COMS7053A: Reinforcement Learning - Assignment Minihack the planet

Leantha Naicker (788753) & Clerence Mathonsi (2512711)

November 2021

Abstract

This report looks at the performance of two reinforcement learning agents, namely Deep-Q Network and REINFORCE methods, to solve a complex task environment in the MiniHack framework named *MiniHack-Quest-Hard-v0*. Unfortunately neither agent is able to actually complete the task; and whilst the DQN agent struggles to get out of the maze of the environment by getting stuck at dead ends and walls, the REINFORCE agent is able to learn and gain rewards over time. Further work and recommendations are made to try to improve the performance of the agents.
The codes used in this project were greatly aided by the labs conducted in this course along with the work of Cheng Xi Tsou [14].
The GitHub repository for this project can be accessed at github.com/COMS7053A_Assignment.

1 Introduction

Advances in Deep Reinforcement Learning has driven the development on novel simulation environments which introduced new challenges for methods and demonstrated their limitations. This sparked the research for novel exploration methods and learning from demonstrations. This paper looks at a complex environment called the *MiniHack-Quest-Hard-v0* dungeon task in MiniHack and documents the experiment of comparing Deep-Q Networks (DQN) and REINFORCE agents to navigate it. The success of the agents are determined by the point accumulation.

2 What is MiniHack?

MiniHack is a sandbox framework for easily designing controlled Reinforcement Learning (RL) environments for RL experiments that widely range in size and complexity. It is essentially a wrapper of the NetHack Learning Environment (NLE) and whilst MiniHack comes with a sufficient list of challenging tasks; the primary innovation of it is to ease the design of new ones [1]. The MiniHack github page contains useful instructions on how to install and begin using the framework [here].

The Gym environment observation space is achieved by wrapping a low-level NetHack object responsible for features, rewards and episode terminal dynamics. The Gym wrapper, by default, only exposes the following data in the form of the tuple (glyphs; stats; message; inventory).
Glyphs are ids in the range of 0 to MAX_GLYPH that support displaying different objects in the task map as different to the tiles. Stats is an integer vector containing the coordinates of the character and other character statistics. Message is a byte vector representing the current message displayed to the player. Inventory is the characters inventory collected and acquired.

The Minihack action space uses ASCII inputs as predefined in it's documentation [2]. Below is a small example:

Name	Value	Key	Description
DOWN	62	>	go down (e.g. a staircase)
DROP	100	d	drop an item
EAT	101	e	eat something
INVENTORY	105	i	show your inventory
PICKUP	44	,	pick up things at the current location
OPEN	111	o	open a door
FIRE	62	>	fire ammunition
KICK	100	d	kick something
SEARCH	101	e	search for things
WIELD	105	i	wield (put in use) a weapon
SIT	243	M-s	sit down
ZAP	112	z	zap a wand

Table 1: Possible Actions

The *MiniHack-Quest-Hard-v0* environment is one of MiniHack's *Quest* tasks. In the this task, the agents need to learn to cross a river of lava, across the river it needs to find a key and use it to open a hidden chest in order to locate the Wand of Death (WoD). This is required to kill the monster in order to collect the goal [3].
The skills acquisition needed for this task environment is navigation, pick-up, inventory and direction.
An image of the Quest-Hard dungeon is shown in Figure 1.

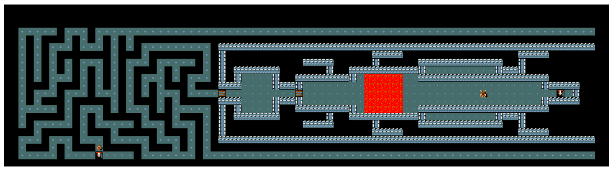

Figure 1: Quest-Hard: A complex task in MiniHack requiring multiple skills [3][4]

2.1 Related work

Reinforcement learning has extensively made use of videos games for RL algorithm testbeds. These games are however computationally expensive, not easily modifiable and often result in over overfit policies due to having only a fixed set of levels [1].
ProcGen [5] addresses the issue by providing a collection of 2D games with generated levels; but the games are still limited to minor modifications. This is unlike the environment creation capabilities of MiniHack.
MiniGrid [6] addresses computational efficiency by providing a collection of generated grid-world tasks; however the complexity of the environments are still limited with few entities and a small action space.
Based on the uses of NetHack and the NetHack Learning Environment (NLE), Minihack is 16 times faster than ALE [7] and faster than ProcGen by 10% [1].

