Vrije Universiteit Amsterdam
Faculty of Science
Deep Learning

Assignment 5C
20 December 2020

Reinforcement Learning:
An in-depth analysis regarding the CartPole-v1 environment.

Names:
Richie Lee *(kle680)*
H.K. Shi *(hsi560)*
Qamar Suleri *(qsi400)*

Instructor:
Emile van Krieken

Abstract

This report compares different algorithms for Reinforcement Learning (RL) with respect to OpenAI's CartPole-v1 environment. The performances of simple/standard REINFORCE, Advantage Actor-Critic with shared parameters (A2C) and various Deep Q-Learning (DQN) algorithms are compared. The networks aim to optimize the moving average episode length. Using A2C with ADAM optimization with a learning rate of 0.001 and a discount rate of 0.98, we find a moving average maximum episode length of 489 out of 500 after 1000 episodes. The implementations and findings of the algorithms are discussed.

Keywords: A2C - DQN - CartPole-v1

1 Introduction

While a number of algorithms for reinforcement learning (RL) have been proposed and a small number of applications developed, there has been very little rigorous empirical evaluation of performance and limitations of these algorithms (Vamplew et al., 2011). This paper aims to compare the performance of simple/standard REINFORCE, Advantage Actor-Critic (A2C) with shared parameters and three Deep Q-Learning (DQN) algorithms; one standard implementation (DQN1), one with experience replay (DQN2) and one with experience replay and a target network (DQN3). Therefore, we state the following central research question:

R: *What algorithm achieves the best average reward after 1000 episodes in the CartPole-v1 environment?*

The simple REINFORCE algorithm does not do any credit assignment: whenever a high total reward is received, the log-probability of all actions performed in that trajectory are increased evenly. As the standard REINFORCE algorithm looks at the expected onward reward instead of the total expected reward, we expect the standard REINFORCE to perform better. Deep Q-learning implements a deterministic policy as it uses a neural network to approximate the Q-value function. However, this might lead to divergence when using the same network to calculate the predicted value and the target value. As a solution a target network can be used to separately estimate the target. In addition, experience replay can be used to lower the correlation amongst the samples and increase sampling efficiency. Lastly, the Advantage Actor-Critic algorithm models actions through policy networks, while rewards are modelled through the critic, which is obtained by estimating the value function. This introduces a slight bias, but this is traded off by a large decrease in variance. Hence, we present the following hypothesis:

H: *The Advantage Actor-Critic with shared parameters achieves the best performance.*

When applying this model to OpenAI's CartPole-v1 environment we find that the Advantage Actor-Critic with shared parameters indeed achieves the best performance (a moving average over 50 episodes with a length of 489 episodes). This is closely followed by the standard REINFORCE algorithm. However the standard REINFORCE algorithm shows to be a lot more volatile.

In Section 2, we describe the environment we use to apply our algorithms to. Subsequently, we describe the algorithms and the methods used to analyze their performance in Section 3. In Section 4, We present and discuss our results. Lastly, Section 5 contains the conclusion.

2 Environment

One of the defining properties of RL models is their ability to learn tasks without being fed data. Instead, these models develop by taking *actions* in environments while seeking to complete objectives. Without prior knowledge, these algorithms will continuously polish a strategy (also known as *policies*) to maximize *rewards* through successful performance.

We will apply this model to the CartPole-v1 environment (Brockman et al. (2016)). This is a game were we attach a pole to a cart. This cart moves in a single dimension along a track and is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright.

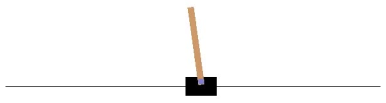


Figure 1: The CartPole-v1 environment

3 Methodology

Before delving into the RL algorithms, we explored its mathematical motivations. Afterwards, we were ready to commence its implementation in Python (PyTorch). In this report, we first examine a simple RL algorithm, which we then extend with Advantage Actor-Critic and Deep-Q-learning attributes (of which we provide pseudocodes in Appendix C).

