Restless Multi-Armed Bandits

Ipsita Mohanty imohanty@uwaterloo.ca
Faculty of Mathematics
University of Waterloo
Waterloo, Ontario, Canada

Editor: Leslie Pack Kaelbling

Abstract

This paper considers the multi-armed restless bandit problem (RMABP) with a finite horizon where the state of each arm continues to evolve even when it is not being played. We model the RMAB environment and evaluate various classical and deep learning-based RL techniques for complexity, performance, and ease of use.

Keywords: Restless multi-armed bandits, Reinforcement Learning, DQN, PPO

1. Introduction

Restless multi-armed bandits (RMAB) is a popular paradigm in many resource allocation problems arising in engineering, as discussed in [NM07; LZ10; NMV11]. It has myriad applications, including but not limited to Job Allocation in a server, Channel Detection in a wireless network, Health Care Systems, Dynamic Posted Pricing, and dynamic UAV routing. Unlike the classical multi-armed bandits (MAB), which remain frozen when not used, restless bandits do not. We discuss both techniques briefly in the following subsections.

1.1 Multi-arm Bandits

The classic multi-armed bandit (MAB) with an i.i.d. reward model has N independent arms and a single player. Each arm, when played, offers an i.i.d. random reward drawn from a distribution with an unknown mean. At each time, the player chooses one arm to play, aiming to maximize the total expected reward in the long run. This problem involves the well-known trade-off between exploitation and exploration, where the player faces the conflicting objectives of playing the arm with the best reward history and playing a less explored arm to learn its reward statistics.

A commonly used performance measure of an arm selection policy is called *regret* and is defined as the reward loss concerning the case with a known reward model. It is clear that under a known reward model, the player should always play the arm with the largest reward means. The essence of the problem is thus to identify the best arm without engaging other arms too often. Any policy with a sublinear growth rate of regret achieves the same maximum average reward (given by the largest reward mean) as in the known model case. However, the slower the regret growth rate, the faster the convergence to this maximum average reward, indicating a more effective learning ability of the policy.

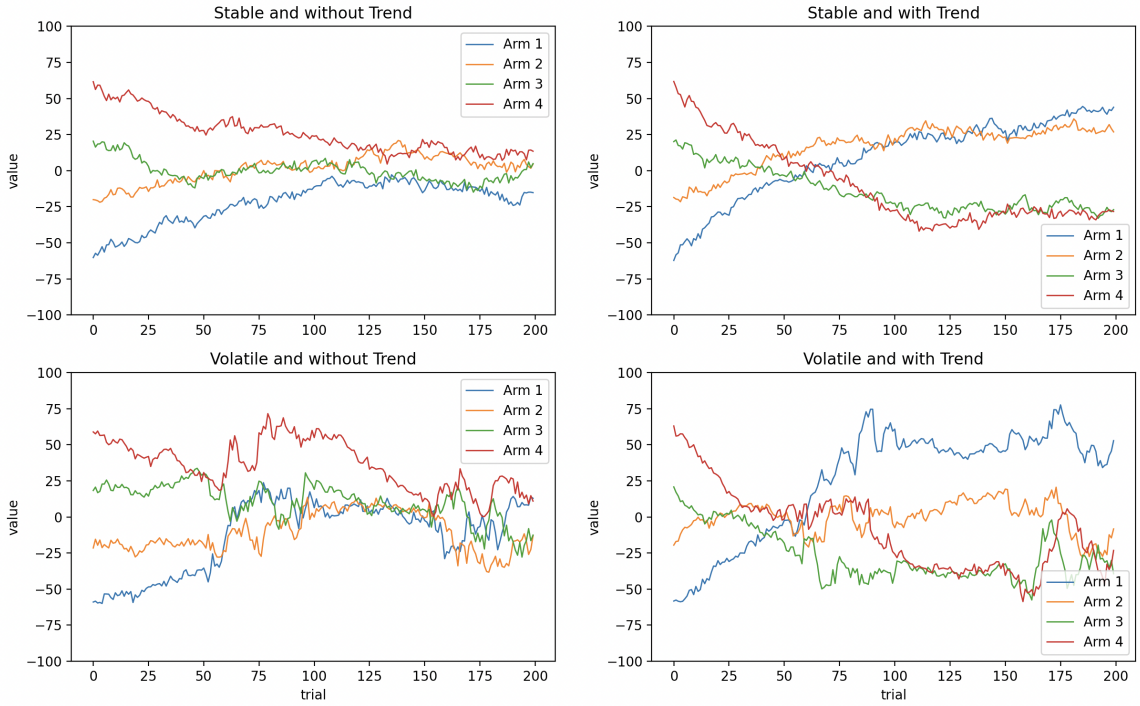


Figure 1: Example rewards in the four-armed restless bandit task

1.2 Restless Multi-Armed Bandits

This paper considers Restless Multi-Armed Bandit (RMAB), a generalization of the classic MAB. In contrast to the rested Markovian reward model, in RMAB, the state of each arm continues to evolve even when it is not played. More specifically, the state of each arm changes according to an unknown Markovian transition rule when the arm is played and according to an arbitrary unknown random process when the arm is not played. We will formulate the problem formally in section 3.

The paper is organized as follows. section 2 talks about previous work concerning the RMAB problem and briefly discusses the approach used. In the next section, we mathematically formulate the problem statement. We then discuss the environment modeling and specific implementation details for all the algorithms used in section 4. section 5 presents the effect of changing model parameters on an agent's performance and compares all the techniques for complexity and performance. Due to space constraints, a few extra plots are included in the Appendix (section 6).

