Project 3 - FYS-STK4155 - Looking for Grid Cells & Place Cells in Recurrent Neural Networks

Markus Borud Pettersen

December 17, 2020

Abstract

By training recurrent neural networks to perform a navigation task, the formation of grid cell- and place cell- like responses was studied. A total of three RNNs were studied, and the baseline model achieved a test mean absolute navigation error of 0.028 for short paths (100 timesteps) and 0.12 for longer paths (1000 timesteps). The best model was found to be an identity initialized RNN, achieving a mean absolute error of 0.016 for short paths, and 0.040 for long ones. An additional RNN was trained with radial basis function inputs, but this model performed slightly worse than the identity network, and was deemed unnecessarily complicated. Finally, it was found that the identity network developed place-cell like output responses during navigation, but no evidence was found of any grid-cell-like activation. Preliminary findings suggest that the place cell centers of the identity network were the same, regardless of the path involved.

Introduction

In 2014, the Nobel prize was awarded to May-Britt and Edvard Moser alongside John O'Keefe, for their role in discovering so-called grid cells, a specialized type of neuron, which is thought to be essential to our ability to navigate our environment [1]. What makes grid cells particularly interesting is the fact that they are only active, or *fire* at certain locations in space, and that these *firing fields* map out a hexagonal grid in space. Furthermore, different grid cells can have slightly offset and/or scaled grids, in such a way that an ensemble of grid cells completely cover or tile the space in which the animal moves.

It is perhaps equally interesting that similar responses were recently found in the hidden layer activations of recurrent neural networks (RNNs) trained on a simple navigation task [2]. In fact, the navigation task consisted of performing so-called path integration; the network received information about its velocity (speed and heading), and was tasked with predicting its position. Since neural networks are directly inspired by biological systems, it seems reasonable that the artificial grid cells found in RNNs are related to grid cells in actual animals, and that one could study the emergent structures in navigating RNNs to learn more about how our brain performs navigation.

What makes these results difficult to interpret, is the fact that [2] used the somewhat complicated long short term memory (LSTM) architecture. However, [3] trained a regular vanilla RNN to do the same task and also found grid-cell-like responses. They, on the other hand, trained with so-called *place cell*-like labels. Place cells are another specialized neuron which only fire at a single region of space. A possible problem with the approach in [3] was that the place cells were randomly and uniformly distributed across the environment with firing fields specified by the authors, and not an emergent feature of the network. Inspired by both of these findings, I want to explore the possibility of training a simple recurrent neural network to perform path integration, while making as few assumptions about the network's place cells as possible. More specifically, I want to see if I can train the network to navigate, under the assumption that it develops place-cell like outputs, and then convert said representation to Cartesian coordinates, and use these coordinates to perform supervised learning.

As the RNN considered in this project seems to behave similarly to its biological counterpart, I also want to consider making the model slightly more realistic, by switching from a velocity input signal, to speed and head direction input *cells*. In the brain, such cells are only responsive for certain input values. In order to emulate this, I will apply so-called radial basis functions (RBFs) to the inputs of the network.

To begin the project, a brief overview of place cells, grid cells and their relationship to each other is given in the context of navigation. Following this, I attempt to outline a possible procedure for decoding the predicted position of a navigating RNN, under the assumption that it has developed a place-cell like representation of space. Finally, I give a short introduction of radial basis functions.

Background

Grid Cells, Place Cells, & Navigation

As mentioned, grid cells are thought to be a critical part of how animals perform navigation. But, there are a host of other cells involved in this process, including so-called place cells, border cells, speed cells and head direction cells [2]. As the name suggests, a place cell is a certain type of neuron which only fires at a certain place in space. Unlike grid cells, however, a place cell is not periodic, and typically only fires in a single small region. As such, the firing rate of a place cell can be modelled using a two-dimensional Gaussian

$$f_{pc}(\mathbf{r}) = Ae^{-(\mathbf{r}-\boldsymbol{\mu})^2/(2\sigma^2)}, \tag{1}$$

where A is some amplitude, $\mathbf{r}$ is the position of the navigator, $\boldsymbol{\mu}$ is the center of the place cell, i.e. the location at which it is maximally active, while σ determines the width of the place cell.

Fig. 1 shows an illustrated place cell modelled according to (1), with $A = 1$, and $\boldsymbol{\mu} = (0.8, 0.4)$, and $\sigma = 0.05$, for input coordinate vectors $\mathbf{r} = (x, y)$, with $x, y \in [0, 1]$. The figure also clearly shows how a place cell gets its name; it is almost completely inactive outside of a small region around $\boldsymbol{\mu}$. Intuitively, a place cell coincides with our sense of location: most people do not think of their position in terms of an absolute coordinate, but rather in terms of being "here" or "there".

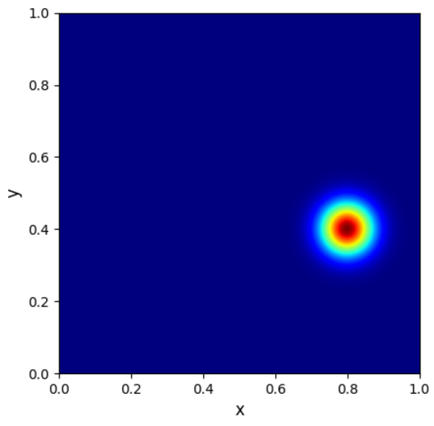

Figure 1: Two-dimensional Gaussian function, centered at $x = 0.8, y = 0.4$, illustrating the response of an idealized place cell in a square chamber.

In the same vein, Fig. 2 shows an illustration of a grid cells in the same region as the place cell. One can notice that the firing rate is maximal at points lying on a hexagonal grid

Interestingly, a grid cell can be modelled as a sum of three plane waves, as long as their wave vectors have a phase offset of $\pi/3$. For the example in Fig.2, the firing rate was calculated as

$$f_{gc}(\mathbf{r}) = \sum_{i=1}^{3} \cos(25\mathbf{k}_i \cdot \mathbf{r}),$$

where the wave vector $\mathbf{k}$ was taken to be a two-dimensional unit vector on the unit circle, with each vector $\mathbf{k}$ rotated 60 degrees relative to the next, i.e. $\mathbf{k}_1 = (\cos\pi/3, \sin\pi/3)$, $\mathbf{k}_2 = (\cos 2\pi/3, \sin 2\pi/3)$, and so on.

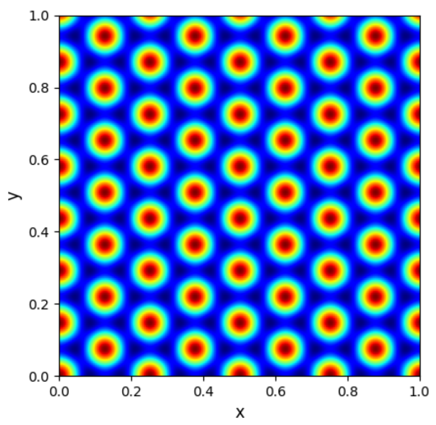

Figure 2: Interference pattern for a sum of three plane waves with wave vectors offset by 60 degrees relative to each other, illustrating the firing pattern of an idealized grid cell in a square chamber.

The next central question is of course how grid and place cells allow us to actually navigate: As mentioned, place cells allow for a direct representation of space, as the act of a place cell firing, indicates that the animal is in the location corresponding to the center of that place cell. Grid cells, on the other hand are not as readily interpreted. As stated, the firing fields, i.e. locations where a grid cell is active, spans across the environment in a periodic fashion (as seen in the example of Fig 2). Furthermore, different grid cells may have slightly different firing fields, as a given firing field may be rotated, shifted or scaled relative to another. All this means that a set of grid cells completely cover the environment in which the agent navigates.

As for actually mapping space, grid cells, through input from speed and head direction cells, perform path integration (i.e. they receive information about their current speed and heading, and from this, update their representation of position). Intriguingly, this process is not directly dependent on sensory inputs such as vision, which one might be able to appreciate: If left in a dark room, one could keep track of ones position relative to the starting point, by keeping track of your speed and heading at all times.

From this very brief introduction, we can gather that grid cells completely map space, and perform path integration. As to how they are key to our able to determine our position, it is thought that grid cells also code for place cells. Using a simple model, we can model this relationship as

$$f_{pc}^i = \sigma\left(b_i + \sum_j W_{ij} f_{gc}^j\right), \tag{2}$$

where f_{pc}^i denotes the firing rate (or activation) of place cell number i, while b_i is the firing threshold (or bias) of said place cell, f_{gc}^j the firing rate of grid cell number j, while W_{ij} is the weight relating grid cell j to place cell i. σ is some nonlinear activation function. Said differently, a place cell is perhaps nothing but a non-linear combination of grid cells.

Recurrent Neural Networks

In order to path integrate, one needs to keep track of one's own position, and possibly speed and heading over time. In order to capture this time-dependence, a recurrent neural network may be used. At its heart, a recurrent neural network is nothing more than a simple difference equation, but as with its deceptively simple relative, the standard feed forward neural network (FFNN), RNNs are capable of solving highly non-trivial tasks, such as natural language processing [4].