3.1 Model Introduction

The models that we use are:

1. **Simple/Standard REINFORCE:** *A basic reinforcement algorithm (with look-ahead). This could be considered a "vanilla" reinforcement algorithm.*

   The policy network has the following architecture:

   $$Input \rightarrow Linear(4, 128) \rightarrow ReLU \rightarrow Linear(128, 128) \rightarrow ReLU \rightarrow Linear(128, 2) \rightarrow Softmax \rightarrow Output$$

   The proof that REINFORCE with correct credit assignment is an unbiased estimate of the policy gradient is given in Appendix A.

2. **Advantage Actor-Critic (A2C):** *These models utilise both value and policy-based modelling by assigning an actor and critic. The actor selects the actions, which will then be evaluated by its critic alongside adjustment instructions.*

   The policy network has the following architecture:

   $$Input \rightarrow Linear(4, 128) \rightarrow ReLU \rightarrow \begin{cases} \rightarrow Linear(128, 2) \rightarrow Softmax \rightarrow Output & \textbf{Actor} \\ \rightarrow Linear(128, 1) & \textbf{Critic} \end{cases} \tag{1}$$

3. **Deep Q-Learning (DQN):** *These models estimate Q-values for action-state pairs in a given environment, essentially combining Q-learning and deep neural networks.*

   We apply the epsilon greedy algorithm, which introduces a random probability of sampling a random action. This ensures exploitation of well-performing trajectories, while not neglecting exploration of new ones. During training this epsilon decreases linearly from 10% to 1%, dropping 1% per 100 episodes. In addition, we will also experiment with experience replay. This extension stores the agent's experience and samples from its memory, which significantly reduces correlation among sequential samples. And finally, we also implement a target network, which is trained alongside the main network improving training stability.

   The Q networks have the following architecture:

   $$Input \rightarrow Linear(4, 128) \rightarrow ReLU \rightarrow Linear(128, 128) \rightarrow ReLU \rightarrow Linear(128, 2) \rightarrow Output \tag{2}$$

3.2 Performance comparison

We evaluate the quality of learning for the Cart-Pole problem per algorithm based on the *episode length*, which is the number of time-steps from the start of the episode ($t = 0$) until the pendulum on the Cart-Pole has fallen over or moves more than 2.4 units from the center. As we use the CartPole-v1 version, the maximum step length is set to 500.

The parameters are updated by using ADAM optimization (where the hyperparameters $\beta_1 = 0.9$ and $\beta_2 = 0.99$ are left to their default value) with discount rate of $\gamma = 0.98$. The performance of the networks based on learning rates $\alpha = 0.0001, 0.0003, 0.001$ and 0.003 are compared. For the REINFORCE and A2C models, the discounted rewards are normalized per episode. Then all models are trained over 1000 episodes. As RL algorithms tend to be very volatile, we train each network for 5 runs and take the mean of the moving average over 50 episodes per run. First, we obtain the learning rate for which each algorithm performs best. Then, we plot these tuned networks to further compare their performances and volatilities.

4 Results

Table 1 shows the maximum performance achieved per algorithm per learning rate. We observe that A2C gives the best performance (489.0) when using learning rate $\alpha = 0.003$, which is in line with our hypothesis. This performance is closely followed by the standard REINFORCE algorithm (481.6) when using learning rate $\alpha = 0.001$. For deep Q-learning we see that both experience replay (DQN2) and target networks (DQN3) contribute to higher scores, as expected. However for experience replay the scores seem unstable except with a low learning rate of $\alpha = 0.0001$ (Fig. 2).

Table 1: Max performance for all algorithms.