2. Related Work

The restless multi-armed bandit (RMAB) problem was introduced by Whittle in 1988 [Whi88]. The main result involves formulating a relaxation of the problem and solving it optimally using a heuristic called the Whittle Index policy. This policy is optimal when the underlying Markov Decision Processes satisfy indexability, which is computationally

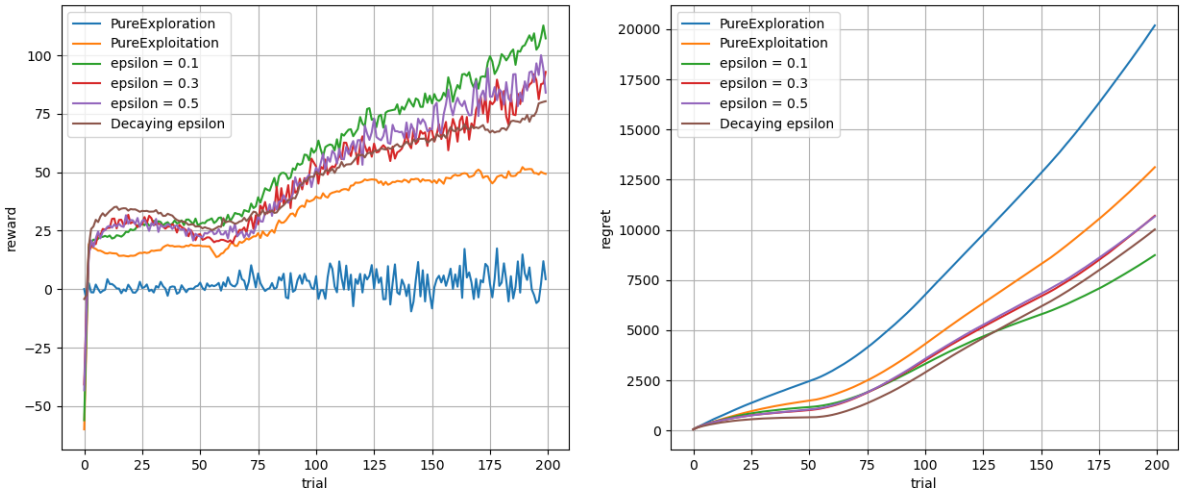


Figure 2: Comparison over different epsilon values for epsilon-greedy algorithm in an environment with high trend and volatility

intensive to verify. Moreover, [PT94] showed that solving RMAB is PSPACE-hard due to the state-dependent reward and combinatorial action space.

In [FNMT19] and [BC18], the authors build their work upon this Whittle Index Policy which requires the knowledge of the underlying MDP, which is rarely known in a practical setting. The main idea was to assign an index to every arm, choose the arm with the largest index, update all indices, and so on. Parallel Q-learning recursions are utilized to learn the whittle indices, which could then solve sub-problems with reduced state space to maximize the average reward of the whole system in the long run. Whittle index policy is asymptotically optimal under certain conditions as given in [WW90].

[WHL20] implements a Restless-UCB framework with a guaranteed regret bound of polynomial complexity in contrast with the exponential complexity of previous methods. Moreover, this framework works with a general online restless bandit problem where the underlying MDP is unknown. Their Restless-UCB algorithm achieves a regret upper bound of $O((N+M^3)T^{\frac{2}{3}}$ where N is the number of arms, M is the Markov chain state space size and T is the time horizon.

[KXBT21] generalizes the RMABP problem by developing a *minimax* regret objective, which essentially minimizes the worst possible regret for the optimal policy and Proximal Policy Approximation (PPO). It also takes the cost involved with choosing a particular action into account. This approach is further extended to a multi-agent setting, and it is shown that their approaches work well in experimental domains.

3. Problem Formulation

We formulate the problem exactly as defined in [SK15]. We have one player and K independent arms in the centralized setting. On each trial t, the reward R_k associated with

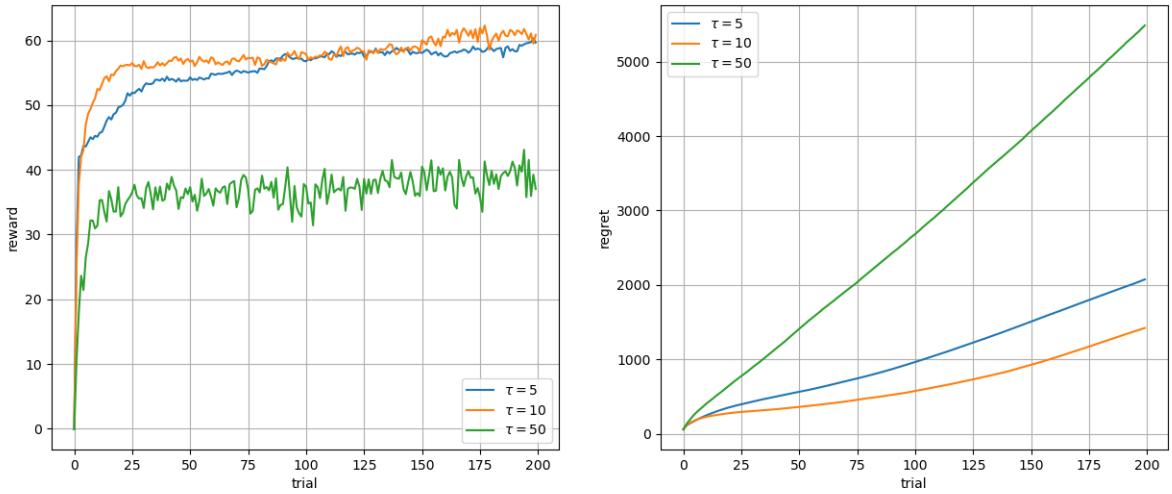


Figure 3: Comparison over different τ values for Softmax selection in an environment with high trend and volatility

each arm $k \in \{1, \ldots, k\}$ is drawn from a Normal distribution, with a mean μ_k that changes over trials according to a random walk. That is -

$$R_k(t) = \mu_k(t) + \epsilon_k \qquad \epsilon_k \sim \mathcal{N}(0, \sigma_\epsilon) \tag{1}$$

$$\mu_k(t) = f(\mu_k(t-1)) + \zeta_k \qquad \zeta_k \sim \mathcal{N}(0, \sigma_\zeta) \tag{2}$$

where f is a linear function. There are two sources of variability in this process: the reward variance σ_ϵ, reflecting the extent to which rewards vary around their mean, and the innovation variance σ_ζ, reflecting the volatility of the environment (time-dependent variation in the mean rewards associated with each arm).

We formulate the decision-making in a restless multi-armed bandit task similar to that used by [DOD+06]. The linear function f is defined as $f(x) = \lambda x + \kappa$. More precisely, the reward mean $\mu_k(t)$ for each arm k is now given by

$$\mu_k(t) = \lambda \mu_k(t-1) + \kappa_k + \zeta_k \qquad \zeta_k \sim \mathcal{N}(0, \sigma_\zeta) \tag{3}$$

where a decay parameter $0 < \lambda < 1$ is used so that values remained closer to 0 than with a pure random walk and κ_k is the fixed trend value associated with the arm k. Four versions of the task were constructed in which (a) the average rewards of the arms changed either completely unpredictably or with a small trend (κ is non-zero), and (b) the volatility of the changes was either stable or there were periods of relatively high volatility (ζ_k is high).

