Vector-Based Navigation Inspired by Directional Place Cells

Harrison Espino[1(✉)] and Jeffrey L. Krichmar[1,2]

[1] Department of Computer Science, University of California, Irvine, Irvine, CA, USA
{espinoh,jkrichma}@uci.edu
[2] Department of Computer Science, Department of Cognitive Sciences, University of California, Irvine, Irvine, CA, USA

Abstract. We introduce a navigation algorithm inspired by directional sensitivity observed in CA1 place cells of the rat hippocampus. These cells exhibit directional polarization characterized by vector fields converging to specific locations in the environment, known as ConSinks [8]. By sampling from a population of such cells at varying orientations, an optimal vector of travel towards a goal can be determined. Our proposed algorithm aims to emulate this mechanism for learning goal-directed navigation tasks. We employ a novel learning rule that integrates environmental reward signals with an eligibility trace to determine the update eligibility of a cell's directional sensitivity. Compared to state-of-the-art Reinforcement Learning algorithms, our approach demonstrates superior performance and speed in learning to navigate towards goals in obstacle-filled environments. Additionally, we observe analogous behavior in our algorithm to experimental evidence, where the mean ConSink location dynamically shifts toward a new goal shortly after it is introduced.

Keywords: Navigation · Reinforcement Learning · Hippocampus

1 Introduction

Place cells in the CA1 area of the hippocampus have shown heading based sensitivity in goal-oriented tasks [3,9]. Recent work in rats quantified this sensitivity as a vector field converging to a specific point, known as a ConSink [8]. In an obstacle-free environment, ConSinks of these direction sensitive place cells (which we call "ConSink cells") organized around the goal, and shifted towards a new goal when it was changed. We suggest that this spatial representation may benefit new learning algorithms for navigation and other domains.

Optimal navigation to a goal is also a prominent benchmark in the field of computer science. A common approach to navigating unknown and complex environments is reinforcement learning (RL), such as Deep Q-Networks (DQN) [7] or Proximal Policy Optimization (PPO) [10]. These algorithms aim to learn a policy which determines optimal actions to navigate towards a goal given observations of the agent's surroundings. These algorithms are provided either partial or complete information about the environment. While they have demonstrated efficacy in domains such as robotics [2] and playing games [5], some open issues still persist. Notably, algorithms reliant on training deep neural networks are often computationally intensive, and require substantial training before achieving acceptable performance.

Inspired by the ConSink finding, we propose an algorithm to achieve fast and robust performance on goal-directed navigation tasks in simulated environments. In the Dyna maze environment, our model learns to navigate to the goal from randomly selected start locations in less than 50 trials, which is substantially faster than DQN and PPO. By testing on multiple environments with randomly placed obstacles, we find our that our model performs better than other state-of-the-art (SoTA) RL algorithms. Our model demonstrates the same ability to adapt as alternatives and exhibits similar behavior to real place cells by shifting the population ConSink location towards the new goal.

In summary, our paper makes the following contributions:

1. We introduce a biologically-inspired algorithm for learning goal-oriented navigation tasks that outperforms DQN and PPO, two state-of-the-art RL algorithms. Our model reaches the goal in fewer training epochs, maximizes reward faster, and adapts to changes in goal quickly.
2. We find similar behavior in our model to biological ConSink cells on experiments when the goal changes locations. Individual ConSinks move closer to the new goal, causing the mean ConSink location to shift to the new goal location after training.
3. Based on our model, we predict that ConSink cells in the rat hippocampus may have directional sensitivity away from the goal towards important "subgoals" in environments with obstacles requiring non-goalward movement.

2 Methods

2.1 ConSink Place Cell Model

In [8], rats that were presented two possible choices of travel selected the direction which aligned with the vector from the population of ConSink cells. When there was no available path to the goal, the rat physically surveyed possible directions and were found to choose the direction with the highest population activation, and consequently that nearest towards the goal.