Many rouge-like games, based on generated dungeon levels and grid-based movements have been proposed as RL benchmarks; such as Rougueinabox, Rogue-Gym, NLE and gym-nethack .Whilst MiniHack is not the first to provide a sandbox for developing environments; technologies like PyVGDL, Griddly and GVGAI do exist, MiniHack has a larger collection of predefined objects, monsters, items, etc and more complex environment capabilities from NetHack. This hits the sweet spot between the ability to customize and richness of entities and environment dynamics [1].

3 Methods Investigated

There are a variety of methods in the reinforcement learning space that could be used to find a model for the Quest-Hard environment. These include value-function, model-based, policy and hierarchical methods [4].
Value-based methods learn a value function in order to determine a policy; model-based methods learn optimal behavior indirectly by learning a model of the environment; policy-based methods attempts to learn the optimal policy directly and hierarchical methods decompose a RL problem into a hierarchy of subtasks [8].

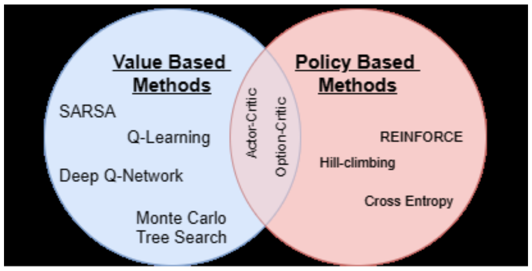

Figure 2: Examples of RL methods classed as Value-based or Policy-based methods [10]

In the end, algorithms Deep-Q Network (DQN) from the value function methods and REINFORCE from the policy methods where chosen for this experiment to be investigated further to solve for an agent in the MiniHack environment. Details of these methods and their implementations are given in the next respective sections.

4 Chosen Method

Off all the methods existing, it was decided to see what a DQN agent and a REINFORCE agent would do with the MiniHack task, given that these aspects were the last two topics of the course labs. It was hoped that the experience gained there would aid with the project.
Both agents were trained over a 1000 episodes, with 1000 steps per episode. Some of the actions given to each agent can be seen in Table 1 . Besides the cardinal directions of moving North, South, East and West; the actions of FIRE, KICK, SEARCH, PICKUP, OPEN and ZAP were given to agents to help them navigate and complete the task.

4.1 Chosen Method:DQN

Deep-Q Networks or DQN were first proposed by DeepMind back in 2015 in attempt to bring the advantages of deep learning to reinforcement learning (RL) [9].
To understand DQN, the root method must first be

understood i.e. Q-Learning; which aims to learn the action-value function Q(s, a). Q-learning is an off-policy Temporal Difference (TD) method [10]. Q-Learning builds a memory table Q[s, a] to store Q-values for all combinations of states and actions for the agent to look up when deciding how good an action is in a particular state. In simple terms, an action is sampled in the current state and a reward R is obtained (if any) with progression to the next state s'. The memory table is then used to determine the next action a' that has the maximum Q(s', a') [11-13].

If state-action combinations are too large, the memory and computation requirements for Q will be too high; this is where DQN comes in to approximate Q(s, a). This means that the Q-value function is approximated rather than remembering a table of solutions. This method also has performance improvement techniques such as *experience replay* and *target network*. Experience replay saves a buffer and uses mini-batch samples from the buffer to train the deep network. Doing these random samples makes data more independent and identically distributed (i.i.d) [11].
Target network creates two deep networks: θ- which retrieves Q-values and θ for all updates in the training. θ- requires synchronization with θ after a set number of updates; for the purpose of fixing the Q-value target temporarily. This removes the problem of chasing moving targets [11].
Target network improvement is important but not as critical as the experience replay which provides the largest performance improvement[11-13].

For the value function method used in this experiment, DQN was selected with implementations of experience replay and target networking. The target and policy network are created using the following layers, which are each separated by Rectified Linear Units (ReLU).
The input layer is a convolutional layer, with 16 filters and a stride of 1. The first hidden layer is a fully connected linear layer with an output size of 512. Lastly, the output layer is a fully connected linear layer with a single output that translates to an action for the agent [10].
Sparsity is reduced in the observation space by cropping a box around the agent's position at each timestep. Default rewards are used for the agent.

The hyper parameters for this experiment are set out in the table below. Most of these parameters where taken from the lab such as alpha and gamma and others from research papers such as the crop size.

