Reinforcement Learning- Final Course Project

Jonathan Erell (ID. 207044447),Etay Todress (316517689)

Final project report for the RL course, Reichman University, 2023

1 Introduction

Sokoban is a classic puzzle game that originated in Japan in the 1980s. In the game, the player controls a warehouse worker who must push crates or boxes around a maze-like warehouse to their designated storage locations. The game is played on a grid, and the worker can push and pull boxes, one box at a time. This paper explores the use of reinforcement learning (RL) to solve the game of Sokoban. Specifically, we investigate the challenges of training an RL agent in a sparse reward world, where rewards are only provided at the end of the game. In reinforcement learning, agents learn to perform tasks by receiving feedback in the form of rewards. However, in such environment, rewards are almost given only at the end of the game, leading to sparse reward scenarios. This presents a challenge for training RL agents, as they may struggle to learn from delayed feedback. To address these challenges, researchers have proposed various techniques, such as curriculum learning, intrinsic motivation, and hierarchical reinforcement learning. Our experiment included various techniques to address this issue, but couldn't find a one-size-fits-all solution, for all random environments.

1.1 Related Works

In recent years, researchers have proposed various techniques, such as intrinsic motivation, curiosity-driven exploration, and reward shaping, to encourage agents to explore and learn in the absence of immediate feedback. Priority experience replay, which prioritizes important transitions in the agent's memory buffer, has also been shown to be effective in sparse reward scenarios. In our work, we drew inspiration from existing literature to tackle the sparse reward problem in the game of Sokoban. Specifically, we were inspired by Shangtong Zhang Github, who used priority experience replay to improve agent performance in a sparse reward environment, and [1], who developed a Sokoban solver using reward shaping. However, we found that while these approaches worked well in some scenarios, they were not universally effective, and a one-size-fits-all solution remains elusive.

2 Solution

2.1 General approach

To solve the project missions, we tried multiple algorithms, ranging from basic naive algorithms such as DQN to more advanced algorithms such as Dual Double Deep Q-Network (D3QN) and Proximal Policy Optimization (PPO). Our strategy was to start with the smaller and lighter models first and check if our approach worked. We tackled the easier types of environments first, with each environment changing for every different seed. This allowed us to gain a sense of how our agents learned and how long it took them to learn. Once we had this knowledge, we were able to tackle the main tasks with more confidence and a better understanding of the problem.

2.2 Design

Our code was developed and trained on the Google Colab platform, with limitations in runtime space requiring us to optimize for efficiency. We trained multiple models, including DQN, D2QN, and D3QN, using a common architecture consisting of three convolutional layers, ReLu activation, flatten, and dense layers. We experimented with different loss functions and optimizers and trained models for a limited number of episodes (100) to minimize training time.

For the second exercise, we added an implementation of the PPO algorithm using the same base architecture and added additional layers specific to PPO. We also experimented with using a VGG net as a pretrained model and adding layers on top. However, we couldn't find a one-size-fits-all solution, and we continued to experiment with different techniques to optimize agent performance in different environments. Throughout the project, we encountered technical challenges related to limited runtime space and the sparse reward structure of the Sokoban environment.

3 Experimental results

3.1 Exercise 1 - Single Environment

To evaluate the performance of our models in the first environment, we conducted a series of experiments. First, we trained three different types of models - DQN, D2QN, and D3QN - using the same hyperparameters. We first compared the impact of prioritized experience replay (PER) on each algorithm and found that they all converged significantly faster when trained with PER compared to without. We then evaluated the performance of each model in terms of convergence time and exploration, to gain insight into which approach might be more effective for later use.

