Final Project
Simulation of the Morris Water Maze using Reinforcement Learning

ECE 517 - Reinforcement Learning

1st Christoph Metzner
The Bredesen Center
The University of Tennessee
Knoxville, USA
cmetzner@vols.utk.edu

2nd Andrew Strick
Department of Kinesiology, Recreation & Sport Studies
The University of Tennessee
Knoxville, USA
astrick@vols.utk.edu

***Abstract*—Studying the ability of spatial navigation facilitated by the hippocampus in humans and animals usually involves the use of water maze tasks, such as the classical neuroscience Morris Water Maze experiment. This tasks requires the study subject to find a hidden platform with a reward in a pool of cloudy water. Studies using living animals are costly and time-intensive. A practical solution to overcome these constraints is the application of reinforcement learning methods to model the behavior of animals in the water maze. This work applies double deep Q-learning to simulate a rats (agents) behavior when confronted with the task of finding a reward platform. Specifically, the task of transfer learning, using learned knowledge to navigate in a novel environment under the conditions of blocked and random practice. The main results are that agents trained with blocked practice and constrained to a transfer environment with the same practice condition are faster and more stable in finding the optimal policy compared their counterparts trained with random practice.**

***Index Terms*—reinforcement, deep learning, spatial navigation, morris watermaze**

I. Introduction

In motor behavior, the environmental context is a key element in understanding how people or organisms learn [3]. One of the key components in evaluating the environment is via spatial navigation. In order to investigate the importance of spatial navigation many neuroscience experiments placed rats or mice in obstacle courses and saw how they behaved. Further research led to what is now known as the classic Morris watermaze [4]. What Morris tested was to see if spatial navigation and memory are held in the hippocampus when rats are searching in a maze. The maze design was circular, filled with water with an island at the center. On the island the rats would receive some food reward for navigating the aqueous terrain. What Morris found was the hippocampus activation changed based on the conditions the rats were in (blocked versus random practice).

In motor learning and neuroscience research blocked and random practice signify two ends of a continuum. Blocked practice can be thought of as repeated the same task with the same environmental constraints before moving onto a new task. Random practice on the other hand can be thought of as two or more separate tasks done in non-repeating interlevened format. For instance, if there are three separate tasks *A*, *B*, and *C*, blocked practice would follow the following format (*AAA*, *BBB*, *CCC*), whereas random practice would follow a random assignment of tasks (*AAC*, *BCA*, *CBB*) [3].

Taken the concepts of spatial navigation and practice conditions together Morris found that rats that had started in random locations in the water maze learned the location of island better than rats starting at the same location. The learning took longer for the random group but it lead to a better spatial abstraction of the water maze than starting a fixed location.

II. Previous Work

Previous work comparing the effects of blocked and random practice have been thoroughly studied. The effects of in humans have been studied in a variety of tasks such as spatial navigation, coincidence timing tasks, golf putting, and pursuit tracking [5]–[8]. From motor behavior and neuroscience work we have found an odd effect where the random practice group performs poorly in practice but does better on the test compared to a blocked practice group. This effect is known as the contextual interference effect [11]. In order to understand the fundamental neurological mechanisms underpinning the findings in motor behavior, neuroscience researchers have primarily used rats and mice using the Morris watermaze paradigm [4]. Since the inception of the water maze experimentation many researchers have tried a variety of changes to the environmental context [9]. Others have focused on lesions to the brain in order to understand which brain structures are important for learning spatial navigation and memory [10]. Regardless, of the methodology used the findings were clear that the hippocampus is involved in spatial navigation, learning and memory.

However, none of these experiments have been able to test 2D to 3D transfer of learning effects, nor controlled for each

condition separately. In order to investigate this premise we first aim to recreate the Morris watermaze experiment using reinforcement learning. Second, following previous research we will create two separate practice conditions (Blocked and Random) and test the agents to navigate a 2D pool to find a reward platform located centrally. Moreover, we will investigate the transfer of learning effect from a 2D to a 3D environment and see if the agents can generalize the 2D context to a 3D space.

III. Description of the Markov Decision Process

The problem can be framed as a Markov Decision Process (MDP). The MDP = {S, A, R, P} can be described by its state space (S), action space (A), rewards (R), and transition probabilities (P).