Considering the most basic, or vanilla RNN, its governing equation can be written as

$$\begin{aligned}\mathbf{u}_t &= W_R\mathbf{h}_{t-1} + W_I\mathbf{v}_t + \mathbf{b}\\ \mathbf{h}_t &= \sigma(\mathbf{u}_t),\end{aligned} \tag{3}$$

where W_R is a matrix containing the *recurrent* weights, W_I a matrix of input weights, while $\mathbf{b}$ is a vector of biases. $\mathbf{u}_t$ is the hidden state activities at timestep t, while $\mathbf{h_t}$ is the hidden state activations at the same step. Here, $\mathbf{v}_t$ is a vector of inputs at at time t. What makes RNNs slightly different from the related FFNNs, is the fact that the hidden state at time t depends on its own state value at the previous timestep, h_{t-1}. This, coupled with the fact that the RNN receives a new input at each step, is what enables it to solve time-series problems.

In order to train RNNs in a supervised manner, one makes use of its similarity to regular FFNNs, by *unrolling* the network in time, and considering each timestep as a layer in a feed forward network. On the other hand, the weights between these "layers" are always given by the recurrent weight matrix, which complicates gradient calculations (as the derivative with respect to a weight depends on the activations at every timestep). Even so, training is still reminiscent to that of a FFNN, and one can use gradient descent and a version of backpropagation called backpropagation through time (BPTT) to train RNNs. BPTT is in essence also similar to the regular backpropagation algorithm, an outline of which can be found in [5].

As RNNs can be viewed as slightly modified, very deep feed forward neural networks, and are trained in almost the same way using stochastic gradient descent, it should be no surprise that RNNs tend to suffer from two major problems often associated with deep neural networks: exploding, and vanishing gradients. As the name suggests, exploding gradients occur when gradients become very large, and possibly overflow, causing training and gradient descent to diverge. The opposite problem occurs when gradients become very small, and vanish. To see explicitly why this can easily occur in gradient calculations during BPTT, see for example [6]. For our purposes, the important takeaway is that in order to investigate navigation over longer timescales, i.e. long paths, an RNN architecture capable of overcoming the vanishing and exploding gradients issue is needed. One way of achieving this, is by using the more complicated LSTM architecture mentioned in the introduction, which was specially designed to do just this. However, LSTMs are more complicated than vanilla RNNs, and therefore harder to interpret.

Another possible way of addressing the problem of vanishing gradients in vanilla RNNs, is to use so-called identity RNN (IRNN) initialization, as proposed by [7]. The authors show empirically that by initializing the recurrent weight matrix to the identity, the biases to zero, and using the rectified linear unit (ReLU) as the recurrent activation function, a simple RNN can learn very long-term dependencies, and even outperform more complicated architectures such as LSTMs for certain tasks [7]. Note that the authors initialized the input weight matrix according to a normal distribution $\mathcal{N}(0, 0.001)$.

Radial Basis Functions

A radial basis function is, put simply, some function where the output somehow depends on the *distance* to some point, or center, in its domain. In keeping with the scope of this text, only real-valued functions, alongside the Euclidean distance is considered. Before asking which point distance should be measured relative to, it is worth considering that we have already seen an example of a radial basis function, in the form of the two-dimensional Gaussian used in (1), where the center of the radial basis function was the actual center of the place cell.

The reason this is interesting, is that radial basis functions not only allow for a natural way of encoding space, but also other variables, such as speed and head direction. To investigate whether adding more biologically plausible features to the navigating RNN improves performance, it would therefore be of interest to use radial basis functions to mimic speed and head direction cells, i.e. specialized neurons which are only responsive to particular input ranges. To model this in a simple manner, one can imagine that instead of having a single input neuron coding for speed, let's say, one has many neurons, each reactive to a certain range of speed values.

To see how this might be implemented, consider the illustration in Fig. 3, where a simple, linearly increasing speed signal $v(t) = 5t$, has been feed into four different simulated speed cells. For this illustration, each speed cell is taken to be an unnormalized Gaussian with a standard deviation of 0.1. The four different "cells" have centers at, or are most responsive to, a velocity of 0, 0.5, 2.5, and 4, respectively.

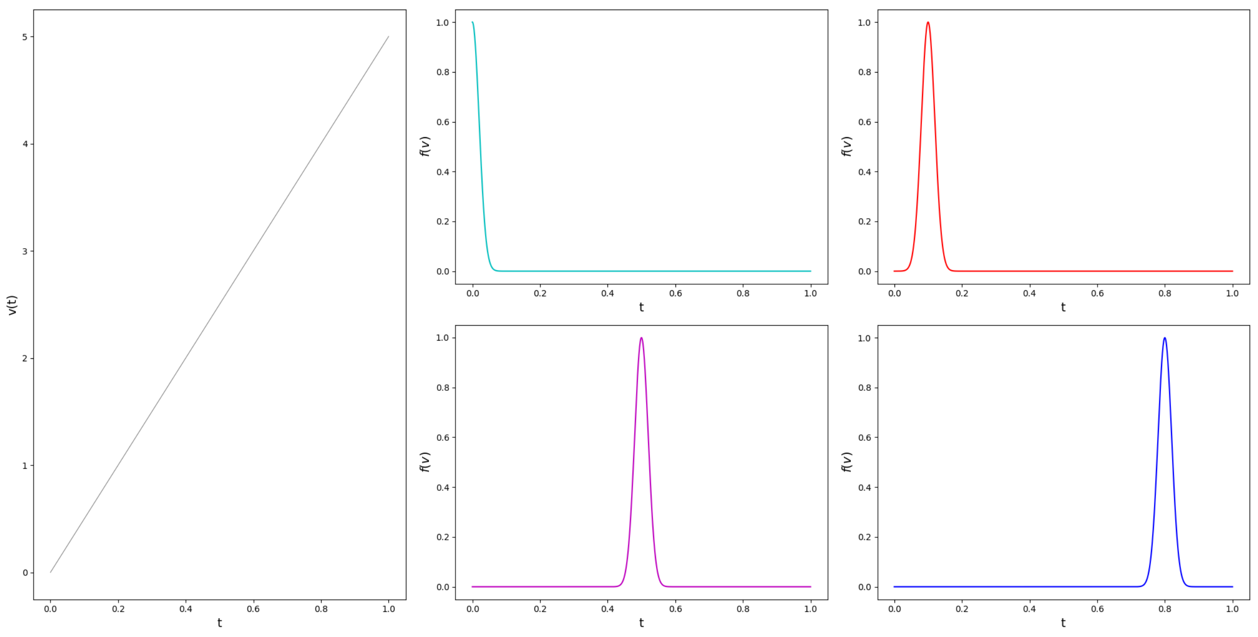

Figure 3: Response of Gaussian speed cells for a linear input signal $v(t) = 5t$. Each cell has a standard deviation of 0.1, and the cell centers correspond to $v = 0$, 0.5, 2.5, and 4, respectively.

As one can easily judge from Fig. 3, the simulated speed cells only react to certain input values, and are almost completely inactive for values far from their center. Based on the set of four speed cells in the figure, one can also tell that there is a clear loss of information in the representation of $v(t)$, as there are some swaths of the signal's range which are not covered by any speed cell. As such, one would either need more speed cells, or possibly wider radial basis functions to have a useful basis set.

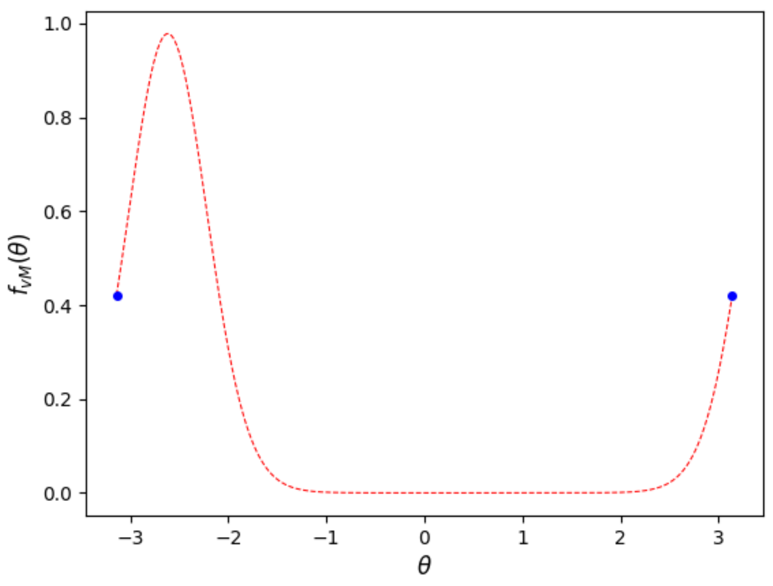

Figure 4: Response of a simulated head direction cell, for input head direction angles in the interval $[-\pi, \pi]$. The shown response is in actuality the probability density of a Von Mises distribution on the same interval.

Since speed is readily converted into a radial basis function representation, one may wonder if the same can be done for the slightly more exotic head direction. Since head direction is measured in terms of radians (angles), it is 2π-periodic, which in essence means that 2π is the same head direction as 0. This is however no obstacle, as there are probability distributions which also have this property. The simplest of which is probably the Von Mises distribution, which can be thought of as a normal distribution, but on a (one-dimensional) ring. An example of a Von Mises distribution is shown in Fig. 4, where the correspondence to a normal distribution is hopefully obvious, as is the periodicity (indicated by blue markers). For the distribution in the figure, the centre of the distribution is at $\theta = -5/6\pi$, and the width parameter, often denoted κ was taken to be 2π, which is roughly equivalent to a Gaussian with $\sigma = 1/\sqrt{2\pi}$, due to the fact that $\sigma^2 \approx 1/\kappa$.

