Reinforcement Learning: NetHack Environment

1st Sarah Wookey
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
1675256@students.wits.ac.za

2nd Rifumo Mzimba
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
1619542@students.wits.ac.za

3rd Caston Nyabadza
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
1069634@students.wits.ac.za

I. Introduction

For this project, we researched on applying different Reinforcement Learning agents to learn and play the NetHack game, namely a Deep Q-Network (DQN) and an Actor Critic (AC). NetHack is a terminal based game which is very complex with multiple dynamic levels and hard to solve. Not only is the action and state space very large, there is no effective pattern of actions that always yield better rewards [1] and state transitions dynamics are very stochastic [2]. Every time the game is restarted, the agent has to explore dungeons that are structured differently [1]. This game is also characterised by low win rates and has sparse rewards in between actions. Thus, the NetHack Learning Environment (NLE) was developed as it provides fast, simple graphic simulations, making it a good environment to apply reinforcement learning.

A. NetHack Game

The goal of this game is for the player to descend all levels of the dungeon, retrieve the amulet, and ascend back to first level [2]. The chosen hero among multiple options, has to descent 50 levels to retrieve the amulet and back. Although the structure of the dungeon is linear, the exact locations of places of interest are randomly generated hence introducing a degree of complexity to the game [2].

B. Related Work

NLE is a relatively new environment with not many published results on the environment. Two baseline models were present in [2], where the agents policy were trained using IMPALA [3] which is an off-policy actor-critic framework, and Random Network Distillation (RND) [4].

It is seen that methods performing well on Atari learning environment tasks were not able to successfully learn in this sparse-reward environment [2]. The NLE environment requires surpassing the limits of deterministic or repetitive settings.

Action space elimination is one of the few techniques to try and improve agent performances. It allows the agent to overcome some of the main difficulties in large action spaces, namely: Function Approximation and Sample Complexity [5].

C. NetHack Learning Environment

The NLE environment is a complex environment with hundreds of different possible symbols, resulting in an enormous combinatorial observation space. While the action space includes 93 different actions which include 77 command actions and 16 movement actions [2]. There are 23 default NLE actions detailed bellow, comprising of “one-step” and ”move far” compass directions, where one takes one step in a certain direction while the other keeps going in a direction until it hits an obstacle.

#	Action
0	More
1	North 1 step
2	East 1 step
3	South 1 step
4	West 1 step
5	North-East 1 step
6	Sout-East 1 step
7	South-West 1 step
8	North-West 1 step
9	North max
10	East max
11	South max
12	West max
13	North-East max
14	Sout-East max
15	South-West max
16	North-West max
17	Go up a staircase
18	Go down a starcase
19	Wait
20	Kick
21	Eat
22	Search

The observation space consists of four elements namely the glyphs, stats, message and inventory [2].

- Glyphs: which is a matrix representing the (batched) 2D symbolic observation of the level. It contains integer values ranging from 0 up to 5991 representing monsters, objects and floor types.

- Stats: which is a vector of size 23 of agent coordinates and other character attributes detailed below.
- Message: which is a tensor representing the current message shown to the player and the
- Inventory: which is a list of padded tensors representing inventory items that are requested by the player .

#	Stat
0	x Coordinate
1	y Coordinate
2	StrengthPercentage
3	Strength
4	Dexterity
5	Constitution
6	Intelligence
7	Wisdom
8	Charisma
9	Score
10	Hitpoints
11	MaxHitpoints
12	Depth
13	Gold
14	Energy
15	MaxEnergy
16	ArmorClass
17	MonsterLevel
18	ExperienceLevel
19	ExperiencePoints
20	Time
21	HungerLevel
22	CarryingCapacity.

II. A SUMMARY OF THE DQN ALGORITHM

The DQN algorithm uses a convolutional neural network to approximate the optimal action-value function Q. During the training process, the Agent, interacts with the environment by making actions and recording the resultant states, which are used during the learning of the Q-network [6]. The network is trained with a variant of the Q-learning algorithm, with stochastic gradient descent to update the weights. The actions are chosen from an epsilon greedy policy in order to decide how to act [6].

