

For folds 7 and 8 our FNN model's loss is higher than the educated guessing and our linear base model. This is still somewhat justifiable, if we take a look at Figure 10 we can see that the confidence interval is quite large for fold 7. This means that sometimes the model is quite accurate and other times it is way off. This seems to support the idea that some locations will be impossible for the model to deduce from the data. A likely cause for this is that not all place cells will be present.

In fold 8, both the linear model's loss and our FNN model's loss are drastically higher than the educated guessing loss with a value of approximately 16.8 and 18.6 cm respectively. A possible reason for this is mentioned in 5.3, where we mention the influence of parts of our data set not being equal in terms of visited areas.

Figure 9 shows the testing and training coordinate points of the mouse in its cage. It is very clear that whenever the loss of our FNN model is high, the testing points are very widely distributed (see data for folds 1 and 9). When the loss is lower, however, we can clearly see that the total set of testing points is a small subset of the total training points set (i.e. the mouse does not move much during the time frame used for collecting these testing points; meaning the data points are very close together and form a 'compressed' area for testing. This makes the mouse's location easier to predict; see folds 3 and 4).

We chose to visualise only these 4 folds because they can be utilised very well to show the relation between the spread of the testing data and the height of the loss.

## 5 Discussion

### 5.1 Data processing

Perhaps the most important part of data processing that we did was the easiest, just selecting the data which was complete (no missing values) was quite important. At first we attempted to take the whole data set, remove all the missing values, and use it as is. This meant the mice would sometimes go missing for hundreds, if not thousands of data-points, and then reappear possibly in a different location. This essentially means that during that time anything could have happened to the mouse, including human intervention. This also means that the perception of space of the mouse could be completely different causing the data set to essentially be completely different af-

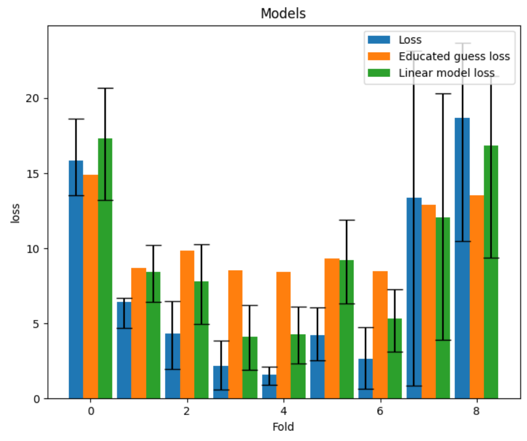


Figure 8: 9-fold cross validation results. Testing loss in cm plotted per fold. The error bars represent the 68.2% confidence interval (roughly 1 standard deviation)

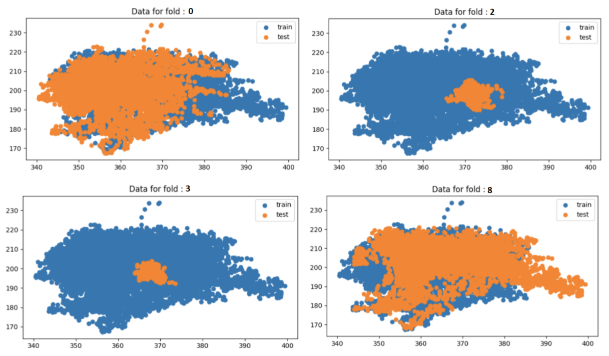


Figure 9: testing and training data represented per fold. from left to right, top to bottom: folds 0, 2, 3, and 8. axes are in cm. (only 4 out of 9 folds taken for clarity

ter a certain point, which obviously is not ideal for a static FNN.

At the start we used the mean of the electrophysiological data in between each point, which meant a strong dimensional reduction, reducing the amount of data-points from the electrophysiology by 32 and removing all spectral components of the data. This seemed like an O.K. solution at the start, we had reached loss values which were sub 20cm for the first time, which looked promising. However, after realizing that we should compare this to chance, and creating the educated guess mechanism, we realized that our model was pretty terrible. Taking the FFT with only 32-33 samples did not help, we only saw improvement past the educated guessing when we halved the sampling frequency and took 65 data-points instead.

## 5.2 Model Hyper-parameters

Working on the hyper-parameters for the model was quite time consuming, for our poor GPU at least.

**Network size**: We initially tried a simple network with one layer that contained 25 neurons. This already performed quite decently. However, we found that adding another layer of 10 improved performance significantly. Further changes to the network's topology did not impact performance in a meaningful way.

**Regularization rate**: We found that $\alpha = \sqrt{5 * 10^{-7}}$ ($\lambda = 5 * 10^{-7}$) was a good balance between reducing overfitting and preserving our good results, such that the folds that previously performed worse than the educated guessing had a lower loss, and without increasing the loss of the folds that previously performed better than the educated guessing. Increasing this value reduced overfitting, but also greatly reduced the performance of folds that performed better than the educated guessing.

