Reinforcement Learning Project

Feziwe Shongwe
BSc Hons Computer Science
2135313@students.wits.ac.za

Phola Bavuma
BSc Hons Big Data Analytics
1848739@students.wits.ac.za

Suraksha Motilal
BSc Hons Computer Science
2108903@students.wits.ac.za

Derrin Naidoo
BSc Hons Computer Science
2127039@students.wits.ac.za

I. Introduction

MiniHack is a sandbox framework containing a large collection of diverse and sophisticated Reinforcement Learning environments [Samvelyan et al., 2021]. These environments become a playground for various Reinforcement Learning agents to learn and conquer. The complexity of these environments vary greatly from crossing a lava river and slaying a large beast, to simply eating an apple.

In this project, we have compiled reinforcement learning models to effectively explore and learn the MiniHack Quest Hard environment. This environment includes various sub-tasks, which need to be achieved for the agent to complete the environment. The first sub-task details a procedurally-generated maze in which the agent will need to navigate its way out of. The second sub-task includes crossing the river of lava, and the third includes having to defeat a demon guarding the exit.

The three models which we have explored in this paper are DQN, REINFORCE with baseline and TD ACTOR-CRITIC.

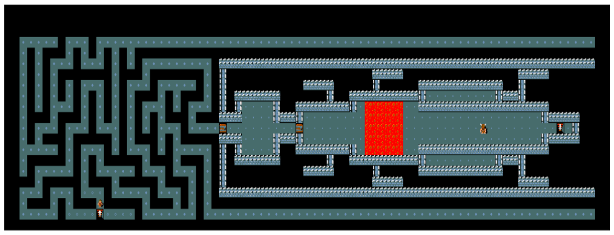

Fig. 1. Minihack environment

II. Deep Q-Network

A. Algorithm Description

The Deep Q-Network is a function approximation technique to tackle Reinforcement Learning problems. The principle idea behind the algorithm is to take in the state of the environment as input, then through the use of Neural Networks, and output the evaluated Q-values for each action. It is known as deep neural network adaption for the Q-Learning algorithm. Attempting to learn a Large environment using regular Q-Learning would be highly impractical as storage would need to be allocated for each state and action pair. The DQN model solves this issue by learning the most optimal features to match state and action pairs to produce the optimal policy for the environment. This is a highly effective approximation of the Q-Learning rather than storing the exact solutions.

The Q-Learning algorithm encompasses a Q-value table that stores the probability distribution of each state and action that will in turn maximize the expected return for the agent in the environment. Each state and action pair is updated using the Bellman Equation:

$$Q(S, A) \longleftarrow +\alpha[R + \gamma max_a Q(S', A) - Q(S, A)] \qquad (1)$$

The DQN algorithm follows a similar path to the Q-learning algorithm. The main adaption is that the Q-value table is replaced with a function approximator. Now with every update, the model updates multiple state and action values instead of one as seen in the Q-learning model.

The loss function for the DQN is described as:

$$L(\Theta) = \mathbb{E}_{s,a,r,s'}[(y_i - Q(s_i, a_i, \Theta))^2] \qquad (2)$$

where the target is depicted as:

$$y_i = r_i + max_{ai+1}\gamma Q(s_{i+1}, a_{i+1}, \Theta') \qquad (3)$$

To avoid the issue of constantly changing targets, the algorithm incorporates a second neural network to estimate the current target. This network is updated after n iterations and adopts the weights of the current neural network.

The weights of the DQN are optimized through the use of stochastic gradient descent. Various problems come about when incorporating this technique in RL. Stochastic gradient descent works best with samples that are independent and identically distributed but in RL, sampling experiences from the environment would lead to sequential samples. This issue would cause the network to see multiple samples of one kind and forget the others. For example: Imagine an agent which learned the optimal policy to complete level one and is exposed to level two for the first time. The second level looks and behaves differently from the first so the agent inevitably forgets how to play the first level and learns the second. This is why an Experience Replay buffer is used. Samples stored within the buffer are randomly sampled for learning. This procedure ensures the data is independent and identically distributed and helps break the temporal correlation of training samples.

