An Application of Reinforcement Learning to Minihack Environment

Devesh Jeaven (2312213)
School of Computer Science and Applied Mathematics
University of Witwatersrand

Thisra Pillay (2110898)
School of Computer Science and Applied Mathematics
University of Witwatersrand

Github Repository : https://github.com/deveshj48/RL_assignment_MiniHack

***Abstract*—MiniHack is a sandbox framework which can be utilised to create custom environments for reinforcement learning (RL). This paper explores the application of two different RL methods in this framework. The methods investigated are a Deep Q Network (DQN) and Soft Actor Critic (SAC). The DQN and SAC agents were trained on smaller environments and then applied to the *MiniHack-Quest-Hard-v0* environment. These smaller tasks are called sub-tasks and while the agents were able to solve simpler navigation tasks, they did not perform well in skill acquisition environments.**

I. Introduction

Reinforcement learning is a domain of machine learning in which the agent aims to learn a behaviour that maximises the accumulated reward. Environments in this domain can vary greatly in terms of action space and state space. Additionally, these environments can have positive or negative rewards.

MiniHack is a sandbox framework for generating custom environments for RL. It leverages the NetHack dynamics and entities to create unique environments [1]. This paper showcases the application of two types of RL agents in the *MiniHack-Quest-Hard-v0* environment. The first agent is a DQN, which is a value function method. The second agent is a SAC, which is a hierarchical method. In this implementation, sub-tasks are investigated.

Section II describes the DQN agent, while section III details the SAC approach. Following this, section IV and section V discuss the results and analysis.

II. DQN

A. DQN Algorithm

The DQN algorithm begins with the initialisation of the replay buffer, the action-value function and the target action-value function. For a number of episodes, an action is selected from the current state using an epsilon-greedy policy. This action is executed and a reward and observation is produced. This tuple of (current state, action, reward, next state) is stored in the replay buffer and will be used for training. Many tuples of this nature are generated from the specified environment and stored as training data in the replay buffer. Subsequently, a random batch of these tuples are sampled from the replay buffer. This batch contains a mixture of old and new tuples to promote diversity and avoid the Catastrophic Forgetting problem where sequential data results in the network forgetting what it has learned earlier. The batch of samples is fed into both the Q-network and the target-network. The Q-network uses the current state and action from the sample to predict the Q-value for the action in the sample. The target network uses the next state from each sample to predict the optimal Q-value out of all the actions possible from the next state in the sample. the loss is calculated using the output from the Q-network, target-network and the reward. The DQN is trained by optimising the loss calculated. Additionally, after a number of updates, the target-network is made to be a clone of the Q-network [2].

B. Q-Network Architecture

The architecture of the Q-network was treated as a hyper-parameter. Initially, a convolutional neural network inspired by the structure in [3] was chosen. A structure consisting of 3 convolutional layers and 2 linear layers, each with the rectified linear function (ReLU) as an activation function was chosen. However, upon evaluation of the initial rewards a strictly linear approach was chosen. This network consists of 4 linear layers, each with the ReLU activation function to accommodate for negative rewards. The first layer is comprised of 81 input features and 64 output features. The second layer consists of 32 output features, the third layer has 16 output features, and lastly the fourth layer has a number of output features which is equivalent to the size of the action space. In this implementation, the starter code provided for the DQN lab was used.

C. Hyperparameters

There are a range of optimisers used in reinforcement learning problems. Two notable optimisers are the Adam optimiser and RMSProp. Using the *MiniHack-MazeWalk-9x9-v0* environment, the effect of the aforementioned optimisers was investigated. It can be seen that the Adam optimiser is a more suitable choice when compared to the performance of the RMSProp optimiser. This is shown in Figure 1 and Figure 2.