**Epochs**: We found that 3 epochs was enough to make the loss values plateau. Past this point, the model would frequently just start overfitting. Before this point, the model would not have the time to reach an optimal solution.

**Learning rate**: As discussed in Section 3.2.2, the learning rate was split across epochs. We found that using this rudimentary method was more intuitive than using exponential decay. A learning rate of 1.0 at the start (for only 1 epoch) was ideal for the model would rapidly converge to a somewhat decent model. However, the model would overshoot if this learning rate was used for the following epochs. Therefore, the next two epochs had a learning rate of 0.1. A lower learning rate would yield similar results, but would be more prone to overfitting.

### 5.3 Cross validation

There are multiple problems with how cross-validation is done, mainly because the mouse is a living organism and changes its internal state over time, whereas we do not do online learning. This means that by testing on the later or earlier recorded data, we are very likely to find data from before or after the mouse has "re-learnt" its environment, which can cause the data to be a bit distorted. This is especially evident at the start, which is further discussed in Section 4. Another issue is that some parts of the maze are visited less frequently than others. This can lead to multiple problems with prediction. For example, when the mouse goes to areas which are sparse in the training data but appear frequently in testing data, the model usually does not predict these positions accurately. This is because it is almost impossible for the model to learn to accurately predict data points that rarely appear in the training data (see Figure 10).

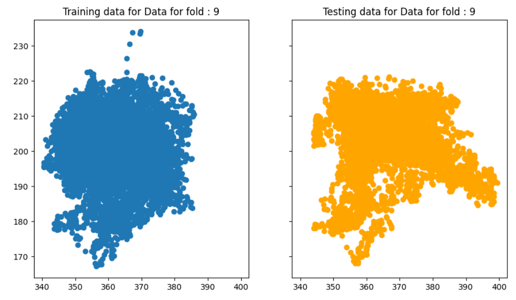


Figure 10: Training and testing position data for the 9th fold of the k-fold algorithm. We can see that the testing data clearly goes over areas that the training data does not go over (right side).

### 5.4 Future improvements

There are multiple future improvements that we could do, the first improvement that we may want to try is to use a principal component analysis (PCA) in order to reduce the dimensionality of the data, this would allow us to work with more relevant data and less noise, which would stop the model from overfitting as well as allow our current hardware to do more.

Another point of improvement that we may want to consider is using a RNN though this would be more complicated to work with. The reason for using a RNN is that these are more appropriate for time dependent input of which the model should "remember" certain information. However the improvement we would see based on this idea may not be significant or even improvement at all, it may not be necessary to "remember" anything.

The last improvement that we will discuss is that of online learning. Online learning would allow the neural network to adapt to the mouse's mental changes, if this was done every so often then perhaps we would get better performance in the folds where the mouse leaves the area that is densest with data. However this may be somewhat unrealistic as it would essentially mean that we know where the mouse is which would defeat the purpose of such an application.

### 5.5 What we learned

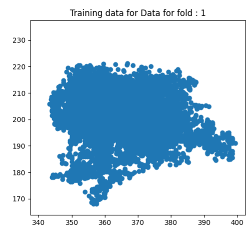
The majority of the workload consisted of preprocessing and cleaning up the data. This is therefore the topic we learnt most about during this project; we understood what works and what doesn't for feeding data into a NN (as discussed in the Data and detailed in the Discussion section, where we mention initially removing missing values), we also better understand the value of actually doing a PCA as opposed to what we have done now. Another area of interest where our experience grew over the course of this project was using Keras to create a FNN, in particular looking into many different aspects of the NN such as the optimizers, regularization, and learning rate.

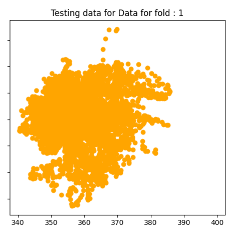
## 6 Appendix

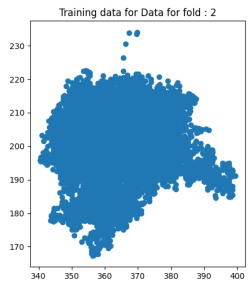
### 6.1 GitHub Repository

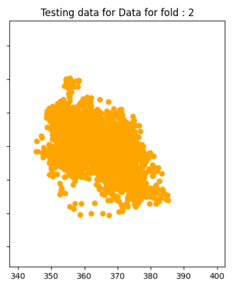
All of the code for this project can be found in the GitHub repo: `https://github.com/OThioux/NN_project_mice_maze`

