Implementation of Hippocampally Dependent Temporal Difference Navigation Models

Hong, Joon Hwan; 260832806

Background

A paper by Foster, Morris, and Dayan proposes a model of hippocampally dependent navigation using a temporal difference learning rule (Foster, Morris, & Dayan, 2000). The hippocampus houses place cells; cells that fire when the animal occupies a specific region (place fields) of an environment. While hippocampal place cells have been implicated in playing a role in spatial learning, the method which a system of place cells – only individuating different locations – can provide navigation without the computation of specific spatial quantities such as distance or direction to a goal is unknown (Foster, Morris, & Dayan, 2000).

A type of neural network learning rule, the temporal difference learning (TD), is a type error-driven learning in a feed-forward neural network. As a background, feed-forward neural networks are instances of neural networks which the node edges do not form a cycle. In particular, TD learning results in a reinforcement learning model, a reward-based change of variables based on an error function. TD learning utilises an error function based on the difference between two sequential output values in a timestep (hence a temporal difference). Consequently, there is potential for TD learning to bridge between the location-based activity of place cells and the goal-directed navigation behaviour in animals (Foster, Morris, & Dayan, 2000). The paper proposes two mathematical models of rodent navigation that uses place cells and TD learning in the Morris watermaze task: The Actor-Critic model (AC), and the Combined

Coordinate & Actor-Critic model. For the term paper, the models are explored by being implemented in Python (Jupyter Notebook) and simulated. The issues with each proposed model are explored as well.

Alternative learning rules were considered such as backpropagation and other supervised methods. Supervised learning methods describe an error function (learning) based on the difference between the neural network output and a target output. This results in a circumstance where a target output is necessary at all timesteps, which lead to a much more computationally expensive process. In addition, it required an "oracle" – an external process to the simulated animal navigation model – to determine the target/correct output. This would not be satisfactory to simulate an animal-like behaviour. As a result, it was discarded for the exploration. An alternative to TD learning is also discussed in the paper: the trace memory learning rule (Brown & Sharp, 1995). It is an alternative supervised learning rule where place cells, regardless of their distance from the goal location, can learn the expected reward by maintaining a trace memory of its activation and decay slowly: it maintains a history to use later when the model reaches the goal. The alternative learning rule is discarded due to its extreme variability in learning and its increased computational cost of maintaining a trace compared to solely comparing sequential timesteps of TD learning (Foster, Morris, & Dayan, 2000).

If a similar behaviour compared to real life data can be generated, it serves as a proxy for how well the model emulates the use of hippocampal place cells in navigation. The mice learn to navigate to the platform location from an arbitrary starting location and results in short escape latency/path length from starting position to the goal after some trials. In addition, if the platform is changed to a new position, the mice relearn which results in a temporary

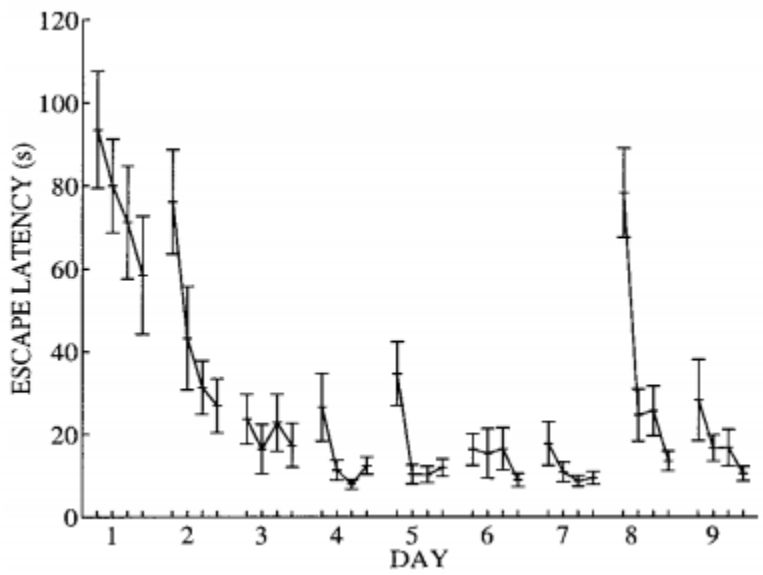
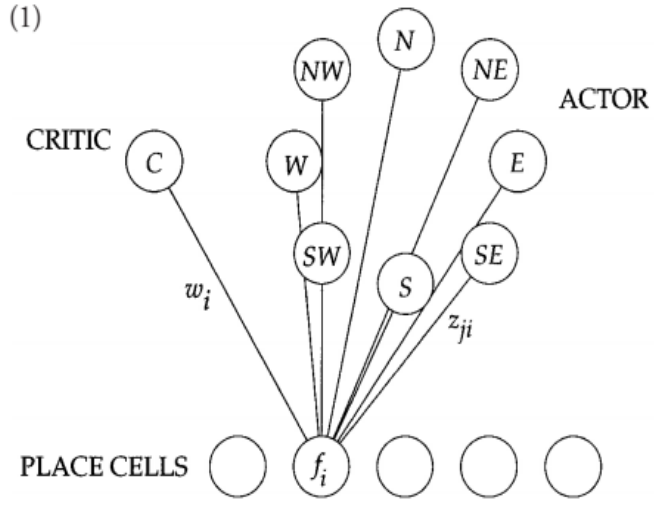