The DQN has many hyperparameters, as shown in Table II. Each hyperparameter can be described as follows:

- env : Specify which environment to use

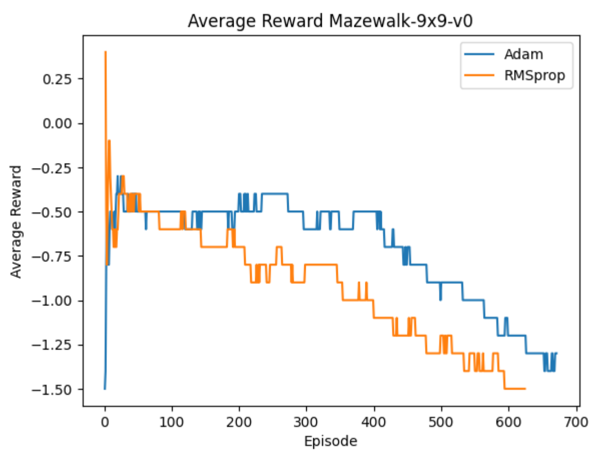

Fig. 1. Comparision plot of optimisers showing the average reward

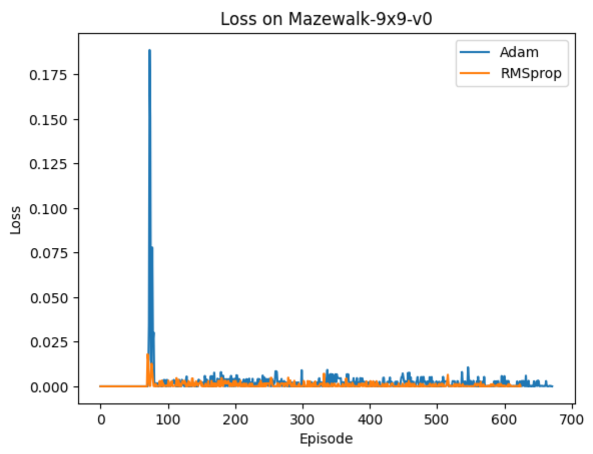

Fig. 2. Comparision plot of optimisers showing the loss

- Replay Buffer Size : Specifies the size of memory available to store sequences of actions and observations used for training
- Learning Rate : Defines a step-size in which the weights are updated when using an optimiser. This value affects how much the weights of the network are adjusted during weight updates
- Discount Factor : Defines how rewards are prioritised. If this value is closer to zero then immediate rewards are seen as more valuable. If this value is closer to one, then long-term rewards are more valuable
- Number of Steps : Total number of steps for which the environment is run
- Batch Size : Number of transitions to optimise simultaneously from the replay buffer during training
- Learning-starts : Number of steps before learning starts in which sequences of transitions are collected in the replay buffer
- learning-freq : Number of iterations between every optimization step

Hyperparameter	Value
env	MiniHack-Room-5x5-v0
Replay Buffer Size	5000
Learning Rate	0.0001
Discount Factor	0.99
Number of Steps	1000000
Batch Size	256
Learning-starts	10000
learning-freq	5
use-double-dqn	True
target-update-freq	1000
eps-start	1.0
eps-end	0.01
eps-fraction	0.1

TABLE I
HYPERPARAMETERS OF DQN

- use_double_dqn : Whether or not a Double DQN is used
- target-update-freq : Defines the number of iterations before which an update for the weights of the target network occurs
- eps-start : Start threshold for epsilon-greedy
- eps-end : End threshold for epsilon-greedy
- eps-fraction : Defines the amount of the training period for which the value of epsilon is reduced. In this way, exploration and exploitation occur but exploration is reduced gradually over time as the agent learns more about its environment

For the tuning of these hyperparameters, the *MiniHack-Room-5x5-v0* environment was used. Initially, the values in Table II were used, however, through manual tuning improvements in performance were achieved. Increasing the size of replay buffer increases the range of samples that are used in the DQN, thus increasing performance. The learning rate was increased to 0.001. The discount factor remained the same, to promote the value of long-term rewards. Due to limited GPU access, the number of steps for which the environment was run was set to 500000. As a result, batch size (64), learning-starts (5000), learning-freq (2), target-update-freq (100) and eps-fraction (0.3) were adjusted to accommodate for a shorter training period. Additionally, double Q-learning was used to accommodate for the maximisation bias in DQN agent.