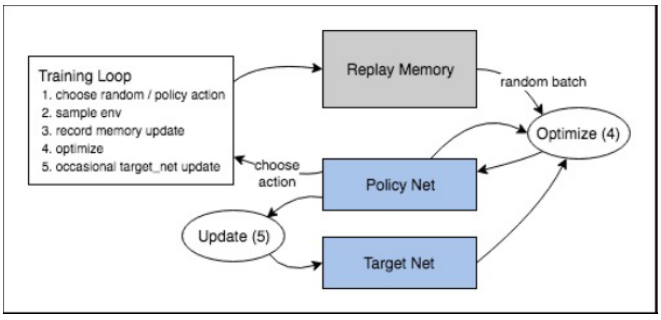


Fig. 1. Training loop overview of DQN

The training process: This diagram loops through for a given number of steps, for as many episodes as required.

1) For each step, we sample an action. Actions are chosen either randomly or based on a policy. The selected action will be based according to an epsilon greedy policy. Thus, we'll sometimes sample an action uniformly at random or use our policy for choosing the action.
2) Execute action and get the next step sample from the gym environment.
3) Record the results of the transition into the replay memory (the next screen and the reward obtained from the step sample). The replay memory acts as a buffer that holds the recent transitions observed.
4) Optimization picks a random sample batch from the replay memory to do training of the new policy. The optimization of the policy network is performed once for every set number of steps and updates the policy network. The expected Q values used in the optimization are computed from the target network.
5) The target network has its weights kept frozen most of the time but is updated with the policy network's weights occasionally to keep it current. This is usually a set number of steps or episodes.

A deep q-learning network has 3 main elements: Replay buffer, Target network and clipping rewards. The replay buffer stores past experiences with the environment including state transitions, rewards and actions, which are necessary data to perform Q learning. A sample batch from the replay buffer is used to update neural networks. This technique expects the following merits. This technique is used to reduce forgetting past transitions and avoid over fitting [6].

The target function is changed frequently with the dqn. Unstable target function makes training difficult, thus the target function parameters are replaced with the latest network every set number of steps.
Each game has different score scales, were winning could give mass points while losing could be only -1, this difference would make training unstable. Thus Clipping Rewards technique clips scores, which all positive rewards are set +1 and all negative rewards are set -1.

There are 2 networks used in the DQN algorithm. During the learning process we use two separate Q — networks(policy-network and target-network) to calculate the predicted value (weights θ) and target value (weights θ'). Both networks have the same architecture. The target network is only updated after a set number of iterations with the weights from the policy network which helps stabilize the training [6].

III. A SUMMARY OF THE ACTOR CRITIC ALGORITHM

Actor Critic Algorithms update at each step (TD Learning) instead of waiting until the end of the episode as done in Monte Carlo REINFORCE. To do this, there are two networks that are trained simultaneously.

- Actor- A policy function, controls how our agent acts.
- Critic- A value function, measures how good these actions are.

Because we have two models that must be trained, it means that we have two set of weights (θ for our action and w for our Critic) that must be optimized separately [7].

- The Policy takes the state, outputs an action (A), and receives a new state (S+1) and a reward (R+1).
- The Critic computes the value of taking that action at that state the Actor updates its policy parameters (weights) using this q value.
- Thanks to its updated parameters, the Actor produces the next action to take at A+1 given the new state S+1. The Critic then updates its value parameters.

The Advantage function captures how much better an action is compared to the others at a given state, while the value function captures how good it is to be at this state [7].

Thus the critic learns the Advantage values. That way the evaluation of an action is based not only on how good the action is, but also how much better it can be.

The reason of the advantage function is that it reduces the high variance of policy networks and stabilize the model [7].

The advantage function is defined as:

$$A(s,a) = Q(s,a) - V(s,a) \tag{1}$$

It is modified so it doesn't requires two value functions - Q(s,a) and V(s). Using the TD error as a good estimator of the advantage function.

$$A(s,a) = r + \gamma V(s',a) - V(s,a) \tag{2}$$

The diagram below illustrates the process described above.