To model this behavior, neurons in the model are characterized by place sensitivity and orientation sensitivity, each of which are calculated individually and multiplied together to a generate the place cell's total activity. In biological neurons, "sensitivity" is characterized by increased firing rate. For our model, it is represented by a scalar value. Place activity at point (x, y) is calculated by (1):

$$v_{place} = \mathcal{N}(\sqrt{(p_x - x)^2 + (p_y - y)^2};\ 0,\ \sigma^2), \tag{1}$$

where the distance between the place cell location (p_x, p_y) and the agent's location (x, y), is passed through a Gaussian activation function. The hyper-parameter σ controls how sharply the activity of the place cell decreases as the distance from the cell's preferred location increases. This is dependent on the scale of the environment, and is set to 2.0 in our experiments. Each grid in our environment had a length of 1 unit, so this allowed for sampling in a wide range around the agent's location.

Orientation sensitivity is defined by a response array of 8 values each representing the cardinal and ordinal directions of travel. From these values, a response vector $\hat{n}$ can be generated as:

$$\hat{n} = \frac{\sum_i w_i \hat{u}}{\| \sum_i w_i \hat{u} \|}, \tag{2}$$

where $\hat{u}$ is the unit vector corresponding to one of 8 directions and w_i is the value for this direction. Consistent with the experimental data in [8], values of the response array are initialized to have a preference towards the goal location. This is achieved through:

$$d(p, p_0, d) = \left| \frac{(\mathbf{p} - \mathbf{p}_0) \cdot \mathbf{d}}{\|\mathbf{d}\|} \right| \tag{3}$$

and

$$w_i = \mathcal{N}(d(g_{x,y}, p_{x,y}, \hat{u_i}); 0, \sigma^2) + U(\alpha_{min}, \alpha_{max}). \tag{4}$$

Equation (3) defines a projection function which returns the minimum distance from point p to point p_0 in the direction d and Eq. (4) is the initialization of response vector value w_i as the minimum distance from the place cell's location $p_{x,y}$ to the goal $g_{x,y}$ through the corresponding unit vector $\hat{u_i}$, plus a random amount of noise given by sampling from a random variable U with minimum and maximum values (n_{min}, n_{max}). For our experiments, $(\alpha_{min}, \alpha_{max})$ is set to $(-0.5, 0.5)$ and σ of the activation function is set to 5.0. We found that a shallow-sloped activation function placed sufficient weight on directions that were close but not exactly in the correct direction.

The orientation-based activation for direction θ can then be calculated as in (5), or the cosine similarity between the response vector and the orientation. This value was normalized between 0 and 1 so that we can achieve a total cell activation between 0 and 1 when it is multiplied with the place activation.

$$(\frac{\theta \cdot \hat{n}}{\|\theta\| \cdot \|\hat{n}\|} + 1) \cdot 0.5 \tag{5}$$

2.2 Vector Navigation with ConSink Place Cells

To facilitate navigation, a population of directional place cells is initialized, each with place sensitivity for a random location in the environment as defined by Eqs. 1 through 4. Locations of place cells are constrained such that each cell is a minimum distance from its neighbors. This minimum distance is dependent on the size of the environment and the number of place cells, and was necessary to ensure place cells are sufficiently distributed throughout the environment.

At each simulated timestep, an action is chosen by sampling the population of cells at the current location for each possible direction of travel. The values for each direction are passed through a softmax activation function to determine the probability of selecting the corresponding action. The temperature of this softmax function is set to 1.0 in our experiments.

2.3 Eligibility Trace and Reward

Learning is achieved by changing the values of the response array according to the reward signal received by the environment. To do this, each neuron must maintain an eligibility trace e defined by:

$$e = \begin{cases} v_{place} & \text{if } v_{place} > \beta \\ e - \frac{e}{\tau} & \text{otherwise} \end{cases}, \tag{6}$$

where β determines the minimum place activity to set the eligibility trace and τ determines the rate of decay of the eligibility trace. We set β to 0.5 and τ to 100 in our experiments. This β value allowed the eligibility trace of nearby neurons to be set again in the case the agent visited the same location twice in quick succession, and a large τ led to a slower decay and greater updates to neurons involved in earlier decisions.

The eligibility trace of each neuron is updated after an action is taken. The purpose of this eligibility trace is to determine the contribution of each cell to the previous action, as well as solve the credit assignment problem when a reward signal is used to update the modeled ConSink neurons. When the neuron's eligibility trace is updated to v_{place}, a unit vector $\hat{l}$ between the neuron's preferred location and the agent's location is saved so that it can be used to update the directional sensitivity.