Algorithm	$\alpha = 0.0001$	$\alpha = 0.0003$	$\alpha = 0.001$	$\alpha = 0.003$
`Simple REINFORCE`	27.8	**30.4**	28.5	25.5
`Standard REINFORCE`	195.7	418.3	**481.6**	329.2
`A2C`	41.4	190.9	453.5	**489.0**
`DQN1`	284.2	**285.9**	267.5	272.1
`DQN2`	**323.2**	257.2	260.8	201.8
`DQN3`	228.2	365.7	**371.6**	293.8

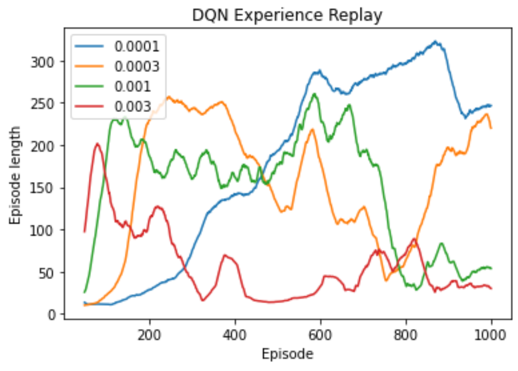
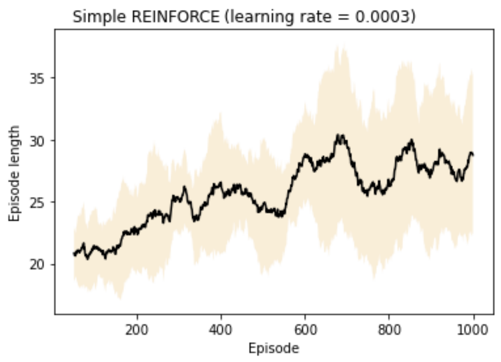
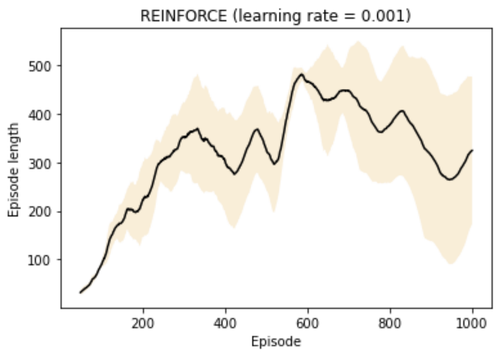


Figure 2: Instability with high learning rates for DQN with experience replay.

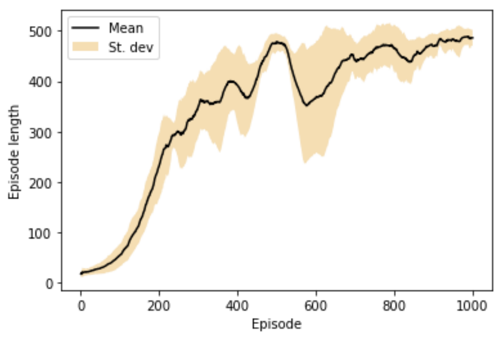
Fig. 3 shows the networks trained on their optimal learning rate. When comparing the two best implementations A2C and standard REINFORCE, we observe that standard REINFORCE is clearly more volatile. After 1000 episodes standard REINFORCE has a standard deviation greater than 100 episodes, whereas A2C shows very minimal volatility. Furthermore, we observe how simple Reinforce clearly is the worst implementation in terms of both episode length and standard deviation. Lastly, we observe that the standard DQN and DQN with experience replay and target network both achieve relatively low volatility.



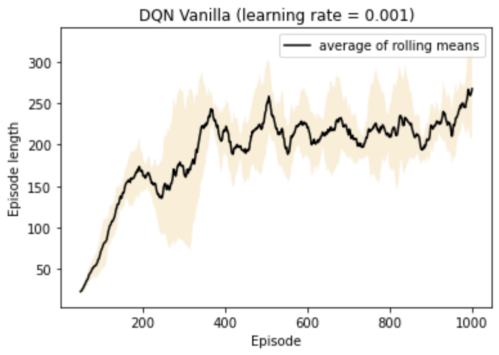
(a) Simple REINFORCE, with learning rate $\alpha = 0.0003$.



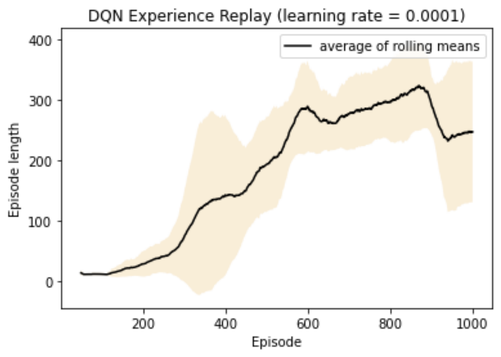
(b) Standard REINFORCE, with learning rate $\alpha = 0.001$.



