Reinforcement Learning for Tower Defence

Andrew Wallace
101210291
andrewwallace3@cmail.carleton.ca

Derrick Zhang
101232374
derrickzhang@cmail.carleton.ca

Mohammad Rehman
101220514
mohammadrehman@cmail.carleton.ca

Manal Hassan
101263813
manalhassa@cmail.carleton.ca

Abstract

We investigate and evaluate the effectiveness of various reinforcement learning (RL) algorithms in a custom high-dimensional tower defence environment, featuring a large discrete action space, stochastic dynamics, sparse rewards, and an automatic tower level-up mechanism. Algorithms such as Q-learning with Q-target networks, SARSA with Q-network approximation, Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Deep Q-Networks (DQN) were evaluated in their ability to learn an optimal policy. Our results show that tabular-inspired methods such as Q-learning and SARSA struggle due to the environment's non-stationary dynamics, yielding high variance and poor convergence. Policy gradient-based algorithms such as A2C and PPO achieved better results, but exhibited instability and sensitivity to hyperparameters. Overall, DQN was the best performing algorithm, combining experience replay and target networks to achieve high returns.

1 Introduction

1.1 Project focus

Tower defence is a strategy game where the goal is to place defensive structures to obstruct and defeat waves of enemies from reaching a goal. This is done by placing and upgrading towers along the attacker's path. The towers are stationary objects that the agent places to attack enemies. These towers have a cost to create, meaning the agent must gather resources to do so. In traditional tower defence games, the player starts with a set number of resources and can obtain more after successfully defending from a wave or defeating enemies. The defensive towers can vary in range and damage.

The objective of this project is to train an agent using reinforcement learning (RL) methods to learn an optimal policy for tower placement that maximizes survival and minimizes damage taken from enemies. Unlike traditional tower defence games with manual upgrades, our environment features towers that automatically evolve and strengthen based on the number of enemies they eliminate, creating a feedback loop between decisions and long-term strategy when it comes to positioning.

Preprint. Under review.

1.2 Importance of the problem

Exploring RL for tower defence is extremely valuable. Applying RL methods to high-dimensional state and action spaces, challenging reward structures, and balancing short and long-term investment provides valuable insights into RL's capabilities in highly transferable domains.

1.3 MDP specifications

The state space is represented as a 3d box the shape of [10, 10, [8]], 10 and 10 representing the x, y coordinate of a cell on the grid and the third dimension being an array containing:

Tower ID, tower level, number Of enemies, average health of enemies, enemy with the highest health value, if the cell has an enemy, if the cell is the base, and the health of the base.

The state space would be the addition of all the max possible values of the third dimension to the power of cell on the grid:

$$(3 + 6 + 11 + 51 + 51 + 2 + 2 + 41)^{100} = 167^{100}$$

Actions taken in this space follow a Markov decision process (MDP) as seen in Figure 1.

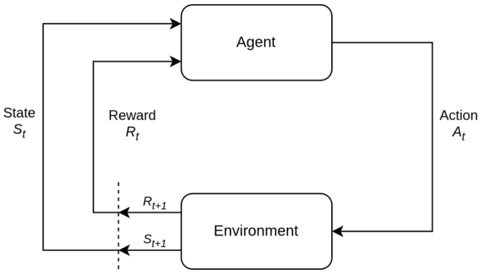

Figure 1: Markov decision process

The action space is of size 97. The agent can either do nothing or place one of two types of turrets on the 49 viable positions on the grid. The agent has the choice of placing down a single target tower, an area of effect tower or do nothing. In this project the agent acts in phases, there's a build phase and a defence phase. In the build phase, the agent is allowed to place one of two types of towers or be inactive. While in the defence phase, the agent is allowed to place towers so long as he has the budget to, but the waves of enemies will begin spawning.

The reward structure is the following:

Enemy Defeated = +10
Enemy Reaches Base = -15
Enemy Damaged = +1
Tower Level up = +5
Wave Cleared = +20
All Waves Cleared = +200
Base Destroyed = -50
Invalid or No Action = -1

2 Approaches

2.1 Prior work

Previous work by Maria Manolaki also focused on implementing reinforcement learning (RL) for tower defence for learning an optimal policy [1]. However, in their implementation, the enemy was the artificial intelligence agent. The agent's goal was to learn an optimal policy to win the game. This involves the enemy avoiding towers and finding the best path to the base. They implemented the A2C algorithm to provide the enemy with the intelligence needed to reach the base. Our implementation differs, focusing only on the player agent. It would be interesting to build on this by deploying a multi-agent interaction between a player and an enemy agent and analysing their interaction.