A. States - S:

The state space of the present MDP is discrete and depends on the dimension d of the environment, i.e., the size of the maze. The environment is designed to be either a square 2-dimensional grid or a 3-dimensional cube, where each cell in the environment represents one state. The state space comprises the local position of the agent which are x, y coordinates in 2D or x, y, z coordinates in 3D. In both environments, the location of the platform and the spatial cue area are unknown to the agent. At the start of each single experiment, the location of the platform was placed in the center of the environment and the larger spatial cue area superimposed on that location. If the agent decides to move outside the grid, the agent returns to its previous position, i.e., previous state.

The indices of any state are as follows:

2D-Grid: **S(y-position of agent — x-position of agent)**
3D-Cube: **S(y-position of agent — x-position of agent — z-position of agent)**

B. Actions - A:

The action space is discrete and consists of either four, in the case of the 2-dimensional grid, or six, for the 3-dimensional cube, actions. In 2D-environment the agent can move from its current location to any adjacent cell via these four directions *north*, *south*, *east*, or *west* direction. In addition to these actions, the agent can move either *up* or *down* one cell in the 3D-environment.

C. Rewards - R:

The reward structure was designed to incentivise the reinforcement learning agent to emulate the behavior of rats moving and searching for rewards in a water pool. Therefore, specific rewards were introduced that simulate energy exhaustion due to swimming or diving, the potential threat of dying if the platform (terminal state) was not found, agent moving inside spatial cue area (i.e., smelling or seeing the cheese), and the agent finding the platform (i.e., finding the cheese). The inclusion of a reward structure simulating a spatial cue area was thought of an attempt of generalization and help the agent understand, where the location of the platform potentially could be. If the agent moves towards the platform inside the spatial cue area the reward would increase. Preliminary studies showed, however, that such a reward must be carefully designed to avoid the agent exploiting it. This issue during training is highlighted in Section VII-A1. The main rewards are listed below:

- Positive reward for finding the platform: $R = 500$
- Negative reward for not finding the platform: $R = -1000$
- Gradually decreasing reward for moving: $R_{t+1} = R_t - 0.01$
- Gradually increasing reward for moving towards the platform if inside the spatial cue area: $R_{t+1} = R_t + 2$
- Gradually decreasing reward for moving away from the platform if inside the spatial cue area: $R_{t+1} = R_t - 2$

D. Probabilities - P:

The dynamics of the environment are unknown. The only thing that is known is that the probability of moving from one cell taking a specific action is one.

- $P_{move} = 1$

IV. Simulation Experiment Setup

The simulation experiments aim to study the ability of transfer learning of reinforcement learning agents. Specifically, we wanted study the agent's ability to generalize its strategy in finding the reward platform from 2D-environment to 3D-environment. The learning ability is stored in the trained weights of the neural networks of each agent. In order to test the transfer of learning in each condition, the blocked and random practice groups in 2-D were tested to the same conditions in 3-D. Furthermore, the experiments are setup in a way to emulate blocked and random practice. Agents faced with blocked practice will always begin their search on the same starting position, whereas agents under the condition of random practice will start their search on any possible location in the environment. The dimension of the environments is set to 12. The 2D-environment is a square, while the 3D-environment is a cube. The starting position in blocked practice was set to (0, 0) in 2D and in 3D to (0, 0, 0).

V. Reinforcement Learning Methods

The ability of spatial navigation in humans and non-human animals is controlled by the hippocampus of the brain. In our study, the neurons of the hippocampus is simulated using deep neural networks, such as the deep Q-network. A deep Q-network uses a multi-layered artificial neural network to output a vector of action values $Q(s, .; \theta)$ for a given input state s, where θ are the parameters of the network. Deep Q-learning suffers from overestimation bias (or maximization bias), that is, the algorithm overestimates the Q-values used for the update rule by always selecting the action that maximizes the Q-value using the same network weights that we are trying to evaluate. The addition of a second network with sole purpose of selecting the appropriate action used for the update combats this overestimation bias. This algorithm is known as

double deep Q-learning, which we apply in our simulation [1]. The first network is used to evaluate the weights in an online fashion, see equation 1. The second network, the target network, is used to compute the action used for the weight update, see equation 2. The weights of the target network are periodically updated by copying the weights of the online network every n episodes.