B. Design Decisions

The observation space selected for this environment is the Gylphs observation space which returns a 21X79 depiction of the current agent state. The states are flattened and reshaped to the size accepted by our Network. For the Choice of Architecture of our model, we implemented two Convolutional layers that adopt the Relu activation functions followed by two fully connected layers. We have also incorporated max pooling within our Convolutional layers. The output of our Network is of the size of the available action space The reason for this implementation is for its simplicity and effectiveness when learning the environment. Also, this model's learning has outperformed our previous implementations. Here is a short summary of the architecture:

- Conv: Conv2d(1,20,Kernal-size=(5x5))
- Relu
- Maxpool: Maxpool2d(Kernal-size=(2x2),stride = (2x2))
- Conv: Conv2d(20,50,Kernal-size=(5x5))
- Relu
- Maxpool: Maxpool2d(Kernal-size = (2x2),stride=(2x2))
- Linear: Linear(1600,500)
- Relu
- Linear: (500,Action-space)

C. Hyperparameters

- Learning rate: 0.01
- Episodes to play for reward evaluation: 10
- Max episodes for training: 1000
- Minimum batch: 500
- Max Buffer storage: 1000000
- Discount factor: 0.8
- Collection epsilon:0.2
- Update target = 20

D. Results

After 200 iterations the agent seems to present some progress with the environment and has displayed an increase in average rewards.

III. REINFORCE

A. Algorithm Description

REINFORCE is a policy gradient-based method, also known as the Monte Carlo Policy Gradient algorithm. A basic implementation of this method would be to create a Policy that takes a state as input and outputs the probability of performing an action. [Yoon, 2018] describes policy gradient approaches as common in model-free reinforcement learning algorithms, and they appear frequently in recent studies.

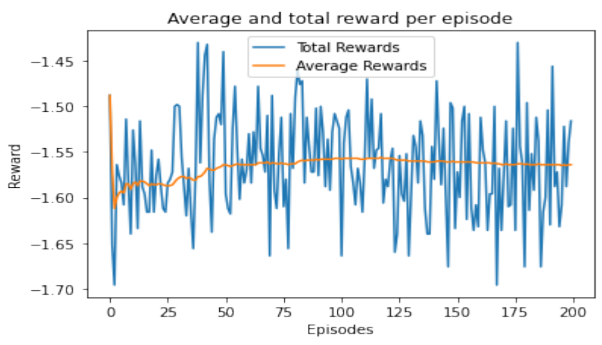

Fig. 2. QDN Plot

Since the policy gradient method is also an "actor" in Actor-Critic methods, understanding it is essential for researching reinforcement learning [Yoon, 2018]. In the following section, Actor-Citric is implemented. The REINFORCE Monte Carlo update rule equation 4 is presented below as well as the REINFORCE with baselines equation 5:

$$\theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \tag{4}$$

$$\theta_{t+1} = \theta_t + \alpha(G_t - bS(t) \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \tag{5}$$

We used baselines in this study because the Monte Carlo method has high variance and is slow when learning (slow convergence). [Sutton and Barto, 2018] compares the two methods when using REINFORCE with baselines and the Monte Carlo-based one based on an environment discussed. We picked the REINFORCE method since it is simple and does not need us to model the environment, which is difficult in the Quest-Hard setting. Due to the obvious stochastic policy representation, policy gradient algorithms in general are known to naturally explore.

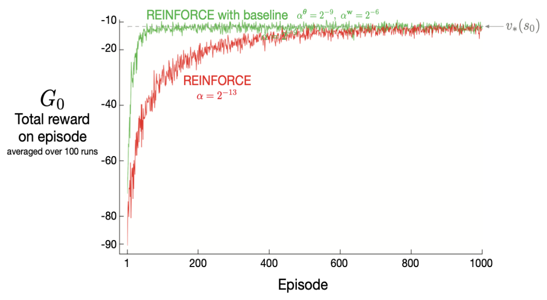

Fig. 3. Comparison of the REINFORCE with baselines and the one based on Monte Carlo for the gridworld environment discussed in [Sutton and Barto, 2018]

B. Design Decisions

We created a Neural Network to act as a policy network. The policy network will accept a state vector as input and provide a (discrete) probability distribution across all feasible actions. The model consists of only two linear layers, with the first layer including a ReLU activation function and the latter layer containing an Adam optimizer.