Figure 1: Performance of rats (N=12) with 4 trials/day. As expected, the rats slowly learn and approach an asymptotic performance seen in the stagnation of escape latency in the later trials per day. Standard deviation is shown as the error bars. Each day, a new platform location is set.

increase in escape latency/total path length. However, the mice are able to relearn after several trials to once again easily find the changed platform location. This is shown in *Figure 1*, where the performance of rats is plotted over days with four trials per day. The figure is provided from the paper.

Implementations

I) Actor-Critic Model (AC)



$$(2)\quad f_i(\mathrm{p}) = \exp\left(-\frac{\|\mathrm{p} - \mathrm{s}_i\|^2}{2\sigma^2}\right)$$

$$(3)\quad \mathrm{C}(\mathrm{p}) = \sum_i w_i f_i(\mathrm{p})$$

$$(4)\quad \delta_t = R_t + \gamma C(\mathrm{p}_{t+1}) - C(\mathrm{p}_t)$$

$$(5)\quad \Delta w_i \propto \delta_t f_i(\mathrm{p}_t)$$

Figure 2: (1) an illustration of the feed-forward neural network. The place cells act as an input layer with 2 output "groups": A critic output and an actor output cluster, composed of eight neurons representing 8 directions which the model can take given a location defined by the place cell. This figure is from the paper. (2) place cell activity (the network input) is defined as a Gaussian function parameterized by the location, p. (3) the critic activation function, which is the weighted sum of the place cell inputs. The goal is to adjust the weights to approach and learn a correct "value function" for the navigation model to use. (4) the temporal difference prediction error definition. (5) the definition of change in weight according to the place cell activation input and the temporal difference. This ensures that C(p) → V(p) as it continually changes until the difference is 0.

The first model is the Actor-Critic Model, which is a basic feed-forward neural network with N=493 input place cells, with each connected to nine output cells. The nine output cells are separated into two sections, the first being the critic output cell, and eight actor cells representing the eight directions the navigation model can take given a location.

The simulation is handled with a central Simulation python class, a full list of class variables is available in *Appendix A*. Mainly, it houses the various constants and changing variables as class variables, as well as the code for the maze and the model (actor-critic or combined). The entire notebook code is available at: https://github.com/Joon-Hwan-Hong/NEUR-503-Final-Project

Place cell activity is modelled as a gaussian function to represent as a probability density function. Place cells map to a position of the agent, which are the place fields in an environment. In *Figure 2(2)*, σ is a hyperparameter that is arbitrarily chosen. It is the radius of a boundary where the firing rate would be 61% of its maximum and s_i is the global coordinate of the center of the place field which the place cell maps to. `self.s` is the coordinate of the place field at the given position p. The paper models with σ=0.16m. The function implementation is shown below as an example, the others can be seen in the repository link:

```
def activity(self, p):
    '''
    calculates equation (1) in the paper, given the position p.
    '''
    return np.exp(-(np.linalg.norm(p-self.s, axis=1)**2)/(2*(self.sigma**2)))
```

The critic output cell attempts to learn the value function parameterized by the location *p*, V(p). Its equation can be seen in *Figure 2(3)*. The value function is the ideal evaluation of the action chosen by the actor; it is not given to the model and it must learn and adjust the weights so that the output function $\mathrm{C(p)} \rightarrow \mathrm{V(p)}$. The value function of the move/position can be

modeled as the summation of a Boolean reward function with a "discount factor" hyperparameter of the steps from now to when the goal is found given that it takes *T* steps:

$$V(p_t) = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots = \sum_{t}^{T} \gamma^{1++} R_t$$

Where $R_t = \begin{cases} 1 \text{ if goal reached} \\ 0 \text{ else} \end{cases}$ is the Boolean reward function where if the current position is the goal/platform, it returns a 1, else it returns a 0. The discount factor γ should range between $0 < \gamma < 1$ to penalize when the reward is found at later timesteps, decreasing the overall reward earned from the Boolean reward function as the number of timesteps to reach the goal increases. It is an arbitrary hyperparameter, and the value of 0.9 was chosen for the model implementation. Thus, the value function (critic) punishes the "actor" from choosing needless actions that would increase the number of steps to reach the goal, penalizing movement that is in the wrong direction.