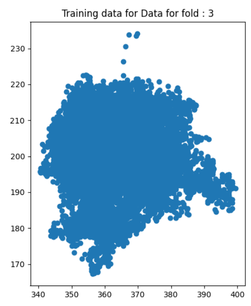
### 6.2 Additional Graphs

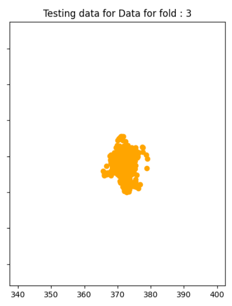


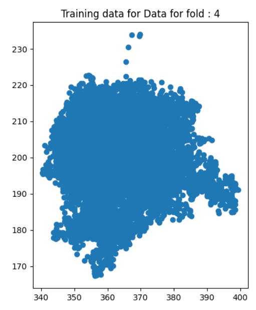


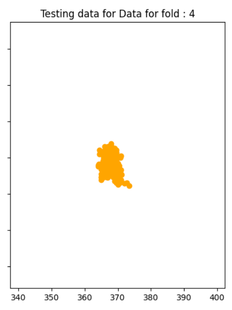


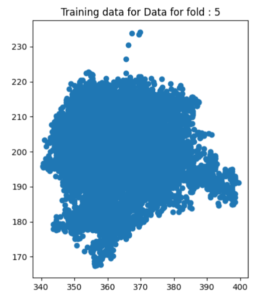


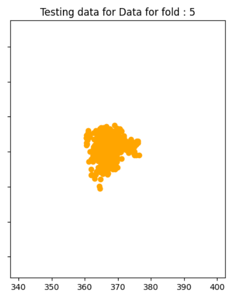


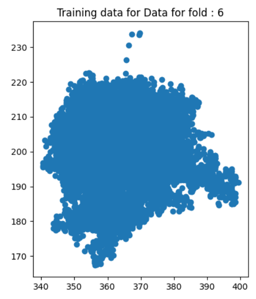


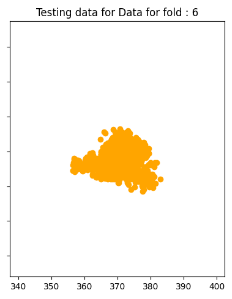


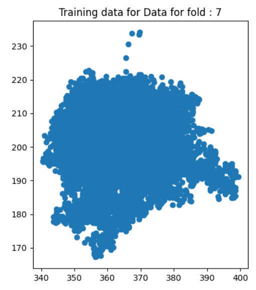


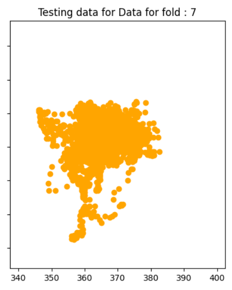


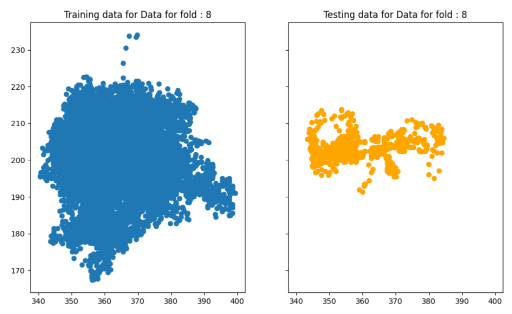


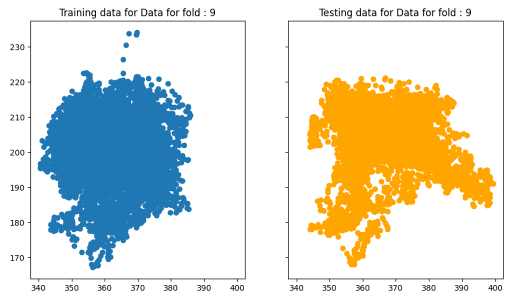












Additional plots for all the folds, note that the folds are numbered 1-9 and not 0-8.

## References

*Action potential.* (2021, Jun). Wikimedia Foundation. Retrieved from `https://en.wikipedia.org/wiki/Action_potential`

Berényi, A., Somogyvári, Z., Nagy, A. J., Roux, L., Long, J. D., Fujisawa, S., … Buzsáki, G. (2014, March). Large-scale, high-density (up to 512 channels) recording of local circuits in behaving animals. *Journal of neurophysiology*, *111*(5), 1132—1149. Retrieved from `https://europepmc.org/articles/PMC3949233` doi: 10.1152/jn.00785.2013

Muller, R. (1996). A quarter of a century of place cells. *Neuron*, *17*, 979–990.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Senzai, Y., & Buzsáki, G. (2017). Physiological properties and behavioral correlates of hippocampal granule cells and mossy cells. *Neuron*, *93*(3), 691–704.