III. Discrete Soft Actor-Critic

Many reinforcement algorithms struggle from high sample complexity and very sensitive convergence properties. The soft actor-critic (SAC) algorithm aims to solve these challenges by maximising both expected reward and entropy [4]. SAC is an actor-critic method whereby a Q-function is trained (critic) and a policy is simultaneously optimised (actor). It is a popular algorithm for solving complex tasks in a continuous action space. Since MiniHack contains many challenging environments but uses a discrete action space, we have decided to implement a discrete adaptation of SAC.

SAC is an off-policy algorithm. It is thus able to learn from past samples and replay buffers. Since it does not require new samples after each policy update like on-policy learning, it is highly sample-efficient. In MiniHack environments, we can produce many random samples without much effort, which proved to be advantageous when using SAC [4].

SAC makes use of various efficient and innovative algorithms, such as target networks, double Q-trick and entropy regularisation [5].

The target networks help us predict the next state actions and Q-values.

The double Q-trick consists of the algorithm containing two Q-functions and choosing the minimum value between the two approximators.

Entropy regularisation is the main feature of SAC. Entropy is a measure of randomness, and SAC can control the randomness of the agent. This is related to the exploration-exploitation trade-off - the higher the entropy results, more exploration is done [6].

Putting together the key elements of SAC, we calculate the loss for each Q-value as follows:

$$L(\phi_i, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{E}[(\mathrm{Q}_{\phi_i}(s,a) - y(r,s',d)^2]$$

where the target y is:

$$y(r,s',d) = r + \gamma(1-d)(\underset{j=1,2}{min}\mathrm{Q}_{\phi_{targ,j}}(s',a') - \alpha log\pi_\theta(\tilde{a}'|s'))$$

In the target algorithm, $\mathcal{D}$ represents the replay buffer, $-\alpha log\pi_\theta(\tilde{a}'|s')$ represents the entropy and Q is a modified value function equation including entropy [6].

A. SAC Architecture

The definitions for some of the hyperparameters, not mentioned previously, are as follows:

- episodes : Number of episodes
- buffer_size : Size of the replay buffer
- lr : Learning Rate
- hidden_size : Number of nodes in hidden layer for the neural network model
- interpolation_factor : Also known as the smoothing factor, this value contributes to how often the target weights are updated to match the current value function weights [7].
- clip_grad_param : This is used to bound the gradient, for gradient clipping. Gradient clipping prevents the exploding gradient problem.

Hyperparameter	Value
episodes	500
buffer_size	1000000
seed	42
batch_size	256
discount	0.99
lr	0.0002
hidden_size	256
interpolation_factor	0.005
clip_grad_param	1
max_episode_steps	1000

TABLE II
HYPERPARAMETERS OF SAC

The hyperparameters were manually tuned on the MiniHack-MazeWalk-9x9-v0 environment.

SAC makes use of neural networks for the actor and critic (usually the same). In order to sufficiently explore the use of SAC in this context, we have implemented two different approaches.

For the first implementation, we used a neural network (NN) with three fully connected layers. The actor network had the addition of a softmax layer, as it determines the action to choose. The number of hidden layers was a hyperparameter. The neural network architecture, as well as the foundation of the code was drawn from here. This approach was considered as fully connected layers learn from all combinations of the features from its previous layers, thus encapsulating as much information as possible.

The second implementation consisted of a mixture of convolutional layers and fully connected layers. It was based off of the Lenet-5 convolutional neural network (CNN) architecture. This implementation was drawn from here [8]. This network consists of 2 convolutional layers, two max-pooling layers and four fully connected layers. This approach was considered because it is computationally more efficient than a fully connected layer. It relies on creating meaningful relationships locally, which can be reflected on a global scale. It also learns features so that it detects patterns that are otherwise difficult to do manually. This network was appropriate for this context as many Minihack environments consist of complex tasks and require methods that can solve these tasks in a computationally efficient way.

IV. RESULTS

MiniHack provides a variety of environments to train agents, which are divided into three task categories. These include, the navigation task environments where the goal is to reach a position or navigate through a complex maze, the skill acquisition task environments where the agent interacts with NetHack objects, such as, monsters, and lastly, the ported task environments. This paper focuses on the first two environments.