Another application of RL for tower defence was implemented by Timothée Blondiaux et al. for Plants vs. Zombies, a popular tower defence game consisting of plants as towers and zombies as enemies [2]. They implemented a wide array of RL methods, including A2C, Monte-Carlo policy gradient, Deep Q-Networks (DQN), and Double Deep Q-Networks (DDQN). They observed good results with the DDQN algorithm, beating difficult levels 65% of the time [2].

A more general approach was taken in a study by Abhinav Wasukar and Sumitra Jakhete, looking at the application of the DQN and PPO algorithms in various game environments, including tower defence. They found that PPO significantly outperformed DQN, achieving a much higher average return. However, DQN showed lower variance and was more consistent across runs [3].

One interesting (and indirect) study done by Baylor Wetzel studied Transfer Learning in a tower defence environment. Transfer learning is the process of using knowledge gained while solving one problem to solve a new, previously unencountered problem [4]. In this study, they collected data from human-reported solutions to a tower defence game. They used these solutions to create computational agents to model human strategies. Then, a domain expert was asked to tell apart the human-made tower placements from the computer-generated ones - they could not tell the difference [4]. This approach focuses on modelling human behaviour rather than achieving optimal performance.

A similar study by Augusto Dias et al. utilised game metadata as inputs to a Deep Q-Learning agent rather than video frames or pixel information [5]. They found that no other work reported developing autonomous agents to play tower defence games (2020) and that using metadata as input reduced computational complexity [5]. This is similar to our implementation, where the state is represented by the game metadata.

2.2 Our approaches

Our project differs from these works through an automatic tower level-up mechanism and a unique approach to action and state space reduction. In our environment, towers level up automatically based on the number of enemies they kill without explicit user input. This adds a layer of complexity for learning optimal solutions. That is, balancing strategic placement for tower level-ups and successful long-term defence.

The tower defence environment has a high-dimensional and continuous state, with a large discrete action space. This makes tabular-based learning methods infeasible. Conventional methods, such as Tabular Q-Learning or SARSA, cannot store Q-tables for an infinite space. So, we first chose SARSA with function approximation to handle the high-dimensional state space. Also, SARSA is an on-policy algorithm that provides sufficient coverage of the large action space. Note that linear function approximation struggles with the large state space, so a Q-network was used. Next, we looked at the Advantage Actor-Critic Algorithm (A2C), which handles continuous state spaces and discrete action spaces [6]. Actor-Critic algorithms utilise policy gradient to update the policy directly with learned parameters. This allows us to tune the policy to reduce variance, demonstrated by methods such as SARSA (e.g., reducing actor step size).

However, SARSA and A2C are on-policy algorithms, making convergence to a stable optimal policy difficult. We then explored Deep Q-Learning with Q-Networks to avoid the on-policy nature of SARSA, and utilized a replay buffer to reduce variance.

3 Empirical studies

3.1 Q-learning

We implemented Q-learning using a simple 3-layer Q-network as our function approximator and a target network to stabilize updates. The Q-network allowed us to learn in a high dimensional state and action space where tabular Q-learning would be infeasible. The space complexity for tabular learning would be $O(|S|x|A|)$.

The target network was implemented to reduce variance and stabilize learning across episodes. However, this approach did not yield good results. As seen in Figure 2, no convergence and instability in learning.

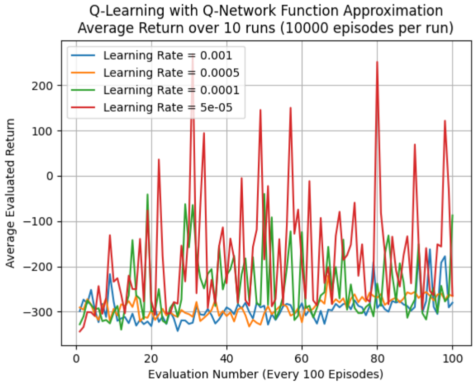

Figure 2: Q-learning performance with different learning rates.