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta_t))\nabla_{Q_t} Q(S_t, A_t; \theta_t) \quad (1)$$

$$Y_t^{DoubleDQN} \equiv R_{t=1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t), \theta_t^-) \quad (2)$$

VI. Data Structures and Code Design

In this section, the main design choices for the data structures and for the developed functions of the code are shown.

A. Data Structures

The state space is a single list of 2 or 3 indices depending on the dimensions of the environment. The actions are individual numbers ranging from 0 to 5. The rewards are float numbers.

B. Code Design

In the following section all major functions of the code are introduced and explained. For a more detailed representation of the code, the reader is referred to the appended code. The code is split in two modules, 1) executing the main learning script (main.py) and 2) a script containing all learning, memory methods of the DDQN algorithm (DDQN.py). The source of the Python3 implementation of the DDQN network follows the tutorial of. the Github repository from Phil Tabor.

1) Function - `get_reward`*:* This function is used to indicate what the reward will be for the agent given the location of the agent, the smell range, and the platform.

2) Function - `platform`*:* This function is used to produce the lower and upper bound indices of the reward platform. In a 2-D environment the platform will be a square. In a 3-D environment the platform will be a cube.

3) Function - `smell_range`*:* This function returns the lower and upper bounds of the a reward smell range for the agent. Like the platform function the smell range will produce a square in 2-D and a cube in 3-D. The smell range can not be smaller than the platform in order to function properly in either environment.

4) Function - `center_location`*:* This function returns the indices for the center of either the 2-D or 3-D environment(s). This function is used as a utility for the following manhatten distance function.

5) Function - `manhatten_dis_center`*:* This function returns the manhatten distance (L1 norm) from the agent's position to the center of the environment.

6) Function - `get_next_state`*:* This function is used to move the agent. In a 2-D environment the agent is allowed four possible actions selections (*north*, *south*, *east*, *west*). In a 3-D environment the agent has all action from the 2-D space with the addition of two more actions (*up*, *down*). If the agent is on the border or boundary of the environment and tries to move beyond, the agent will return to its location. The agent must move until it reaches the terminal state or until it exhausts the number of steps allowed per episode.

7) Function - `get_terminal`*:* This function checks if the next state of the environment is terminal or not. In other words, it checks once the agent moves if the agent moved onto any point of the platform.

8) Function - `init_starting_state`*:* This function initializes the position of the agent. It checks if a starting state is given or not. If a starting state is given, the agent starts on position $S_{2D} = [0, 0]$ or $S_{3D} = [0, 0, 0]$ for all episodes. Should that not be the case, the agent will start on random locations in the environment.

9) Function - `execute_learning`*:* This function performs the learning for n episodes. Here the function, specifically calls DDQN method. This functions stores the calculated returns per episode in an array and returns the array of all the calculated returns for plotting the learning progress.

10) Function - `plot_episode`*:* This function plots the episode given a starting state and a trained policy. First, a trajectory is sampled until the terminal state is reached. Second, each visited state since the starting state is plotted for the sampled trajectory from the first step.

11) Function - `plot_total_returns`*:* This function plots the total return per episode for all episodes of training.

12) Function - `init_ddqn_agent`*:* This function initializes the double deep Q-learning agent depending on the input dimensions, i.e., whether the environment is a 2D-grid or a 3D-Cube. The agent follows an ϵ-greedy policy. The DDQN agents are initialized with certain fixed hyperparameters, which are the following:

- Learning rate - α: 0.0005.
- Discount factor - γ: 0.99.
- Number of actions: 4 (2D) or 6 (3D).
- Exploration factor - ϵ: 1.0, which is gradually decreased to a minimum of 0.01.
- Batch size: 64.
- Input dimensions of environment: 2 or 3.

13) Class - `ReplayBuffer`*:* The class object *ReplayBuffer* is used to maintain a memory of all the transitions an agent had made during training. A individual transitions contains is a tuple with information about the visited state, selected action, received reward, moved next state, and whether the next state was terminal or not. This class has the two methods `store_transition` and `sample_buffer`. The first one stores the transitions after each move during training and the latter one is used during weight updating by sampling a batch of memorized transitions.

14) Function - `build_DQN`*:* This function helps to build a sequential deep learning model using tensorflow and keras.

The models are rather shallow and possess three dense layers, where the last one is the output layer. The dimensions of the layers are 128. The used activation function is rectified linear unit or *Relu*. Adam was selected as the optimizer with a learning rate specified in section VI-B12 and the *mean-squared-error-loss* for the loss function.