An ideal Bayesian learner with knowledge of the properties of this process would update her belief about the average rewards based on the observed rewards corresponding to the chosen arm at each time step; $p(\mu_k(t)|\mathbf{R}, \mathbf{C})$; the posterior distribution of arm k's average reward, conditional upon the obtained rewards $\mathbf{R} = (R_1, \ldots, R_t)$ and choices made $\mathbf{C} = (C_1, \ldots, C_t)$. This posterior distribution, together with the structural model, allows the

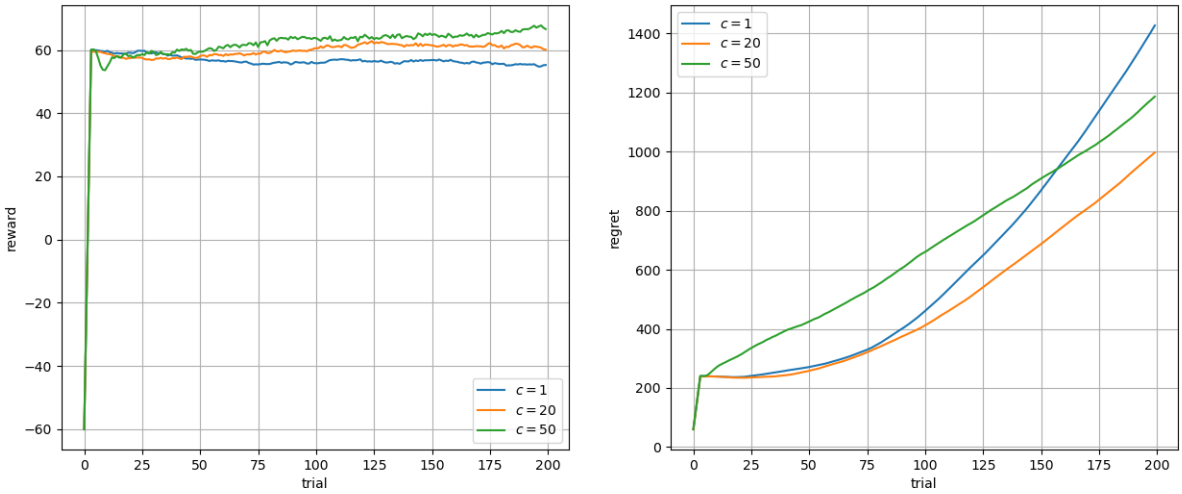


Figure 4: Comparison over different c values for UCB in an environment with high trend and volatility

agent to derive a prior distribution $p(\mu(t+1)|\mu(t))$ for an arm's average reward on the next trial. In turn, these prior distributions can be used to derive prior predictive distributions for the actual rewards that can be obtained by playing each arm:

$$p(R_k(t+1)|\mathbf{R},\mathbf{C}) = \int p(R_k(t+1)|\mu_k(t+1)) \cdot p(\mu_k(t+1)|\mu_k(t)) \cdot p(\mu_k|\mathbf{R},\mathbf{C})$$

4. Environment Details and Implementation

In this section, we present details regarding the implementation of the model along with details corresponding to some well-known reinforcement learning algorithms. Specifically, we implement the ϵ-greedy algorithm, Upper Confidence Bound (UCB) algorithm,

Environment: The implementation metrics are directly taken from [SK15]. We consider a four-armed bandit environment (i.e., $K = 4$). The total number of trials in a game is set to 200, and in each trial, we choose any one out of the four arms. The rewards associated with each choice are given by Equation 1 and Equation 3. The averages of the arms are initialized as $\mu = -60, -20, 20, 60$ for arms $k = 1, \ldots, 4$, respectively and the reward variance σ_ϵ^2 is set to 16. A decay parameter $\lambda = 0.9836$ is used so that values of μ remain bounded. For different sets of values of trend κ and innovation variance σ_ζ, there can arise four cases - *(1) Stable innovation variance with a trend (ST), (2) Volatile innovation-variance with a trend (VT), (3) Stable innovation variance without trend (SN), and (4) Volatile innovation variance without trend (VN)*. For ST and VT case, $\kappa = \{0.5, 0.5, -0.5, -0.5\}$ and 0 otherwise. The innovation variance is $\sigma_\zeta^2 = 16$ for all trials in stable volatility conditions (SN and ST). In the variable volatility conditions, the innovation variance is the same in half of the

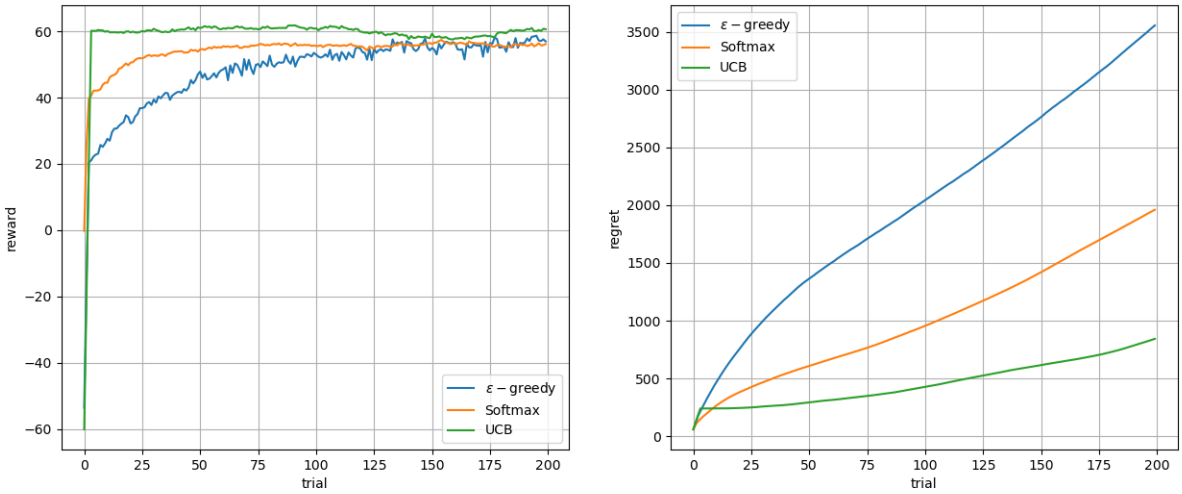
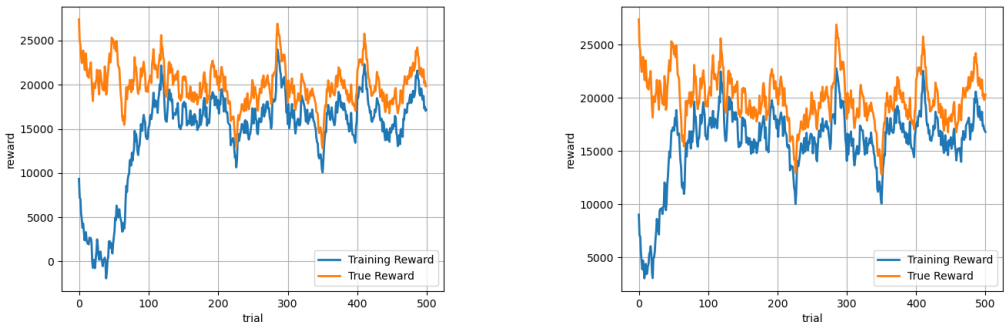