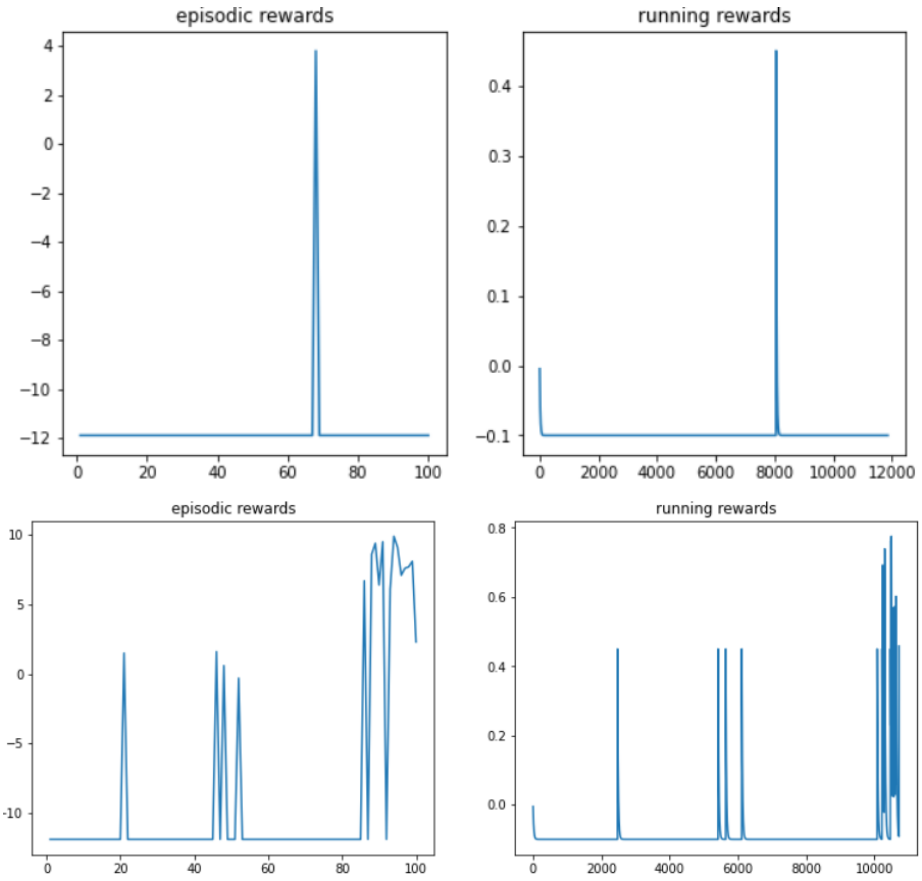

Figure 1: Caption for Images 1 and 2

As shown in Figure 5, we observed that models that did not use PER, such as DQN and D2QN, didn't converge during 100 episodes. This is likely due to the extremely sparse reward structure of the environment, which makes it difficult for the models to learn without prioritized sampling.

Secondly, we conducted a hyperparameter search to identify the optimal parameter values for the D3QN algorithm. Due to resource limitations, we were unable to run this experiment for all of our models. Instead, we selected D3QN, which we believed to be the most promising approach based on our previous evaluations.

We found that D3QN outperformed DQN and D2QN in terms of convergence time and final reward. This may be because D3QN introduces a dueling architecture, which reduces noise in the value estimates by dividing the state-action value from the values of all other actions. This allows for better exploration and exploitation of the state space.

For our hyperparameter tuning, we tried a variety of parameters to opti-

mize the performance of the D3QN algorithm. We experimented with different values for the gamma discount factor, including 0.8, 0.9, and 0.99, to determine its impact on the learning process. We also explored the epsilon greedy concept, using an epsilon greedy approach with epsilon decay for exploration when true. This allowed us to balance exploration and exploitation during the learning process and achieve better convergence times. Finally, we investigated the impact of different reward initialization methods, testing three different values: "positive," which replaced the default rewards with larger ones without negative rewards; "negative," which punished the agent more than the default rewards; and a default approach that used the standard reward scheme provided by the environment. By tuning these hyperparameters, we were able to gain more knowledge on which algorithm to move on with.

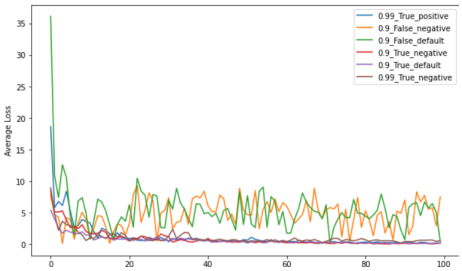

Figure 2: Average Loss as a function of episode's number

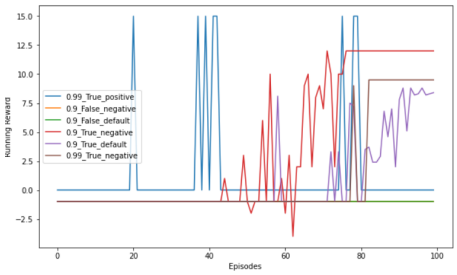

Figure 3: Average reward as a function of episode's number

To explore the impact of different hyperparameters on the performance of our models, we plotted a selection of randomly chosen hyperparameters. Each model was named based on the hyperparameters used, including gamma, epsilon-greedy (boolean value), and reward initialization (type). Our plots revealed several insights about the learning process. Firstly, we found that models that did not use the epsilon greedy method often diverged due to the sparse reward structure of the environment. Secondly, even as the loss decreased, some models did not always converge due to the nature of the D3QN model, which may learn biased or inaccurate target values. We also observed that larger gamma discount factors tended to lead to faster convergence, as they gave more value to rewards that arrived later in the trajectory. Finally, we were able to observe the learning process by tracking the moment each model achieved their first final reward - we found that all models tended to perform similarly, learning quickly after the first significant reward was achieved.