Hyperparam	Value	Description
Gamma	0.99	The Discount Factor for rewards
Learning rate	0.02	To update the model parameters
Crop dimensions	14x14	The height and width of the cropped agent view
Total steps	1000 x 1000	The number of steps used to train the model
Batch size	32	The number of transitions per update step
Target network update	1000	The number of iterations before each target network update

Table 2: Hyperparamerters for Dueling DQN model

4.2 Chosen Method: REINFORCE

The REINFORCE algorithm belongs the family of Policy Gradient Algorithms within reinforcement learning. It estimates the optimal policy, parameterised by weights θ, through gradient ascent [8].
The policy is usually a Neural Network, which acts to take in a state and generate the probabilities of taking an action as the output [8][14]. The objective of the policy is to maximize the expected reward.
REINFORCE is built using trajectories instead of episodes because maximizing expected return lets the method search for optimal policies for episodic and continuing tasks. Trajectories are denoted by τ and the return for a trajectory denoted by R(τ).

$$R(\tau) = (G_0, G_1, ..., G_H) \tag{1}$$

The expected return is denoted by U(θ) and can be defined by the equation below:

$$U(\theta) = \sum_{\tau}(\mathbf{P}(\tau; \theta)R(\tau)) \tag{2}$$

One way to determine the weights of the Neural Network θ, that maximises U(θ) is through gradient ascent; which iteratively takes smalls steps in the direction of the gradient. This is expressed by the equation below, where alpha, α, is the step size.

$$\theta = \theta + \alpha \Delta_{\theta} U(\theta) \tag{3}$$

In gradient methods where some probability p can be formulated, the log probability log(p) should be optimized instead of the probability p for some parameters θ. The gradient of log(p(x)) is generally more well-scaled than p(x).

The REINFORCE algorithm was implemented with a cropped observation space of a box around the agent's position at each timestep. The Neural Network architecture is created using the following layers, which are each separated by the action function, Rectified Linear Units (ReLU).
The input layer and 3 hidden layers are all linear layers. The output layer has neurons corresponding to the reduced action space. The output layer is activated by the softmax activation function.
Default rewards are used for the agent.
Below is a list of the hyperparameters used in the training of the agent. Most of the parameters were taken from the REINFORCE lab, except the crop dimensions which came from [10].

Hyperparam	Value	Description
Gamma	0.99	The Discount Factor for rewards
Alpha	1	Learning rate to update θ
Max steps	1000	Maximum number of steps allowed per episode
Number of Episodes	1000	Number of episodes method was trained for
Crop Dimensions	14x14	The height and width of the cropped agent view

Table 3: Hyperparamerters for REINFORCE model

5 Experimentation and Results

Whilst the Quest-Hard environment had many interesting things to learn, explore and achieve; unfortunately neither of the agents created where able to reach the goal at the end of the task, or even get very far in the task at all. The subtasks are listed below with agent performance:

Navigated Maze: REINFORCE method managed to navigate through the maze, DQN method failed to navigate through the maze.
Find Objects: Both REINFORCE and DQN failed to achieve this sub-goal
Lava Crossing: Both DQN and REINFORCE did not cross the lava
Find Key: Both methods failed to achieve this subgoal
Get Wand: The two agents did not get wand
Defeat Demon: Both agents did not make it to this sub-goal
Collect Goal: No goal for DQN and REINFORCE
Score: DQN 0/10, REINFORCE 3/10
As seen above the agents did not get very far in the quest; with the DQN agent never actually getting out of the maze. The following subsections highlight the results in a little more detail.

5.1 DQN results

To determine the results for DQN method we used Glyphs and Chars observation spaces. The method ran through 1000000 steps to reach 1000 episodes. The results of how the method performed are presented in Table 4. For conducting the performance of the model we focused on the loss, steps taken and rewards assigned to the model.

Episode	Average loss	Reward	Score
50	0.02	-4.7	0
200	4.72	-6.7	0
400	0.67	-6.8	0
600	2.09	-4.2	0
800	5.11	-5.5	0
1000	0.08	-6.9	0

Table 4: Performance results for DQN method

The graph for DQN method resulted in a blank graph due to the score staying 0 and gaining no rewards from the environment. Clearly something buggy in the code that could not be found in time before the submission, or just a really poorly designed agent.