15) Class - `DDQNAgent`: This class builds the double deep Q-learning agent in the environment. This object performs the learning with method `learns`, the *remembering* of past transitions using the method `store_transition` of the object ReplayBuffer, chooses the action with method `choose_action`, and updates the target network weights by copying the weights of the evaluation network using method `update_network_parameters`. The method `learns` applies the double deep Q-learning algorithm and weight update.

16) Function - `set_weights_3D_2D`: This function sets the weights of an untrained agent initialized to find the platform in 3D space for the x- and y- inputs equal the trained weights of the agent trained in the 2D-grid environment. It also updates the weights connected with the output layer for the first four actions (form 0 to 3, or actions north, south, west, and east).

VII. Results Analysis

In this section, the simulation results of the Morris Water Maze using Reinforcement Learning to model the agent are presented. A total of four experiments were conducted. The success of these experiments is defined as the agent's ability to learn in terms of time, that is, the higher expected return for one episode the better. Since the position of the platform is unknown the agent has to randomly search the state space for the platform or in a greater extent a spatial cue area surrounding the platform. The spatial cue area is the only external element in the environment providing the agent a positive feedback and the property to lead the agent to towards the platform.

The following section first talks about the main issues during the training process that were the agent staying in the spatial cue area to accrue large positive rewards and not being able to find the spatial cue area and thus the platform in 3D environment. The second subsection provides the results of the experiments for agent trained in blocked and random practice for both the 2D- and 3D-environment.

A. Issues during Training

The main two issues that were encountered during training are talked about in this section. After careful investigation the following list provides a quick overview of the potential causes for those issues.

- Ratio between the size of the spatial cue area and the environment was too large.
- Ratio between the size of the platform and of the spatial cue area was too small, i.e., the environment helped the agent to little to accomplish its goal.
- Unfavorable design of the reward structure to guide the agents actions and movements in the environment
  - Gradual increase and decrease of reward inside the spatial cue area moving to or away from platform
  - Reward for reaching the platform
- Type of training condition being blocked or random, i.e., starting position of agent

1) Agent decides to stay in spatial cue area: A main issue during the training process was the agent spending all its moves in the spatial cue area and accruing a large number of reward for each move. The old design of the reward structure gave gradually increasing (or decreasing) rewards to the agent for advancing closer (further) to the platform. However, the agent learned a behavior that enabled it to increase the reward indefinitely without ever going to the platform until all moves per episode were exhausted. This resulted in unintended very large expected returns per episode (see Figure 1).

In attempt to alleviate this issue, we decided to increase the terminal reward significantly to +500. In the same breath, the reward for exhausting all moves during one episode, i.e., the agent dying, were changed from -200 to -1000. The design of the reward structure for the agent moving inside the spatial cue area was redesigned. Previously, the reward structure was that it would increase by Reward = Reward + 2 for each step the agent would make in the spatial cue area. Now the formula made in the current code only allows for a subsequent increase for a reward in the spatial cue area if the agent moves toward the platform (which occurs when the agent is in the spatial cue area or coming from outside into the spatial cue area). Also, the reward will decrease or return to original values if the agent comes into the spatial cue range and then leaves it.

2) Agent not able to find the platform in 3D-Environment: The second main issue during training was the issue with the transfer of learning from 2-D to 3-D environments. Simply put, in our original design the agents could not find the platform or spatial cue area. This could be due to a variety of factors. The agent's way of finding the platform is by randomly searching the environment for cues. Therefore, as the environment becomes larger, by adding more dimensions, the agent's chance of randomly stumbling across the spatial cue area becomes smaller. This leads to the next factor, the size of the platform and spatial cue area in relation to the size of the current environment. If the sizes of the platform and spatial cue area remain the same in 3D-environment as they were in 2D-environment, they cover a much smaller percentage of the total space of the 3D-environment. In alleviating this problem, we decided to reduce the dimensions of the environment to 12 (i.e., 12 cells per dimension) and increase the dimensions of the platform and spatial cue area to cover roughly the same percentage of space in 3D as they did in 2D-environment. If the spatial cue area is not big enough the agent may never be able to find the platform unless stumbling across it randomly.