Upon receiving a positive or negative reward, the values of each neuron's response arrays are calculated by:

$$w_i = w_i + \frac{\hat{u_i} \cdot \hat{l}}{\|\hat{u_i}\| \cdot \|\hat{l}\|} * r * e_i * \alpha, \tag{7}$$

where r is a negative or positive reward from the environment and α is a learning rate (set to 0.001 in our experiments). The cosine similarity between the saved direction of travel and the response array value's unit vector determine its contribution to the agent's action. The eligibility trace additionally scales the learning by how active the neuron was during traversal.

2.4 Environment

We use the OpenAI Gym framework to build environments capable of training our models and Reinforcement Learning models for comparison [1]. Agents traverse a grid-world environment and are allowed to move to neighboring squares

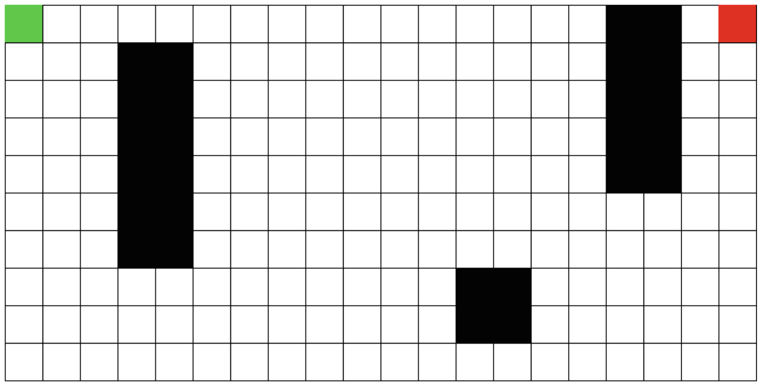


Fig. 1. The Dyna maze environment. Black squares represent intraversable obstacles. Red square is the goal. The agent starts randomly along the leftmost column (the green square represents one starting location). (Color figure online)

in 8 directions if that square is not occupied by an obstacle or out of bounds. We test our algorithm in three different environments.

The first environment is the Dyna maze, which was introduced to test the Dyna Reinforcement Learning algorithm and has been used in subsequent navigation papers [6,12]. Our implementation can be seen in Fig. 1. The agent (green) started randomly at any space in the leftmost column of the environment, and the goal (red) was always in the top-right space.

To further assess the robustness of our algorithm, we train multiple instances of the model on environments with randomly placed obstacles. Obstacles are placed randomly according to a density percentage (10% in our experiments) and then iteratively moved to ensure a valid path from the start to end exists. In this case, the agent's start position is always on the top-left space, and the goal on the bottom-right space. A sample environment of this kind can be seen in Fig. 2.

Lastly, we train models in a completely open environment similar to that used in the rat experiments [8], which was a 10 X 10 grid. The environment and models are initialized with the goal in the center of the environment (5, 5). The goal is moved to a different location (2, 3) after 100 epochs of training. At each episode, the agent started randomly at the edge of the environment.

Agents received an observation in the form of its current grid location (x, y) and 8 values corresponding to the distance to the closest boundary or wall in the directions of movement. The reward signal for the environments is as follows. Reaching the goal results in a reward of 1.0. Attempting to move into a space occupied by an obstacle results in a reward of -0.8, and -0.75 for attempting to move out of bounds. If the agent moves to a previously visited space, it receives a reward of -0.25. The agent receives a reward of -0.04 in all other cases. An epoch in this environment lasted until the goal is reached, or the total reward went below -100. This reward structure was chosen to encourage fast planning towards the goal while discouraging wasteful actions and encountering obstacles.

Individual values were fine-tuned for convergence on our model's comparisons. To ensure fairness, all models had the same observation space and rewards.