This is due to the deadly triad combination of function approximation, bootstrapping, and off-policy learning. Even when attempting to stabilize bootstrapping updates, there was no consistency in learning. One interesting observation was that higher learning rates yielded more consistent, but lower returns. On the other hand, lower learning rates showed high variance and less consistency. With a lower learning rate, the immediate rewards propagate slowly through the Q-network, so stochasticity in the environment dominates. This demonstrates Q-learning's sensitivity to hyperparameters and overall instability.

3.1.1 SARSA

After implementing the SARSA algorithm with Q-network function approximation, we evaluated and observed the following results in Figure 3.

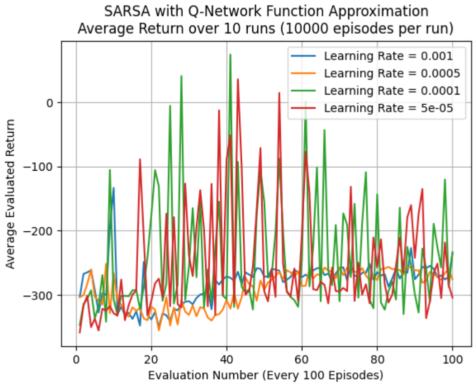

Figure 3: SARSA performance with different learning rates.

First, the environment has a delayed and sparse reward structure, causing SARSA's on-policy TD(0) update to struggle with delayed reward signals. Also, as wave difficulty increases, an action taken in one state may signal a vastly different reward in another state. For example, in an early wave when enemy health is lower, placing a tower in a cell that results in the elimination of the enemy will return a large positive return. However, in later waves, when enemy health is high, placing the same tower in the same cell may not yield the same result. This inhibits SARSA learning, since it expects environment dynamics to be stationary to guarantee convergence. Furthermore, SARSA is an on-policy algorithm. It depends directly on actions taken. In early exploration, actions are random, which leads to poor estimates, noisy updates, and high variance.

Due to the on-policy nature of SARSA, we do see more coverage of the action space compared to off-policy methods such as Q-learning. Overall, SARSA performed poorly in learning an optimal policy. Reducing the state dimensionality and ensuring stationary environment dynamics are required for better performance. Other methods should be considered before implementing SARSA in a tower defense environment.

3.1.2 Advantage Actor-Critic (A2C)

After running the A2C algorithm, it was observed in Figure 4 to have rough convergence to an optimal policy.

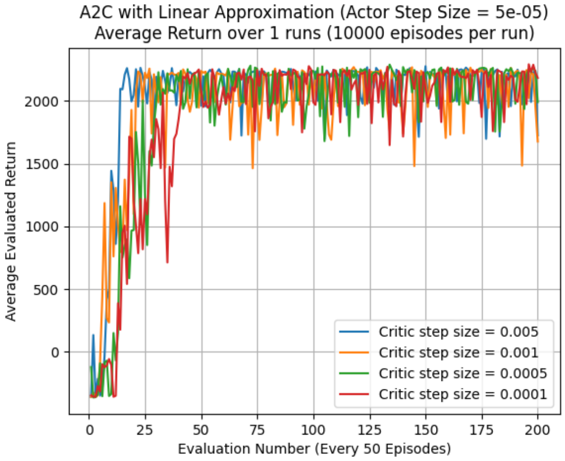

Figure 4: A2C performance with different learning rates.

In this reward structure, the estimated maximum value you could reach when interacting with the environment was 3700. In Figure 4, the algorithm performed well in achieving close to maximum reward values across runs. However, we do see high variance regardless of the critic's step size. This is due to the on-policy nature of the algorithm, where updates to the actor policy impact the policy for all other actions. Another reason is that A2C uses a SoftMax policy. The action is sampled from a distribution, so there's always a chance that the policy will deviate from the optimal action, creating variance in the average evaluated return.

We also noticed the training time for A2C was considerably longer compared to Q-learning and SARSA.

Overall, the A2C algorithm performed well with this environment, likely due to the critic's ability to make less accurate state-value estimates without drastically impacting the actor policy.

3.1.3 Proximal Policy Optimization (PPO)

Proximal Policy Optimization is a type of policy gradient method, particularly useful for complicated or large environments. Our tower defence environment has a large discrete action space making this algorithm suitable as it allows for efficient state exploration through stochastic sampling rather than argmax selection over Q-values [7]. We implemented this algorithm using the Stable Baseline library.