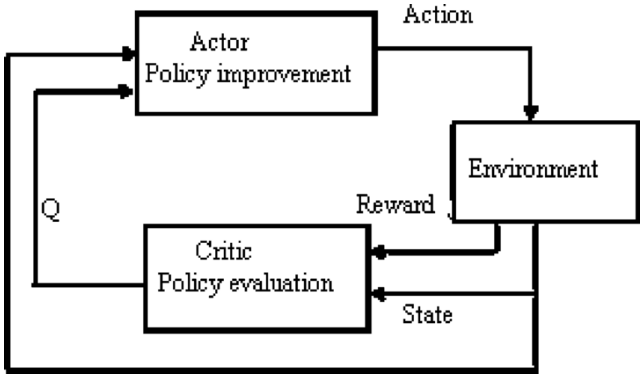


Fig. 2. Training loop overview of AC

IV. Method

The RL agent has to be implemented in a such a way that it sufficiently generalises its optimal policy that it learns through training [1] in order to cater for the highly stochastic environment of the NetHack game. It has to generalise abstract sub goals such as navigating the staircase, collecting gold, discovering unseen parts of the dungeon and finding the oracle [2]. It also has to be efficient and should balance between cost and benefit of exploration [8]. Therefore the complexity is driven less by the need to be thorough and more by the need to balance the time spent exploring a space with respect to the amount of benefit accrued e.g. area revealed and items collected [9].

Since the state-action space was very large, the simpler Q-learning algorithm is less attractive as more computational power will be required to maintain the action-value table. Instead a function approximator was preferred. Learning with function approximation also allows for experience with a smaller subset of the large state space in order to generalize over a much larger subset with reasonable accuracy [9] . It does not only modify the action-value of the current state but also of other related states [9]. The function approximator would need to infer which states are similar to other states so that one update will have effect on the related states.

For this project, the hero used was the monk but there are various different heroes that could have been used such as valkyrie, wizard and tourist.
The two algorithms that will be used are the DQN and Actor Critic algorithms to solve for the optimal policy in the NetHack Learning environment. They were chosen precisely because they address the issues of using the off-policy Q-learning i.e instability [6]. Actor-Critic comes to be more important because of its ability to converge quicker [7]. Our observation space included only the glyph identifiers and vector containing the agent's stats. We reduced the observation space from 4 items to 2 items in-order to reduce computational complexities.
Both algorithms used the same network architecture that was used in the baseline model from [2] which is described below.

As embedding dimension of the glyphs we use 32 and for the hidden dimension for the observation o_t and the output of the LSTM h_t, we use 128. For encoding the full map of glyphs as well as the 9 x 9 crop, we use a 5-layer ConvNet architecture with filter size 3 x 3, padding 1 and stride 1. The input channel of the first layer of the ConvNet is the embedding size of the glyphs (32). Subsequent layers have an input and output channel dimension of 16. We employ a gradient norm clipping of 40 and clip rewards using $r_c = tanh(r = 100)$. We use RMSProp with a learning rate of 0.0002 without momentum and with $\epsilon_{RMSProp} = 0.000001$ [2].

Network Architecture
(crop): Crop()
(embed): Embedding(5976, 32)
(extract_representation): Sequential
(0): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ELU(alpha=1.0)
(2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ELU(alpha=1.0)
(4): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(5): ELU(alpha=1.0)
(6): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(7): ELU(alpha=1.0)
(8): Conv2d(16, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(9): ELU(alpha=1.0)
(extract_crop_representation): Sequential
(0): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ELU(alpha=1.0)
(2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(3): ELU(alpha=1.0)
(4): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(5): ELU(alpha=1.0)
(6): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(7): ELU(alpha=1.0)
(8): Conv2d(16, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(9): ELU(alpha=1.0)
(embed_blstats): Sequential
(0): Linear(in_features=25, out_features=32, bias=True)
(1): ReLU()
(2): Linear(in_features=32, out_features=32, bias=True)
(3): ReLU()
(fc): Sequential
(0): Linear(in_features=13952, out_features=512, bias=True)
(1): ReLU()
(2): Linear(in_features=512, out_features=512, bias=True)
(3): ReLU()
(core): LSTM(512, 512)
(policy): Linear(in_features=512, out_features=23, bias=True)
(baseline): Linear(in_features=512, out_features=1, bias=True)

A visual representation of the Network is presented below.

