Learning To Play Atari 2600 Bowling Through Deep Reinforcement Learning

Logan Bell 6047211, Aws Al Jumaily 6459861

Abstract

REINFORCE, Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) are three fundamental deep reinforcement algorithms. In this paper, we compare and contrast the performances of these algorithms on the Atari 2600 game Bowling. Testing these algorithms gives us a chance to compare the performance of a policy-based algorithm with value-based algorithms, as well as the chance to explore the improvements that DDQN has over DQN. We observed that DQN had the best performance but the least stability, while REINFORCE had the worst performance but the most stability. DDQN was more stable than DQN, however, it was unexpectedly outperformed by DQN, which is known for overestimating action values. We present possible explanations for this, namely that not enough training sessions were used for training, which is the main limitation of our experiment.

1. Introduction

Bowling is a video game that was created for the Atari 2600 in 1979. In this game, the player's goal is to throw a ball to knock over as many pins as possible in order to achieve the highest score that they can. The score is directly correlated to the number of pins knocked down, with additional points allocated if all pins are knocked over within two throws. Pins are reset after either two consecutive throws or all pins are knocked over in one turn. This cycle occurs 10 times in total before the game is over. On the 10^{th} turn, if all pins are knocked over in one turn, the player gets up to three extra throws including his original attempt. The player has the following six controls: moving the character up or down, throwing the ball, curving the ball up or down after the release and no-operation. A curve can only be applied once per throw anytime after the release of the ball.

Reinforcement learning (RL) is a sub-field of machine learning that is based on the trail-and-error process. In RL, an agent interacts with the environment by starting off at some state, taking an action, and in turn, receiving a reward and a new state. A policy defines the agent's behaviour by indicating the actions to pursue at any given state. A reward signal defines the immediate value of taking an action. The action value function defines the long-term value of taking an action and the state value function defines the long-term value of being in a specific state. The goal of an RL agent is to learn an optimal policy which maximizes the cumulative rewards over time (Sutton & Barto, 2018, pp. 1-13). In our experiment, Atari 2600 Bowling is the agent's environment. It is an episodic environment which consists of one full game per episode. The maximum number of time steps that an episode can consist of is 10,000.

The agent selects actions from the previously stated action space and is rewarded based on how many pins are knocked down. The state space of this environment is a three dimensional array of size (210, 160, 3), which is the height and width of the screen in pixels, as well as the RGB colour value of each pixel. To introduce stochasticity, the number of frames that a time step can last for is between two to four frames at random. In this paper, we explore how the REINFORCE, Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) algorithms perform in the Atari 2600 Bowling environment.

2. Methods

2.1. REINFORCE

REINFORCE is a policy gradient algorithm that learns the parameters of a policy and directly uses the policy to select actions in a given state (Sutton & Barto, 2018, p. 321). The neural network used in this algorithm is a policy network which outputs a discrete action probability distribution based on the inputted state. The goal of this algorithm is to positively reinforce actions that yield the highest rewards for a given state. An intuitive consequence of having a discrete action probability distribution is that increasing the probability of taking desirable actions will automatically decrease the probability of taking undesirable actions. When the agent encounters the same state or a similar state in the future, it will be able to refer to its previous knowledge of the state and perform a favourable action (Zai & Brown, 2020, pp. 90-108).

REINFORCE uses Monte Carlo (MC) sampling to sample trajectories. This requires the completion of a full episode before using the experiences gained from the episode for

Algorithm 1 REINFORCE With Baseline

Input: learning rate α;$MAX\_EPISODE$;
Output: policy π_θ;
initialize weights θ of a policy network π_θ;
for $episode = 1, 2, ..., MAX\_EPISODE$: **do**
  sample a trajectory $\tau = s_0, a_0, r_0, ..., s_T, r_T, r_T$;
  Set $\nabla_\theta J(\pi_\theta) = 0$;
  for $t = 0, ..., T$ **do**
    $R_T(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r'_t$;
    $\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T} (R_t(\tau) - b(s_t)) \nabla_\theta log \pi_\theta(a_t|s_t)$
    where $b = \frac{1}{T} \sum_{t=0}^{T} R_t(\tau)$;
  end for
  $\theta = \theta + \alpha \nabla_\theta J(\pi_\theta)$;
end for