3.2 Exercise 2 - Multiple Environments

In this exercise, our goal was to build a model that could generalize well across all environments. We knew that in the world of RL, such problems are typically addressed using actor-critic methods, which have the ability to learn the policy itself rather than a Q function like DQN algorithms. To this end, we added

the PPO algorithm to our mix of methods. In addition, we followed the reward shaping theory proposed by [1] and implemented a reward shaping function. Reward shaping is a technique that helps to converge in a sparse rewards world by modifying the initial rewards given by the environment in a way that facilitates learning, even if the final reward is not achieved. Our reward shaping function was based on a measure of the distance from our agent to the box, as well as the distance from the box to the target. This way, we motivated our agent to reach the goal at every step and punished it more otherwise.

3.2.1 Experiment 1

As we did in exercise one, we wanted to understand the environments and the challenges they posed. Therefore, we conducted a small hyperparameter tuning experiment on each of the models - DQN, D2QN, D3QN, and PPO - using the same discount factors, learning rates, and other hyperparameters. Specifically, we evaluated the impact of prioritized experience replay (PER) and reward shaping on these models, using the environment from the first exercise.

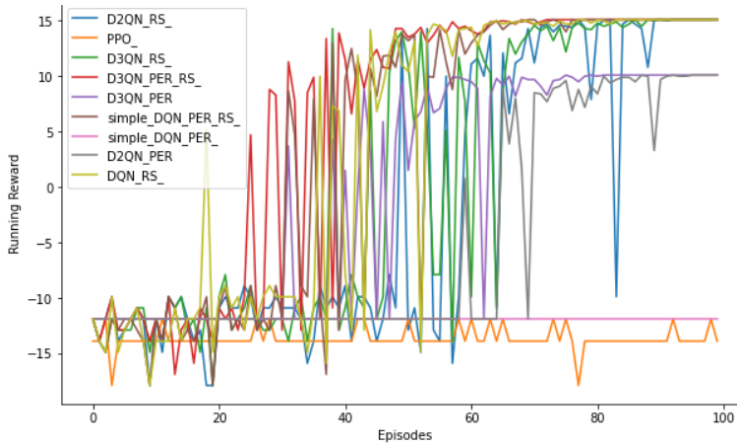

Figure 4: Episode's average reward

The following plot 5 provide a wealth of insights. Each color in the plot corresponds to a model with a specific name convention. If "PER" is included in the name, the model used prioritized experience replay, and if "RS" is included, the model used reward shaping. First, we observe that every model that used reward shaping as a reward method was able to converge. In addition, we see that convergence occurred much faster when both PER and RS were used, as the first final reward was achieved in around 20-30 episodes. Furthermore, we observed that the PPO algorithm did not perform well and was one of the only models that did not converge. Therefore, we decided to proceed with the D3QN model, which uses both PER and RS. We randomly selected 50 different

environments to test the model on.

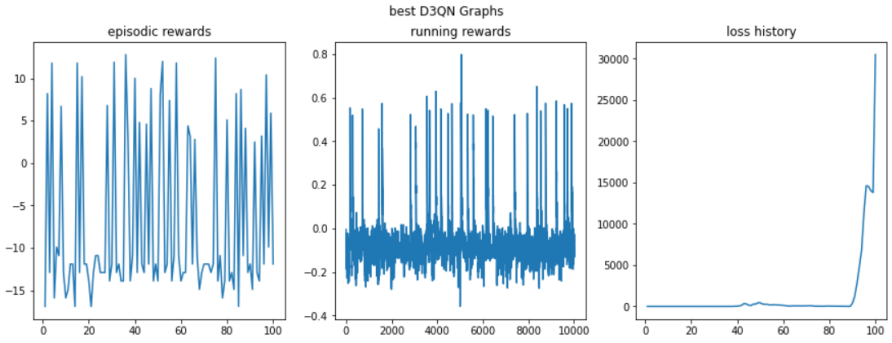

Figure 5: Average rewards, running rewards and loss - given episode

Although this specific model was able to solve each environment individually, it was unable to solve them all together. This demonstrates the difficulty of generalizing across multiple environments and underscores the importance of continued research in this area.

3.2.2 Experiment 2

For the second experiment, we utilized a pretrained D3QN model that had been trained on several easy levels. We had previously defined five environments that were solvable in a single step and trained a model using PER and RS on them. Our idea was that since this type of algorithm was able to solve each environment individually but struggled when presented with multiple environments, a pretrained model might help it converge more quickly. The pretrained model was able to converge successfully on the five easy levels it had been trained on. However, when we ran it on different levels that the model had not seen before, we observed that it not only failed to converge, but also appeared to diverge. This suggests that the model was unable to generalize to new environments, further highlighting the challenges of achieving robustness and generalization in these types of environments.