The value function can be used to derive the prediction error between C(p) and V(p) from the weights. Given that the value function is an incrementing series, it can be expressed as a recursive function:

$$V(p_t) = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} = R_t + \gamma(R_{t+1} + \gamma R_{t+2}) = R_t + \gamma V(p_{t+1})$$

In addition, given that the goal is for $C(p)$ to converge to $V(p)$, the same can be expressed for the critic cell's output:

$$C(p) \to V(p) \implies C(p_t) = R_t + \gamma C(p_{t+1})$$

Therefore, given that the critic output function is approaching, the following equation for the prediction error for C(p) in *Figure 2(4)* can be derived:

$$C(p_t) = R_t + \gamma C(p_{t+1}) \;\&\; C(p) \text{ not converged to } V(p) \implies \delta_t = R_t + \gamma C(p_{t+1}) - C(p_t)$$

The model then can adjust the weights according to the prediction error, seen in *Figure 2(5)*, and the multiplication by the place cell activation function (*Figure 2(1)*) ensures that only the place cells that were activated are adjusted by the prediction error. Consequently, the critic output cell is able to learn from the actions the actor cells take, and in response adjust the output function to learn a proper value function for each position by reinforcing good actions while penalizing actions that results in increasing the escape latency or path length for the navigation model; a case of reinforcement/reward-based learning.

The paper described the model being capable of taking eight different directions at each timestep for convenience of computation, seen in *Figure 2(1)*. The actor in the model is composed of eight action/direction cells. Similar to the critic output cell. The eight actor cells also have an output function as well as a weight adjustment mechanism, seen in *Figure 3*.

$$(1)\quad a_j(\mathrm{p}) = \sum_i z_{ji} f_i(\mathrm{p}) \qquad (2)\quad P_j = \frac{\exp(2a_j)}{\sum_k \exp(2a_k)} \qquad (3)\quad \Delta z_{ji} \propto \delta_t f_i(\mathrm{p}_t) g_j(t)$$

Figure 3: (1) the activity function for the direction j. This is again the summation of the weight place cell input function, with the distinction from the critic being that w_i is replaced with z_{ji}, the weight from place cell i to each individual action cell j. (2) then the direction taken by the actor is stochastic/probabilistic, each direction probability given as the fraction of the activity value for direction j over the total activity value of all eight directions. (3) the weight adjustment relation.

Each individual direction cell at each position *p* have a probability of being chosen, defined by P_j seen in *Figure 3(2)*. It can be interpreted as the strength/level of preference that the direction is chosen given the position. The reason the direction is chosen in a stochastic mean is to force the actor to "explore" and try various directions to learn. Thus, the weights from the place cell *I* to each action cell *j* can be modified similar to the critic weight adjustment relation, seen in *Figure 3(3)*. Without loss of generality from the critic weight adjustment relation, the change in

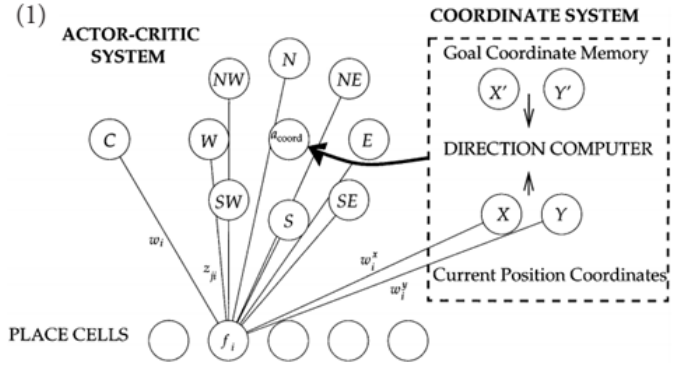
weight can be derived with the addition of a Boolean function $g_j(t) = \begin{cases} 1 \text{ if direction chosen} \\ 0 \text{ else} \end{cases}$ to ensure that only the direction taken is modified accordingly.

This results in an interesting situation of exhibiting a form of Hebbian learning. If the action results in a negative prediction error $\delta_t < 0$ in the critic cell, assuming that the reward was not reached ($R_t = 0$), implies that $C(p_t) > \gamma C(p_{t+1})$:

$$\delta_t = 0 + \gamma C(p_{t+1}) - C(p_t) \Longrightarrow C(p_t) + \delta_t = \gamma C(p_{t+1}) \Longrightarrow C(p_t) > \gamma C(p_{t+1})$$

Subsequently, as the critic weight adjustment relation (*Figure 2(5)*) results in $C(p) \rightarrow V(p) \Longrightarrow V(p_t) > \gamma V(p_{t+1})$; the prediction error obtained from taking that action decreases the value function. A negative prediction error would also decrease the activity function of the action cell give the specific place cell, decreasing the probability of that direction being favoured. The converse, $\delta_t > 0$ would then imply that $V(p_t) < \gamma V(p_{t+1})$ without loss of generality, meaning that the action increases the value function. Subsequently, the action cell weight adjustment relation, seen in *Figure 3(3)* would then increase the activity function in *Figure 3(1)* and increase the probability of that direction being favoured/activate more often. Consequently, this results in a situation where the edge/synapse between the input place cell vertex and an actor's action cell vertex is strengthened if they both fire (activate) and results in an improved value function, while the edge between is conversely weakened if they both fire and results in a decreased value function. From the mathematical description from above, a function for running the simulation can be implemented. It is implemented as `Simulation.run_simulation()`.