In our experiments, we tested two key parameters, learning rates and clip ranges. In the first experiment we tested 3 different clip ranges with a fixed learning rate of 0.0003. The experiment was set up to run for 1,000 episodes, with evaluation performed every 15 episodes, and a single run per configuration.

At the first clip range of 0.1, the agent achieved an early return of approximately 7,200 with another peak at around episode 750. Although this configuration had a long period of near 0 return between the two peaks it achieved the highest single evaluation return. The next clip range of 0.2 demonstrated a slower initial learning compared to 0.1, with lower peaks through learning episodes but showed a major peak at the midpoint of training. Lastly, the clip range of 0.3 showed very slow

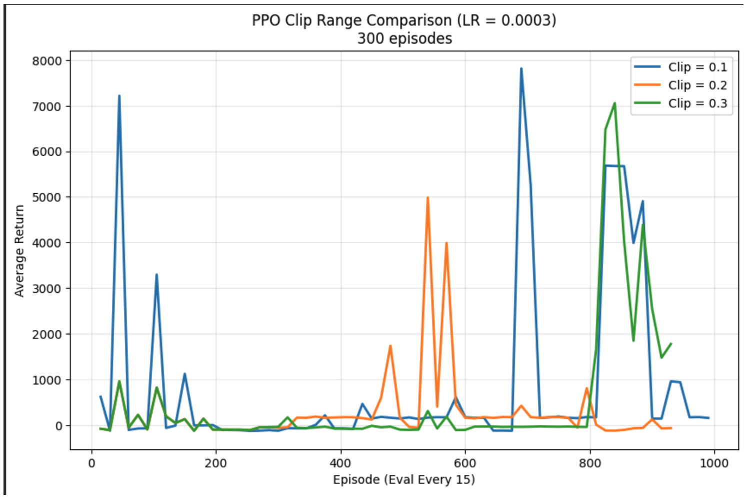

Figure 5: PPO performance with 3 different clip ranges.

learning performance during early episodes and a major return peak near episode 800, maintaining sustained returns of 1,500-3,000. Although there is slow initial learning, 0.3 shows promising visible improvement in learning and high returns with more training. Overall, smaller clips values show a faster initial return but are less stable, and higher clip values learn slowly but show improvement with more training.

In the second experiment, we had a fixed clip range at $\epsilon = 0.2$ and tested with 3 different learning rates.

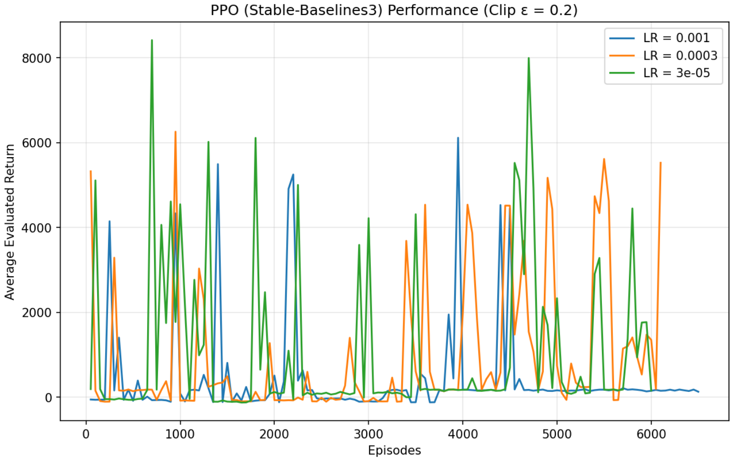

Figure 6: PPO performance with a fixed clip range.

The first learning rate of 0.001 showed high early returns but unsustained early in the training, this likely occurred because large policy updates caused the agent to overshoot good tower placement before exploring all action space. The second learning rate of 0.0003 also showed similar sparse and occasional peak returns throughout training, however towards the end of the training we can observe more consistent returns, which can be explained by the sparse rewards of the environment. The last learning rate of 0.00003, demonstrated the largest returns with a peak of 8,000. This likely occurred because small gradient steps allow the policy to not be vulnerable to environmental stochasticity. From both experiments the prolonged near 0 return can be explained by the environment's large state space as well as the sparse rewards making it difficult to have consistent results.

