Domain Generalization in RL: A Case Study with Snake

Gopal K. Goel and Owen Dugan

December 6, 2023

Abstract

We examine the problem of generalizing to unseen domains in reinforcement learning by considering the game of snake. We train an agent on an open board and test whether it can translate its play to environments with obstacles or with other players. We observe a range of generalization abilities, depending on training hyperparameters, leading to insights into overfitting and generalization in reinforcement learning.

1 Introduction

In recent years, deep learning neural networks have risen to prominence for their ability to accurately perform complex tasks such as image recognition and natural language processing. Key to their success is vast amounts of demonstration data to train on. Unfortunately, in many tasks, demonstration data is costly or impossible to collect, making supervised neural network learning challenging.

Reinforcement learning allows for the training of neural networks to perform tasks without demonstrations. It does so by allowing the neural network to try actions and rewarding the network based on the quality of its performance. In this way, the network produces its own demonstration data and progressively discovers successful behaviors.

One challenge of reinforcement learning is that agents trained through reinforcement learning can struggle to generalize to environments outside of the domain of the environments on which the agents have been trained [1]. This can lead to challenges, especially in applications where training is done in simulated environments [2].

In this paper, we examine the problem of generalizing to unseen domains in reinforcement learning by considering the game of snake. We train an agent on an open board and test whether it can translate its play to environments with obstacles or with other players, seeking to gain insight into generalization and overfitting in reinforcement learning.

2 Methodology

2.1 Environment Description

We use a modified version the Gym-Snake environment taken from [3]. Gym-Snake is an implementation of the classic game snake with a state/action/reward interface, as specified by the OpenAI gym specification. In particular, for a single snake, the action space at any time is $\{\uparrow, \leftarrow, \downarrow, \rightarrow\}$. Each time the snake crosses over a piece of food, it's length increases by 1, and a new piece of food is randomly spawned at an empty grid location. If the snake head enters a snake cell or exits the frame, the snake dies. The snake gains a reward of $+1$ for eating a piece of food, and a reward of -1 for dying, at which point it no

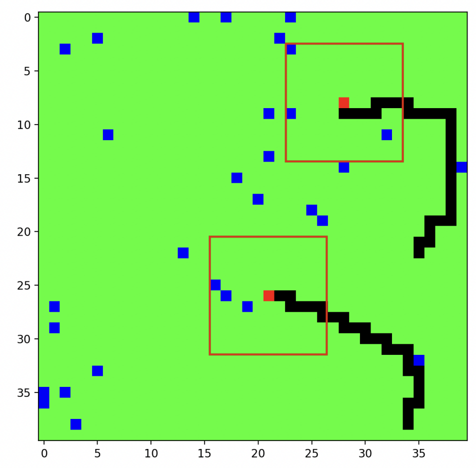

Figure 1: Single Frame from multi-snake run. Observation windows are overlayed on top of the environment.

longer takes any actions.

The environment supports multiple snakes playing together. An example frame from an environment run is shown in Figure 1.

2.2 Agent Overview

Our goal is to train agents that observe the state of a snake, and take actions that optimize the discounted reward of the snake over time. In particular, if the snake eats food at times $t_1, \ldots, t_n$, and dies at time T, then the discounted reward is

$$\left(\sum_{i=1}^{n} \gamma^{t_i}\right) - \gamma^T, \tag{1}$$

where $\gamma < 1$ is the discount factor.

We make the simplifying assumption that the agents can only observe an 11×11 grid centered at the head of the snake. Parts of the observation window that spill off the grid are colored black (i.e. the agent sees black outside of the grid confines). See Figure 1 for a visualization of the observation windows.

The observation window serves two purposes. First, if there are multiple snakes on the board, the agent can easily know which one it is controlling, namely the one at the center of its observation window. Second, it reduces the snake observations to a much more manageable size of 11×11 instead of 40×40.

The agent's policy is a neural network whose input layer takes in an observation, and whose output layer outputs a probability distribution over the action space.