In the same way as with the speed "cells", one can therefore mimic the behaviour of head direction cells by creating an ensemble of differently centered Von Mises distributions (corresponding to the angle to which the cell responds maximally), and pass the head direction signal through said ensemble before feeding it to the RNN.

As a small side note, a radial basis function representation of a continuous variable such as speed does come at the cost of being very high-dimensional; we require many nodes to encode a single quantity, and there is also some redundancy, in the sense that cells may overlap, and therefore encode the same information. However, this might also be a useful feature in biological systems as it may help to guard against noise, as multiple cells hold a "piece" of the information being processed.

Decoding Place Cell Centers

As described in the introduction, the goal of this project is to train an RNN to perform path integration, and investigate whether both grid- and place cells can emerge as a solution to the navigation problem. To do this, the approach will be to *assume* that the output of the network is place cell-like, and convert/decode the output of the network in terms of Cartesian coordinates, and have this be the final output of the network. This might sound straight-forward; one could be tempted to use the argmax function to simply extract the location of greatest activation and denote this as a place cell center, for instance. Once place cell centers are found in this manner, one can simply decode the position as the position of the most active cell at any given time. However, argmax is not differentiable, and so we need to develop a differentiable alternative, in order to be able to train models using gradient methods.

To do this, consider the fact that one can think of a place cell's activation as being proportional to the probability of finding yourself at the cell's center: If a place cell fires strongly, it is very likely that you are close to the point in space where the place cell has its center. Going one step further, and assuming that an ensemble of N such cells constitute a probability distribution at any given time (i.e. their activations sum to one at each timestep), one may hope that the expected value

$$\mathbb{E}_{PC}[\mathbf{r}_t] = \sum_{j=1}^{N} p_t^j \mathbf{r}_{pc}^j, \tag{4}$$

is a good estimate of the position $\mathbf{r}$ at time t. Here, $\mathbb{E}_{PC}$ is the expected value over all place cells (at time t), and $\mathbf{r}_{pc}^j$ is the center of place cell j. Note that the place cell centers are fixed in time. It seems unreasonable to think that the outputs of our network (which is only assumed to be place cell-*like*) sum to one without any modification. Therefore, inspired by the way the softmax function is determined, one can calculate the normalized outputs or probabilities of place cell i, as

$$p_t^i = \frac{f_{pc,t}^i}{\sum_{k=1}^{N} f_{pc,t}^k}, \tag{5}$$

where $f_{pc,t}^i$ denotes the firing rate of place cell i at time t. For the physically inclined, this definition means that $\mathbb{E}_{PC}$ is nothing but the center of mass of a collection of particles with masses given by f_{pc} and locations by $\mathbf{r}_{pc}$.

Fig. 5 shows the above scheme applied to motion along a circle of radius 0.5, for three simulated place cell with fixed centers, at angles $\theta = 0, \pi/4$ and $\pi/2$, respectively. Also shown is the activation of each place cell, taken to be non-normalized Gaussians, each with standard deviation 0.1, alongside the corresponding output normalized according to (5). Alongside the place cell activations is a motion along the circle (solid line), the place cell centers (large dots), and the decoded position along the circle (small blue dots).

From the place cell activations it is evident that as one moves close to a given center, the normalized activation becomes almost unity (and all other activations consequently close to zero), which "picks out" the center of that place cell as the decoded position through (4). One can also see that for the regions between cells, there is some degree of interpolation of position; the decoded position is mostly a weighted sum of the two closes place cells, and so on. Finally, one can also see that the scheme fails far from a place cell: the decoded position far from the three place cells fall on a line between pc_0 and pc_1, but this is because the normalized activation once again goes to unity for the closest place cell, and almost zero for the remaining cells. This goes to show that this scheme depends on having place cells that cover the entire environment in order to get an accurate position estimate.

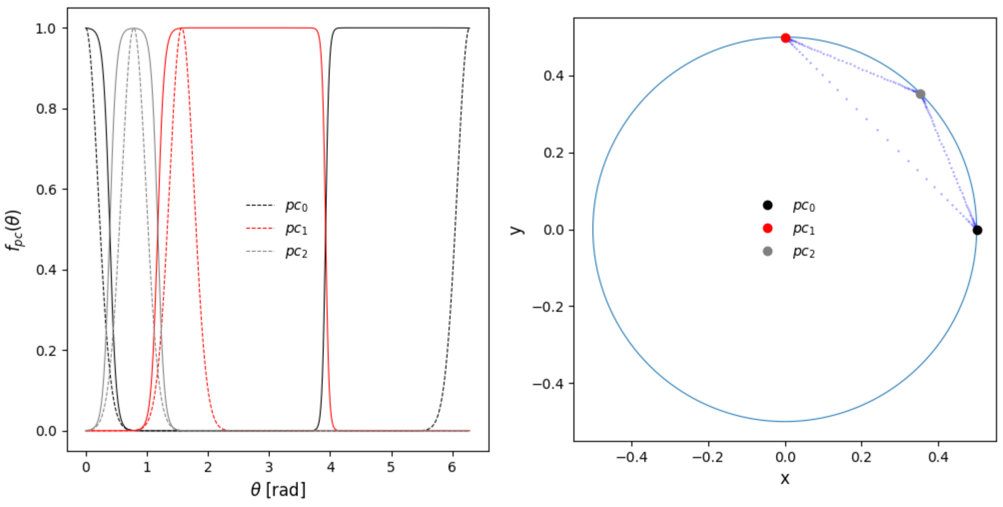

Figure 5: Example of using place cell activation to decode position. The left pane shows the activation an ensemble of three simulated, Gaussian place cells. Illustrated is both the cell activation (dashed line), and activation normalized over the ensemble (solid line). The right pane shows the motion that generated the responses of the left pane (blue circular path). Also illustrated is the centers of each place cell (large dots), as well as the decoded position (small blue dots).

While the example in Fig. 5 hopefully illustrates the decoding of position from place cells, a method for actually finding the place cell centers is needed. To find a such a scheme, we once again assume that our network is trained, and is producing some place-cell-like output. Finding the centers of these place cells is actually a similar problem to the position decoding, but this time we want to study the entire time-series response of a given place cell, and from this response deduce the place cell center. Following the same logic as before, but instead taking the expectation value over time for a single place cell, we arrive at the estimate

$$\mathbb{E}_{time}[\mathbf{r}_{pc}^k] = \sum_{t=1}^{T} P_t^k \mathbf{r}_t, \tag{6}$$

where $\mathbb{E}_{time}[\mathbf{r}_{pc}^k]$ is the expected value over time of the center of place cell k, and $\mathbf{r}_t$ is the target position at timestep t, while T is the number of timesteps, and we interpret P_t^k to be the probability of the place cell center being at position $\mathbf{r}_t$. The same normalization procedure as before is used, but this time, as we want to take the expected value over time, we normalize the place cell activation in time:

$$P_t^k = \frac{f_{pc,t}^k}{\sum_{t=1}^{T} f_{pc,t}^k}, \tag{7}$$

where $f_{pc,t}^k$ is the activation of place cell k at time t. This approach can then be applied to each place cell separately, yielding an estimate for all place cell centers.

At this point, this might seem like a hopeless approach; in order to determine the place cell centers, one needs the target position, i.e. the thing the network is trying to predict. However, the goal of this network is not strictly to path integrate; it is to form grid cell/place cell firing patterns in space while doing so. In other words, the goal is to have the navigator spawn at some random location in the room, path integrate *relative* to its starting position using an internal place-cell/grid cell representation of space. Then, in order to train the network, a way of measuring how well the network is actually navigating is needed. This is where the decoding process comes in; by decoding the internal representation in terms of Cartesian coordinates, one can measure of the network's performance (in terms of absolute prediction error), while still having the network use its own representation of space and only navigate relative to its starting point.

Methods

Simulated Paths

In order to investigate whether RNNs performing path integration develop grid- or place cell like hidden unit activations, a dataset was simulated. The dataset consisted of semi-random, semi-smooth paths in a square, 1×1 chamber. The paths themselves were created using python and the Numpy library. For a given path, the starting coordinate was sampled randomly from a uniform distribution in the chamber, and both initial speed and head direction was sampled from uniform distributions.

At each timestep, the head direction was updated by adding random turn angle (in radians), sampled from a normal distribution $\mathcal{N}(0, 0.25)$, i.e.

$$\phi \to \phi + \varepsilon,$$

where ϕ is the head direction and ε the added turn angle. Also at each timestep, the distances to all walls were computed. If the distance of any coordinate component was smaller than 0.05, or the agent was outside the arena, an additional turn angle was added to the head direction, so as to keep the agent inside the arena, and avoid the walls. This was done by determining the difference in angle between the closest wall normal and head direction, and turning counterclockwise if the difference was positive, and clockwise otherwise. This way, if the head direction for example was $\pi/3$ and the corresponding top wall normal angle was $\pi/2$, then the agent[1] turns clockwise, eventually facing away from the wall.