II) Combined Coordinate & AC Model



$$(2)\quad X(\mathrm{p}) = \Sigma_i w_i^X f_i(\mathrm{p}) \qquad Y(\mathrm{p}) = \Sigma_i w_i^Y f_i(\mathrm{p})$$

$$(3)\quad \Delta w_i^X \propto (\Delta x_t + X(\mathrm{p}_{t+1}) - X(\mathrm{p}_t)) \sum_{k=1}^{t} \lambda^{t-k} f_i(\mathrm{p}_k)$$

$$\Delta w_i^Y \propto (\Delta y_t + Y(\mathrm{p}_{t+1}) - Y(\mathrm{p}_t)) \sum_{k=1}^{t} \lambda^{t-k} f_i(\mathrm{p}_k)$$

Figure 4: (1) an illustration of the combined coordinate and actor-critic model. Figure is obtained from the paper. The model adds to the pure Actor-Critic system with an inclusion of two new output nodes from each place cell (X and Y). It attempts to learn the global coordinate representation of the current position given the place cell activation. The goal coordinate memory vertices are a memory cell storing the coordinate which the last platform was found. The DIRECTION COMPUTER is a means to compute an output direction given the coordinates of the model and the last remembered goal position. (2) Learning mechanism for X and Y coordinates. Just like previously, it is a weighted sum of the place cell activation. (3) the weight adjustment relation.

An issue encountered by the pure Actor-Critic model is that there is no mechanism in place such that the experiences of previous trials contribute to learning new platform positions. However, this is not true in real behavioural observations; previous experience is used by mice to guide their spatial navigation. Animals are, generally speaking, able to generalize from their initial experiences and improve future performances. As a reminder, this is observed in *Figure 1*. This indicates that mice are capable of generating an allocentric – independent from their starting position – representation of the maze/test environment over trials. Consequently, the combined coordinate and actor-critic model proposes a method to acquire a coordinate system independent of the starting point of the simulation from the model's self-motion. In other words, the model needs to have a system in place to learn a global coordinate system from relative self-motion derived from the series of place cell activations. The model illustration is shown in *Figure 4(1)*.

Two new output cells (for the X and Y dimension respectively) are added in the combined model to learn global coordinate representation of the current position based on the place cell firing rate, seen in *Figure 4(2)*. Both cell outputs are the weighted summation of place cell activity. An important thing to note is that although for the purpose of illustration they are labelled X and Y, the only requirement is that the coordinate system is derived from two orthogonal unit vectors/directions. Unfortunately, this adds variability per simulation.

On the assumption that the movement of mice in the maze is constant, which the paper states for practicality, the difference in global coordinate after a timestep is equivalent to the relative self-motion per timestep. Consequently, it can be exploited to implement weight adjusting relations. Shown in *Figure 4(3)*, it can be achieved by once again doing a weighted summation of place cell activity from the initial timestep to current, with the λ hyperparameter determining the penalizing factor to earlier activity patterns, given that it is to the power of $t - k$. In other words, the value of the hyperparameter shapes to what extent are past timesteps considered. For the paper, a value of $\lambda = 0.9$ was arbitrarily chosen.

Then, the $(\Delta \mathrm{x_t} + X(p_{\mathrm{t+1}}) - X(\mathrm{p_t}))$ factor is sum of the difference between current and the next timestep in X, with $\Delta \mathrm{x_i}$ equaling the self-motion estimate, which in this context becomes equivalent to the difference in global coordinate after the timestep. Essentially, it functions the same as the prediction error for the critic, noting the similarity from *Figure 2(4)*:

$$X(\mathrm{p_t}) = X(\mathrm{p_{t+1}}) + \Delta \mathrm{x_t} \implies \mathrm{Error} = (\Delta \mathrm{x_t} + X(\mathrm{p_{t+1}}) - X(\mathrm{p_t}))$$

The same reasoning for the equations is applied for the y coordinate output cell without loss of generality.