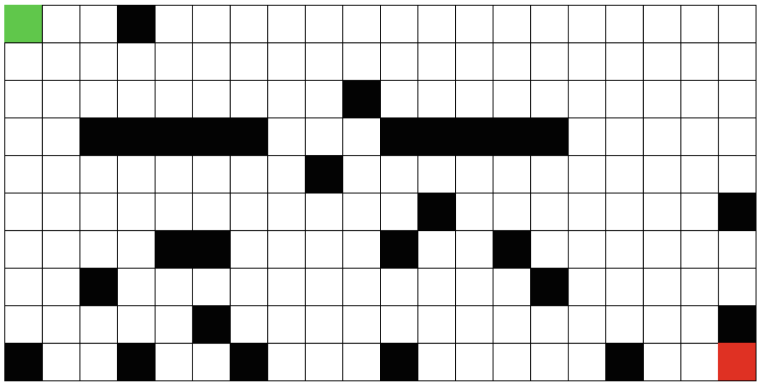


Fig. 2. A sample environment with randomly placed obstacles at 10% obstacle density. The agent's start location is always on the top left, and the goal is always on the bottom right.

2.5 Comparisons

We assess the performance of our model by comparing two other SoTA RL algorithms on the same environment.

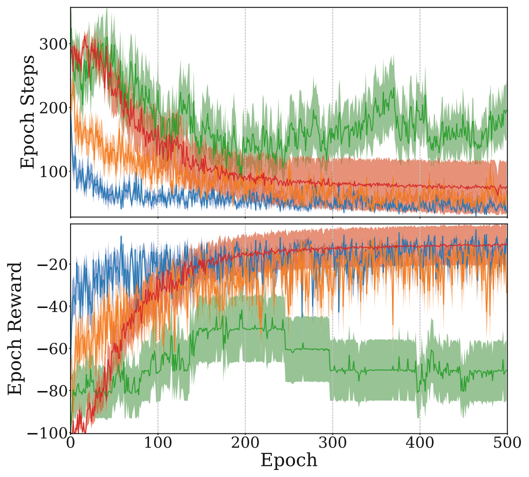
Deep Q-Networks: An RL algorithm combining the Q-learning technique and deep learning [7]. A neural network is trained to approximate the Q-function, which estimates the expected reward for taking an action at a given state. Past experiences are replayed to stabilize training and improve data efficiency. We implemented DQN with a 2 hidden layer network, with 256 neurons at each layer.

Proximal Policy Optimization: An RL algorithm with the objective of approximating an optimal policy which maps specific states to actions [10]. A neural network is trained to learn a probability distribution of each action given the agent's current state. PPO has found great success in robotics and game playing for it's simplicity and efficiency. We implemented PPO with a 2 hidden layer network, with 64 neurons at each layer.

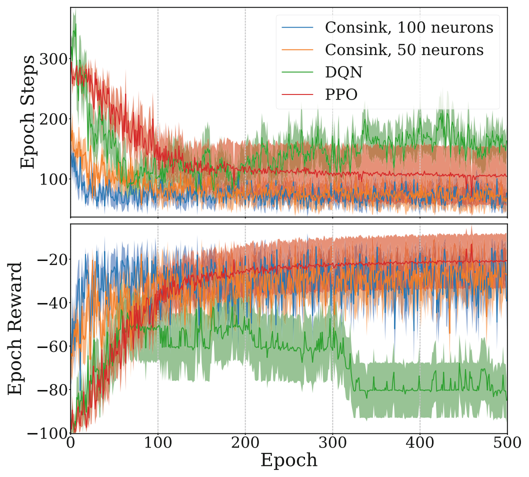
3 Results

3.1 Maze Learning

The result of 500 epochs of training on the Dyna maze are shown in Fig. 3a. Solid lines and shaded areas represent the mean and standard error of 10 agents. Since the reward signal is overwhelmingly negative, it is expected that an agent's



(a) Dyna Maze



(b) Random Maze

Fig. 3. DQN, PPO, and Consink (ours) agents trained on the (a) Dyna maze environment and (b) random mazes for 500 epochs. Top graphs are the total number of steps before reaching the goal or failing, bottom graphs are the total reward at each epoch. The line and shaded area are the mean and standard error of 10 different agents each.

total reward approaches zero as it learns to avoid obstacles and take minimal steps to reach the goal. Agents using our ConSink model learned faster than both PPO and DQN agents, as evidenced by larger reward epochs in a shorter time. This can also be seen when measuring the epoch step count, where our models consistently learned shorter paths. One reason for this is that ConSink place cells have preferred directions initially, as observed by [8].