2.3 Reinforcement Learning Algorithm

We use Proximal Policy Optimization (PPO) [4] with an actor and a critic to train our agents. PPO maintains an actor, the policy $\pi_\theta(a|s)$, and a critic, an estimate of the value function $V_\theta(s)$. At each epoch, it allows the agent to interact with the environment through its previous policy $\pi_{\theta_{old}}(a|s)$, collecting training data $\tau = s_t, a_t, r_t$. It then optimizes the loss function

$$L_\tau(\theta) = L_\tau^P(\theta) + L_\tau^V(\theta) + L_\tau^S(\theta), \tag{2}$$

where $L_\tau^P(\theta)$ is the PPO policy function loss, $L_\tau^V(\theta)$ is a value function loss, and $L_\tau^S(\theta)$ is an entropy loss to promote exploration. The key innovation of PPO is to define $L_\tau^P(\theta)$ as

$$\mathbb{E}_t \left[\min\left(A_t r_t, A_t \texttt{clip}(r_t, 1-\epsilon, 1+\epsilon)\right)\right], \tag{3}$$

where ϵ is a hyperparameter, $r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, and A_t is computed from $V_\theta(s_t)$ using an advantage estimation algorithm . This effectively incentivizes the new policy $\pi_\theta(a|s)$ to remain close to the previous policy $\pi_{\theta_{old}}(a|s)$. In this loss, the gradient is not propagated through A_t.

We modify an implementation of PPO from the CleanRL [5] library for our experiments.

2.4 Trained Agents

We trained six agents, each of them using actor-critic PPO with 1.44×10^7 timesteps.

Agent Name	Architecture	NN Width	NN Depth	CNN Kernel Size	Starting Length
`BASELINE`	MLP	256	3	N/A	3
`WIDTH-128`	MLP	128	3	N/A	3
`WIDTH-512`	MLP	512	3	N/A	3
`DEPTH-4`	MLP	256	4	N/A	3
`CNN`	CNN	N/A	3	3×3	3
`LENGTH-10`	MLP	256	3	N/A	10

Table 1: Hyperparameters For Trained Agents

All agents are trained in the standard 40×40 environment with 30 foods, and with $\gamma = 0.999$ and 1024 rollout steps. All but the last agent have their runs with starting length 3.

The first agent, `BASELINE`, has an MLP (multi-layer perceptron) architecture with width 256 and depth 3. The other agents can be viewed as perturbations off this agent. Details about the hyperparameters of the various agents can be found in Table 1

The agents `WIDTH-128`, `WIDTH-512`, and `DEPTH-4` have the same architecture as `BASELINE` except with varying widths and depths. The agent `CNN` uses a convolutional neural network in place of a MLP. Finally, the agent `LENGTH-10` has identical architecture to `BASELINE`, except that all of its training runs have the snake start with length 10 instead of length 3.

2.5 Evaluation Framework

To test the performance of the agents, we let them run in test environments and measure the final length at time-of-death.

The main goal of this study is to observe whether the agents perform well in out-of-distribution settings. As so, we propose the following testing suite that focuses on ability to play in an environment with obstacles, and ability to play in a multiplayer setting, both of which can be considered to be domain generalizations over the relatively simple training environment.

`TRAIN-ENVIRONMENT`: This is the same environment used for training, 40×40 grid with 30 foods and starting length 3.

`SIMPLE-MAZE`: This environment introduces a 20×20 black square in the center of the grid, which induces death if the snake touches it. It is intended to test whether the agents can navigate in more constrained settings than the spacious `TRAIN-ENVIRONMENT`.

`HARD-MAZE`: This environment has a long black barrier that initializes the snake in a confined exterior region, and forces the snake to try to navigate into the interior of the barrier to get more food. It is intended to be a serious stress test of the agent's obstacle navigation capabilities.

`MULTI-AGENT-TEST`: This environment is the same as `TRAIN-ENVIRONMENT`, except that we deploy 5 snakes of starting length 3, all using the same policy. We measure the final length of the last standing snake, and report that as our performance metric.

The various testing environments are depicted in Figure 2.