3.1.4 DQN

For the DQN experiments, the agent observes the flattened metadata state vector [5] and we apply per-feature min-max normalization before feeding it to the network. The Q-function is represented by a three-layer feed-forward network with two 128-unit hidden layers and ReLU activations, outputting one Q-value per discrete action. To stabilize learning, we follow the standard DQN design. We use an experience replay memory that stores up to 50,000 transitions, mini-batches of size 64 are sampled for updates, and a separate target network is updated every 10 episodes. Exploration is ϵ-greedy with ϵ decaying exponentially from 1.0 to a minimum of 0.05 over episodes.

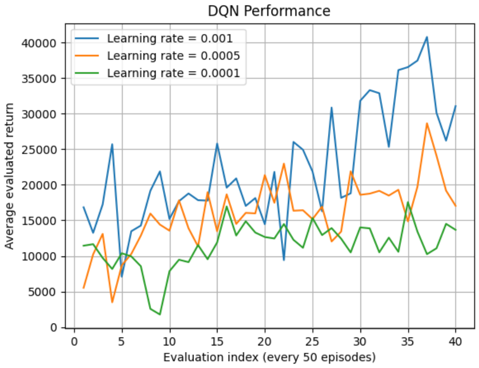

Figure 7: DQN with 3 different performance rates

Figure 7 summarizes the learning curves for the three learning rates (0.001, 0.0005, 0.0001). Each configuration was trained for 2,500 episodes and evaluated every 50 episodes for 1 run. All three learning rates show an overall upward trend, indicating that DQN is able to exploit experience replay and the target network to improve performance over time rather than diverging, unlike the earlier Q-learning baseline. The largest learning rate (0.001) achieves the fastest performance gains and reaches the highest final average return, although the curve being more volatile than the others since the update step is larger. The intermediate rate (0.0005) learns more slowly but exhibits a smoother curve with fewer sharp drops, while the smallest rate (0.0001) produces the most stable but also the lowest returns, this suggests underfitting and a very slow propagation of reward information due to the sparse and delayed rewards.

Overall, these results suggest that DQN is the most effective algorithm that we tested being significantly more effective than our vanilla Q-learning and SARSA with neural function approximation, which showed little convergence and high instability. However, the DQN curves are still quite volatile, reflecting the difficulty of learning in a high-dimensional, stochastic environment with delayed rewards. The sensitivity to the learning rate visible in Figure 7 indicates that larger step sizes can unlock higher returns but at the cost of higher variance, while smaller step sizes are more stable but may fail to fully exploit the environment within a fixed training budget.

3.1.5 Quantile Regression DQN (QR-DQN)

To attempt to address the instability observed in standard DQN, we implemented Quantile Regression DQN (QR-DQN), which is a distributional RL algorithm. Rather than learning a single expected Q-value, QR-DQN models the full distribution of returns using a set of quantiles. This may provide better information about uncertainty in value estimates, which can improve learning stability, especially in a more stochastic environment like this. This algorithm was implemented using the SB3 library with 50 quantiles.

Our experiment ran for 100,000 timesteps, evaluating every 5,000 and averaging over 3 runs. Three learning rates were tested: 0.001, 0.0005, and 0.0001. The results are shown in Figure 8.

The highest learning rate (0.001) showed steady improvement throughout training, reaching an average evaluated return of around 7,000 by the end. Despite an initial dip around 35,000 timesteps, the algorithm recovered and showed consistent upward progress. The middle learning rate (0.0005) exhibited faster early learning, peaking at approximately 6,500 around 55,000 timesteps. However,

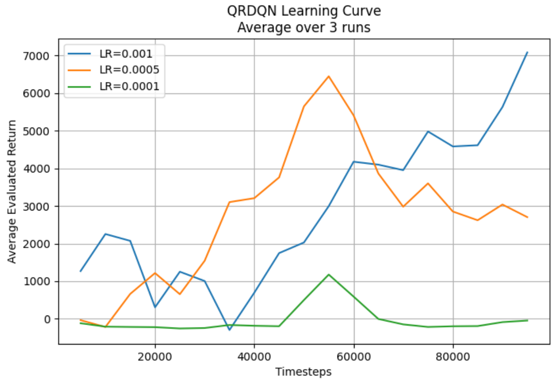

Figure 8: QR-DQN performance with different learning rates.

performance degraded in later timesteps, suggesting that there may have been overfitting or oscillation happening. The lowest learning rate (0.0001) failed to learn effectively, remaining near zero with one spike around 35,000 timesteps, suggesting that there was insufficient adaptation.