In addition, the speed of the agent was updated at each step, given by an update rule similar to that of the head direction:

$$s \to s + \chi,$$

where χ was a random change in speed, sampled from a normal distribution $\mathcal{N}(0, 1)$. To avoid the speed becoming negative or too large, s was clipped between 0 and 1. For the head direction, the modulus was taken with 2π to ensure that the head direction was in the interval $[0, 2\pi]$. Finally, the position of the agent at a given time step t was computed as

$$\mathbf{r}_t = \mathbf{r}_{t-1} + \Delta t \cdot s_t(\cos\phi_t, \sin\phi_t),$$

where the time constant Δt was set to 0.1.

To emulate a real-world scarcity of data, as well as save storage space, small datasets were created with sequence lengths of 100, 1000, and 10000 steps, respectively. Each dataset featured 10000 paths, or samples. In order to train recurrent neural nets on both Cartesian coordinates as well as head direction/speed inputs, datasets were created with both sets of inputs.

[1]When the term "agent" is used, this only denotes the thing that is moving (i.e. the thing that generates the path, an imagined rat), and should not be confused with an agent in reinforcement learning which can perform actions to change outcomes.

Recurrent Neural Networks

In order to investigate path integration in different RNN architectures, three different RNN models were created: one baseline model with uniform initialization, one IRNN model, and a custom RNN which supported radial basis function inputs, an RBFRNN.

All models featured the same basic architecture, composed of two units; one RNN layer, and one output layer. In all cases, models were implemented using Tensorflow 2.1 and Python3. Note that all code is freely available at `https://github.com/markusbp/fys_stk4155/tree/master/project3`.

For the baseline model, the recurrent weight matrix, input matrix, and biases were all initialized according to a uniform distribution, more specifically a Glorot uniform initialization scheme (see for example [5]). For this model, the recurrent activation function was tanh, and there were 100 recurrent nodes.

The output layer was initialized using the same Glorot uniform scheme, but the biases were initialized to zero. The output layer featured 50 output units (i.e. assumed place cells). To enforce a non-negative output similar to that of place cells, the output layer activation function was ReLU.

To allow for training on very long paths, the IRNN architecture described in the background section was used for the second RNN model. As described earlier, the recurrent weight matrix was initialized according to the identity matrix, the biases were set to zero, and the input weights were initialized according to a normal distribution $\mathcal{N}(0, 0.001)$. The recurrent activation function was accordingly ReLU, and the recurrent layer featured 100 hidden units. In addition, an optional L2 weight regularization was added to the recurrent weight matrix. The IRNN output layer was identical to that of the baseline model, both in terms of weight initialization and activation function. As with the baseline model, the number of output nodes, i.e. assumed place cells, was 50.

The RBFRNN was based off of the IRNN design, in order for it to be able to cope with possibly long paths. However, to allow for radial basis function inputs, a custom RNN cell, or RNN equation, was implemented. The custom RNN cell followed the same structure as (3), but featured two sets of inputs, one from an ensemble of simulated speed cells, and the other from a set of simulated head direction cells. The speed cell ensemble consisted of 20 nodes, the responses of which were calculated as a non-normalized Gaussians in a manner similar to that outlined in the background section. For simplicity, the speed cell centers were taken to be linearly spaced in the interval $[0, 1]$, corresponding to the possible values of the dataset paths. To ensure that the full interval of possible speed values were encoded by the speed cells, the width, or standard deviation of each Gaussian was taken to be 1/20, i.e. the range of possible speeds divided by the number of cells.

The responses of the head direction cells were approximated using the Von Mises distribution described in the background section. As with the speed cells, the centers of the head direction cells were linearly spaced along the range of possible head directions as they appeared in the dataset, i.e. in the range $[0, 2\pi]$. The width parameter κ of the distribution of each cell was taken to be 2π, and a total of 50 head direction cells was used.

The activations of the speed and head direction cells were each fed into the RNN hidden state by a respective input weight matrix. Each weight matrix was separately initialized according to the standard IRNN input matrix initialization, i.e. according to a normal distribution $\mathcal{N}(0, 0.001)$. The recurrent weight matrix was initialized to identity, and the bias to zero, and the recurrent layer consisted of 100 nodes, as with the other models. The output layer of the RBFRNN was identical to those of the IRNN and baseline models.

For the IRNN and baseline models, the input to the RNN layer was the Cartesian velocity vector of the agent. In other words, the input at time t was

$$\mathbf{x}_t = s_t(\cos\phi_t, \sin\phi_t),$$

which was done to avoid the discontinuity of the head direction at $\phi = 0 \iff 2\pi$. For the RBFRNN, the speed and head direction were supplied to the head direction and speed cell ensembles for decoding, so that the input vector to the RNN layer at time t was

$$\mathbf{x}_t = (s_t, \phi_t),$$

for a given path.

In addition to head direction and speed inputs, the network itself was supplied with all target positions, so that the place cell/position decoding scheme detailed in the background section could be implemented. Each RNN performed the same decoding process, using the following steps:

1. Compute RNN hidden state activations from head input data and previous state
2. Compute output activations using RNN activations
3. Determine place cell centers from output using eq. (6) and position labels
4. Decode output to Cartesian coordinates using place cell centers and eq. (4)
5. Yield predicted Cartesian coordinates

To avoid numerical instability, a small constant term, $\delta = 10^{-10}$, was added to the normalization factors of the place cell activations (i.e. to the denominators of (5) and 7)). This was done as it was observed that the loss diverged early on in training, suggesting models initially had very small output activations. For all models, the initial state of the recurrent layer was trainable, meaning that each node could learn a unique starting state, but this state was the same for all samples in a mini-batch.

Training & Experiments

To compare the performance of all networks on the navigation task, all networks were trained on datasets corresponding to paths of length 100 timesteps. All networks were trained using minibatch gradient descent and the Adam optimizer. The cost function was taken to be the mean squared error in all cases, for all models. As mentioned, each dataset consisted of 10000 samples or paths, with each path in a given dataset being the same length. In all trials, 80 % of the dataset was used for training, and the remaining 20 % reserved for testing. For training of RNNs, the batch size was 50 for all models, and after some initial testing[2], the learning rate was taken to be $1 \cdot 10^{-5}$ for the baseline model, and 10^{-4} for the IRNN and RBFRNN models. For these initial runs, no regularization was applied to any of the models. All models were trained for 100 epochs, and the dataset was shuffled between each epoch.

The relatively short 100-timestep paths did not allow for complete exploration of the chamber, and the place cell decoding scheme assumes that place cells emerge as the navigator moves around the chamber. Therefore, models were also trained on a dataset containing 1000-step paths, using the same training parameters described in the previous paragraph. Note that lower learning rates were tested for the longer paths, but this only seemed to slow down learning for the baseline model. For the IRNN based models, a lower learning rate strangely seemed to cause the models to stop learning completely.

To visualize the activations of the RNN and output layers as a function of space, trained models were applied to a single path, and the resulting activations were then meaned over in space, by subdividing the arena into 50×50 bins, and dividing the activity by the number of times a given bin was visited. Note that this was done using the SciPy library.

Finally, to study the effects of regularization, a grid search was performed for the IRNN model, for values of the recurrent weight matrix L2 weight penalty factor. In this search, one IRNN model was trained for 20 epochs on the 1000-timestep paths for each L2 penalty factor. The L2 factors were taken to be geometrically spaced values in the interval $[10^{-6}, 10]$.

Results

Simulated Paths

Fig. 6 shows an example of a path taken from the simulated dataset, of length 1000 timesteps. Each step is indicated by a small dot, and so the concentration of dots along the path also gives an indication of the speed of the agent. A large blue dot indicates the starting position, and one can clearly see that the motion constitutes a semi-smooth traversal of the chamber. It is also noticable that there tends to be a rather

[2]Stochastic Student Descent

abrupt turning away from the walls, but turning takes place over a few timesteps and so is less dramatic than bouncing off the walls, for instance.

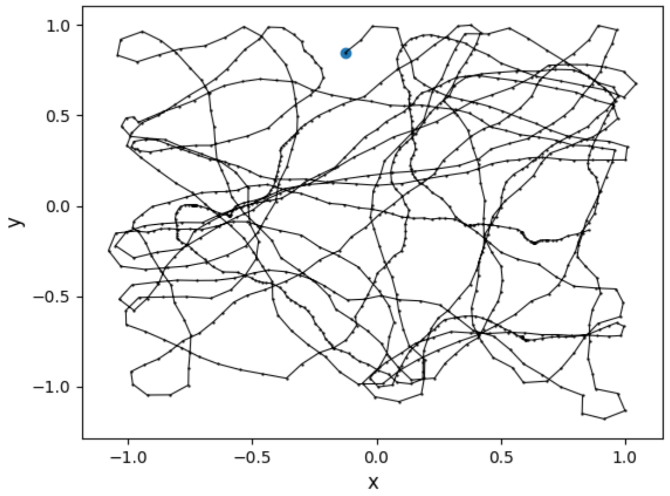

Figure 6: 1000-timestep path from the simulated trajectory dataset. The large blue dot shows the initial position of the navigator, while the small dark dots indicate the position at each timestep.