3 Results

To compute the average final lengths, we run each testing environment $N = 100$ times and compute the mean μ and standard error of the mean, $\sigma/\sqrt{N}$, where σ is the standard deviation of the observed final lengths. Figure 3 shows the results of our experiments.

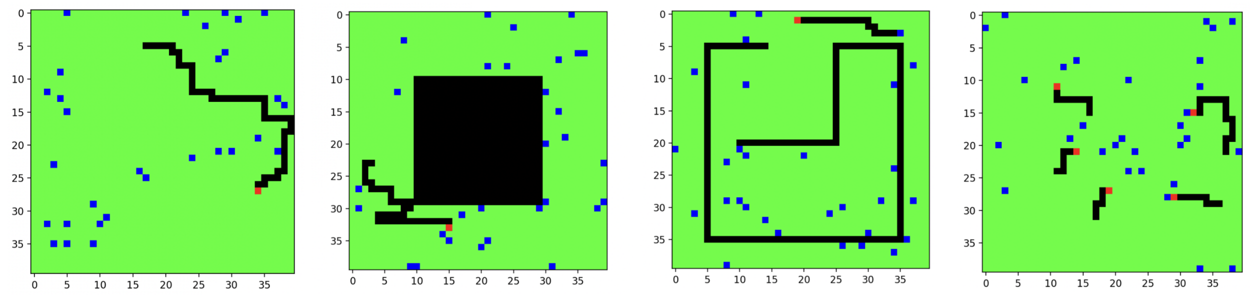

Figure 2: Testing environments `TRAIN-ENVIRONMENT`, `SIMPLE-MAZE`, `HARD-MAZE`, `MULTI-AGENT-TEST` from left to right

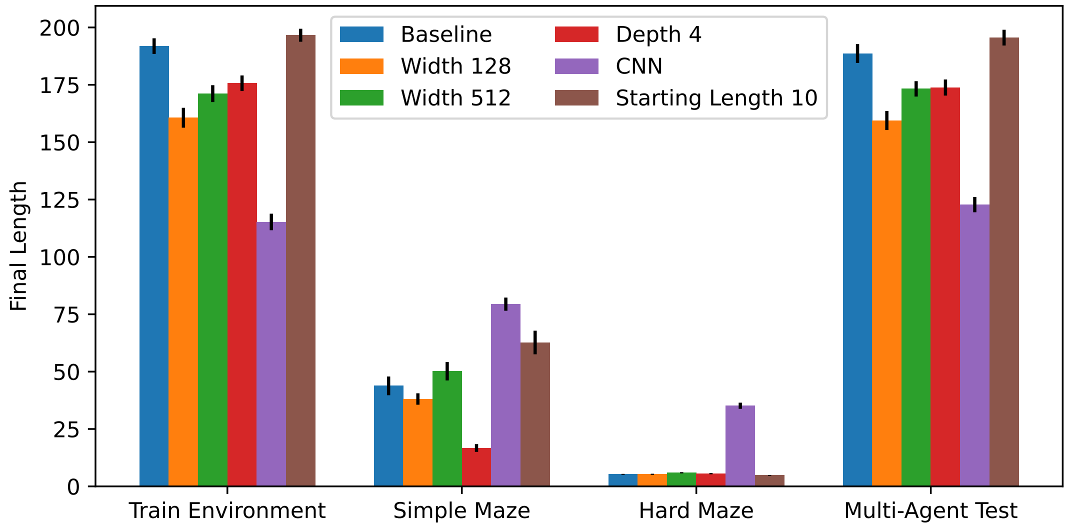

Figure 3: Average final lengths of agents in test environments

Note that most methods have comparable performance on `TRAIN-ENVIRONMENT`, between final length 150 and 200, but `CNN` performs substantially worse, with average final length 115 ± 4.

All methods have substantially smaller final lengths on `SIMPLE-MAZE` and even worse performance on `HARD-MAZE`. Interestingly, however, for both maze environments, `CNN` performs best; this improved relative performance is most prominent in `HARD-MAZE`. We also note that `DEPTH-4` performs noticeably worse than other methods in `SIMPLE-MAZE`.