To resolve some edge cases, when there is no goal coordinate stored in memory (i.e. the first trial), the coordinate controller is hard-set to specify a random exploratory direction. The coordinate system has a single tertiary layer output: the a_{coords} cell which is superimposed on top of the actor's action cells. In addition, to maintain consistency in learning, if the coordinate model's action is chosen, the weights of the system is modified by the prediction error generated by the critic, using the same formula mentioned in the critic section.

III) The Maze Environment

A python class for the watermaze environment was implemented for the purpose of the simulation. To keep the implementation simple, the eight directions possible was defined by numbers ranging from 0-7 in `self.direction`. Six class functions are implemented to simulate a minimum requirement version of a watermaze environment.

The first class function is to implement a way for the model to move in the space given a direction input (`self.move(Action)`). First, the cosine and sine of the direction is obtained, followed by adding in a momentum to emulate swimming and not just instantaneous movement derived from previous direction. Then an edge case is tested to see if the model is attempting to leave the watermaze bounds (`if (np.linalg.norm(newposition) == self.radius)`), which then the model is coded to reflect/bounce off the wall. Finally, the position, timestep, and previous direction is updated.

The second function implemented the bouncing mechanic. It first determines where in the global coordinate system the rat hit the wall. Then the tangent vector from a $-\frac{\pi}{2}$ radian rotation from the direction vector is calculated. Subsequently, the angle between the direction

vector and the tangent vector (`theta`) is calculated and applied to get the reflected direction and position, where the angle is defined to be: `2*(np.pi - theta)`.

The other four functions are short functions. The third function is a helper function for the wall bouncing mechanic implemented. It is used to determine the point in the global space where the rat hits the wall. The fourth function sets the starting position randomly. The fifth function checks if the runtime of the simulation finished. The final function simply checks if the coordinates of the model is the same as the goal coordinates, if the simulated mice is on the platform.

Simulation Results & Discussion

I) Single Platform

Simulation results of both the AC and the combined coordinate and AC model is shown in *Figure 5*. A general trend of both models successfully learning given a single platform location is observed. Notably, less variance is observed in the combined model as the model efficiently learns the correct pathing. The line indicates the average from the 10 runs per trial to simulate multiple mice in the simulation. The Actor-Critic simulation shows that given only place cell activations, a model is able to effectively learn to take more efficient paths across trials. This indicates that given only localization information from place cells, knowing what place field a model is in, it can effectively learn as a navigation model. In relation to experimental data, this implies that place cells in the hippocampus – giving a representation of current position of the mice via place fields – is capable of providing navigational qualities given the environment.

A potential issue in navigational models, the *distal reward problem*, is not observed in the actor-critic and combined model. Given that the reward is associated to place cells (or in

general, a particular coordinate in abstract navigation systems) via conditioning, navigation models can face the issue of lacking a means to propagate goal information to improve navigation behaviour (Foster, Morris, & Dayan, 2000). In other words, as most locations in an environment is not near the goal location, only the place cells/coordinates near a goal have a level of activity as a reward. Consequently, if a model is to initialize too far away from the objective, there is no direct information to decide on a correct direction to start.

A benefit of establishing a learning rule using a temporal difference is effectively bypassing the highlighted issue a navigational model using a pure reward system may face. An explanation to why is not provided in the paper, hence the following is examination given the mathematical properties. The issue with the *distal reward problem* is that the information about the reward is only apparent when approaching or near the goal position. However, the implementation of a critic to generate an output function, learning an unknown value function V(p), which changes per position, a temporal difference in model performance is made regardless of the position. Consequently, the model relies upon the temporal difference of the expected value function between sequential locations as a substitute for a pure position-reward mechanism. Learning can continuously occur. The critic theoretically values positions and states closer to the goal than positions farther away; once $\mathrm{C}(\mathrm{p}) \rightarrow \mathrm{V}(\mathrm{p})$, the TD prediction error shapes actor action cells to correctly reinforce the directions that move the model closer to the goal coordinate, and supress incorrect action cells that decrease the value function (moving away from the goal) regardless the position. This is observed in *Figure 5*, where after approximately ~20 trials, the model is able to consistently have a low path length, indicating a good critic output function convergence to the value function and easily finding the goal.