We see similar results on randomly generated mazes in Fig. 3b. Our ConSink models with both 100 or 50 neurons outpaced DQN and PPO on both total reward per epoch and steps per epoch. Despite random initialization for the environment, the location of cells, and directional sensitivity of each cell, we find striking consistency over multiple runs.

Figure 4 depicts the difference in the neurons' directional sensitivities from initialization to the final training epoch. Place cells are drawn at their specified location, with lines indicating the direction of the response vector. Neurons in the model initially showed scattered directional sensitivity due to the noise from their initial preference towards the goal as described in Sect. 2.1. After training, populations of nearby neurons show similar directional sensitivity to each other. Near obstacles such as the bottom-most square, the neurons' directional sensitivities appear to curve around it, suggesting the model learns the optimal action is to avoid the obstacle while moving towards the goal.

Tables 1 and 2 report the average number of epochs before each model reaches the goal for the first time, as well as the percentage of time the goal is reached after 10, 50, 100, and all epochs. Each epoch in the environment lasts until the agent reaches the goal, or the total reward goes below -100. The results are the mean of 10 different agents.

ConSink models are first to reach the goal from initialization, and reach the goal a larger percentage of the time throughout training. The speed in which our models successfully reach the goal can be attributed to a number of factors. Our models do not rely on training densely connected neural networks like DQN and PPO. Our learning rule facilitates adjustments to the model completely online, rather than during specific training times between epochs. Additionally, initializing each neuron's directional sensitivity to be towards the goal means the model will often start by choosing actions in a straight line towards the goal, which may be correct in many cases.

Table 1. Success reaching goal on Dyna maze. Mean±standard deviation of 10 runs reported. Top row is the epochs to reach the goal. Other rows are the percentage of time the goal is reached after 10, 50, 100, and all epochs.

Model	ConSink-100	ConSink-50	DQN	PPO
First Goal Epoch	**0.7 ± 1.0**	2.11 ± 2.85	29.8 ± 55.4	24.2 ± 28.0
First 10 Goal %	**80.0 ± 21.9%**	35.6 ± 29.5%	22.0 ± 36.3%	2.0 ± 6.0%
First 50 Goal %	**91.2 ± 5.8%**	70.4 ± 24.0%	23.8 ± 37.7%	29.8 ± 23.1%
First 100 Goal %	**93.0 ± 4.5%**	76.1 ± 21.8%	26.9 ± 37.3%	52.8 ± 28.5%
Total Goal %	**96.4 ± 3.1%**	91.1 ± 6.5%	36.0 ± 26.1%	82.1 ± 27.8%

Table 2. Success reaching goal on randomly generated mazes over 10 runs (mean ± stdev). Top row is the number of epochs to reach the goal. Other rows are the percentage of time the goal is reached after 10, 50, 100, and all epochs.

Model	ConSink-100	ConSink-50	DQN	PPO
First Goal Epoch	2.3 ± 3.78	**1.7 ± 2.33**	10.5 ± 8.16	19.3 ± 33.47
First 10 Goal %	**62.0 ± 42.4%**	58.0 ± 36.3%	10.0 ± 16.1%	12.0 ± 18.3%
First 50 Goal %	**83.2 ± 15.8%**	75.6 ± 23.0%	29.4 ± 24.7%	34.2 ± 29.3%
First 100 Goal %	**85.6 ± 15.4%**	77.5 ± 25.1%	40.0 ± 33.6%	51.5 ± 32.4%
Total Goal %	**86.5 ± 15.2%**	79.7 ± 32.6%	36.4 ± 32.1%	74.3 ± 37.0%

3.2 Goal-Switching and Adaptability

Our experiments evaluating adaptability in an open environment are shown in Fig. 5. The agents are trained for 100 epochs with the goal at (5, 5), after which the goal is moved to (2, 3). The point at which the goal is moved is indicated by the dotted red line. As with the previous experiments, the line and shaded areas represent the mean and standard error of 10 agents. Because the environment