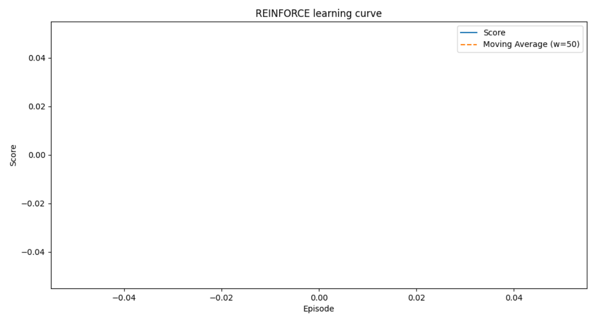

Figure 3: The learning curve for the DQN method

From the high score and rewards of the DQN method it is evident that our model struggles to learn in the environment. Our DQN could confirm the findings of [3], [13] that faced problems with agents being stuck at a wall once they have reached them. We note that during the training of a thousands episodes, the losses oscillate for each update and the model never finds an approximation for the value function.

5.2 REINFORCE Results

To determine which observation space to use for the reinforce method, experiments were conducted on Glyphs, Chars, and Message. The result of each experiment conducted is presented in Table 5.From the results we can see that the REINFORCE method trained on the message observations produced the best result for this method. Therefore for the remainder of this study, the REINFORCE method presented refers to the method trained on the message observation space.

Observation Used	Average Reward
Glyphs	-9.00
Chars	19.04
Message	28.06

Table 5: Results obtained for Reinforce Method

As stated above REINFORCE method was trained using message observation space, it was chosen because of its higher average reward compared to Glyphs and Chars. Table 6 below show the performance results for the REINFORCE method, the table displays the episodes, scores and losses.

Episodes	Score	Losses
50	-7.19	-748554.0
200	-7.53	-787073.9
400	-8.07	-865682.2
600	-7.77	-825391.9
800	-7.22	-753891.6
1000	-7.60	-811427.5

Table 6: Performance results for REINFORCE method

Figure 4 shows the rewards returned after training the REINFORCE method for 1000 episodes capped at 1000 steps per episode. The graph shows that the agent learns and is able to gain rewards that terminate at different stages of the task for each episode.

6 Failed requirements

Due to the lack of data from the DQN agent, it is near impossible to do a comparison plot as it would certainly just look like the REINFORCE reward graph.

Agent video recordings for DQN and REINFORCE in the MiniHack-Quest-Hard environment requirements were not met due to the inability to get the code working to generate the gif. Unfortunately the tools usually used did not work well with recording the MiniHack environment.

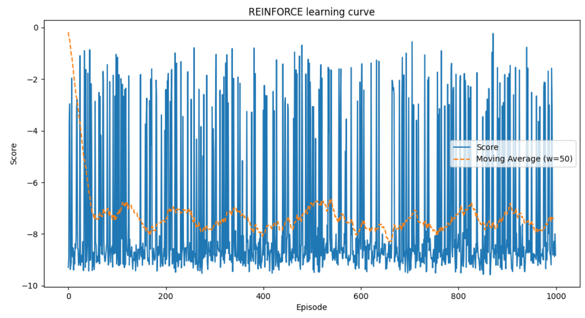

Figure 4: The learning curve for the REINFORCE method.

Attempts where made to use many online solutions and help was even sort for using the custom wrapper provided by a classmate. That wrapper resulted in .json files which can be found in the github repository.

6.1 Future work

This project leaves room for agent improvement for both DQN and REINFORCE. First lets address the biggest issue which is the DQN agent. Result suggest that code itself may not have been working correctly due the agent gaining no rewards at all. In order to further improve performance of the DQN agent; there are two possible features to look into.
1. Implementing Dueling DQN over DQN.
In Dueling DQN, Q is computed with the formula below, where V is the value function and A is a state-dependent action advantage function [11]:

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{A}\sum_{a=1}^{A}(A(s,a)) \tag{4}$$

Instead of learning Q; V and A are used. Whilst DQN updates the Q-value function of a state for a specific action only; Dueling DQN updates V, which other updates in Q(s, a') can take advantage of.

2.Replacing ϵ-greedy with a NoisyNet.
DQN uses ϵ-greedy to select actions, however, using a NoisyNet can replace ϵ-greedy by adding parametric noise to the linear layer. This aids in exploration. The NoisyNet uses a greedy method to select actions from the Q-value function but for the fully connected layers of the Q approximator, trainable noise is added to explore actions [11][12]. Adding noise to a deep network is equal to or better than adding noise to the action via the

ϵ-greedy method [13].

Whilst the REINFORCE agent did actually do much better than DQN it still not achieve the tasks it set out to, i.e. collect the goal at the end of the task. Adding more actions to the action space may have helped the agent in that the complexity of the task probably required more than the watered down action space provided.
Custom reward functions could also aid both agents in learning better if they were properly done.