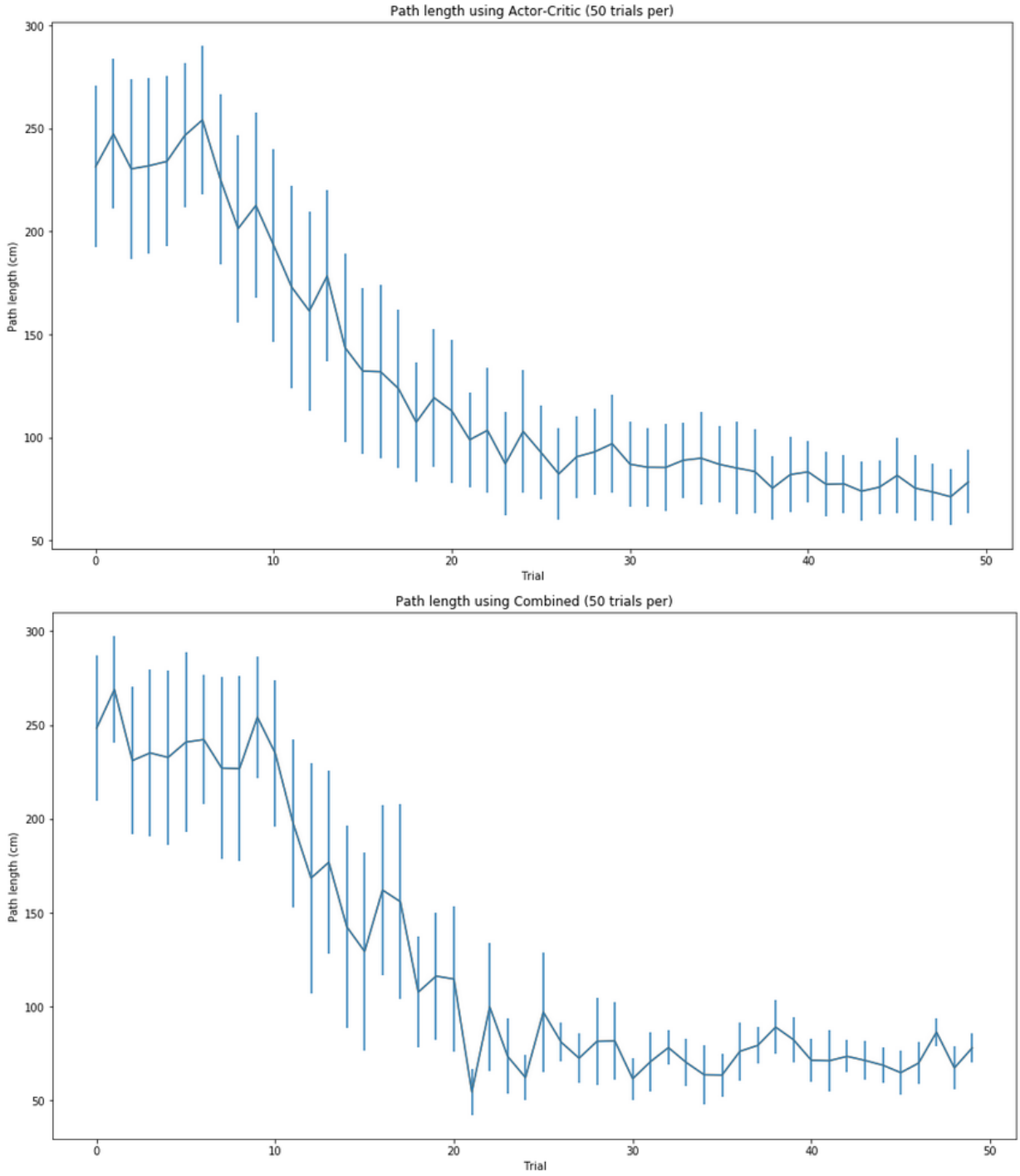


Figure 5: Pathlength from starting position to goal/platform over trials. Per platform position 50 trials are conducted, with N=10 runs per trial to emulate having 10 separate mice. The error bars represent the standard deviation. For debugging and standardizing purposes, the platform is positioned at [15,15] XY coordinates. If desired, a random integer position can be generated with:

```
platform coords=random.randint(-60, 60, size=(1,2))
```