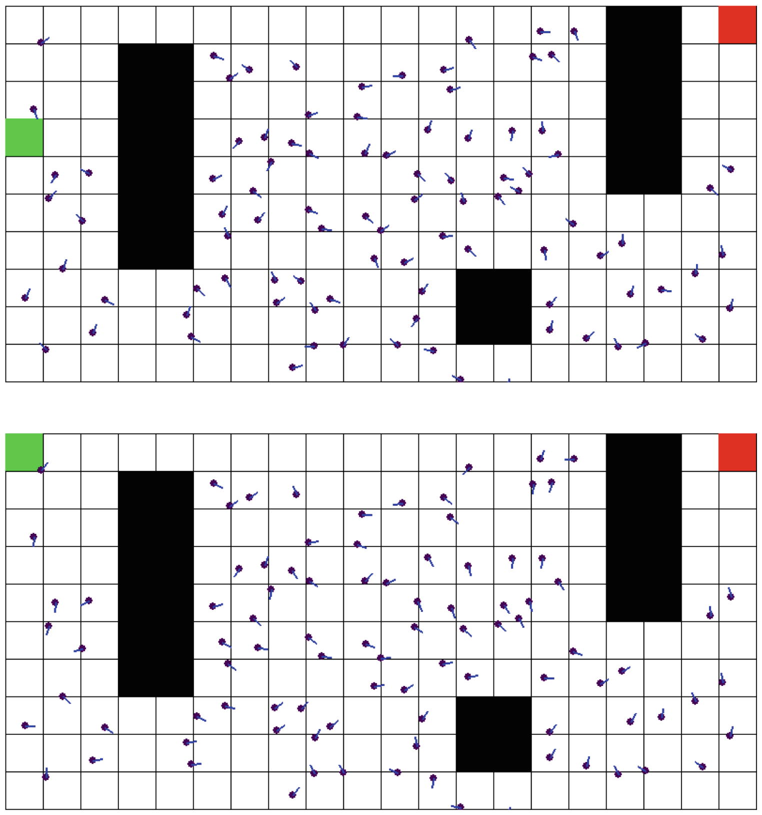


Fig. 4. Top image shows the initial directional sensitivity of place cells. Bottom image shows directional sensitivity after 500 epochs of training. Place cells are drawn in purple, with a line indicating the direction of highest activity.

contains no obstacles and the agent can reach the goal in minimal steps, our models and DQN quickly found success navigating. After the goal switch, we see a dramatic decrease in epoch reward and an increase in epoch steps as our models initially fail to reach the new goal. However, after less than 20 epochs, they return to performing better on reward and steps taken than DQN and PPO.

The high epoch step count and low reward from our models immediately following the goal shift can be attributed to a lack of exploratory behavior from the trained model. This could potentially be alleviated by implementing a variable softmax temperature for action selection based on the total accumulated reward in an epoch. This way, the model exploits its learned action policy unless it has accumulated negative rewards, in which case it will switch to a more explorative strategy.

Figure 6a shows the change in distance of the mean ConSink location to the goal before and after the goal switch. A neuron's ConSink location is calculated by multiplying the neuron's response vector by its distance to the goal. ConSinks are initially tuned to the location of the original goal, and over the first 100 epochs the mean ConSink location continues to shift closer to the center of the goal. After the goal switch we see the mean ConSink location shift in