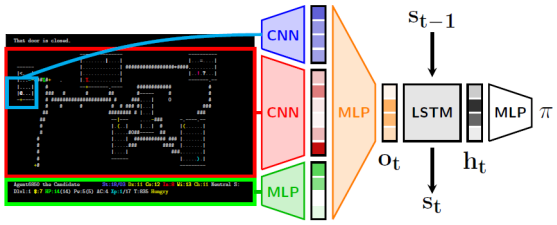


Fig. 3. Overview of the core architecture of the baseline models released with NLE

A. DQN

The DQN was implemented using the 23 default nle actions in action space with just the glyphs and stats. We reduce the sparsity in the observation space by only using the Glyphs observations, and cropping them to a fixed box 9 x 9, around the agent's position at each timestep. The hyper parameters used are described in the table bellow.

Hyper-parameters

Parameter name	Value	Description
Seed	1,2,3,4,5	The seed for the random function and environment
Replay-buffer-size	10000	Stochastic gradient descent (SGD) updates are sampled from this number of most recent frames
Learning-rate	0.00048	Learning rate for Adam optimizer
Discount-factor	0.99	Discount factor Gamma used in Q-learning update
Num-steps	10^6	Total number of steps to run the environment for
Batch-size	256	Number of transitions to optimize at the same time (number of training cases over which each SGD update is computed)
Learning-starts	10000	Number of steps before learning starts
Learning-freq	5	Number of iterations between every optimization step
Target-update-freq	1000	Number of iterations between every target network update
Eps-start	1	Epsilon-greedy start threshold. The initial value of ϵ in ϵ-greedy exploration
Eps-end	0.01	Epsilon-greedy end threshold. The final value of ϵ in ϵ-greedy exploration
Eps-fraction	0.50	Fraction of num-steps
Print-freq	10	Number of iterations before a print statement (used for progress)

B. DQN Experiment 2

The above DQN agent was modified to take in a smaller action space, removing all the max directions as they can be made up with just single steps. The wait and more were also removed, leaving an action space of size 13. All the same parameters were used in the simulations, just with a reduced action space.

C. AC

The Actor Critic was implemented using the 23 default nle actions in action space with just the glyphs and stats. We reduce the sparsity in the observation space by only using the Glyphs observations, and cropping them to a fixed box 9 x 9, around the agent's position at each timestep. The hyper parameters used are described in the table bellow.

Hyper-parameters

Parameter Name	Value	Description
seed	1,2,3,4,5	The seed for the random function and environment
Discount factor	0.99	Discount factor Gamma used in Q-learning update
Learning rate	0.00048	The learning rate to update the model parameters.
Batch size	1	The number of transitions per update
max_episodes	1000	Number of episodes to train for

V. Results and Discussion

After training the above agents, the below graphs of the average reward was obtained for each of the three agents.

Fig. 4. DQN (full action space) mean average reward for every 25 episodes , plotted every 5 episodes

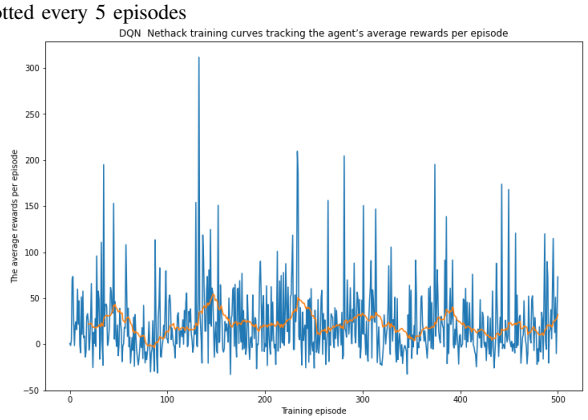


Fig. 5. DQN (reduced action space) mean average reward for every 25 episodes, plotted every 5 episodes