B. Simulation Results

The following section presents and discusses the experimental results of the simulation for the task of transfer learning

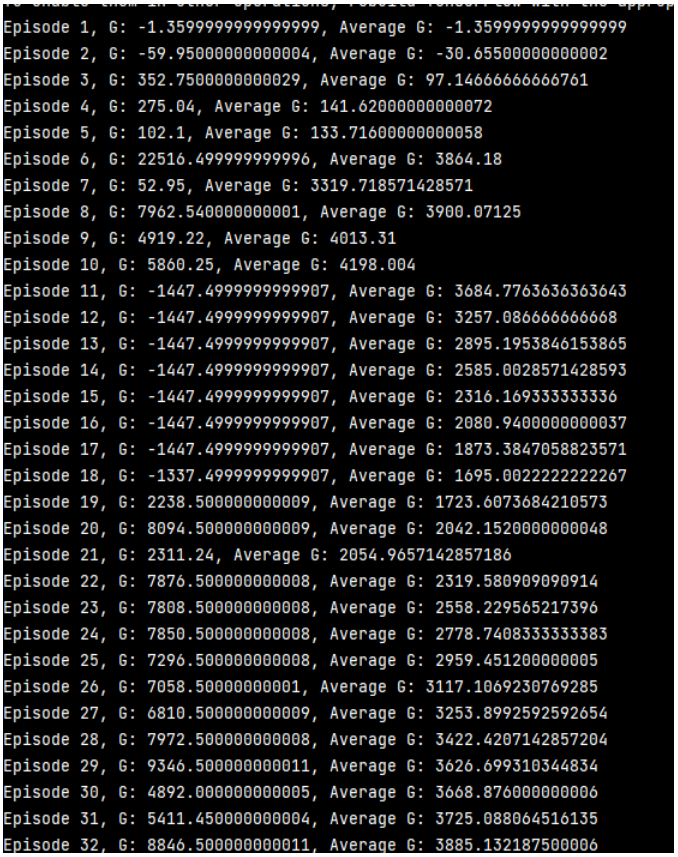

Fig. 1. Expected returns during training for several episodes. Issue during training due to the agent staying in the spatial cue area and abusing the gradually increasing reward structure for getting close to the platform.

from the 2D-environment to the 3D-environment. The main finding is that agents trained with blocked practice were more efficient and stable in finding the terminal platform than the agents trained with random practice. Recall that we controlled for each condition to be constrained within each training context. Which means that the blocked 2D trained agent only transferred to a blocked 3D environment and vice versa for the random agent. Given the results, it is unclear if agents were able to abstract the learned behavior (i.e., weights of the neural networks for x- and y-direction) and perform transfer learning. In other words, we are unsure if providing the weights of the 2D-agents to the 3D-agents helped in learning quickly the new environment. Due to lack of time and the extremely long computing time for the 3D-environment training we were unable to let agents run without using weights of the 2D-agents. This is considered to be a limitation of this study. Experimental studies conducted during the completion of this work showed, that training time and success measured in expected return per episode heavily depends on the size of the environment as well on the whether the agent moves in 2D or 3D. Agents under blocked practice trained faster than agents under random practice and agents in 2D-environment compared with agents in 3D-environment (see table I).

TABLE I
COMPUTING TIME OF THE INDIVIDUAL AGENT'S TRAINING FOR BLOCKED AND RANDOM PRACTICE IN 2D- AND 3D-ENVIRONMENT.

Environment	Practice	Time [minutes]
2D	Blocked	27
2D	Random	69
3D	Blocked	90
3D	Random	360

1) Results of Simulation of Agents in 2D-Environment: The agent under the condition of blocked practice outperformed in terms of time and stability its random practice counterpart. The blocked practice agent found an optimal policy in roughly 25 episodes with a maximum expected return of -0.36 (see Figure 2. This expected return can be understood as the minimum cumulative negative reward for the agent moving towards the platform, i.e., each move costs energy and exhausts the agent. The significant drop in expected return around episode 100 can be explained by the agent still exploring new actions. In this episode the agent could have taken several exploring moves and moved into a new area of the environment completely unknown. In other words, the deep neural network has not experienced the sequence of visited states before and therefore, the agent did a poor job in computing the appropriate action for the given weights. This is due to the fact, that the weights were optimized for the agent visiting a different sequence of states as indicated by the bright cells of the contour map in Figure 3. The contour map indicates that the agent learned that the fastest way to the platform is to move north to cell (3, 0) from the starting position (0, 0) and then east to cell (3, 3). The final move is north towards the platform and terminal state.