RNNs & Experiments

Table 1 shows the test set mean absolute error (MAE) for all trained models, when applied to datasets containing length 100- and 1000 timestep paths. Note that the mean absolute error was not the loss function used for training, it was only used for gauging performance, as the MAE gives the average (coordinate) distance between prediction and target. Immediately, is is evident that the baseline model is outperformed by the IRNN models, both for long and short paths. Considering the 100-steps case, the MAE is almost twice that of the IRNN, while for the long paths, the average error is approximately triple, at 0.12 for the baseline and 0.04 for the IRNN. Interestingly, the IRNN and RBFRNN are comparable in performance for the 100 timestep path, with the RBFRNN performing ever so slightly better. We see that the opposite is true for the 1000-timestep paths, as the IRNN slightly outperforms the RBFRNN in terms of MAE.

Table 1: Test set mean absolute error for each RNN model after 100 epochs of training, for paths of length $T = 100$ and $T = 1000$ timesteps.

Steps	100	1000
Baseline	0.028	0.120
IRNN	0.016	0.040
RBFRNN	0.015	0.053

Fig. 11 in the Appendix shows the output activations of a trained baseline model, as applied to a sample 100-timestep path from the test dataset. In addition to activations, the actual path is also shown, as well as the decoded place cell centers. Lastly, the distance to the decoded place cell center is shown alongside the activation. As one can tell, the decoded centers are scattered neatly along the travelled path, and the distance/activation plots indicates that the place cell center decoding scheme is working as intended: For most cells, the firing rate is at a maximum where the distance is smallest, and close to zero. As such, the decoding appears to pick the center to be the location at which the activation is largest. However, one can also see that these are not true place cells; they only appear to be sharply peaked at a single location in *time*, but from the distance plot some cells are approached multiple times, which should result in multiple firings if the results were place-cell like. One can also note that some cells are zero at all times, and appear not to contribute.

The IRNN and RBFRNN exhibit the same behaviour as the baseline model when trained on the 100-timestep dataset (see Fig. 12 in the Appendix for the response of a trained RBFRNN model); the activations are sharply peaked in time, but do not show repeated firings when the center is approached multiple times. However, the activations of the RBFRNN do appear slightly more smooth and less sharply peaked than that of the baseline model.

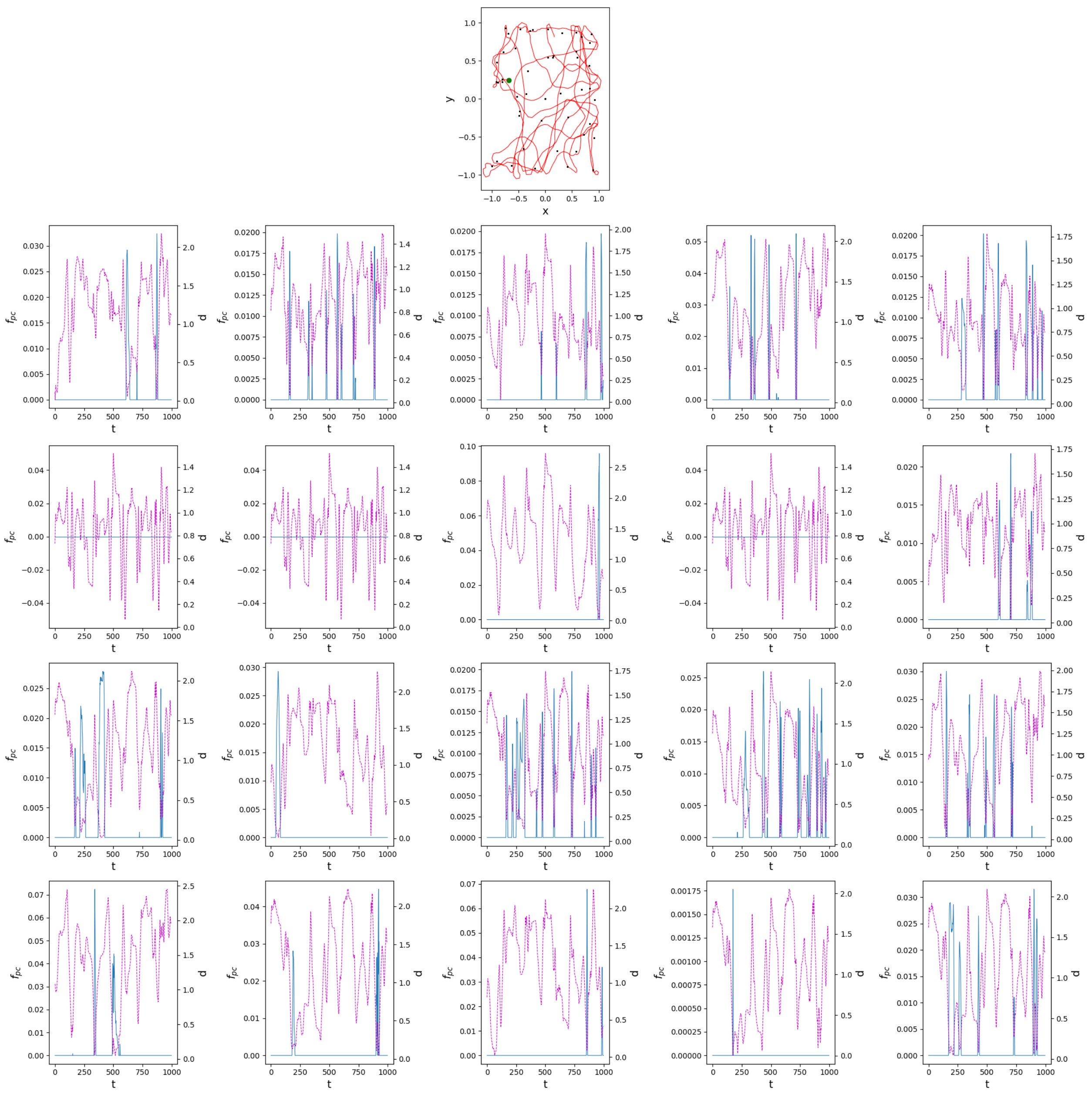

Figure 7: Activations of the first 20 output nodes of the IRNN model trained on paths of length 1000 timesteps for 100 epochs, as a function of time. Also shown, in the top pane, is the actual path (in red), the starting point of the path (green dot) and the decode place cell centers (black dots). Alongside the firing rates (blue, left y-axis) f_{pc}, is the distance d to the decoded center of that cell (magenta, right y-axis).

Fig. 7 shows the output activations in time, for an IRNN model trained on paths of length 1000 timesteps. For this run, one can tell that the chamber is explored much more fully, and this time the decoded centers are not perfectly aligned with the path, but rather scattered somewhat uniformly around the chamber. Also unlike the 100-timestep case, is that one can now observe that some of the output nodes are now active at multiple points in time, which seem to coincide with a close passage to the node's decoded center (i.e. a small distance). It is this behaviour one would expect from a node exhibiting place-cell like behaviour. Even more encouraging is the fact that the response of a cell increases with decreasing distance. It is also interesting to note that the response of the top-left node does not appear to respond to a very close passage at time $t = 0$, but fires strongly at later times.

Fig. 8 shows the average output activation of the IRNN trained on 1000-timestep paths, as applied to a single, 10000 (10^4) timestep path, plotted as a function of the target coordinate. Confirming the observations made for Fig. 7, it appears that the IRNN has indeed formed place cells as solution to the navigation problem. Comparing to the idealized place cell of Fig. 1, these place-cell-like responses are not perfectly circular, but are quite reminiscent. They also all appear to be of roughly the same size.

The output activations of the baseline model also displayed some repeated firing for close passages, but for this model the extrapolated centers were clustered close to the walls, and the output nodes were mainly active along the borders. As they do not bring much to new to the table, these results are not included here, but are available at `https://github.com/markusbp/fys_stk4155/tree/master/project3`.

Fig. 9 displays the average hidden unit activation of the first 20 nodes in the RNN layer of the IRNN model trained on 1000-timestep paths, plotted as a function of space. Once again, this activation map was produced by applying the model to a single path of length 10000 timesteps. As one can see, there is little evidence of a grid cell-like response in this layer. Rather, the response appears to be strongly linked to the walls and corners, with some units being highly active along the rightmost wall, some along the top, and so on.

As for the RBFRNN, the spatial and temporal responses were very similar to those of the IRNN, both for the output and RNN layers. For the intrigued reader, these responses can be found at the repository linked to previously.