Figure 5: Comparison over different classical RL methods in an environment with high trend and volatility

trials. On trials 51–100 and 151–200, the innovation variance is increased to $\sigma_\zeta^2 = 256$. One example of the resulting rewards in the four conditions is provided in Figure 1. A similar schedule was used by [DOD+06], but it did not include trends or changes in volatility and used only positive rewards.

Epsilon-Greedy: The ϵ-greedy policy is a policy that chooses the best action (i.e., the action associated with the highest value) with probability $1-\epsilon \in (0,1)$ and a random action with probability ϵ. When it chooses the random actions (i.e., with probability ϵ), it chooses them uniformly (i.e., it considers all actions equally good), even though certain actions (even excluding the currently best one) are better than others. In our implementation, we use a decaying ϵ to consider our past experiences more as time progresses. We ran the experiment with different epsilon values - $[0.1, 0.3, 0.5]$ and a decaying epsilon = 1/iteration number. $\epsilon = 0.1$ gave the best results. Thus, we use that for all further plots.

Softmax action selection: Although ε-greedy action selection is an effective and popular means of balancing exploration and exploitation in reinforcement learning, one drawback is that when it explores, it chooses equally among all actions. This means that it is as likely to choose the worst-appearing action as it is to choose the next-to-best action. This may be unsatisfactory in tasks where the worst actions are very bad. The obvious solution is to vary the action probabilities as a graded function of the estimated value. The greedy action is still given the highest selection probability, but all the others are ranked and weighted according to their value estimates. These are called softmax action selection rules. The most common softmax method uses a Boltzmann distribution. It chooses action on the t^{th}



(a) DQN Training (b) PPO Training

Figure 6: Environment with trend and high volatility

play with probability

$$\frac{e^{Q_t(a}/\tau}{\Sigma_{b=0}^{K} e^{Q_t(b)/\tau}}$$

where τ is a positive parameter called the temperature and $Q_t(a)$ is the estimated value of action 'a' at time step 't'. High temperatures cause the actions to be all (nearly) equiprobable. Low temperatures cause a greater difference in selection probability for actions that differ in their value estimates. For different values of $\tau \in \{5, 10, 50\}$, $\tau = 5$ gave the best result. We use this value for all further plots.

Upper Confidence Bound: Rather than performing exploration by simply selecting an arbitrary action chosen with a probability that remains constant, the UCB algorithm changes its exploration-exploitation balance as it gathers more knowledge of the environment. It moves from being primarily focused on exploration, when actions that have been tried the least are preferred, to instead concentrating on exploitation, selecting the action with the highest estimated reward.
With UCB, 'A_t', the action chosen at time step 't', is given by:

$$A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

where $Q_t(a)$ is the estimated value of action a at time step t, $N_t(a)$ is the number of times that action a has been selected before time t and c is a confidence value that controls the level of exploration. Here we choose $c = 20$ out of $\{1, 20, 50\}$

Deep Reinforcement Learning: Our aim will be to train a policy that tries to maximize the discounted, cumulative reward $R_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t$, where R_{t_0} is also known as the return. The discount, γ, should be a constant between 0 and 1 that ensures the sum converges. A lower γ makes rewards from the uncertain far future less critical for our agent than the ones in the near future that it can be reasonably confident about. It also en-

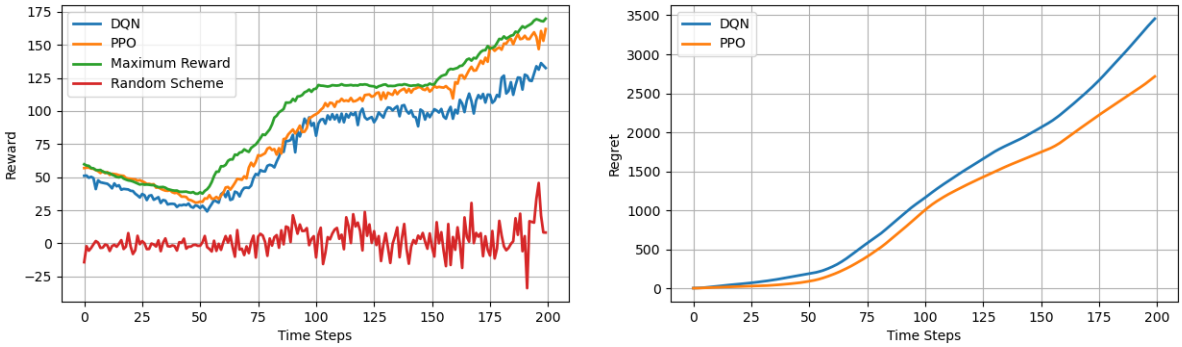


Figure 7: Comparison over different Deep RL methods in an environment with high trend and volatility

courages agents to collect rewards closer in time than equivalent rewards temporally future away.