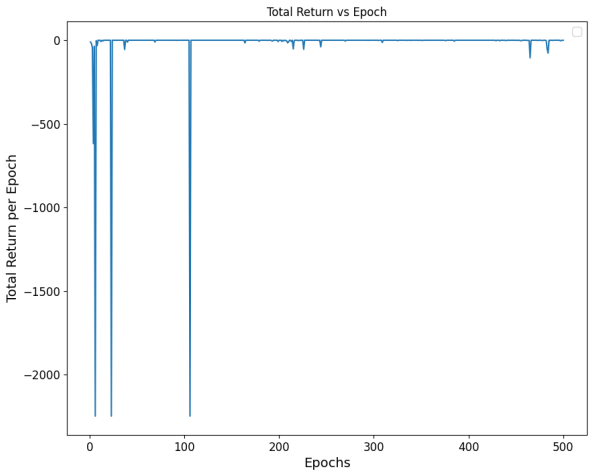

Fig. 2. Trajectory of learning of the agent searching the 2D-environment for the reward platform. Expected return plotted against epochs (n=500) for blocked practice, i.e., agent has fixed x- and y-starting positions of (0, 0).

The random practice agent found an optimal policy in roughly 50 episodes, twice as long as the blocked practice

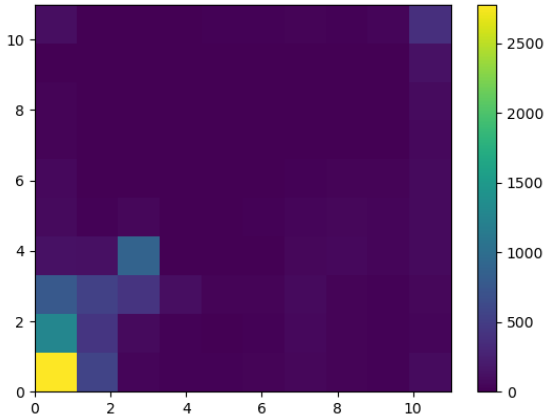

Fig. 3. Contour plot to highlight the visited states of the agent moving inside the 2-dimensional environment for blocked practice. Brighter cells indicate states with higher frequency of the agent visiting them.

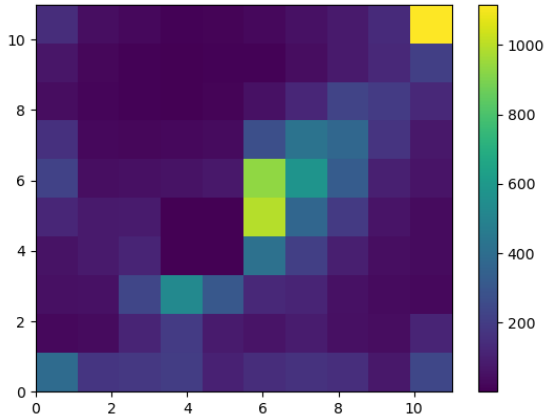

Fig. 5. Contour plot to highlight the visited states of the agent moving inside the 2-dimensional environment for random practice. Brighter cells indicate states with higher frequency of the agent visiting them.

agent, with a maximum expected return of -0.36 (see Figure 4. The more frequent drops in expected returns can be explained by the random starting position of the agent and the resulting knowledge gap of the deep neural network,i.e., the weights were trained to generalize but were not particularly helpful in those starting positions. Figure 5 shows the contour map and indicates the visited states of the agent. It seems that the agent learned to move towards cell (5, 6) and connects that cell with the terminal state. In comparison, this agent is visiting many more states than the blocked agent.