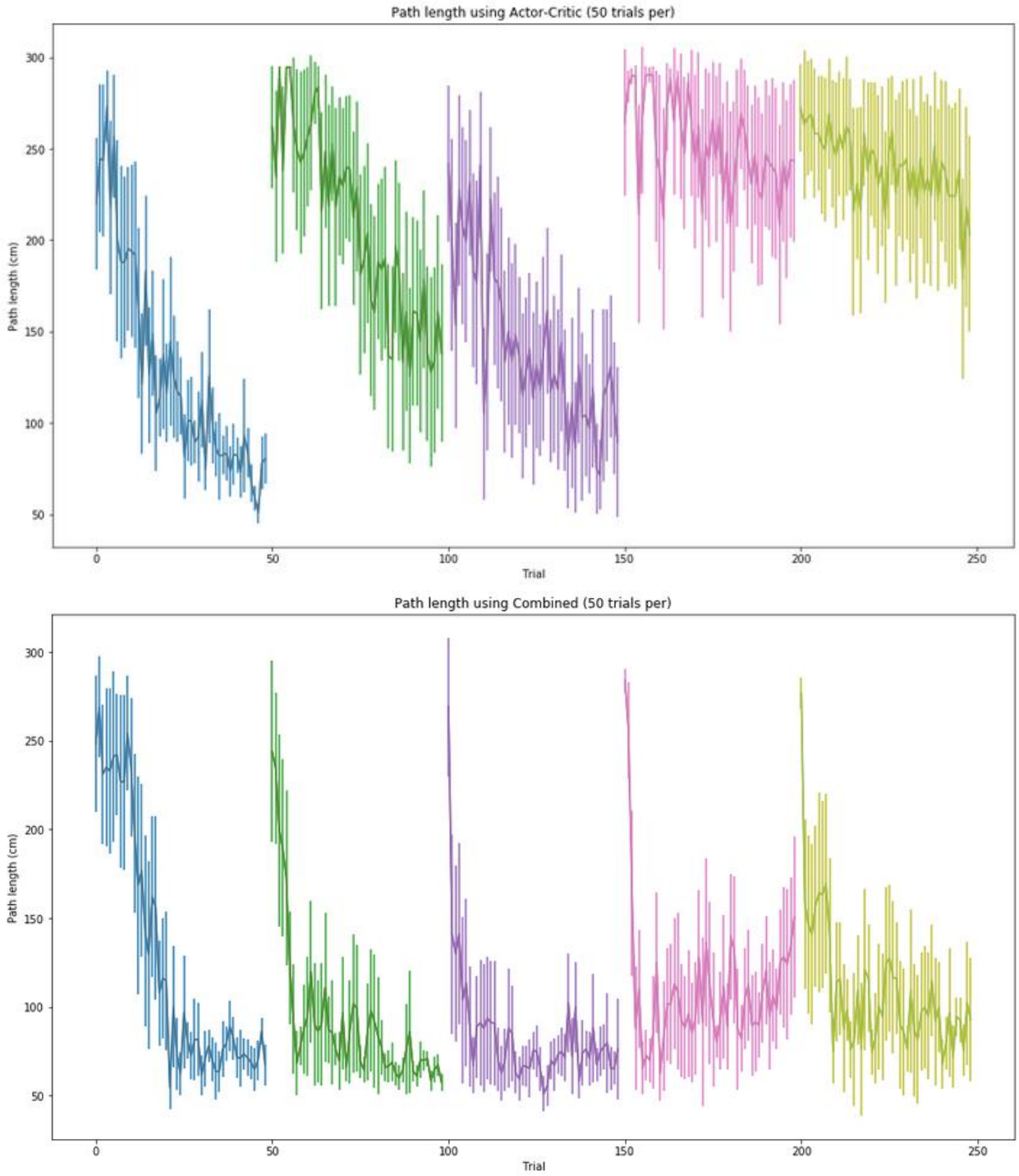


Figure 6: Pathlength from starting position to goal coordinate over trials. Per platform position 50 trials are conducted, a new platform position is set afterwards: [[15,15], [-15,-15], [30,10], [-10, 30], [-40,20]]. This is arbitrarily set to standardize and for debugging. If random platforms are desired, the following code should be implemented:
`platform_coords=random.randint(-60, 60, size=(5,2))`

II) Changing Platforms

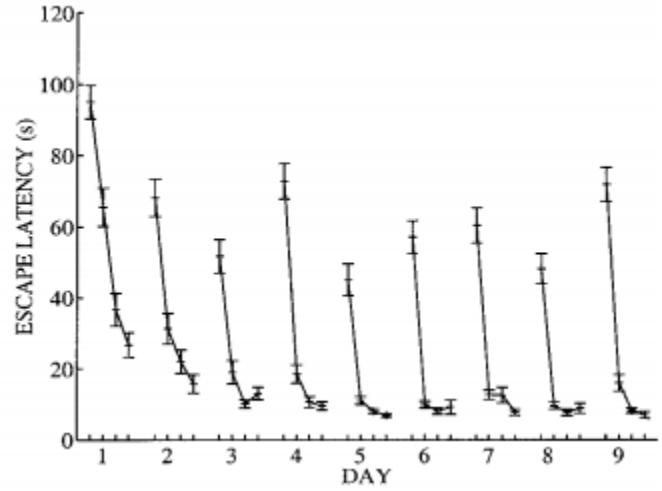


Figure 7: Experimental results of mice escape latency given changing platforms (multiple platforms). Animals are able to learn changing platform locations on successive days; this implies that they avoid interference from previous knowledge and learn consistent global coordinates. Figure is from the paper.