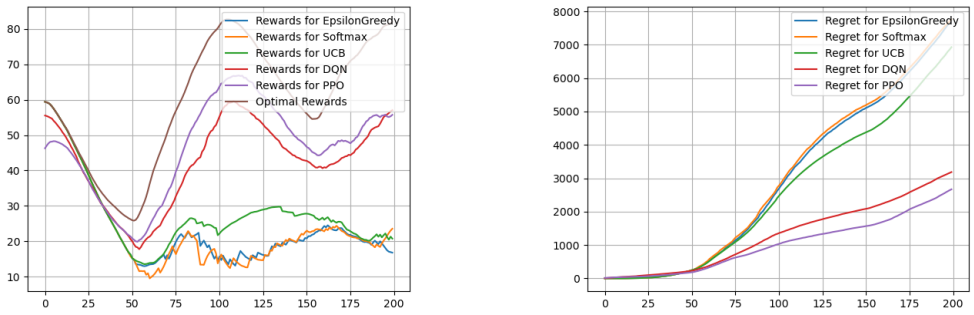
Our key challenge was to model the environment so that we could use deep learning-based methods for the RMAB problem. Looking at the problem statement, it's very intuitive that the agent needs some knowledge about rewards history to predict the following optimal action. So, we define the game of 200 steps as an episode and draw an arm at each step. The state of the environment at a timestep t is defined to be the collection of previous m actions and corresponding observed rewards, i.e. $s_t = [(a_{t-1}, r_{t-1}), \ldots, (a_{t-m}, r_{t-m})]$. The value of m has to be chosen to contain minimum information to predict the next optimal arm. In an extreme case, m can be 200, which means that the agent considers the whole gameplay. We have used this m as a hyperparameter for our model and trained the agent using Deep Q-Networks (**DQN**) and Proximal Policy Optimization (**PPO**). We observe that the agent performed better on increasing the value of m but becomes stagnant after a certain value. We fix $m = 8$ for the further plots.

5. Experiments and Evaluation

This section extensively evaluates all the techniques implemented for solving the RMAB problem. The first part talks in detail about the classical RL algorithms - $\epsilon-$greedy, Softmax, and UCB- and explains why the specific parameters were chosen in section 4. The second part compares the two Deep Reinforcement Learning algorithms, namely DQN and PPO, and illustrates their improvement over classical RL algorithms.

5.1 Classical RL algorithms

For $\epsilon-$greedy, we ran the experiment with pure exploration, pure exploitation, $\epsilon = \{0.1, 0.3, 0.5\}$, and a decaying epsilon. For an environment with trend enabled and volatile innovation variance (high σ_ζ), we get the results as shown in Figure 2. We observe best values for both regret and rewards for $\epsilon = 0.1$, we use it to compare it against all other algorithms. The plots for the other three types of environment configuration as described in section 4 (ST, SN, and VN) are included in section 6.



(a) (b)

Figure 8: Comparing all RL algorithms for the environment with trend and high volatility

For Softmax action selection in an environment with high trend and volatility, it is evident from Figure 3 that $\tau = 5$ gives the best reward and the least regret. We use this parameter in all further plots.

For the UCB algorithm, as shown in Figure 4, $c = 20$ gives a reasonable reward and regret pair.

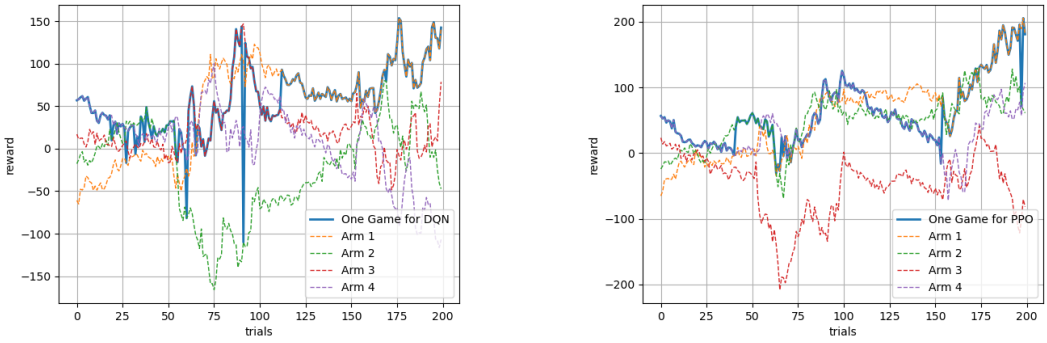
On comparing the three algorithms, it can be easily seen from Figure 5 that UCB performs better than $\epsilon-$greedy and Softmax algorithm. However, these techniques get stuck at some sub-optimal level. We say this because all algorithms have extremely high regret values. We thus move to deep learning-based algorithms to check if we can get better results than this.

5.2 Deep Learning algorithms

We implemented a DQN-based algorithm for solving the RMAB problem by taking ideas from [MKS+13] followed by the PPO approach as defined in [SWD+17]. We defined the input vector as a collection of n consecutive actions and rewards. The agent was able to choose optimal actions in almost all cases for both DQN and PPO. It was also observed that setting $n > 10$ took too long to train, and for $n \leq 4$, the results were suboptimal. Setting $n = 8$ gave the best results. Thus, we fix that hyperparameter for all further plots.

itself. Figure 6a illustrates the training performance of the DQN agent by comparing the mean of obtained rewards with the maximum achievable reward. For the first 100 episodes, we decrease ϵ linearly from 1 to 0.01. We can clearly observe from the figure that as soon as epsilon settles, the DQN performance peaks as it exploits the learned policy, and then the average reward becomes constant.

Figure 6b shows the training performance for the PPO agent. PPO uses parameter space noise as discussed in [PHD+17] and can achieve peak performance much before 100 episodes, unlike DQN. This is because PPO uses clipping while updating the gradients, thus, avoiding abrupt changes and gradually moving towards the optimal policy instead of fluctuating within sub-optimal actions.



(a) DQN (b) PPO

Figure 9: Evolution of the agent in one full game

Figure 7 evaluates the trained DQN and PPO agents on 100 episodes and compares the average. We also compare the performance of our trained agents with the maximum achievable reward and a random scheme in which an action is chosen at random at every time step. It can be seen that PPO outperforms DQN, especially in the later part of the game when trend and volatility take full effect. We also compare the regret observed by each agent. The regret measure of PPO is also better than DQN, and in general, these Deep Reinforcement learning methods perform far better than classical RL methods, as shown in Figure 8.

Figure 9 shows one evolution of rewards associated with all four arms in a full-length game and how the agent chooses the most optimal arm. We observe that as soon as the value of the arm that the agent is following falls, the agent immediately switches to the most optimal arm for both PPO and DQN.

5.3 Remarks

It is clear from the above discussion that deep RL methods can easily outperform classical methods for environments with high stochasticity. The latter, however, is easier to implement and takes significantly less time to train. In the scope of this paper, due to space constraints, we have only considered a volatile environment with a trend (VT). We have chosen the VT environment as it ensures maximum stochasticity in the RMAB setting; thus, it provides an accurate metric for evaluating the agent.