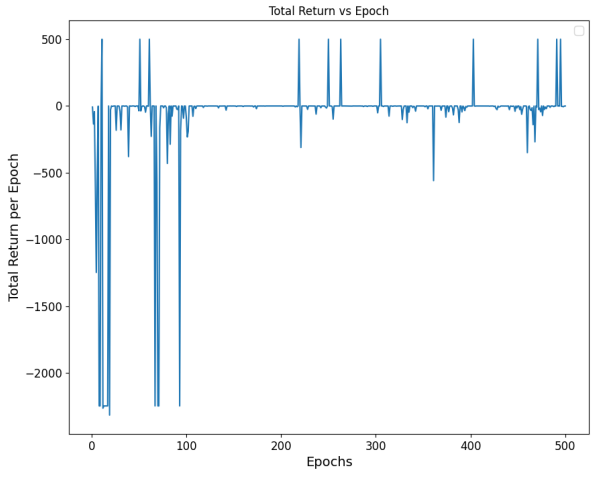

Fig. 4. Trajectory of learning of the agent searching the 2D-environment for the reward platform. Expected return plotted against epochs (n=500) for random practice, i.e., agent starts at any random x- and y-position in environment.

2) Results of Simulation of Agents in 3D-Environment: The results of the agent trained in 3D-environment mirror their 2D-counterparts in terms of learning ability (i.e., time) and stability between the blocked and random training conditions. The blocked agent seems to have a much easier time to understand where the platform is compared to the agent trained in random practice.

The agent under the blocked practice condition outperformed in terms of time and stability its random practice counterpart. The blocked practice agent found an optimal policy in roughly 45 episodes with a maximum expected return of -0.36 (see Figure 2. Initially, it seems that the agent might may something out of the provided 2D-weights. Nevertheless, the agent is only able to identify the optimal policy after spending a certain period in the 3D-environment. The significant drops in expected returns around episode 180 and 310 can be explained by the agent still exploring the environment using the epsilon-greedy policy. Especially, around 310 could be interpreted as the agent learning a new composition of the weights. These weights may have been used for a couple of episode until the agent stabilized again and return to its old policy. The variation at the end can be explained with the chance for exploration when choosing an action for a given state.

The random practice agent received the *maximum* negative reward in the first 50 episodes and thus, it seems that the agent did not profit by providing the learned weights about the x- and y-direction in 2D. Figure 4 shows the expected return plotted against epoch. The random agent on average has a large negative expected return indicating that it either finds the platform after many taken moves or completely exhausting all moves. Furthermore, the agent has a much harder time determining the optimal policy and finding the platform than the blocked practice agent. These difficulties could stem from the increased size of the environment and the effects of the random starting position on finding the platform. The agent is incapable of learning an optimal policy (i.e., updating weights

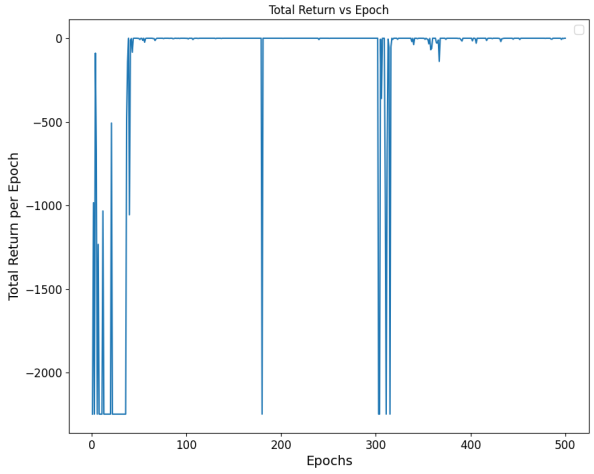

Fig. 6. Trajectory of learning of the agent searching the 3D-environment for the reward platform. Expected return plotted against epochs (n=500) for blocked practice, i.e., agent has fixed x-, y- and y-starting positions of (0, 0, 0).

accordingly) since it always starts in a new location that might be completely unknown. Furthermore, the path towards the platform is increased due to the bigger environment.

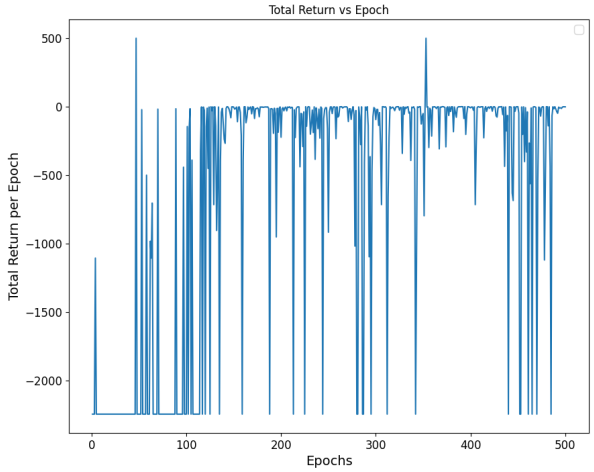

Fig. 7. Trajectory of learning of the agent searching the 3D-environment for the reward platform. Expected return plotted against epochs (n=500) for random practice, i.e., agent starts at any random x- and y-position in environment.