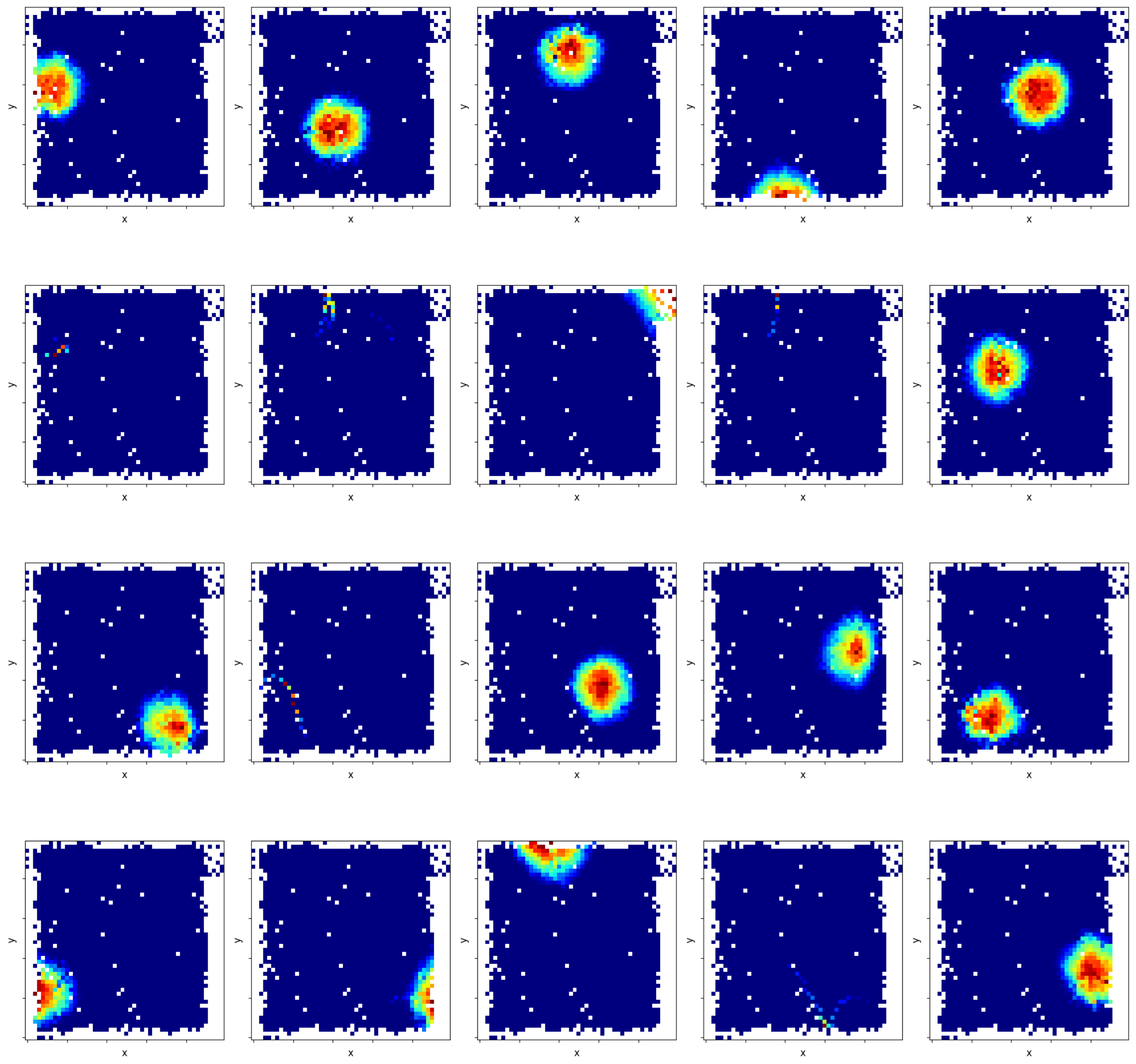

Figure 8: Activations of the first 20 output nodes of the IRNN model trained on paths of length 1000 timesteps for 100 epochs, plotted as a function of space in a 1×1 chamber. For increased resolution, this firing map was produced by running the model on a path of length 10000 timesteps. White regions in the firing maps indicate regions not visited.

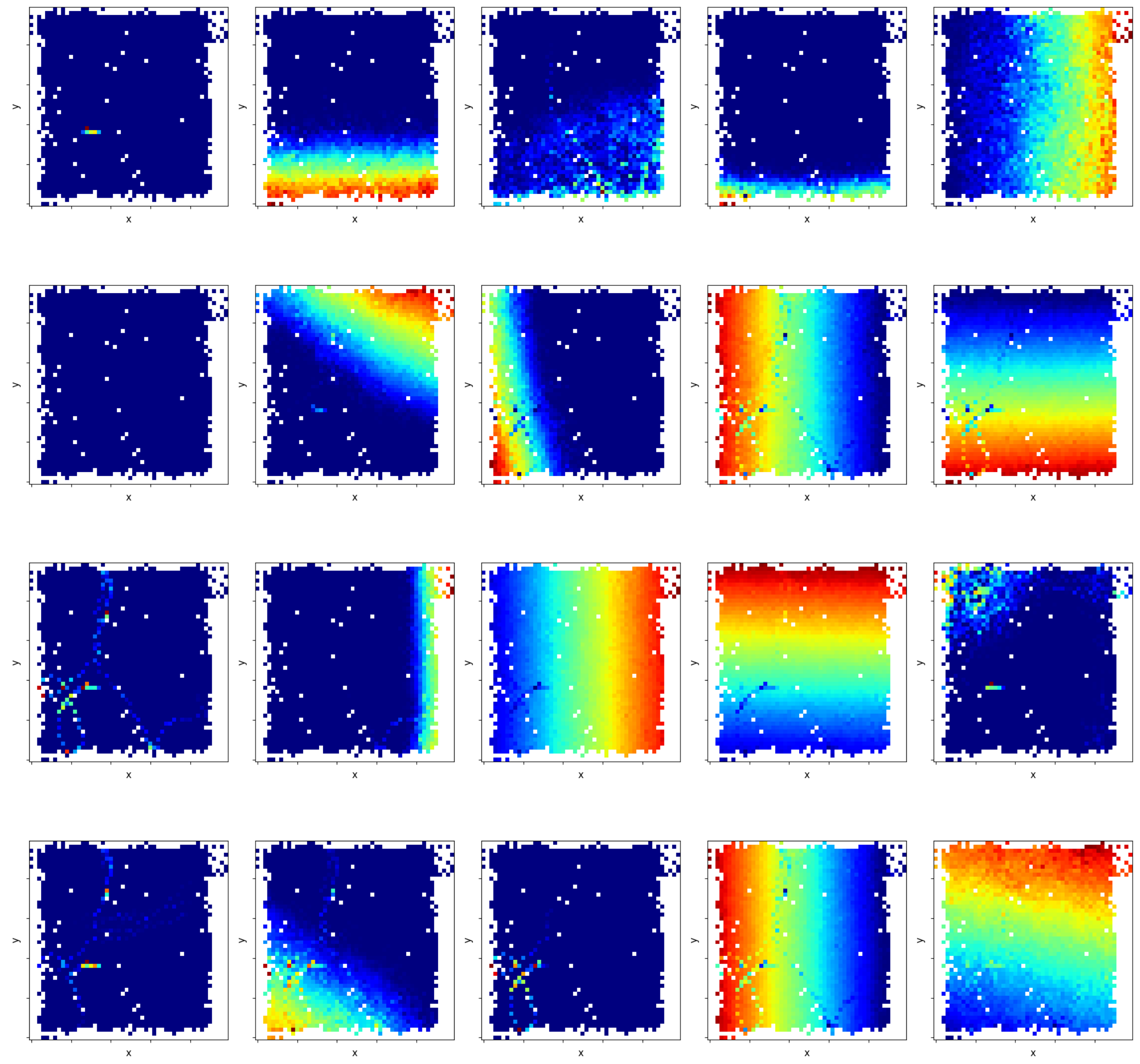

Figure 9: Activations of the first 20 RNN nodes of the IRNN model trained on paths of length 1000 timesteps for 100 epochs, plotted as a function of space in a 1×1 chamber. For increased resolution, this firing map was produced by running the model on a path of length 10000 timesteps. White regions in the firing maps indicate regions not visited.

Finally, Fig. 10 shows the results of the grid search for the L2 weight regularization factor for the IRNN models trained on the 1000-timestep path dataset, with each model being trained for 20 epochs. While there is a clear trend that applying some weight regularization seems to improve the MAE, especially for $L2 \approx 0.1$, this improvement did not translate to any difference in the spatial responses of the RNN or output layers. In other words, every model trained for Fig. 10 displayed spatial responses similar to those in Figures 9 and 8.

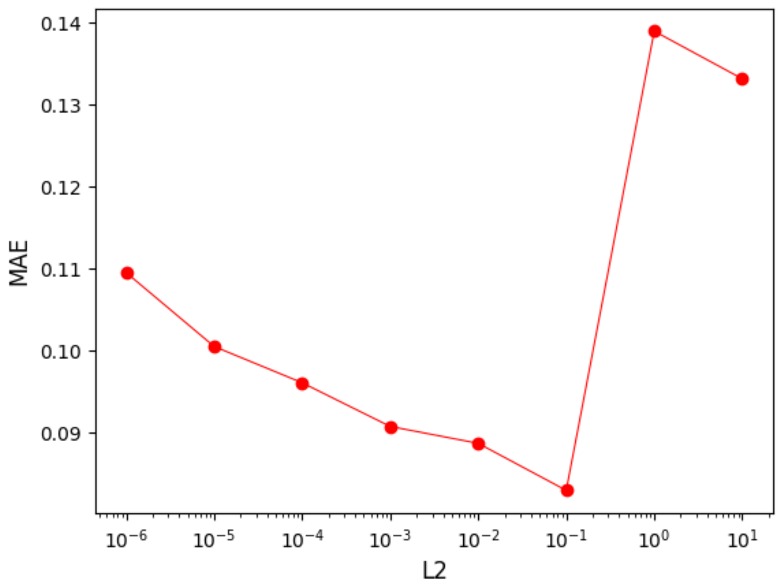

Figure 10: Results from grid search of the L2 weight regularization factor applied to recurrent weight matrix, for the IRNN model trained on 1000-timestep paths.