The overall implementation is hosted on GitHub, and the plots corresponding to the other three environment settings can be found in section 6.

6. Conclusion and Future Directions

This paper evaluates different RL techniques to solve the Restless Multi-Armed Bandits problem. We see that deep RL-based methods like DQN and PPO significantly outperform classical algorithms like $\epsilon-$greedy, Softmax, and UCB. However, as the environment is highly stochastic, the agent can never exactly achieve the maximum possible reward.

Treated as an optimal control problem, the optimal solution of the RMABP is known to be computationally intractable. However, we observe that deep RL methods like DQN and PPO can achieve near-optimal performance even in such high-entropy environments.

It can further be inferred that the agent not only learns to choose the optimal action based on the past observed rewards on selecting an action but it also takes the memory matrix into account. The memory matrix allows the agent to get an idea about the nature of each arm, and aids in solving the exploration v/s exploitation problem.

In this paper, we have only considered the single agent setting where a single arm was selected out of the K available arms. This can be extended to a case where n out of K arms are selected at each trial. Further, we can consider a decentralized setting where there are m players who have access to only their own results at any timestep.

Most existing work on RMABP solves the offline setting where the underlying working of the game is known. In our case, too, we have fixed the number of trials to 200; thus, we are trying to solve the problem in an offline setting. We will have to train our model over an infinite horizon if we consider an online setting. It will be interesting to extend this work further to an online setting.

References

[BC18] Vivek S. Borkar and Karan Chadha. A reinforcement learning algorithm for restless bandits. In *2018 Indian Control Conference (ICC)*, pages 89–94, 2018.

[DOD+06] Nathaniel D. Daw, John P. O'Doherty, Peter Dayan, Ben Seymour, and Raymond J. Dolan. Cortical substrates for exploratory decisions in humans. *Nature*, 441(7095):876–879, 2006.

[FNMT19] Jing Fu, Yoni Nazarathy, Sarat Moka, and Peter G. Taylor. Towards q-learning the whittle index for restless bandits. In *2019 Australian & New Zealand Control Conference (ANZCC)*, pages 249–254, 2019.

[KXBT21] Jackson A. Killian, Lily Xu, Arpita Biswas, and Milind Tambe. Restless and uncertain: Robust policies for restless bandits via deep multi-agent reinforcement learning, 2021.

[LZ10] Keqin Liu and Qing Zhao. Indexability of restless bandit problems and optimality of whittle index for dynamic multichannel access. *IEEE Transactions on Information Theory*, 56(11):5547–5567, 2010.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[NM07] José Niño-Mora. Dynamic priority allocation via restless bandit marginal productivity indices. *TOP*, 15(2):161–198, 2007.

[NMV11] José Niño-Mora and Sofía S. Villar. Sensor scheduling for hunting elusive hiding targets via whittle's restless bandit index policy. In *International Conference*

on NETwork Games, Control and Optimization (NetGCooP 2011), pages 1–8, 2011.

[PHD+17] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration, 2017.

[PT94] C.H. Papadimitriou and J.N. Tsitsiklis. The complexity of optimal queueing network control. In *Proceedings of IEEE 9th Annual Conference on Structure in Complexity Theory*, pages 318–322, 1994.

[SK15] Maarten Speekenbrink and Emmanouil Konstantinidis. Uncertainty and exploration in a restless bandit problem. *Topics in Cognitive Science*, 7(2):351–367, 2015.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[Whi88] P. Whittle. Restless bandits: activity allocation in a changing world. *Journal of Applied Probability*, 25(A):287–298, 1988.

[WHL20] Siwei Wang, Longbo Huang, and John C. S. Lui. Restless-ucb, an efficient and low-complexity algorithm for online restless bandits, 2020.

[WW90] Richard R. Weber and Gideon Weiss. On an index policy for restless bandits. *Journal of Applied Probability*, 27(3):637–648, 1990.

Appendix

A. Environment without trend and a stable innovation variance (SN)

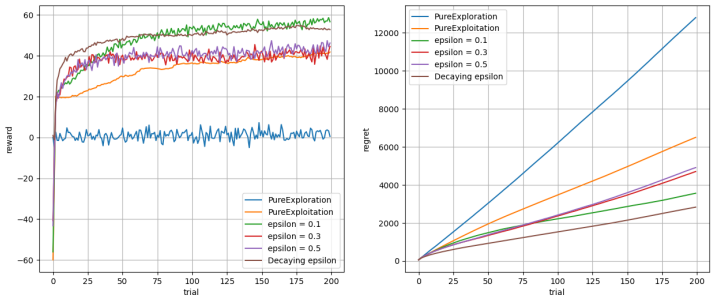


Figure 10: Comparison over different epsilon values for epsilon-greedy algorithm

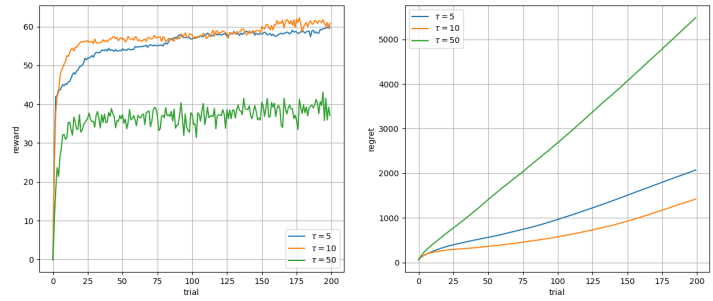


Figure 11: Comparison over different τ values for Softmax selection

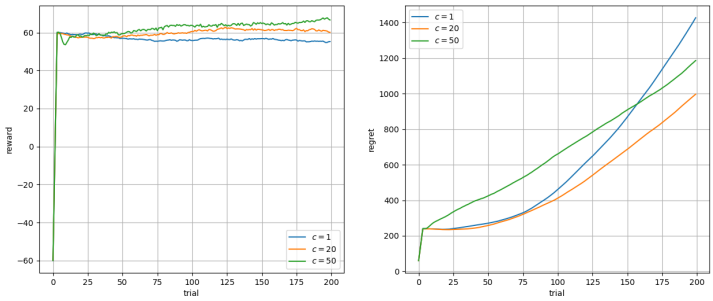
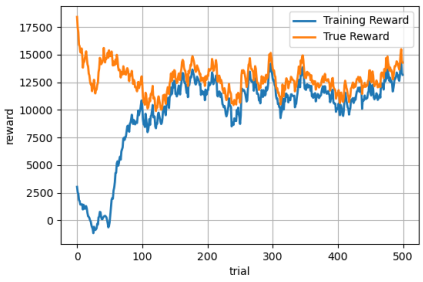
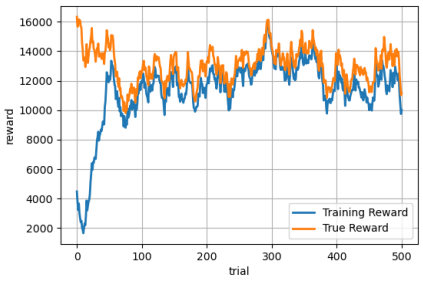