As seen from figure 6, it appears that we were far from achieving convergence. The episode rewards only seem to succeed on the 1-step environments, which may be due to the algorithm's greediness. Moreover, the loss tends to increase, indicating that the model is not effectively learning from its experiences. These results demonstrate the difficulties of generalization in RL environments and underscore the need for further research in this area.

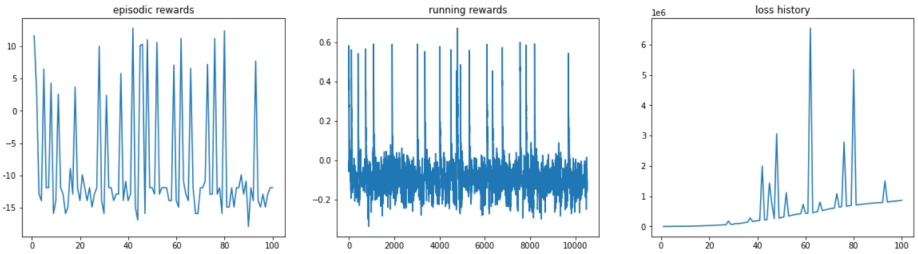

Figure 6: Average rewards, running rewards and loss - given episode

3.2.3 Experiment 3

For the third experiment, we changed our approach to the problem. We believed that if a model was able to succeed on each environment individually, it was likely because it had learned the underlying patterns and features well enough to predict the appropriate actions in any given situation. To test this hypothesis, we decided to use a pretrained VGG19 neural network architecture. We chose this specific DNN because it had been trained as a classification network, which is relevant to our goal of selecting actions based on the current environment situation. We added the D3QN head layers on top of the pretrained VGG net and began training on multiple environments. Unfortunately, we were once again unsuccessful, with the model failing to converge.

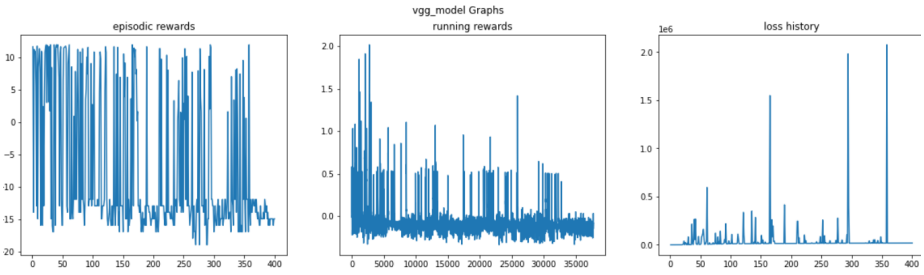

Figure 7: Average rewards, running rewards and loss - given episode

The graphs from figure 7, shows that there was not a change in the three measures we used - average rewards, running reward, and loss. While this doesn't necessarily indicate that the model diverged, the limited amount of time we had to run the experiment makes it difficult to determine whether convergence was achieved.

4 Discussion

In this project, we delved into the world of deep RL and faced various challenges along the way. Our initial approach was to tackle the easiest environments using the most basic and naive method, but we quickly realized the challenge of sparse rewards. To overcome this challenge, we implemented a prioritized experience replay (PER) to help with exploration (without such method, some environments are rarely gets to the finish line, where they receive positive reward), and reward shaping to help the agent learn faster. Without these techniques, our models struggled to converge and failed to learn effectively.

In our second exercise, we encountered a new challenge where the model struggled to learn multiple environments at once, even simple ones. We realized that we did not place enough emphasis on the importance of the convolutional neural network (CNN) in solving such problems. We believe that training a network to identify the object in the scene perfectly could be the key to achieving convergence. Therefore, we plan to experiment with different CNN architectures in future projects.

Throughout the project, we discovered that the process of experimentation and evaluating different techniques is crucial in deep RL. We learned the importance of setting hyperparameters and tuning them to optimize the performance of our models. Additionally, we found that a pre-trained model can be useful in some scenarios, but it does not necessarily guarantee success on new environments.

Overall, we gained a deeper understanding of the challenges and importance of deep RL, and we are excited to continue exploring this field in future projects. We believe that the techniques and insights gained from this project will serve as a foundation for further exploration and experimentation in the field of deep RL. Last but not least, we acknowledge that our results were limited by our resources, such as the runtime space and hardware we had access to. With better resources, we could have run our models longer and trained deeper neural networks, which could have brought us closer to convergence.

5 Code

Google Colab's Code

References

[1] Zhao Yang, Mike Preuss, and Aske Plaat. Potential-based reward shaping in sokoban. *arXiv preprint arXiv:2109.05022*, 2021.