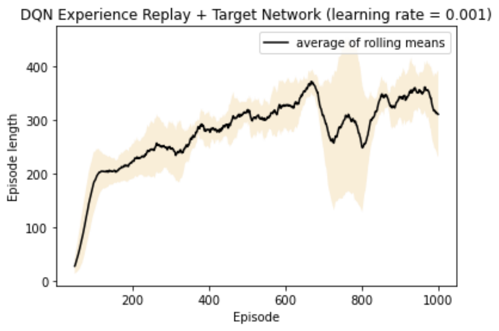
(c) A2C, with learning rate $\alpha = 0.001$.



(d) DQN, with learning rate $\alpha = 0.0003$.



(e) DQN with Experience Replay, with learning rate $\alpha = 0.0001$.



(f) DQN with experience replay and Target Network, with learning rate $\alpha = 0.001$.

Figure 3: The performance per algorithm based on the optimal learning rate.

5 Conclusion and Discussion

We have presented and elaborated six different RL algorithms to answer the research question:

R: *What algorithm achieves the best average reward after 1000 episodes in the CartPole-v1 environment?*

We have found that Advantage Actor Critic with shared parameters achieves the best performance (489.0) when using learning rate $\alpha = 0.003$. Hence, we can not reject our hypothesis. However this performance is closely followed by the standard REINFORCE algorithm (481.6) when using learning rate $\alpha = 0.001$. As the standard REINFORCE algorithm is simpler and takes a lot less time to compute, this algorithm wins in terms of efficiency. However when comparing the algorithms in terms of volatility we find that standard REINFORCE is a lot more volatile after 1000 episodes. Hence, we still favour Advantage Actor Critic with shared parameters.

One thing to note is that with tuning the learning rate we took the one with the highest maximum of the average of rolling means for 5 runs, without regards to the volatility. This may not be the best choice because the volatility itself may result in a higher maximum score by chance, even with averaging.

Secondly, a one-to-one comparison cannot be made between REINFORCE, DQN and Advantage Actor Critic, because while the amount of episodes is the same, the amount of times the parameters gets updated do not. In REINFORCE and Advantage Actor Critic the parameters are updated after each episode, while with DQN this is done after each step because it is an online algorithm.

Lastly, in this report we have found that Advantage Actor Critic with shared parameters are preferred in the CartPole-v1 environment. However, this might not be the case for different environments. In future research, these RL algorithms can be implemented on different environments to explore how this affects their expected performances. By evaluating various environments, we may be able to draw more generalised conclusions about the model.

References

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., and Dekker, E. (2011). Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning*, 84(1-2):51–80.

Yoon, C. (2019). Understanding actor critic methods.

Appendix A

Proof 1: $\nabla_\theta J(\theta) = \sum_{t'=0}^{T-1} \nabla_\theta \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1}]$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{p(\tau|\theta)}[R_\gamma] = \nabla_\theta \mathbb{E}_{p(\tau|\theta)}[r_1 + \gamma r_2 + \gamma^2 r_3 + ... + \gamma^{T-1} r_T] = \nabla_\theta \mathbb{E}_{p(\tau|\theta)} \left[\sum_{t'=0}^{T-1} \gamma^{t'} r_{t'+1}\right] \\
&= \sum_{t'=0}^{T-1} \nabla_\theta \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1}]
\end{aligned}$$

Proof 2: $\nabla_\theta \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1}] = \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1}] &= \nabla_\theta \sum_t p(\tau|\theta)\gamma^{t'} r_{t'+1} = \sum_t \gamma^{t'} r_{t'+1} \nabla_\theta p(\tau|\theta) = \sum_t \gamma^{t'} r_{t'+1} \nabla_\theta p(\tau|\theta) \frac{p(\tau|\theta)}{p(\tau|\theta)} \\
&= \sum_t \gamma^{t'} r_{t'+1} p(\tau|\theta) \frac{\nabla_\theta p(\tau|\theta)}{p(\tau|\theta)} = \sum_t \gamma^{t'} r_{t'+1} p(\tau|\theta) \nabla_\theta \log p(\tau|\theta) \\
&= \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1} \nabla_\theta \log p(\tau|\theta)] = \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]
\end{aligned}$$