Interestingly, the average final lengths in `MULTI-AGENT-TEST` are almost exactly the same as the average final lengths in `TRAIN-ENVIRONMENT`.

4 Discussion and Conclusion

We hypothesize that the reason that the performance in `TRAIN-ENVIRONMENT` and `MULTI-AGENT-TEST` are nearly identical is that in `MULTI-AGENT-TEST`, most snakes die off quickly, and one snake is left, and it is then playing effectively in `TRAIN-ENVIRONMENT`.

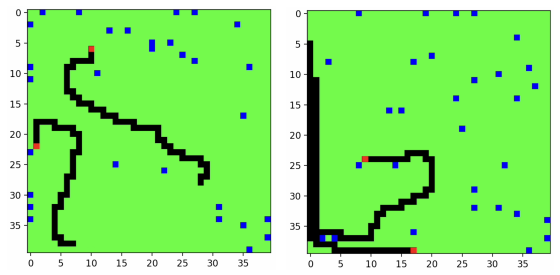

Figure 4: Competitive Dynamics in MULTI-AGENT-TEST. First frame depicts snakes repelling each other, second frame depicts snakes "hugging" each other.

This hypothesis was formed by observing several runs by eye, though there may be ways to study this more systematically. We did not expect this behavior when proposing the testing suite, but it makes sense in hindsight.

Occasionally, we did find that MULTI-AGENT-TEST would end up with two long snakes fighting for survival, and we observed some interesting competitive dynamics. Some examples of these dynamics are depicted in Figure 4.

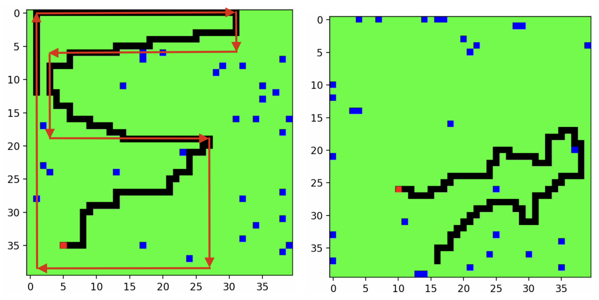

Figure 5: Movement styles of MLP- and CNN-based models. Left: MLP-based models learn a structure that reduces the chances of self-collision. Right: CNN does not learn such a structure.

We speculate that the performance of CNN is less overfit to TRAIN-ENVIRONMENT, which thereby enables it to generalize more effectively to SIMPLE-MAZE and HARD-MAZE. In particular, from examining testing runs, we find that the non-CNN models converge to an elegant strategy which involves repeatedly traveling straight upward and then moving back and forth as the snake travels downward (see Figure 5). We believe that the non-CNN reduces the probability of collisions, which increases performance on TRAIN-ENVIRONMENT. However, we suspect that this also reduces the robustness of the snake's obstacle avoidance – the snake begins to expect a certain structure when it encounters black tiles which does not hold in the maze environments. On the other hand, CNN does not learn the structure, as shown in Figure 5, so it performs more poorly on TRAIN-ENVIRONMENT, but must also learn more robust avoidance techniques for the black tiles. We believe that this obstacle avoidance leads to improved generalization in the maze environments.

These results provide some evidence that convolutional neural networks possess strong generalization capabilities in RL. Future work can investigate variations in CNN hyperparameters to see if a model exists with strong training and generalization performance.

Acknowledgements

We would like to thank Professor Cathy Wu and TAs Chanwoo Park and Gilhyun Ryou for their guidance and support during the course.

References

[1] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A Survey of Zero-shot Generalisation in Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:2111.09794, November 2021.

[2] Mengyuan Yan, Qingyun Sun, Iuri Frosio, Stephen Tyree, and Jan Kautz. How to Close Sim-Real Gap? Transfer with

Segmentation! *arXiv e-prints*, page arXiv:2005.07695, May 2020.

[3] Satchel Grant and Joakim Rishaug. Gym-snake. `https://github.com/grantsrb/Gym-Snake`, January 13 2021.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv e-prints*, page arXiv:1707.06347, July 2017.

[5] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.