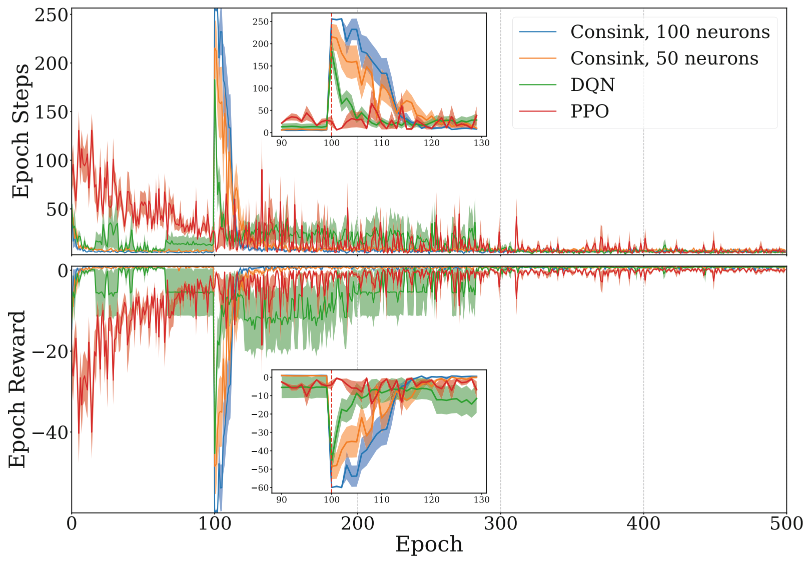


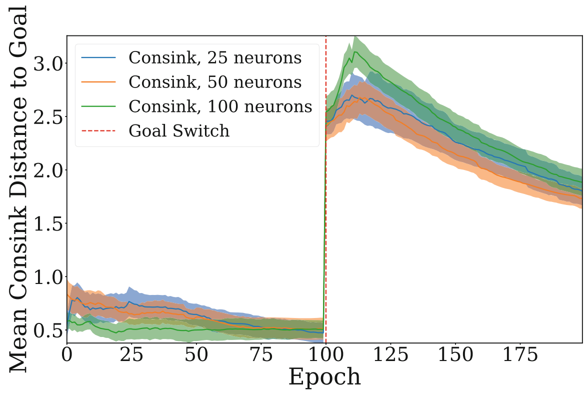
Fig. 5. DQN, PPO, and Consink (ours) agents trained on an obstacle-free open environment for 500 epochs. The goal is moved to an alternate location after 100 epochs, shown by the dotted line. Top graph is the total number of steps before reaching the goal or failing, bottom graph is the total reward at each epoch. The line and shaded area are the mean and standard error of 10 different agents each. Inset graph shows training epochs from 90 to 130.

the direction of the new goal with training, as the place neurons' directional sensitivities begin to change due to learning.

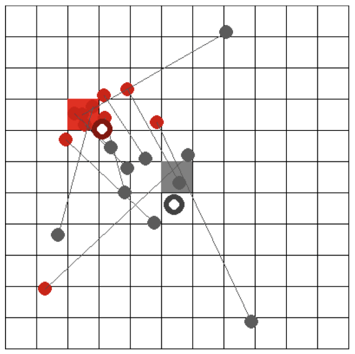
Similar to Fig. 2e from the rodent ConSink experiments [8], we plot the shift of individual ConSinks before and after training in Fig. 6b. The gray square is the location of the initial goal. Filled gray circles are the locations of ConSinks for 10 candidate neurons, and the white-filled gray circle is the mean ConSink location for the entire population. The same is true for the red square and red circles, which represent the new goal and ConSinks after the goal is changed. Gray lines connect the ConSinks of the same neuron. We find that individual ConSinks move closer to the new goal during training, leading the mean ConSink location to shift as well.

4 Discussion

We introduce a navigation algorithm that is capable of rapidly and accurately reaching a goal in obstacle-filled environments. Our model outperforms SoTA reinforcement learning models in maximizing accumulated reward, minimizing steps taken, consistently reaching the goal throughout training. When the goal is moved after training in a completely open environment, our model is able to adapt to this change in few epochs of exploration. Cells in our model also exhibit behavior reminiscent of biological data, as the mean ConSink location moves towards the new goal during training.



(a) Mean ConSink distance to goal during training



(b) Shift in individual ConSinks

Fig. 6. (a) Distance of the mean ConSink location after the goal is changed in an open environment. Consink agents with 100, 50, and 25 neurons are trained for 100 epochs before the goal is changed. The line and shaded area are the mean and standard error of 10 different agents each. (b) Visualization of Consinks before and after the goal is changed. Gray circles represent the initial location of Consinks and the gray square is the location of the first goal. Red circles represent the Consink location of the same place cell after training on the new goal. (Color figure online)

Our findings mirror biological evidence showing adaptation in the ConSink cells recorded in rats. Our novel learning rule achieves this using a combination of eligibility trace, saved trajectories, and reward signal. This suggests that ConSink place cells in the hippocampus may be employing similar reward-based learning to adjust their directional sensitivity.