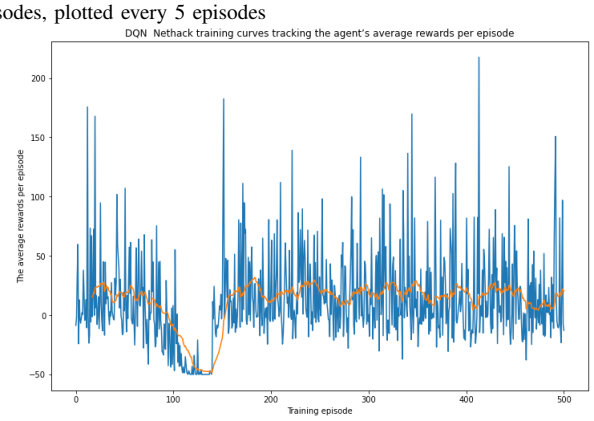


Fig. 6. Actor Critic mean average reward for every 25 episodes

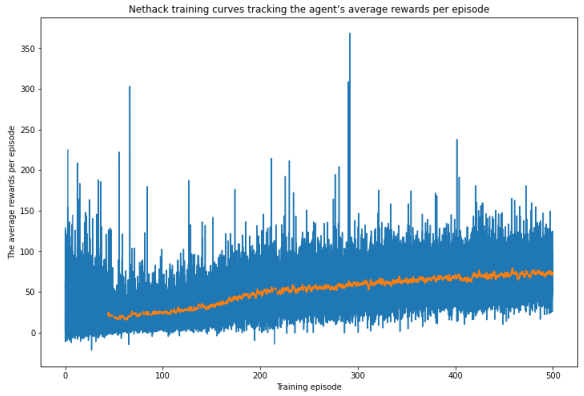
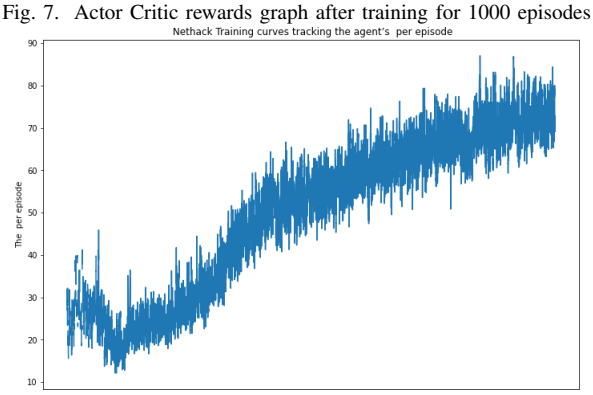


Fig. 7. Actor Critic rewards graph after training for 1000 episodes



put graphs of learning rate and loss

As you can see in the above results, the DQN agent does not do particularly well. As soon as the exploration rate decreases, so did the performance of the agent. After observing the agent, it was noted that it did not manage to learn. Thus the action space was reduced and we retired with a smaller action space. However, the results were similar. Which matches finding of DQN performances in Rogueinabox, whereby they state that DQN agents failed to learn for dynamic levels but did manage to learn what staying on the same level [10].

After a long time fiddling with the hyper parameters, this was the best model for Actor Critic. You can clearly observe a growing trend with the rewards and was not random like above.

VI. Future Works

The first thing we plan to try is to reduce the action space of the Actor Critic, similarly to what was done to the DQN. Since actor critic managed to learn, we are curious as to how much of a difference the results would be.

Next, we plan to try is playing around with the observation space by further restricting the size of the glyphs or increasing it. The current crop size is 9 x 9. Reducing the observation space will allow for better function approximation however increasing it could allow the agent to search and see goal objects further away.

We plan to try change to Sarsa from Q-learning as it will train quicker and we see sooner how our agent is performing, especially when trying to optimise the reward function [6].

We plan to further try reduce the action space to just direction to see how the agent performs as opposed to the current reduced set.

After doing some research, hierarchical DQN performs quiet well in the Rogue environment [10], thus it is a good idea to use that as a starting point for our next model.

VII. Conclusion

The aim of this study was to explore different models on this sparse reward, NetHack environment. It was shown that the deep Q-learning agent did not manage to learn the environment and only did well when sampling random actions. This study aims as a starting point for further research on this environment to be built on.

References

[1] Y. Kanagawa and T. Kaneko, "Rogue-gym: A new challenge for generalization in reinforcement learning," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[2] H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel, "The nethack learning environment," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[3] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," *arXiv preprint arXiv:1802.01561*, 2018.

[4] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[5] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor, "Learn what not to learn: Action elimination with deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 31, pp. 3562–3573, 2018.

[6] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[7] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in neural information processing systems*, 2000, pp. 1008–1014.

[8] J. Campbell and C. Verbrugge, "Exploration in nethack with secret discovery," *IEEE Transactions on Games*, 2018.

[9] J. Campbell, "Exploration and combat in nethack," 2018.

[10] A. ASPERTI and G. PEDRINI, "Rogueinabox: a rogue environment for ai learning."