It would be beneficial to do further analysis on the agents in more basic MiniHack tasks to see how they perform and determine if the agents are just poorly designed in general or poorly designed for the task at hand. Implementing the above recommendations and training the agents for a longer period of time may also aid in improving results.

7 Conclusion

MiniHack provides a wide variety of interesting challenges for exploration, many possible states and numerous environments dynamics to discover. The *Quest-Hard* task lives up to it's name, requiring skill acquisition of navigation, pick-up, inventory and direction.
In this project DQN and REINFORCE agents were created to learn to interact with the MiniHack environment. By comparing the results of the two agents, we can conclude that REINFORCE learns better than DQN in the *MiniHack-Quest-Hard-v0* environment. Both agents were ran using the same number of steps and episodes, from the results conducted in table 4 and table 6 it is clear that by comparing the scores of the two agents REINFORCE is the better option compared to DQN.

References

[1] Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette & Tim Rocktaschel, *Minihack the planet: A sandbox for open-ended reinforcement learning research*, 35th Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1), 2021.

[2] MiniHack, "Action Spaces". [Online]. Available: https://minihack.readthedocs.io/en/latest/getting-started/action_spaces.html. [Accessed: Oct 31, 2021]

[3] MiniHack, "MiniHack Environments". [Online]. Available: https://minihack.readthedocs.io/en/latest/envs/tasks.html, MiniHack, MiniHack Environments. [Accessed: Nov 1-8, 2021]

[4] Reinforcement Learning COMS4047A/COMS7053A MiniHack the planet Assignment Brief

[5] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. *Leveraging procedural generation to benchmark reinforcement learning*, arXiv preprint arXiv:1912.01588, 2019.

[6] Maxime Chevalier-Boisvert, Lucas Willems & Suman Pal, "Minimalistic Gridworld Environment for OpenAI Gym", 2018. [Online]. Available: https://github.com/maximecb/gym-minigrid. [Accessed: Nov 8, 2021]

[7] Marc G. Bellemare, Yavar Naddaf, Joel Veness & Michael Bowling, "The Arcade Learning Environment: An Evaluation Platform for General Agents". CoRR, abs/1207.4708, 2012.
[8] Richard S. Sutton & Andrew G. Barto. (2018). *Reinforcement learning: An introduction* (2nd Edition). The MIT Press, Cambridge, Massachusetts, London, England

[9] Mita Chaturdevi,"What Are DQN Reinforcement Learning Models", Jun 10, 2021. [Online]. Available: https://analyticsindiamag.com/what-are-dqn-reinforcement-learning-models/. [Accessed: Nov 5-7, 2021]

[10] Nicolaas P. Cawood, Clarise Poobalan, Shikash Algu & Byron Gomes, *Trying to Solve the NetHack Unicorn: A Series of Experiments in Reinforcement Learning.* [Online]. Available: https://github.com/Pieter-Cawood/Reinforcement-Learning/blob/master/Submission/Nethack_Project.pdf . [Accessed: Nov 1-7, 2021]

[11] Deep Learning, "RL — DQN Deep Q-network, Jonathan Hui", Jul 16, 2018. [Online]. Available: https://jonathan-hui.medium.com/rl-dqn-deep-q-network-e207751f7ae4. [Accessed: Oct 29, 2021]

[12] David Silver, "Lectures on Reinforcement Learning" 2015. [Online]. Available: https://www.davidsilver.uk/teaching/. [Accessed: Oct 31 - Nov 6, 2021]

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver,

Alex Graves, Ioannis Antonoglou, Daan Wierstra & Martin Riedmiller *Playing Atari with Deep Reinforcement Learning*, DeepMind Technologies. [Online], Available: https://www.cs.toronto.edu/ vmnih/docs/dqn.pdf. [Accessed: Nov 6, 2021]

[14] Cheng Xi Tsou. [Online]. Available: https://github.com/chengxi600/RLStuff. [Accessed: Oct 29- Nov 10, 2021]

[15] A. Asperti, C. De Pieri, and G. Pedrini, “Rogueinabox: an environment for roguelike learning,” International Journal of Computers, vol. 2, 2017.

[16] A. Asperti, C. Pieri, M. Maldini, G. Pedrini, and F. Sovrano, “A modular deep-learning environment for rogue,” WSEAS Transactions on Systems and Control, vol. 12, 2017.