A. Sub-Tasks

The DQN and SAC agents were trained on multiple environments. NetHack has a large action space which increases difficulty in training agents, thus the action space was restricted in each of the tasks to promote skill discovery. The first environment that was used, was the *MiniHack-Room-5x5-v0* navigation task environment, in which the agent is required to reach a goal position. In this environment, the agent receives a reward of -0.01 when a boundary wall is hit and a reward of +1 when the goal position, i.e., the staircase is reached. Additionally, this task has a relatively small action space of 8. This includes the NetHack compass directions, namely, north, north east, east, south east, south, south west, west and north west.

1) DQN Results: Training the DQN on this environment resulted in the average reward in Figure 3. Although the DQN agent performed poorly in the initial episodes, as training continued the average reward converged to a reward of 1.0 showing that the agent was able to solve this task.

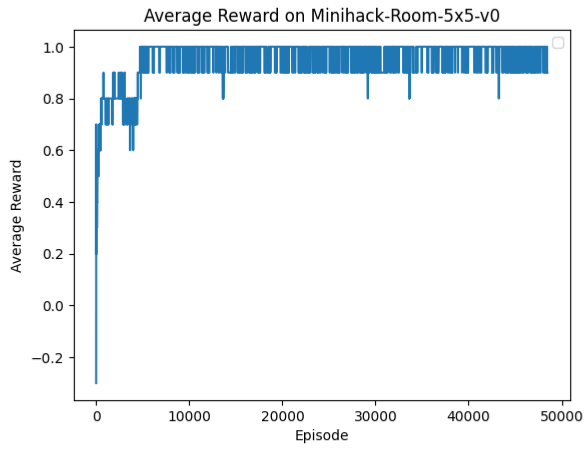

Fig. 3. Average Reward for the MiniHack-Room-5x5-v0 environment

The next sub-task that was investigated was the navigation task environment of *MiniHack-MazeWalk-9x9-v0*. This task has the same action space as the *MiniHack-Room-5x5-v0* environment. The objective of this task, however, is to successfully navigate through the maze generated.

2) DQN Results: Training of the DQN on this environment produced the results summarised in Figure 4. As shown in the figure, the DQN agent was not able to solve this task. The average reward in this case persisted as a negative value and decreased as training continued for 500000 steps.

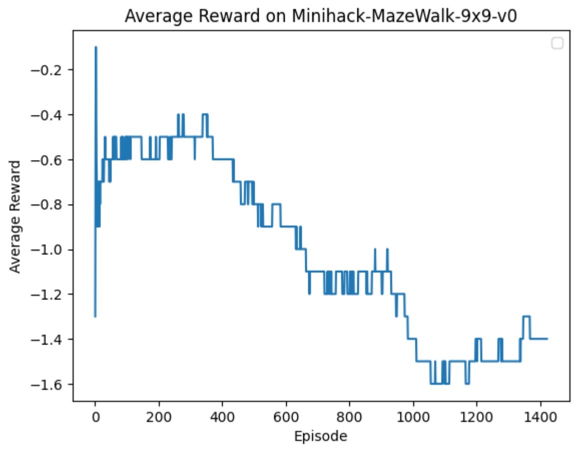

Fig. 4. Average Reward for the MiniHack-MazeWalk-9x9-v0 environment

The last sub-task environment was the *MiniHack-LockedDoor-v0* skill acquisition environment. In this environment, the objective is to kick locked doors. The action space consists of the aforementioned compass directions, as well as the 'kick' action. As shown in Figure 5, the reward decreased as training continued. This shows that the DQN agent was not able to solve this task.