Proof 3: $\mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \neq \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{t'-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$

$$\begin{aligned}
\mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] &= \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{t'-1} \nabla_\theta \log \pi_\theta(a_t|s_t) + \gamma^{t'} r_{t'+1} \sum_{t=t'}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\
&= \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{t'-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\
&\qquad + \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=t'}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\
&= \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{t'-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\
&\qquad + \sum_t p(\tau|\theta)\gamma^{t'} r_{t'+1} \sum_{t=t'}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \\
&= \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{t'-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \\
&\qquad + \sum_t p(\tau|\theta)\gamma^{t'} r_{t'+1} \sum_{t=t'}^{T-1} \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)} \\
&\neq \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{t'-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]
\end{aligned}$$

Proof 4: $\sum_{t'=0}^{T-1} \gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'+1}$
Base Case ($T = 1$)

$$\sum_{t'=0}^{T-1} \gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) = r_1 \nabla_\theta \log \pi_\theta(a_0|s_0) = \nabla_\theta \log \pi_\theta(a_0|s_0) r_1 = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'+1}$$

Inductive Step ($T \longrightarrow T+1$)

$$\begin{aligned}
\sum_{t'=0}^{T} \gamma^{t'} r_{t'+1} \sum_{t=t'}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) &= \sum_{t'=0}^{T-1} \gamma^{t'} r_{t'+1} \sum_{t=t'}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) + \gamma^T r_{T+1} \sum_{t=t'}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \\
&\overset{\textbf{Induction}}{=} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'+1} + \gamma^T r_{T+1} \sum_{t=t'}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \\
&= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma^{t'} r_{t'+1} \quad \forall \quad T \in \mathbb{Z}
\end{aligned}$$

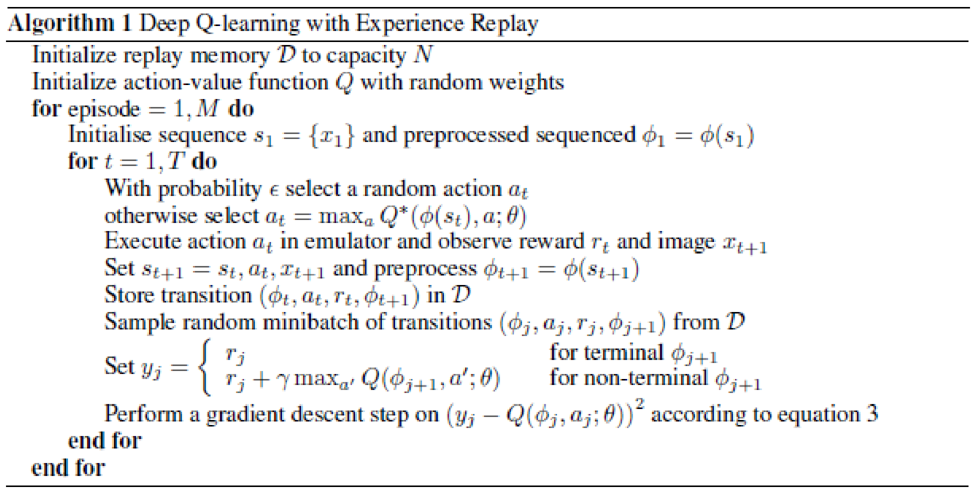
Proof 5: $\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{p(\tau|\theta)}[R_\gamma] \approx \sum_{t=0}^{T} \gamma^t G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$