Algorithm 2 DQN with target network

Input: number of batches B; batch size N; updates per batch U; discount factor γ; target network update frequency F; learning rate α;MAX_STEPS;
Outputs: optimized parameters θ;
for $m = 1 : MAX\_STEPS$ **do**
  gather and store h experiences (x_i, a_i, r_i, x'_i) using ϵ-greedy;
  for $b = 1 : B$ **do**
    sample batch b of experiences from the ERM;
    for $u = 1 : U$ **do**
      for $n = 1 : N$ **do**
        $Y_n = r_n + \delta_{x'_n} \gamma max_{a'} Q^\pi \psi(X'_n, a')$
        $(\delta_{x'_n} = 0 \; if \; x'_n \; is \; terminal, 1 \; otherwise)$;
      end for
      $L(\theta) = \frac{1}{N} \sum_{n=1}^{N} \begin{cases} 0.5(\mathrm{x}_n - y_n)^2 & if |\mathrm{x}_n - y_n| < 1 \\ |\mathrm{x}_n - y_n| - 0.5 & otherwise \end{cases}$
      $\theta = \theta - \alpha \nabla_\theta \mathrm{L}(\theta)$
    end for
  end for
  if $m \bmod F == 0$ **then**
    $\psi \leftarrow \theta$
  end if
end for

training, which is why REINFORCE is limited to being used strictly for episodic tasks. Moreover, REINFORCE is an on-policy algorithm, meaning experiences are generated from the same policy that is being optimized (Li, 2021). This hinders sample efficiency, as experiences must be discarded once they are used (Li, 2021a). The reason for this is there exists a high correlation between experiences when the policy that is being maximized is the same policy that is used for experience sampling (Li, 2021b).

One main weakness of REINFORCE is that it suffers from high variance. To solve this issue, a baseline method can be used to determine if the returns of the current action are indeed more favourable than the average returns (Li, 2021a). The baseline method that was used in this experiment is subtracting the average of previous action returns of the episode trajectory from the return of the current action. A positive number indicates that the current action is a favourable action (Li, 2021a).

2.2. DQN

When working with continuing tasks or environments that have a continuous or an extremely large state space, it would be unrealistic to represent an exact value function or policy. Instead, we must approximate them using a neural network. DQN is an extension of Q-learning, which is a type of temporal difference (TD) algorithm that finds optimal action value functions. Similar to REINFORCE, DQN samples experiences from the environment, however, DQN does not require the completion of an episode to begin collecting samples. It only collects experiences in the form of state, action, reward and next state. This allows for DQN to be utilized in both continuing and discrete tasks. DQN also differs from REINFORCE as it makes use of the recursive property of the action value function. The input of a DQN is a state-action pair and the output is the target, which is the reward from the maximizing action, according to the current estimate of values (Li, 2021b).

DQN is an off-policy algorithm, meaning the policy that is being maximized (the target policy) is different from the policy that experiences is sampled from (the proposal policy) (Li, 2021). In this experiment, the ϵ-greedy policy is used as the proposal policy. As a result of being an off-policy algorithm, DQN has the ability to reuse experiences, making it more sample-efficient than REINFORCE. Experiences are stored in an experience replay memory (ERM) and are uniformly sampled in batches for use (Li, 2021b).

DQN suffers from instability because the Q network is updated in each batch. This implies that although two different batches may have the same experiences, inconsistent targets will be produced. To avoid this weakness, a target network can be used along with the Q network. The Q network continues to be error-evolved, however, the Q network parameters are only periodically used to update the target network (Li, 2021b).

2.3. DDQN

DDQN is an improvement on DQN's action value overestimation. DDQN uses two networks: a training network that is used for action selection and a target network that is used for action evaluation (van Hasselt et al., 2016). The training and target networks are trained with overlapping experiences, but can work as two different networks only if the