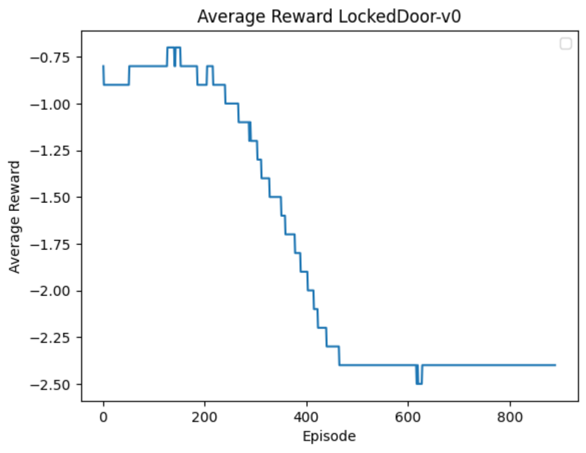

Fig. 5. Average Reward for the MiniHack-LockedDoor-v0 environment

B. MiniHack-Quest-Hard-v0 Environment

The *MiniHack-Quest-Hard-v0* environment is one in which the agent is required to complete a quest. This environment is the most challenging of the quest environments in MiniHack and is composed of many tasks. In this context, the agent needs to navigate through a maze, cross a lava river and use a wand to defeat a monster guarding the goal. While on this quest, the agent needs to perform other tasks, such as, defeat monsters and kick locked doors.

In this environment the action space was composed of the 8 cardinal directions as well as the 'KICK', 'PICKUP', 'APPLY', 'FIRE', 'RUSH', 'ZAP','PUTON', 'READ', 'WEAR', 'QUAFF' and 'PRAY' actions.

1) DQN Results: Figure 6 shows the results obtained from the DQN agent. As in the previous two sub-tasks, the DQN was unable to solve this environment task.

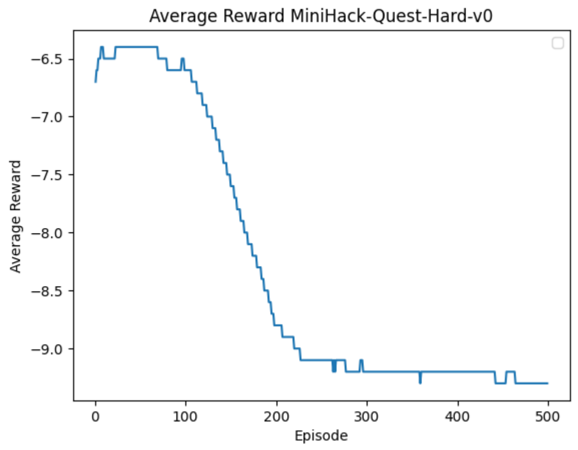

Fig. 6. Average Reward for the MiniHack-Quest-Hard-v0 environment

C. SAC results

The CNN architecture approach completed 500 episodes significantly faster than the fully connected NN, with the

CNN finishing the *MiniHack-MazeWalk-9x9-v0* in 64055 steps and the NN finishing in 343545 steps. The results for the *MiniHack-Room-5x5-v0* consisted of the CNN finishing in 71674 steps and NN in 68475 steps.

It can be seen in Figure 8 that the agent performed poorly in both tasks, even after tuning hyperparameters. These results are from using the CNN for the actor and critic networks.

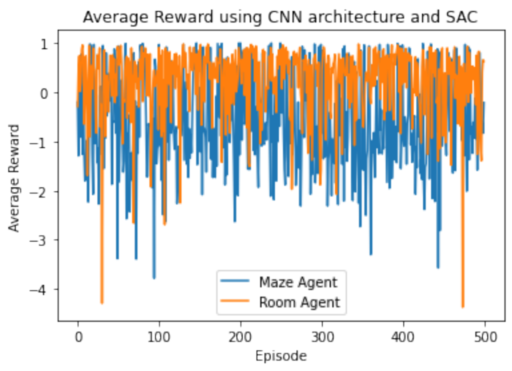

Fig. 7. Average Reward for the MiniHack-MazeWalk-9x9-v0 and MiniHack-Room-5x5-v0 environment using the CNN approach

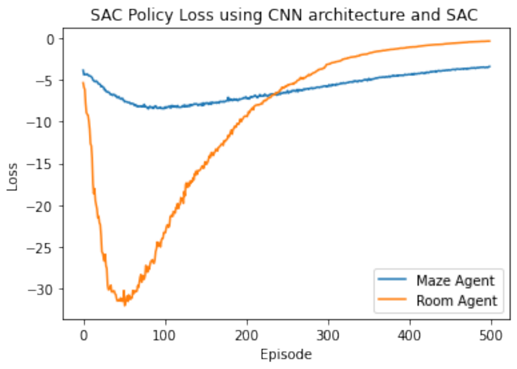

Fig. 8. Policy Loss for the MiniHack-MazeWalk-9x9-v0 and MiniHack-Room-5x5-v0 environment using the CNN approach