C. *Hyperparameters*

- seed = 45 $\rightarrow$ which seed to use
- Environment name: MiniHack-Quest-Hard-v $\rightarrow$ name of the game
- Replay buffer size = $5e3$ $\rightarrow$ replay buffer size
- learning rate= 0.0001 $\rightarrow$ learning rate for Adam optimizer
- Discount factor(γ)= 1 $\rightarrow$ discount factor (long sighted)
- Number of maximum steps per episode = 5000 $\rightarrow$ total number of steps to run the environment for if it does not converge
- Batch size= 256 $\rightarrow$ number of transitions to optimize at the same time
- Steps for learning to start= 100 $\rightarrow$ number of steps before learning starts
- learning frequency= 2 $\rightarrow$ number of iterations between every optimization step
- target-update-freq= 1000 $\rightarrow$ number of iterations between every target network update
- Epsilon start(maximum)= 1.0$\rightarrow$ e-greedy start threshold
- Epsilon end (minimum)= 0.1 $\rightarrow$ e-greedy end threshold
- Epsilon fraction= 0.2 $\rightarrow$ fraction of num-steps
- Print frequency= 25$\rightarrow$ number of iterations between each print out
- Save frequency= 500 $\rightarrow$ number of iterations between each model save

D. *Results*

After 500 episodes the model was still exploring in the environment which indicates that it required more episodes to have positive positive rewards.

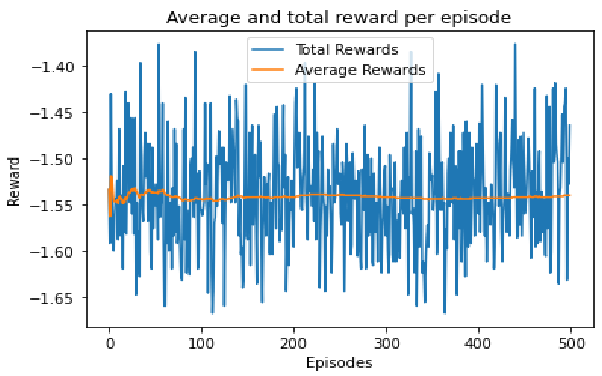

Fig. 4. Average rewards for 500 Episodes for REINFORCE with baselined

IV. Temporal Difference (TD) Actor-Critic

A. *Algorithm Description*

The TD Actor-Critic model is a policy search-based method in which the model aims to learn the optimal policy. This is achieved by integrating two components into the model, namely the actor and the critic. The actor component refers to the policy network where the agent's actions are chosen based on the current policy (on-policy approach). More specifically, the policy returns a probability distribution over the action space where a subsequent action can be chosen by sampling the distribution. The critic component is the value function network, which evaluates the expected return of the corresponding states given actions that the agent took. By updating the weights in the policy network according to some loss, a better policy is estimated. In this model, we compute the TD-Error given by

$$\delta = R_{t+1} + V(S_{t+1}) - V(S_t) \tag{6}$$

to proportionally update the weights for the combined network.

In order to implement the combined network, we utilise an *ActorCriticNetwork* class which describes a feed-forward neural network of various layers to output a value and policy. A number of convolutional layers with consecutive max pooling layers are applied to the "glpyh" observation where it then passes through two fully connected layers. Thereafter, the "message" observation passes through a linear function and gets concatenated with the previous "glyph" output. The concatenated output then passes through one more linear function and ReLU activation before finally passing through the relevent layers to produce a value for the state and the policy. The layers of the neural network contain the following architecture:

1) 2D Convolutional Layer with 20 feature maps, kernal size (5x5) and ReLU activation
   - Applies convolutions to the "glyph" observation
2) 2D Max Pooling Layer with pool size (2x2) and stride (2x2)
   - Downsamples convolutional layer
3) 2D Convolutional Layer with 50 feature maps, kernal size (5x5) and ReLU activation
   - Applies convolutions to the "glyph" observation
4) 2D Max Pooling Layer with pool size (2x2) and stride (2x2)
   - Downsamples convolutional layer
5) Linear Layer with 256 input size and subsequent ReLU activation layer
   - Applies linear function to the "message" observation
6) Linear Layer with 513 input size and subsequent ReLU activation layer
   - Applies linear function to the concatenated observation
7) Dense Layer with 1x1 output size and no activation
   - Returns the value of the state.
8) Dense Layer with action space output size and softmax activation
   - Returns probabilities for every action in the action space.