Algorithm 3 DDQN with target networks

Input: number of batches B; batch size N; updates per batch U; discount factor γ; target network update frequency F; learning rate α; MAX_STEPS
Outputs: optimized parameters θ
for $m = 1 : MAX\_STEPS$ **do**
  gather and store h experiences (x_i, a_i, r_i, x_i') using ϵ-greedy;
  for $b = 1 : B$ **do**
    sample batch b of experiences from the ERM;
    for $u = 1 : U$ **do**
      for $n = 1 : N$ **do**
        $Y = r_n + \delta_{x_n'} \gamma Q^{\pi} \psi(X_n', max_{a'} Q^{\pi\theta}(X_n', a'))$
        $(\delta_{x_n'} = 0 \; if \; x_n' \; is \; terminal, 1 \; otherwise)$;
      end for
      $L(\theta) = \frac{1}{N} \sum_{n=1}^{N} \begin{cases} 0.5(\mathrm{x}_n - y_n)^2 & if |\mathrm{x}_n - y_n| < 1 \\ |\mathrm{x}_n - y_n| - 0.5 & otherwise \end{cases}$
      $\theta = \theta - \alpha \nabla_{\theta} \mathrm{L}(\theta)$
    end for
  end for
  if $m \bmod F == 0$ **then**
    $\psi \leftarrow \theta$
  end if
end for

sum of time steps between the two networks is large enough. The target network is updated by replacement, which means it remains fixed for a set period of time while the training network updates regularly. Once a certain length of time has passed, the target network is then replaced by the training network. This is done throughout the training of the agent (Graesser & Keng, 2019, pp. 106-108).

3. Results

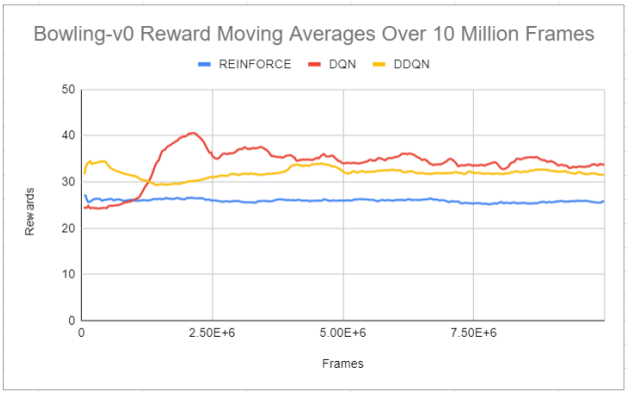

Figure 1. Reward moving averages achieved by the 3 algorithms

Our results are based off total rewards received after completion of one full game. While DQN preformed the best out of the three algorithms, it was the most unstable, with a 40.38 difference between its highest and lowest rewards achieved. DDQN is the second-best performing algorithm and is more stable than DQN, with an 18.06 difference between its highest and lowest rewards. Lastly, REINFORCE had the weakest performance but was the most stable, with a 10.56 difference between its highest and lowest rewards.

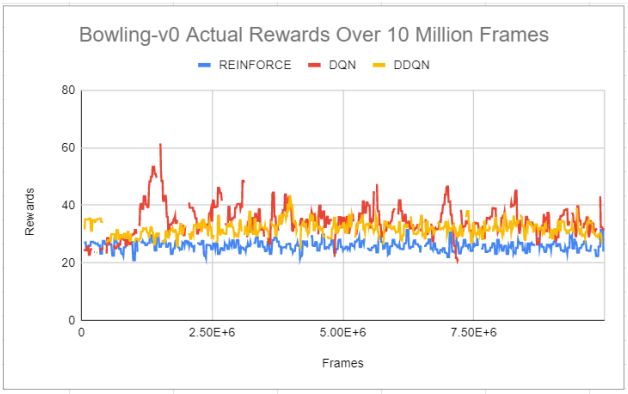

Figure 2. Actual rewards achieved by the 3 algorithms

Table 1. Summary Of Results For REINFORCE, DQN And DDQN

	REINFORCE	DQN	DDQN
Maximum Reward	31.50	61.13	43.19
Minimum Reward	20.94	20.75	25.13
Average Reward	25.91	34.55	31.86

4. Discussion

Two major observations can be made regarding the performance of REINFORCE: (a) convergence was reached very early in the training, and (b) it was out-performed by the value-based algorithms. The action selection method used by DQN and DDQN is ϵ-greedy, which is a very sensitive method. Slight changes in action value estimations may lead to extreme shifts in action selection probabilities. The action selection probabilities in REINFORCE are adjusted in a smooth fashion due to continuous policy parameterization, which is a significant reason for why REINFORCE has more of a guarantee to converge when compared to DQN and DDQN. Moreover, adding a baseline to REINFORCE greatly reduces variance, leading to faster convergence (Sutton & Barto, 2018, pp. 321-324). Figures 1 and 2 show the stronger stability of REINFORCE in comparison to the other two methods. Additionally, as a result of using MC sampling, REINFORCE has the ability to observe the long-term effects of actions rather than only the immediate

effects, which helps with learning the true value of actions (Zai & Brown, 2020, p. 113).