The following results are from using the fully connected NN for the actor and critic networks:

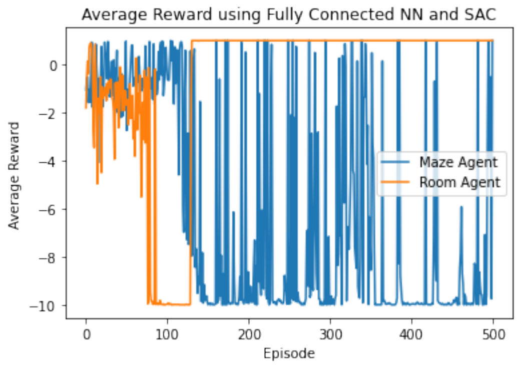

Fig. 9. Average Reward for the MiniHack-MazeWalk-9x9-v0 and MiniHack-Room-5x5-v0 environment using the NN approach

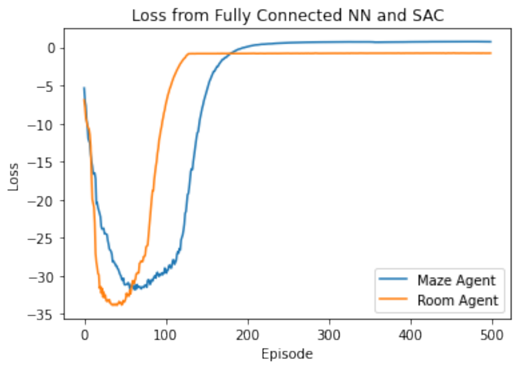

Fig. 10. Policy Loss for the MiniHack-MazeWalk-9x9-v0 and MiniHack-Room-5x5-v0 environment using the NN approach

Section VI shows the comparison plots between the CNN and NN approach.

V. Analysis

A. DQN

As shown in section IV, the DQN performed well on simpler navigation tasks and performed poorly on more complex tasks. For the *MiniHack-Room-5x5-v0* environment, the agent was able to converge to 1.0 and rarely produced negative rewards on average. In the case of the remaining sub-tasks and the *MiniHack-Quest-Hard-v0* environment, the agent only produced negative average rewards, which increased in negativity as training continued. This suggests that the hyperparameters of the DQN could be tuned more effectively to improve performance. A further investigation of network architectures may show promise, as well as exploring the effect of different state representations in the network.

Due to the difficulty of the *MiniHack-Quest-Hard-v0* environment, it is possible that another approach may need to be taken. One such approach, which is an improvement to the DQN, is the Dueling DQN. In addition, exploration of reward shaping may prove beneficial. Reward shaping is a method in which intermediate rewards are given to help the algorithm converge at a faster rate.

B. SAC

From the results of the SAC implementations, it is evident that the agents were not effectively trained. This could be due to the observation spaces not being fed into the networks in a way that they could fully extract important information. The spatial information of the agents could also have been lost, resulting in an ineffective model being trained.

The Average Reward for the *MiniHack-Room-5x5-v0* agent in Figure 9 immediately converged to 1 after around 130 episodes. This caused it to finish faster than the CNN approach for the *MiniHack-Room-5x5-v0*. Although achieving the goal, it could mean that it stopped exploring, which could be detrimental in certain non-stationary environments. It can also be seen that at that time, the policy (actor) loss also immediately converged to 0.

It was also noted that the policy loss from the both SAC implementations inevitably increased after a number of episodes. Multiple learning rate values were tried, however, the same problem occurred. A possible issue may the problem of the moving target, whereby the agent uses the same neural network to estimate the Q-value and target value [9]. The target network could have incorrectly been updated, or the interpolation factor could have not been adjusted enough to compliment the learning rate.

