Winter in Data Science 2022
Hands-on Reinforcement Learning

Mentored by Jujhaar Singh and Siddhant Midha

Shravya Suresh
Dept. of Engineering Physics
IIT Bombay
210260046@iitb.ac.in

***Abstract*—Reinforcement Learning is a unique branch of Machine Learning that helps us create intelligent agents capable of performing various tasks. Reinforcement learning problems involve learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The concept differs from supervised and unsupervised learning, in the sense that it uses training information that evaluates the actions taken rather than information that instructs by giving correct actions. Such evaluative feedback is the basis of methods for function optimization, including evolutionary methods. Through this project, the following topics are explored: the fundamentals of Bandits, Markov Decision Processes, Dynamic Programming (Policy and Value iteration), Prediction and Control methods, and many associated algorithms (ϵ-greedy, Thomas Sampling, Monte Carlo, Q-Learning, SARSA)**

I. Introduction

Reinforcement Learning (abbreviated as RL) is a machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. It is employed by various software and machines to find the best possible behavior or path it should take in a specific situation - to maximise the reward in the given situation. One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best.

II. Concepts Covered

A. Multi-Arm Bandits

Consider this situation: a slot machine with n arms is presented, and each arm of the machine has its own probability of success and failure. Pulling any of the arms yields a stochastic '*reward*' of 1 for success or 0 for failure. By sequentially pulling each arm of the machine, the aim is to maximise this reward.

This is the underlying concept behind n-armed Bandits. In the n-armed bandit problem, each action has an expected or mean reward given that it is selected. This is known as the '*value*' of that action. Knowing the value of an action, one can easily solve the n-arm bandit problem by constantly selecting the action with the highest value. However, a realistic situation would be when the exact values of each action are not known, but they can be estimated.

The action with the highest value estimate is known as the '*greedy action*'. Selecting the greedy action means to '*exploit*' the current knowledge of the values of the actions. Selecting any other action means to be '*exploring*', since this helps to improve the estimate of the value of the non-greedy action. There is always a trade-off between exploitation and exploration. Exploitation is the direct way to maximize the expected reward, but exploration sometimes can produce a greater total reward in the long run.

Some useful algorithms to maximise the reward are:

1) ϵ-greedy Algorithm: The epsilon-greedy agent is designed to give an ϵ chance (say for example 10%) towards exploring bandits randomly at any state, and acts greedily on its current 'best' bandit value estimate for all other times. This mechanism can help the agent focus on its currently most "successful" bandits, and the exploratory mechanism gives the agent to explore for better bandits that might be out there.

This can be represented as follows, for an action a:

A_t = argmax $Q_t(a)$, with probability = 1- ϵ
A_t = any choice of $Q_t(a)$ from all actions,
with probability = ϵ

Fig. 1. plots the performance of the ϵ-greedy algorithm by comparing different values of ϵ on a 10-armed testbed, averaged over 2000 tasks. The structure at the beginning of these curves depends on how actions are selected when multiple actions have the same maximum action value. To do away with any ties or overlapping graphs, a very small degree of randomness is introduced to each of the initial action values, so that the ties or overlaps effectively don't occur.