VIII. Code

Please use the following command in your command line to run the example setup: For environment in 2-dimensions with fixed starting position:

```
python ECE517_RL_Group1_Project3_main.py 2
12 2 2 0
```

For environment in 2-dimensions with random starting position:

```
python ECE517_RL_Group1_Project3_main.py 2
12 2 2 1
```

For environment in 3-dimensions with fixed starting position:

```
python ECE517_RL_Group1_Project3_main.py 3
12 4 4 0
```

For environment in 3-dimensions with random starting position:

```
python ECE517_RL_Group1_Project3_main.py 3
12 4 4 1
```

IX. Conclusions

In this work, we simulated the classic neurosciene Morris Water Maze using reinforcement learning and double deep Q-learning. As other work showed [2], this classic neuroscience experiment can be simulated using reinforcement learning techniques which could have significant impact on how spatial navigation facilitated by the hippocampus is studied in the future. For example, if the model is fine-tuned to the behavior of the rodent used in such experiments, the lives of these animals and money could be saved in the future.

In this study, we tried to answer the question if reinforcement learning models can achieve transfer learning. Based on the results, it is unclear if the agent in 3D was really utilizing the prior information (i.e., model weights) from agents trained in 2D-environment when finding the platform or if the agent recomputed the weights once it started to move inside the 3D-environment and randomly finding the platform.

A limitation of this study is that no additional experiments for comparison were made of agents without pre-trained weights learning about the location of the platform in 3D-environment. Furthermore, more experiments could be done to highlight that training time and expected return depends on the size of the environment, platform, and the spatial cue area.

Future work could deal with the implementation of reinforcement learning techniques that work well with continuous state space and continuous action space. Modelling the action space as continuous comes closer to the movement behavior of a rat faced with the Morris Water Maze Task.

References

[1] Van Hasselt, H., Guez, A., & Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).

[2] Tessereau, C., O'Dea, R., Coombes, S., & Bast, T. (2021). Reinforcement Learning approaches to hippocampus-dependent flexible spatial navigation. Brain and neuroscience advances, 5, 2398212820975634.

[3] Schmidt, R. A., Lee, T. D., Winstein, C., Wulf, G., & Zelaznik, H. N. (2018). Motor control and learning: A behavioral emphasis. Human kinetics.

[4] Morris, R. G. (1984). Developments of a water-maze procedure for studying spatial learning in the rat. Journal of neuroscience methods, 11(1), 47-60.

[5] Rey, P. D., Wughalter, E. H., & Whitehurst, M. (1982). The effects of contextual interference on females with varied experience in open sport skills. Research quarterly for exercise and sport, 53(2), 108-115.

[6] Whitehurst, M., & Del Rey, P. (1983). Effects of contextual interference, task difficulty, and levels of processing on pursuit tracking. Perceptual and Motor Skills, 57(2), 619-628.

[7] Porter, J. M., Landin, D., Hebert, E. P., & Baum, B. (2007). The effects of three levels of contextual interference on performance outcomes and movement patterns in golf skills. International journal of sports science & coaching, 2(3), 243-255.
[8] Shewokis, P. A., Shariff, F. U., Liu, Y., Ayaz, H., Castellanos, A., & Lind, D. S. (2017). Acquisition, retention and transfer of simulated laparoscopic tasks using fNIR and a contextual interference paradigm. The american journal of surgery, 213(2), 336-345.
[9] Bye, C. M., Hong, N. S., Moore, K., Deibel, S. H., & McDonald, R. J. (2019). The effects of pool shape manipulations on rat spatial memory acquired in the Morris water maze. Learning & behavior, 47(1), 29-37.
[10] Logue, S. F., Paylor, R., & Wehner, J. M. (1997). Hippocampal lesions cause learning deficits in inbred mice in the Morris water maze and conditioned-fear task. Behavioral neuroscience, 111(1), 104-113.
[11] Magill, R. A., & Hall, K. G. (1990). A review of the contextual interference effect in motor skill acquisition. Human movement science, 9(5), 241-289.