Discussion

Simulated Paths

While it is very interesting that the IRNN developed place-cell like outputs when trained on long paths, it seems appropriate to first discuss the more technical details of this project. Starting with the generated datasets, it is worth pointing out that the author has little first-hand experience with the path a rat or mouse traces out when confined to move in a square chamber. Therefore, the development of the dataset was largely inspired by the trajectories used in [2] and [3], which hopefully means that the generated trajectories are at least indirectly inspired by the biological case.

However, there are a few possible concerns with the dataset at hand: For one thing, the speed of the navigator could potentially be too high or too low: If the step size is too great, it would seem plausible that grid cells could have a hard time forming a complete tiling of space, as a single step might completely "skip" the firing field of another grid cell. If the speed is consistently too low, the agent hardly navigates the chamber, and the results might be more akin to those of the 100-step case, where the place cells centers fall neatly along the path, but a cell only fires once (in time).

Another possible concern is that no hard boundary was used for the walls of the chamber; instead a soft turning system was used, to emulate the fact that no animal with concern for its own well-being would charge directly into a wall, but rather turn away as it approaches it. The soft turning did however allow for the agent to exit the 1×1 boundary of the chamber, even if only slightly. It might be that these relatively rare occurrences might be damaging to training, and countermeasures should be implemented in the future. As another possible future development, it might be interesting to give the network an explicit signal whenever it was close to the wall, akin to touching the wall. However, it might be that the relatively fast turning shown in Fig. 6 is a sufficient signal. This is also supported in part by the RNN hidden layer response in Fig. 9, which appears to react strongly near the walls.

RNNs & Experiments

Since this project is somewhat exploratory, a lot of the findings presented here are not well understood. For instance, the reason why models trained on shorter paths developed the strange one-shot "pseudo-place cells" is not known. It might be that the relatively high number of output nodes (50) in this situation makes the repeated firings of a place cell obsolete; it might simply be good enough to have adjacent nodes (in space) excite each other, and then simply discard the cell after use. It might therefore be interesting to try to train models with more regularization: either by simply removing some nodes, or apply weight regularization, or perhaps also dropout.

In any case, it would appear that the longer 1000-timestep problem is sufficiently hard that a place cell representation is needed to solve the navigation task effectively. This is also supported by the fact that the baseline model performed slightly worse, and did not exhibit place-cell like responses, but were rather more reminiscent of so-called border cells (i.e. only active along the border). This is not to say that a place-cell representation is the *best* solution, only that the proposed decoding scheme appears to work best with such a representation.

While on the subject of the decoding procedure, it is worth mentioning that the approach does seem to work, but that training of the RNN models was sometimes unpredictable: The baseline model did not appear to converge for a learning rate of 10^{-4}, and strangely, the IRNN based models seemed to struggle for learning rates *lower* than 10^{-5}, but appeared to converge most of the time for a learning rate of 10^{-4}. The reason for this is not understood, but as a speculation, it might be that the lower learning rate causes the models to linger in a region of parameter space where the outputs are small, which might cause numerical instabilities in the normalization of the outputs. Coupled with the fact that the Adam optimizer was used, which adapts and decays the learning rate over time, a somewhat large initial learning rate might get the network out of this danger zone. Regardless of explanation, a future exploration should feature a proper grid search of learning rates, which was not done this time due to time constraints.

A reasonable question in the same vein would be why the softmax function was not used to normalize the outputs. After all, softmax is not as prone to division by zero, and was designed to approximate argmax. The reason for this is somewhat simple; The output activation was taken to be ReLU, inspired by the thought that place cells firings should be non-negative. For non-negative inputs, the smallest value of the exponential function is unity, and if softmax was used to normalize the output, one node would need to fire very strongly to squash the activation of other nodes, which might inhibit learning and decoding. In the future, it would be interesting to remove the ReLU output activation, and attempt to use softmax instead of the center-of-mass approach described in the background section. An added benefit of this might be that gradients might become larger; judging from Fig. 5, the normalization procedure might also cause gradients to become small. The removal of the ReLU output activation might also improve learning, as it can easily cause gradients to become zero throughout the network, easily halting learning. However, based on the results in this project, this does not seem to be a major problem, as even very long time-series (1000 timesteps) problems were learned somewhat easily by the networks.

Another point of contention is the number of nodes used in each model: The 100 recurrent, and 50 output nodes used for each model was simply chosen as to be not too large, and not too small. For the output nodes, as they are supposed to represent place cells, the number of nodes could directly influence the accuracy of the results, as more and more sharply peaked place cells might be used to more accurately represent position. As discussed for the short paths, it might be that a surplus of place cells might be damaging to the formation of place cell responses. It could also be that a smaller or larger number of recurrent nodes could be conducive to grid cells, but this would also have to be studied further in the future.

As for the actual results, Table 1 indicates that an IRNN-based architecture appears better suited for the navigation task than the baseline model. It might of course be the case that other initializations perform even better, but the IRNN setup proved to be a simple way of handling long time dependencies, and also seemed to work well even with the addition of radial basis function inputs.

The addition of radial basis function, on the other hand, did not appear to induce any particular benefit, either in terms of the character of the spatial responses, or in actual navigation MAE. Therefore, it would seem that the addition of RBF inputs is an unnecessary and complicated addition, which does not help in learning. RBFs are on the other hand conceptually somewhat appealing, and it might be that by better initialization of the basis set than the naive one used in this project, results could be improved.

While it is rewarding to find that the responses of the IRNN trained on the 1000-timestep paths does appear to form place cell-like responses, it is still noteworthy that no grid-like responses were found. As such, it will be necessary to investigate possible extensions of the model in the future. One such attempt was made using the grid search for L2 weight regularization factors, but this did not seem to give much change. It might however be that the allotted training time of 20 epochs was too small, and it would be interesting to repeat this search with a longer training time. One other possibility is to add dropout in the model, something [2] reported was necessary for grid cell formation.

As an ending note, a strange feature was found in a last-minute analysis of the results: By generating a new figure, equal to that of Fig. 8, it was found that a *different* 10000-timestep path yielded almost the exact same place cell-like responses. This seems to go against the notion that the place cell representation is generated anew for each path, and that each path generates a unique set of place cells. Rather, it might be that the network in some way orients itself (with no training involved) in the chamber, causing the place cell to form the same mapping of space for all sufficiently long paths. This seems quite mysterious and could either just be a fluke, or a real feature of the network. Regardless, it was deemed sufficiently interesting to include at the last minute, and will be studied further in the future.

Conclusion

In order to investigate the formation of place cells and grid cells in recurrent neural networks, a simple decoding scheme was developed. In short terms, the process involved assuming that the RNN had developed a place-cell like output, and from that extrapolate the centers of said place cells, and use these centers to decode the network's activations into Cartesian coordinate predictions. Under this assumption, the goal was to train RNNs, and see whether grid-cell/place-cell like output activation responses emerged during navigation.

RNNs were trained on simulated paths of an imagined rat navigating a 1×1 square chamber. There were a total of three RNNs, one baseline model, one identity RNN meant to handle long time dependencies, and a radial basis function RNN designed to take radial basis function inputs. The inputs to the baseline and identity RNN were the Cartesian velocity vector, and the last network received speed and head direction inputs, by way of radial basis functions. All networks were trained on paths of length 100-, and 1000 timesteps

The identity RNN performed best in terms of navigational mean absolute error, achieving a final test set error of 0.016 for the 100 timestep paths, and 0.04 for the 1000-step paths. The baseline model achieved a final error of 0.028 in the 100-step case, and 0.120 for the 1000-step paths. The radial basis function network performed comparably to the identity network, albeit slightly worse for the longer paths.

More importantly, the identity network was found to exhibit place-cell-like output activations, but no sign of grid cell-like responses were found. The radial basis function network also displayed the same responses, but generally performed slightly worse, and given the simplicity of the identity network, the radial basis function approach was deemed unessecary.

Interestingly, when the identity network's outputs were studied further for very long paths, they were found to exhibit similar average spatial responses, even for different paths. This suggests either an error in the finding, or possibly an interesting mechanism in the network. Believing the latter to be true, this makes for a very interesting case for future investigations.

References

[1] *The Nobel Prize in Physiology or Medicine 2014*. en-US. 2020. URL: https://www.nobelprize.org/prizes/medicine/2014/press-release/ (visited on 12/16/2020).

[2] Andrea Banino et al. "Vector-based navigation using grid-like representations in artificial agents". en. In: *Nature* 557.7705 (May 2018), pp. 429–433. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0102-6. URL: https://www.nature.com/articles/s41586-018-0102-6 (visited on 11/13/2020).

[3] Ben Sorscher et al. "A unified theory for the origin of grid cells through the lens of pattern formation". In: *Advances in Neural Information Processing Systems* (2019).

[4] Morten Hjorth-Jensen. *Week 42: Convolutional (CNN) and Recurrent (RNN) neural networks*. Oct. 2020. URL: https://compphysics.github.io/MachineLearning/doc/pub/week42/html/week42.html (visited on 12/16/2020).