Evidently, seen in *Figure 6*, the Actor-Critic model fails to emulate the experimental rat data seen in *Figure 7*, while the combined coordinate & AC model succeeds. This is the expected result. An issue solely relying on a temporal difference in reward value is highlighted. The lack of rapid improvement on platform 4 and 5 on the AC model indicates that the model incorrectly predicts that learning a new platform position is supressed due to previous knowledge. The failure of the AC model adapting to new locations can be explained by the mathematical foundation of value function: the learned value function is a bijection between spatial information and reward information. In other words, it is a one-to-one mapping function between specific spatial values and reward. As a result, the value function and the actor's chosen policy/direction can not adapt to a changed reward location; the model's previously learned values interfere with learning new information. The AC model produces a goal-dependent egocentric realization of the environment. It acquires inconsistent coordinates of the environment as a whole and thus has a difficulty learning when encountering a new goal position drastically different than the previous location. This issue is called the *global consistency problem*.

The coordinate model provides a means to generate a goal-independent representation of the environment coordinates. The *global consistency problem* is an issue for a navigation system that relies on temporal difference in reward/objective to build a representation of the environment. However, given the fact that mice are able to adapt to changing platform locations in *Figure 7*, and that animals are able to exhibit a natural basis of learning coordinates from their relative motion (Foster, Morris, & Dayan, 2000), suggest that a neural coordinate representation does exist in the brain. The simulation illustrates that the combined coordinate and AC model sufficiently emulates the experimental result in escape latency.

The mechanism to which the combined model is able to overcome the *global consistency problem* is as follows: from the coordinate system, the agent is able to move towards a goal memory (using X' and Y' cells) regardless of the value function learning in previous trials. In addition, by attempting to establish a globally consistent coordinate system using place cell activities, the AC system of the model effectively relearns the value function and the preferred policy/direction for each place field. The issue of critic's previously learned value function and the actor's policy preference transferring is effectively prevented; interference is no longer apparent. As shown in *Figure 6*, the model effectively learns as platform positions change.

The combined coordinate and AC model is a mathematical model showcasing how hippocampal place cells can contribute to spatial navigation in animals. It is an example of how a global coordinate system can be constructed in the brain given only relative positional information from the history of place cell activation. However effective the combined model may be in modelling a two-dimensional navigation of mice, it is not generalizable to animal

navigation in general. The model assumes a flat topological environment, a two-dimensional space with no barriers, nor is potential change in elevation or other environment features considered. In addition, the watermaze environment ensures that a direct path from the starting position to the goal is available. The mathematical definitions also employ hyperparameters which control the learning process itself, and those variables would change depending on the environment and situation given. The use of constant hyperparameters (γ, λ) is unreasonable given the consistently changing environment an animal encounters; an implementation of a dynamic hyperparameter as a function of the environment is needed. Consequently, the model offers no insight to how animals act in topologically richer representations of space.

Overall, the exploration of the hippocampally dependent navigation model using a temporal difference learning rule showcase the possibility to develop a neurobiologically feasible means of a two-dimensional navigation system solely using hippocampal place cells, which are implicated in spatial navigation in the brain. The combined coordinate and AC model is a feed-forward neural network of two output systems working in parallel, the Actor-Critic system and the coordinate system receiving input from place cell activations to successfully emulate mice in the watermaze task. It is able to successfully emulate the experimental data shown in *Figure 1* and *Figure 7*. While the model can not be extrapolated or generalized to a more topologically rich scenario, it is an interesting example of how reinforcement learning – in the form of temporal difference in the value function – can be applied to the hippocampus as a learning mechanism.

Appendix

Appendix A: the class variables of the Simulation class

```
def __init__(self,algorithm, eta_a=0.1, eta_c=0.01, eta_xy=0.01, d_f=0.9):
        # *** constant vars ***
        self.N = 493 # number of place cells
        self.sigma = 16 # gaussian width for place cells
        self.Lambda = 0.9 # place cell history in coord model

        # *** changing vars ***
        self.goal_xy = [] # goal coordinates
        self.a_c = 0 # preferred action coord

        # *** identifiers ***
        self.maze = None
        self.algorithm = algorithm

        # *** weights ***
        self.w_c = np.zeros(self.N) # critic weights
        self.w_a = np.zeros([8, self.N]) # actor weights
        self.w_x = np.zeros(self.N) # x-coord weights
        self.w_y = np.zeros(self.N) # y-coord weights

        # *** learning ***
        self.eta_a = eta_a # actor learning rate
        self.eta_c = eta_c # critic learning rate
        self.eta_xy = eta_xy # coordinate learning rate
        self.d_f = d_f # discount factor

        # *** place cell coordinate generation ***
        self.s = np.zeros([self.N,2])
        self.create_space()
```

References

Brown, M. A., & Sharp, P. E. (1995). Simulation of Spatial Learning in the Morris Water Maze by a Neural Network Model of the Hippocampal Formation and the Nucleus Accumbens. *Hippocampus*, 171-188.

Foster, D., Morris, R., & Dayan, P. (2000). A Model of Hippocampally Depenent Navigation Using the Temporal Difference Learning Rule. *Hippocampus*, 1-16.