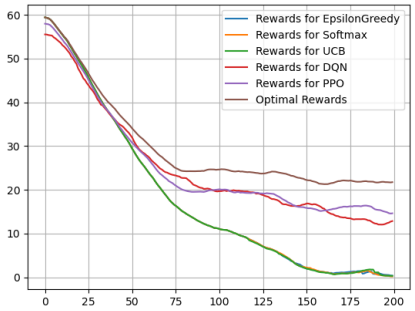
Figure 12: Comparison over different c values for UCB



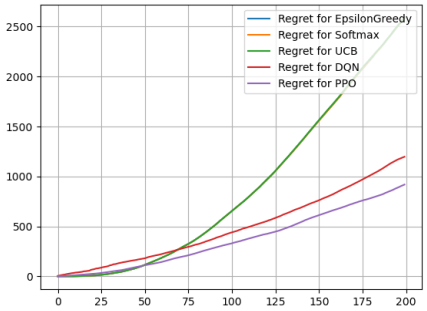
(a) DQN Training



(b) PPO Training

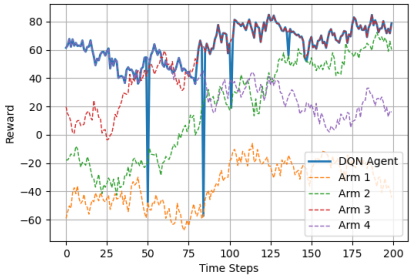


(c)

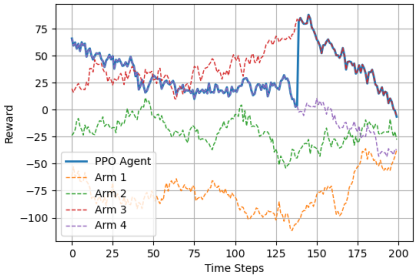


(d)

Figure 13: Comparing all RL algorithms



(a) DQN



(b) PPO

Figure 14: Evolution of the agent in one full game

B. Environment with trend and a stable innovation variance (ST)

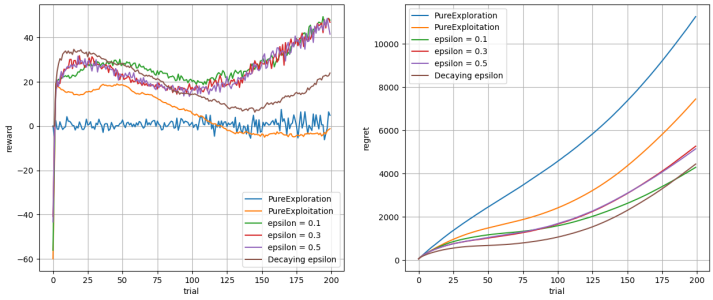


Figure 15: Comparison over different epsilon values for epsilon-greedy algorithm

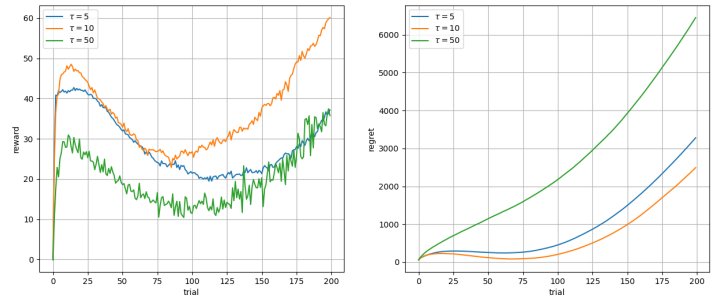


Figure 16: Comparison over different τ values for Softmax selection

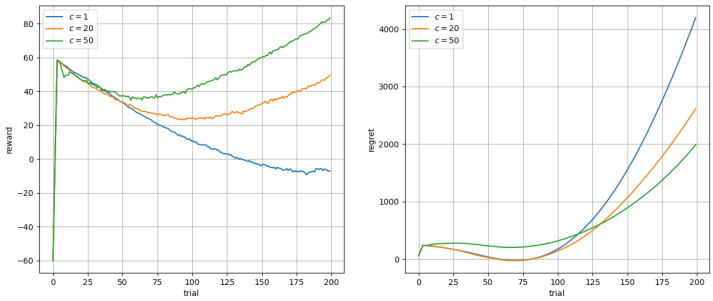
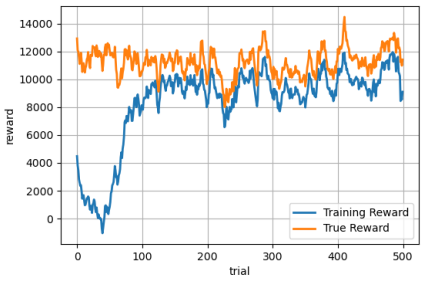
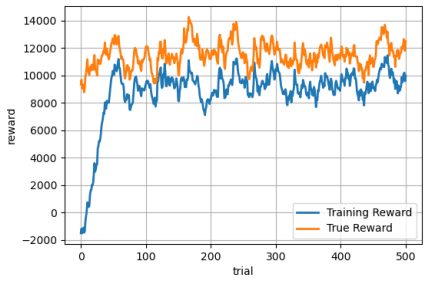


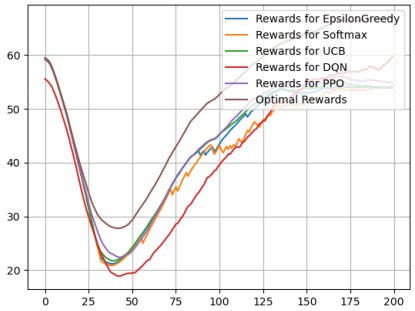
Figure 17: Comparison over different c values for UCB