[5] Markus Borud Pettersen. “Project 2- FYS-STK4155”. Project Report. Oslo: University of Oslo, 2020.

[6] *8.7. Backpropagation Through Time — Dive into Deep Learning 0.15.1 documentation*. URL: https://d2l.ai/chapter_recurrent-neural-networks/bptt.html (visited on 12/17/2020).

[7] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. “A Simple Way to Initialize Recurrent Networks of Rectified Linear Units”. In: *arXiv:1504.00941 [cs]* (Apr. 2015). arXiv: 1504.00941. URL: http://arxiv.org/abs/1504.00941 (visited on 12/14/2020).

Appendix: Activation maps

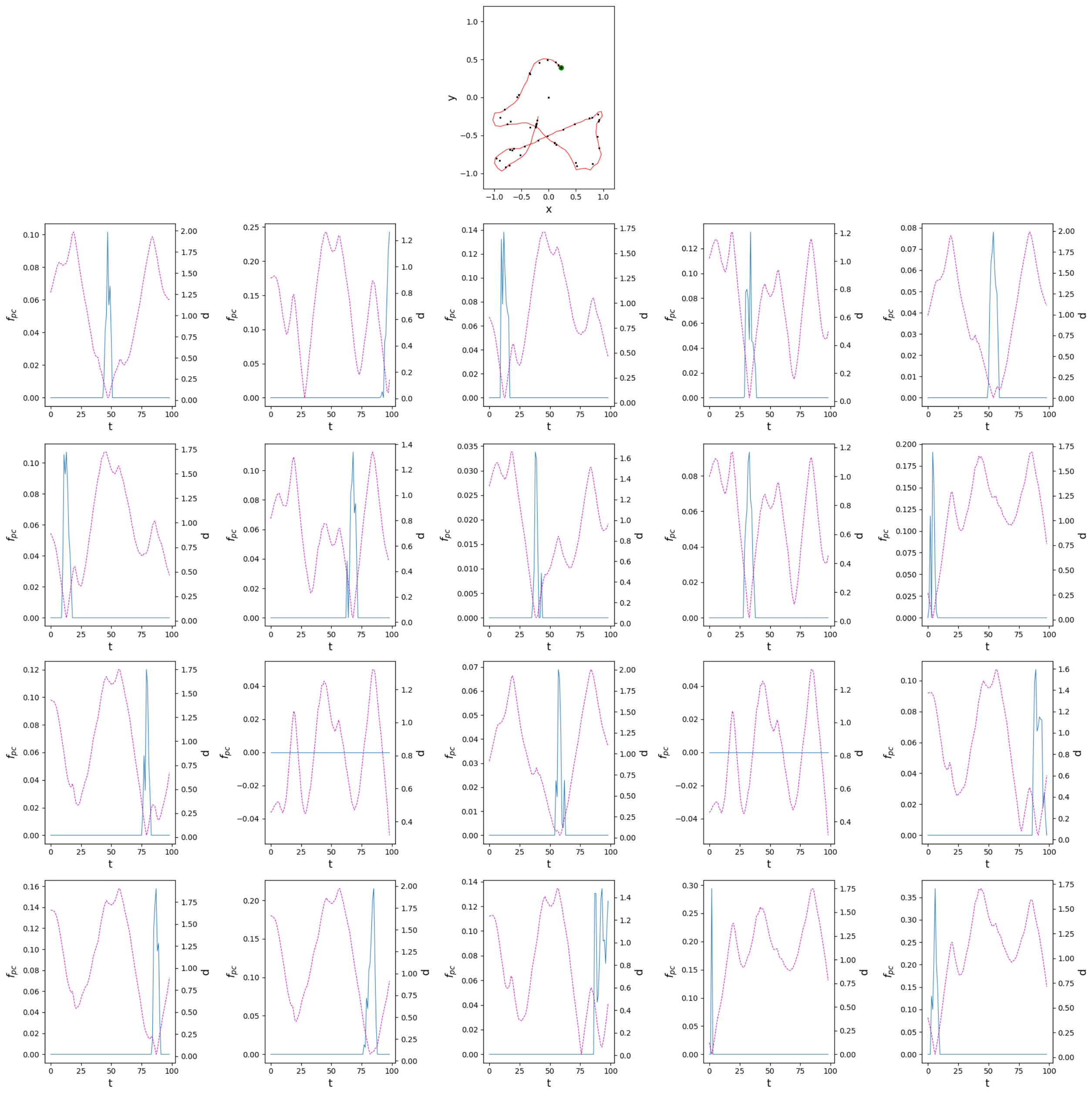

Figure 11: Activations of the first 20 output nodes of the baseline model trained on paths of length 100 timesteps for 100 epochs, as a function of time. Also shown, in the top pane, is the actual path (in red), the starting point of the path (green dot) and the decode place cell centers (black dots). Alongside the firing rates (blue, left y-axis) f_{pc}, is the distance d to the decoded center of that cell (magenta, right y-axis).

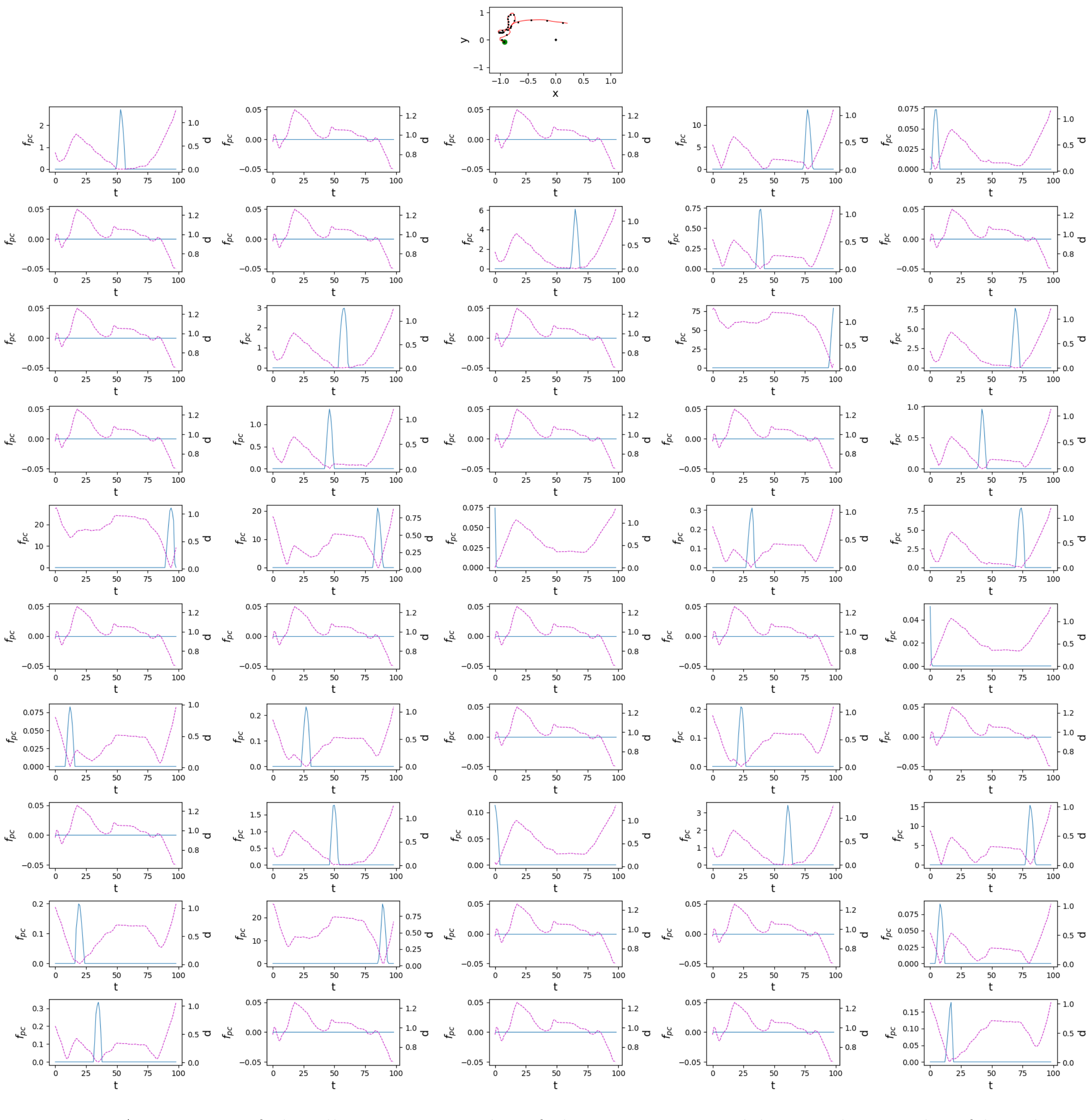

Figure 12: Activations of the all 50 output nodes of the RBFRNN model trained on paths of length 100 timesteps for 100 epochs, as a function of time. Also shown, in the top pane, is the actual path (in red), the starting point of the path (green dot) and the decode place cell centers (black dots). Alongside the firing rates (blue, left y-axis) f_{pc}, is the distance d to the decoded center of that cell (magenta, right y-axis).