Our results also raise interesting questions about the role of directional sensitivity in place cells for complex environments. Previous work investigating place cell directionality in rats were in environments in which a direct route to the goal was always possible [3]. Ormond and O'Keefe used a unique "honeycomb maze" in which the rat was required to make a sequence of binary choices on raised platforms, sometimes requiring the rat to move away from a goal [8].

In both cases, the environment did not have obstacles and was fully visible to the rat. The honeycomb maze removed the ability for the rat to move directly to the goal, however the correct binary choice was always more in the direction of the goal than the incorrect one. Our model predicts that, in environments filled with obstacles, populations of ConSinks near obstacles blocking direct routes to the goal may need to point towards sub-goals in order to facilitate navigation. Behavioral studies have shown that mice utilize sub-goal strategies when learning new environments [11]. Future work might investigate if sub-goal memorization is encoded in place cell directionality.

Our model is similar to traditional Q-learning in that each place neuron is learning an optimal action for the location it is sensitive to, much like a Q-table learns optimal actions for each state. However, our model uniquely takes advantage of the fact that environment states in a navigational problem can be related over two axes of position. This allows for the generalization of unlearned areas by sampling the learned actions of place cells representing nearby areas. Future work may aim to apply our model to non-navigational problems which can be represented in this way. The model may also employ heterogeneous learning rates and eligibility trace functions across neurons to represent dynamic environments.

Deep Q-learning uses a form of experience replay in which previously saved state-action-reward sequences are used to train the model multiple times. PPO employs a similar technique, maintaining a growing replay buffer which is used to train the model every set number of epochs. While such a replay system is not yet implemented in our model, recordings of the rat hippocampus have shown sequential reactivation of place cells resembling past trajectories during sleep or rest states [4]. By saving past experiences, replay can potentially be used to further improve the efficiency of our models. Additionally, the model can serve as a generator for novel trajectories by continually sampling and simulating actions offline.

Acknowledgement. This work was supported by NIH award R01 NS135850-02.

Disclosure of Interests. The authors have no competing interests.

References

1. Brockman, G., et al.: Openai gym. CoRR abs/1606.01540 (2016)
2. Gu, S., Lillicrap, T., Sutskever, I., Levine, S.: Continuous deep q-learning with model-based acceleration. In: Balcan, M.F., Weinberger, K.Q. (eds.) Proceedings of The 33rd International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 2829–2838. PMLR (2016)
3. Jercog, P.E., Ahmadian, Y., Woodruff, C., Deb-Sen, R., Abbott, L.F., Kandel, E.R.: Heading direction with respect to a reference point modulates place-cell activity. Nat. Commun. **10**(1), 2333 (2019)
4. Lee, A.K., Wilson, M.A.: Memory of sequential experience in the hippocampus during slow wave sleep. Neuron **36**(6), 1183–1194 (2002)
5. Lin, C.J., Jhang, J.Y., Lin, H.Y., Lee, C.L., Young, K.Y.: Using a reinforcement q-learning-based deep neural network for playing video games. Electronics **8**(10) (2019)
6. Mattar, M.G., Daw, N.D.: Prioritized memory access explains planning and hippocampal replay. Nat. Neurosci. **21**(11), 1609–1617 (2018)
7. Mnih, V., et al.: Playing ATARI with deep reinforcement learning. CoRR abs/1312.5602 (2013)
8. Ormond, J., O'Keefe, J.: Hippocampal place cells have goal-oriented vector fields during navigation. Nature **607**(7920), 741–746 (2022)
9. Sarel, A., Finkelstein, A., Las, L., Ulanovsky, N.: Vectorial representation of spatial goals in the hippocampus of bats. Science **355**(6321), 176–180 (2017)
10. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR abs/1707.06347 (2017)
11. Shamash, P., Olesen, S.F., Iordanidou, P., Campagner, D., Banerjee, N., Branco, T.: Mice learn multi-step routes by memorizing subgoal locations. Nat. Neurosci. **24**(9), 1270–1279 (2021)
12. Sutton, R.S.: Dyna, an integrated architecture for learning, planning, and reacting. SIGART Bull. **2**(4), 160–163 (1991)