VI. Other Graphs

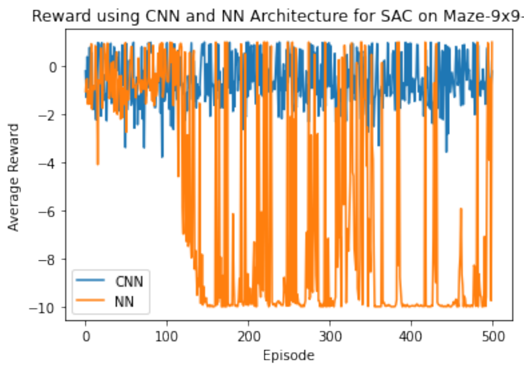

Fig. 11. Reward using CNN and NN Architecture for SAC on MiniHack-MazeWalk-9x9-v0

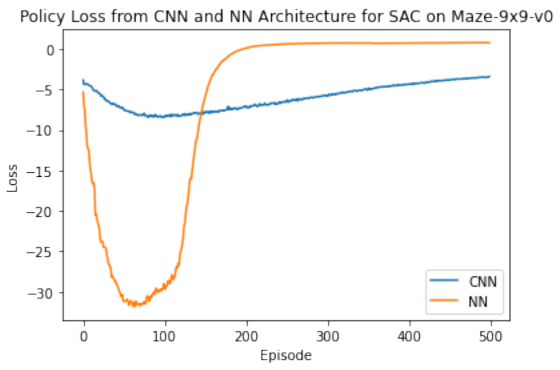

Fig. 12. Loss from CNN and NN Architecture for SAC on MiniHack-MazeWalk-9x9-v0

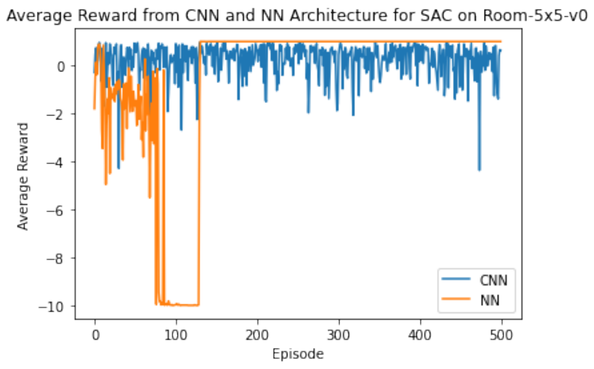

Fig. 13. Reward from CNN and NN Architecture for SAC on MiniHack-Room-5x5-v0

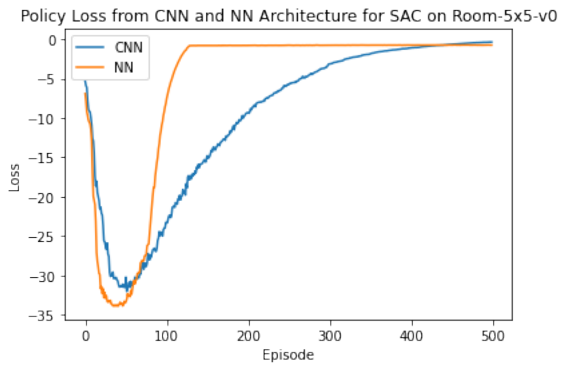

Fig. 14. Loss from CNN and NN Architecture for SAC on MiniHack-Room-5x5-v0

References

[1] M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Küttler, E. Grefenstette, and T. Rocktäschel, “Minihack the planet: A sandbox for open-ended reinforcement learning research,” *CoRR*, vol. abs/2109.13202, 2021. [Online]. Available: https://arxiv.org/abs/2109.13202

[2] K. Doshi, “Reinforcement learning explained visually part 5 deep q networks step by step,” https://towardsdatascience.com/reinforcement-learning-explained-visually-part-5-deep-q-networks-step-by-step-5a5317197f4b, accessed: 2023-10-29.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, pp. 529–533, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:205242740

[4] *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*, 2018.

[5] F. Hofstätter, “Adapting soft actor critic for discrete action spaces.”

[6] OpenAI, “Soft actor-critic.”

[7] S. Santhosh, “Reinforcement learning(part-5): Soft actor-critic(sac) network using tensorflow2.”

[8] B. Budler, T. Packirisamy, K. Nair, and N. Raal, “deep-rl-minihack-the-planet,” 2021.

[9] A. Suran, “Diving into deep reinforcement learning with deep q learning.”