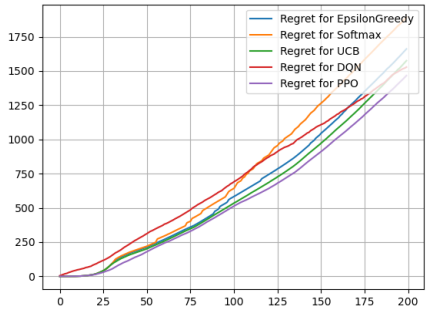
(a) DQN Training



(b) PPO Training

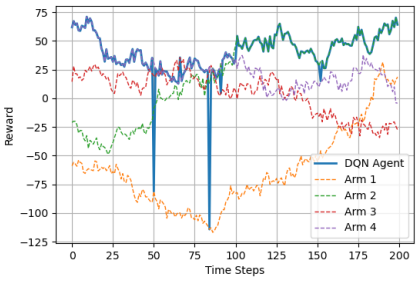


(c)

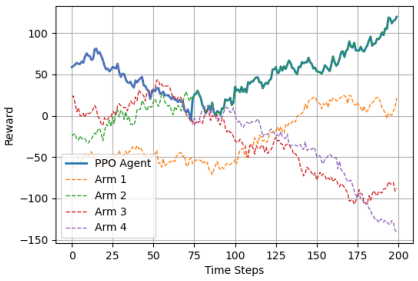


(d)

Figure 18: Comparing all RL algorithms



(a) DQN



(b) PPO

Figure 19: Evolution of the agent in one full game

C. Environment without trend and volatile innovation variance (VN)

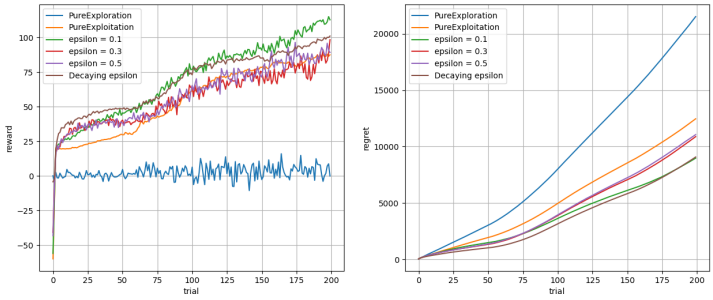


Figure 20: Comparison over different epsilon values for epsilon-greedy algorithm

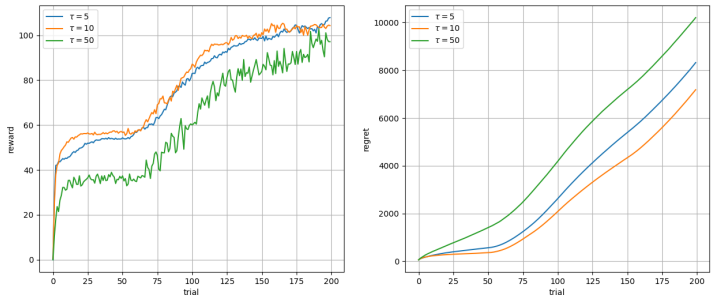


Figure 21: Comparison over different τ values for Softmax selection

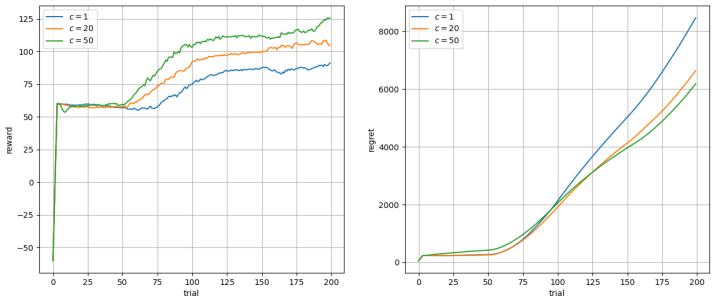
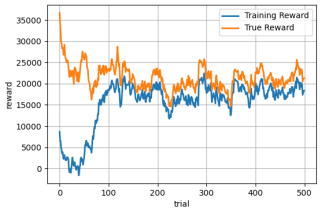
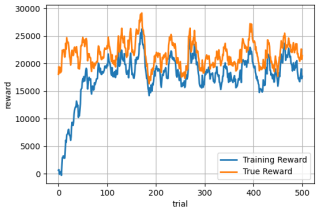


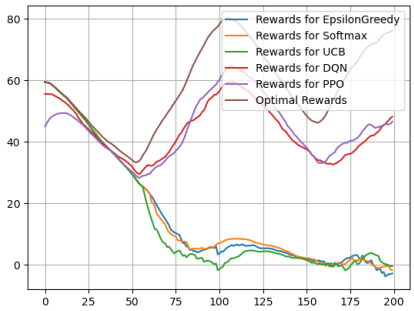
Figure 22: Comparison over different c values for UCB



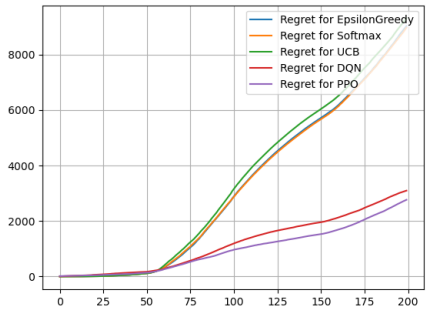
(a) DQN Training



(b) PPO Training

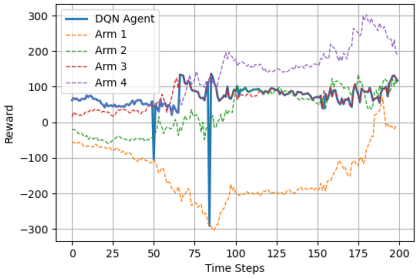


(c)

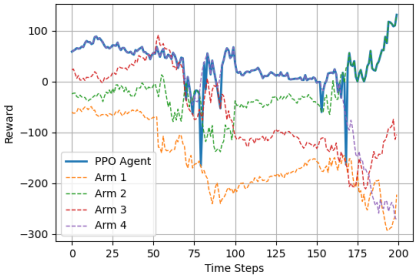


(d)

Figure 23: Comparing all RL algorithms



(a) DQN



(b) PPO

Figure 24: Evolution of the agent in one full game