QR-DQN outperformed both Q-learning and SARSA, but it did not surpass standard DQN, which achieved returns in a much higher range. The more complex modelling of returns may have hindered the learning efficiency within the training budget. As well, the approach of this algorithm added computational overhead and training time without much benefit. The simpler DQN with experience replay appears better suited for this environment.

4 Conclusion

In conclusion, we explored various reinforcement learning (RL) methods to learn an optimal policy for tower defense. This included Q-learning, SARSA, Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), Deep Q-Networks (DQN), and Quantile Regression DQN.

We observed high variability and poor performance for the Q-learning and SARSA algorithms. This is due to the difficulty of bootstrapping in high dimensional state spaces with sparse and delayed rewards. The deadly triad of function approximation, bootstrapping, and off-policy learning proved particularly problematic for Q-learning, while SARSA struggled with non-stationary environment dynamics as wave difficulty increased.

Policy gradient-based algorithms such as A2C and PPO yielded better results. This is due to their ability to learn the policy directly, instead of relying on bootstrapped Q-values. A2C achieved returns close to the theoretical maximum of 3,700, demonstrating that actor-critic methods are well-suited for this environment. PPO showed sensitivity to the clip range hyperparameter, with smaller clip values learning faster but less stability.

DQN emerged as our best performing algorithm, achieving returns in the 10,000-40,000 range. The combination of experience replay and target networks effectively addressed the instability seen in vanilla Q-learning. Interestingly, QR-DQN did not improve upon standard DQN despite its distributional approach, suggesting that modelling return uncertainty provides limited benefit in this environment and adds unnecessary computational overhead.

A key insight from this project is the importance of matching algorithm complexity to the problem. Simpler methods with appropriate stabilization techniques (e.g., DQN with replay buffers) outperformed more sophisticated distributional approaches. Additionally, the automatic tower evolution mechanic created a challenging credit assignment problem, where early placement decisions have delayed effects on tower strength.

In future work, more analysis and design of a strong feature extractor should be considered. This would help reduce the high dimensional state complexity and improve algorithm performance. Reward shaping to provide denser feedback signals could also accelerate learning. Next, other algorithms should be considered. To address overestimation, an algorithm such as Double Deep Q-Networks may be considered. For more parallelized training, Asynchronous Advantage Actor-Critic can be considered. In terms of training, curriculum learning may help agents learn fundamental strategies before facing the full complexity of the environment (i.e., starting with fewer waves and gradually increasing difficulty).

Overall, we gained valuable insights into the application of RL in a tower defence environment. This environment proved to be a challenging testbed for RL due to a combination of strategic planning, resource management, and stochasticity. In particular, we learned about the tradeoffs between an algorithm's complexity, stability, and efficiency.

References

[1] M. Manolaki. Development of a tower defense game with reinforcement learning agents. Technical report, Online report, November 2020.

[2] T. Blondiaux, L. Frank, H. Gebran, and A. Perot. Plants vs. zombies: Reinforcement learning to a tower defense game, 2025. URL `https://hanadyg.github.io/portfolio/report/INF581_report.pdf`.

[3] A. Wasukar and S. Jakhete. Comparative study of reinforcement learning methods: Dqn vs ppo in a game, December 2024. URL `https://pijet.org/papers/volume-2%20issue-1/Final%20Revised%20Paper_Pijet-07_Done.pdf`.

[4] B. Wetzel. Transfer learning in spatial reasoning puzzles. In *Proceedings of IJCAI 2011*, 2011. URL `https://www.ijcai.org/Proceedings/11/Papers/504.pdf`.

[5] A. Dias, J. Foleiss, and R. P. Lopes. Reinforcement learning in tower defense. In *Communications in Computer and Information Science*, pages 127–139. 2022. doi: 10.1007/978-3-030-95305-8_10. URL `https://doi.org/10.1007/978-3-030-95305-8_10`.

[6] Actor-critic methods — mastering reinforcement learning, 2023. URL `https://gibberblot.github.io/rl-notes/single-agent/actor-critic.html`.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018. URL `http://incompleteideas.net/book/RLbook2020.pdf`.