B. Design Decisions

Importantly, the Actor-Critic model is brittle by nature which means that the performance of the model is highly subject to changes in hyperparameters and inputs. It is highly advised that the learning rate of the model be tweaked in future runs for improved training performance. Furthermore, the nature of actor-critic requires many episodes for the agent to undergo effective training. If the resources are available, it is recommend to run the algorithm for a greater number of episodes.

Combining the policy and value function into a single network architecture did show improvements in stability as training did not see any unfavourable spikes indicating that the agent is no longer learning correctly.

- Combined policy network and value function network into a single network architecture. This contributes to the stability of the network during training.
- Used the ”glyphs” and ”message” observations to train the model.
- Used the ”pixels” observations to generate a video for the best, learnt model.

C. Hyperparameters

- Learning Rate: 5e-6
- Discount Factor: 0.99
- Number of Episodes: 500

D. Results

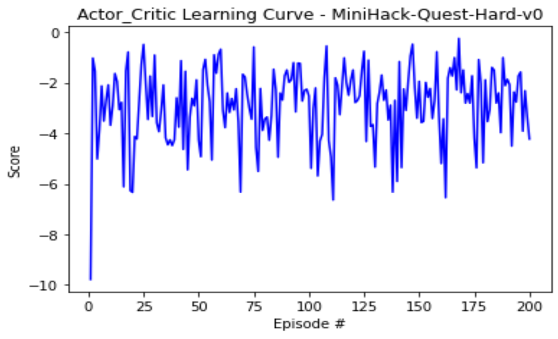

Fig. 5. TD Actor Critic - Learning Curve

While no clear learning trend is observed in the raw learning plot, a subsequent best-fit plot will indicate that learning is taking place.

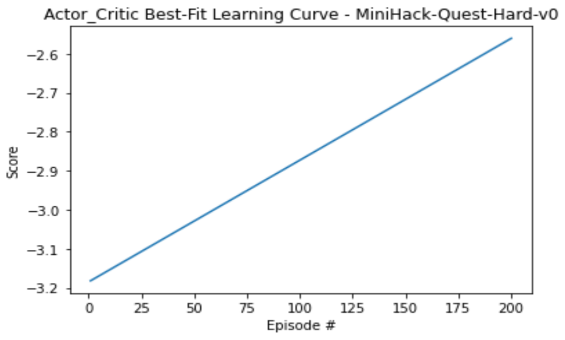

Fig. 6. TD Actor Critic - Best-Fit Learning Curve

It is observed that there is a clear learning trend in the algorithm’s execution. Tuning the hyperparameters may produce more favourable training performance.

V. Code and Results

The code for this project can be found here.

VI. Conclusion

After assessment of the performance of the DQN, REINFORCE with Baseline and Actor-Critic model, the best performing of the set is considered to be the Actor-Critic model. The Actor-Critic Model has been capable of effectively slowly learning the environment including displaying promising results. Constantly improving its position.

References

[Samvelyan et al., 2021] Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Küttler, H., Grefenstette, E., and Rocktäschel, T. (2021). Minihack the planet: A sandbox for open-ended reinforcement learning research. *ArXiv*, abs/2109.13202.

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

[Yoon, 2018] Yoon, C. (2018). Deriving policy gradients and implementing reinforce.

Declaration

We hereby declare the following:

- We as Team Members, Contributed Equally in this Project.
- We are aware that plagiarism (the use of someone else's work without their permission and/or without acknowledging the original source) is wrong.
- We are confirm that the work submitted for assessment is our own unaided work except where we have explicitly indicated otherwise.
- We have followed the required conventions in referencing the thoughts and ideas of others.
- we understand that the University of the Witwatersrand, Johannesburg may take disciplinary action against me if there is a belief that this is not my own unaided work or that I have failed to acknowledge the source of the ideas or words in my writing.