While policy gradient methods are guaranteed to converge to a local optima, they are not guaranteed to converge to a strong local optima. This is because stochastic gradient descent is applied to a non-convex loss function by policy gradient methods, where extremely sub-optimal stationary points may exist. Convergence to a poor local optima by REINFORCE is evident in our findings, and is a reason why REINFORCE was outperformed by DQN and DDQN (Bhandari & Russo, 2019). DQN and DDQN are less likely to have the issue of converging to a local optima due to the use of the ϵ-greedy method. This is because if given enough time, ϵ-greedy will visit every possible state-action pair often enough such that convergence to true optimal action values occurs. This is a result of ϵ-greedy randomly selecting between exploration and exploitation, and when exploration is selected, actions are uniformly sampled (Wang & Vinel, 2020).

Our results suggest that DQN performed marginally better than DDQN. This was not anticipated, as DQN suffered from action value overestimations, which was expected (van Hasselt et al., 2016). The main limitation of our work is that our results are based on conducting one trail of 10 million frames of training for each algorithm. An increase of training trials may have produced more predictable results, however, we were limited by computational resources.

There are two factors that influence the impact that DQN's action value overestimation has on the outcome: (a) the distribution of the overestimation, and (b) the size of the action space (Graesser & Keng, 2019, pp. 107-108; van Hasselt et al., 2016). The closer to having an equal action value overestimation distribution, the more mitigated the negative impacts on the results are (van Hasselt et al., 2016). Moreover, the greater the action space, the more severe the overestimation will be (Graesser & Keng, 2019, pp. 107-108). The action space in the bowling environment is exceptionally small, as it is reduced from six to three (curve up, curve down or no curve) after the ball is released and is reduced to zero once a curve is applied. With that being said, these factors may have affected the severity of the impact of overestimation for DQN.

Our results indicate that DQN is highly unstable compared to DDQN. This aligns with previous findings, which suggest a strong correlation between action value overestimation and instability (van Hasselt et al., 2016).

5. Conclusion

In this experiment, we successfully trained an agent to play Atari 2600 Bowling using three different deep reinforcement algorithms: REINFORCE, a policy-based method, as well as DQN and DDQN, which are value-based methods (Li, 2021a; Li, 2021b). Our results indicate that REINFORCE is the most stable but has the poorest performance. DQN is the least stable but has the best performance. DDQN scored in between the other two algorithms, as it is the second most stable and the second best-performing. REINFORCE was expected to under perform, as it is common for policy gradient methods to converge to poor local optima (Bhandari & Russo, 2019). DQN was not expected to outperform DDQN, as DQN suffers from action value overestimation (Van Hasselt et al., 2016). Although overestimation did occur and immensely hindered DQN's stability, the algorithm still performed better than DDQN. We presented some possible explanations for this, however, the most likely explanation is that not enough training sessions were run for each algorithm. Future work should consider our limitation and present results that are averaged over multiple training sessions for each algorithm.

6. References

Bhandari, J., & Russo, D. (2019). Global optimality guarantees for policy gradient methods. arXiv preprint arXiv:1906.01786.

Graesser, L., & Keng, W. L. (2019). Foundations of Deep Reinforcement Learning. Addison-Wesley Professional.

Li, Y. (2021). COSC 4P83/5P83: Lecture 03 Monte Carlo Model-Free Prediction and Control Methods [LaTeX slides]. School of Computer Science, Brock University.

Li, Y. (2021a). COSC 4P83/5P83: Lecture 05 Policy Gradient Methods [LaTeX slides]. School of Computer Science, Brock University.

Li, Y. (2021b). COSC 4P83/5P83: Lecture 06 Deep Q-Networks for Value Approximation [LaTeX slides]. School of Computer Science, Brock University.

Sutton, R. S. & Barto, A. G. (2018). Reinforcement Learning: An Introduction. The MIT Press.

van Hasselt, H., Guez, A., & Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. Proceedings of the AAAI Conference on Artificial Intelligence, 30(1). Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/10295

Wang, X., & Vinel, A. (2020). Reannealing of Decaying Exploration Based On Heuristic Measure in Deep Q-Network. ArXiv, abs/2009.14297.

Zai, A. A. & Brown, B. (2020). Deep reinforcement learning in action. Manning Publications Company.