$$\begin{aligned}
\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{p(\tau|\theta)}[R_\gamma] &= \sum_{t'=0}^{T-1} \nabla_\theta \mathbb{E}_{p(\tau|\theta)}[\gamma^{t'} r_{t'+1}] = \sum_{t'=0}^{T-1} \mathbb{E}_{p(\tau|\theta)} \left[\gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \\
&= \mathbb{E}_{p(\tau|\theta)} \left[\sum_{t'=0}^{T-1} \gamma^{t'} r_{t'+1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right] = \mathbb{E}_{p(\tau|\theta)} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma^{t'} r_{t'+1} \right] \\
&= \mathbb{E}_{p(\tau|\theta)} \left[\sum_{t=0}^{T} \gamma^t G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \overset{\textbf{MC}}{\approx} \sum_{t=0}^{T} \gamma^t G_t \nabla_\theta \log \pi_\theta(a_t|s_t)
\end{aligned}$$

Appendix B

This part of the appendix contains pseudocodes of the code structure for our Deep-Q and Advantage Actor-Critic implementations. These originate from Mnih et al. (2015) and Yoon (2019) respectively.

Figure 4: Deep Q-learning with Experience Replay (DQN)



```
Algorithm 1 Deep Q-learning with Experience Replay
  Initialize replay memory $\mathcal{D}$ to capacity $N$
  Initialize action-value function $Q$ with random weights
  for episode $= 1, M$ do
      Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
      for $t = 1, T$ do
          With probability $\epsilon$ select a random action $a_t$
          otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
          Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
          Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
          Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
          Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
          Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
          Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
      end for
  end for
```

Figure 5: Advantage Actor-Critic (A2C)

```
Algorithm 1 Q Actor Critic
  Initialize parameters $s, \theta, w$ and learning rates $\alpha_\theta, \alpha_w$; sample $a \sim \pi_\theta(a|s)$.
  for $t = 1 \dots T$: do
      Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
      Then sample the next action $a' \sim \pi_\theta(a'|s')$
      Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$; Compute
      the correction (TD error) for action-value at time t:
          $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
      and use it to update the parameters of Q function:
          $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
      Move to a $\leftarrow a'$ and s $\leftarrow s'$
  end for
```

Appendix C

This part of the appendix contains figures displaying more stable deep-Q training for lower learning rates.

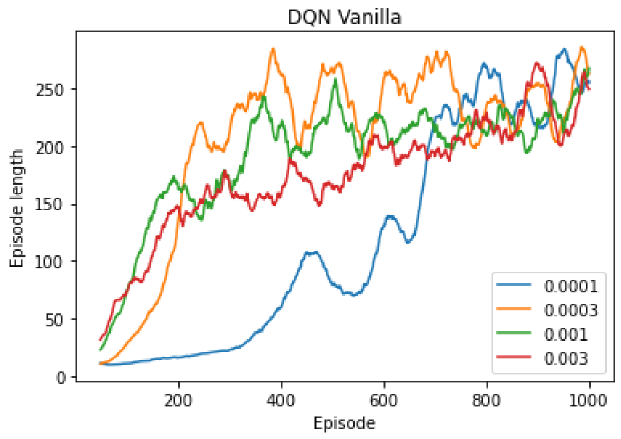


Figure 6: Deep Q Learning (With Experience Replay Target Network)

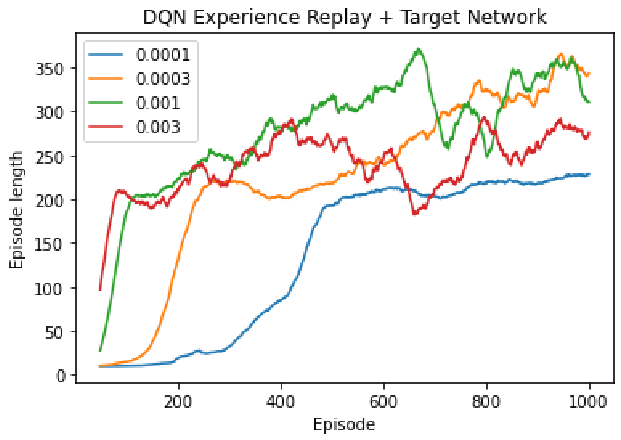


Figure 7: Deep Q Learning (With Experience Replay Target Network)