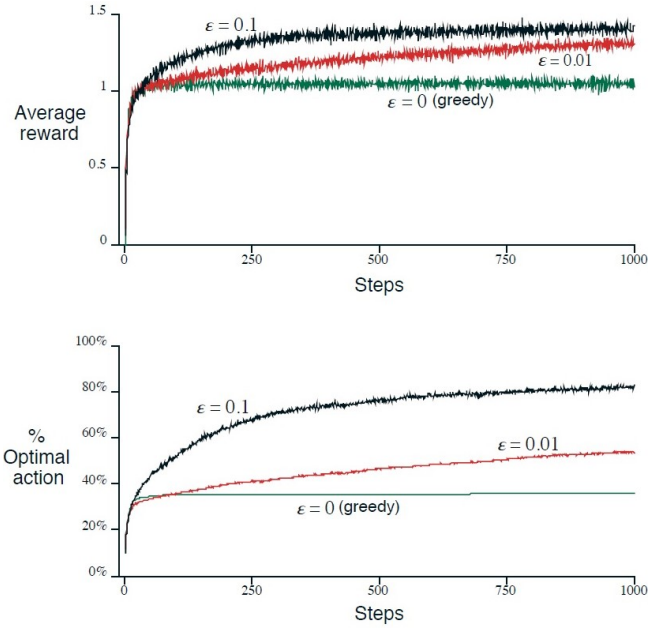

Fig. 1. Average performance of ϵ-greedy action-value methods on a 10-armed testbed, averaged over 2000 tasks.

An advantage of the ϵ-greedy method is with respect to the number of ta that with increase in the number of tasks upto the limiting value, every action will be eventually sampled an infinite number of times. The more the number of tasks, the greater the variance of the rewards, and hence the more the noise. With noisier rewards it takes more exploration to find the optimal action, and hence the ϵ-greedy method would fare even better in comparison to the greedy method.

2) Upper Confidence-Bound Action Selection: Upper confidence-bound (UCB) action selection is built upon the concept that the square-root term is a measure of the uncertainty or variance in the estimate of the action value. In mathematical terms:

$$A_t = \text{argmax } [Q_t(a) + c\sqrt{\frac{lnt}{N_t(a)}}]$$

where lnt denotes the natural logarithm of t, $N_t(a)$ represents the number of times action a has been tried by the t^{th} timestep, and the number c > 0 controls the degree of exploration. If $N_t(a) = 0$, then a is considered to be a maximizing action.

If $N_t(a) = 0$, then a is considered to be a maximizing action. Each time a is selected, the uncertainty is presumably reduced since $N_t(a)$ is incremented and the uncertainty term is decreased. On the other hand, each time an action other than a is selected, t is increased, and subsequently the uncertainty estimate is increased. Albeit the increase gets smaller over time, but it is unbounded.

Figure 2 shows the average performance of UCB action selection on a 10-armed testbed. UCB does perform well, but is harder than the ϵ-greedy method to extend beyond bandits to the more general reinforcement learning settings, such as

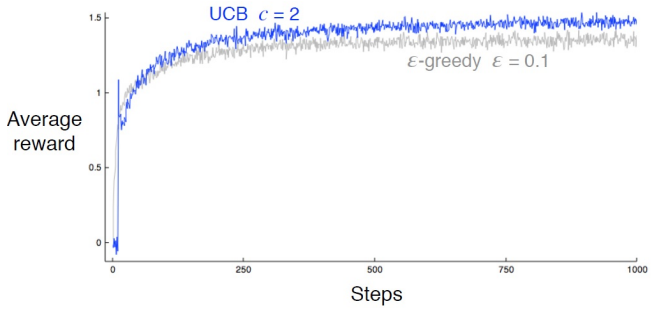

Fig. 2. Average performance of UCB action selection on a 10-armed testbed

nonstationary problems and large state spaces.

3) Thomas Sampling: Thompson Sampling is an algorithm that employs Bayesian probability to the exploration-exploitation trade-off. Here, the optimal arm would be considered as one whose reward is of the form

$$A_t = \text{argmax } N_t(a)]$$

for an action a, where $N_t(a)$ is computed using the beta function on a list of successes and failures of the arms of the bandit. Bayes' Theorem is used to update the respective distributions of the arms, hence maintaining the efficiency of the algorithm.

Thompson Sampling tends to reduce the magnitude of the search as we get more and more information, keeping in line with the desirable trade-off where we want more information in fewer searches. While this algorithm is hence more exploratory-oriented, it exhausts more computational power in its execution.

B. Markov Decision Processes

Markov Decision Processes(MDPs) are mathematical frameworks to describe an environment in Reinforcement Learning. An MDP consists of a set of finite environment states S, a set of possible actions $A(s)$ in each state, a real valued reward function $R(s)$ and a transition model $P(s', s|a)$. Memoryless in nature, Markov Decision Processes predict the next state based only on the current state and not the previous one.

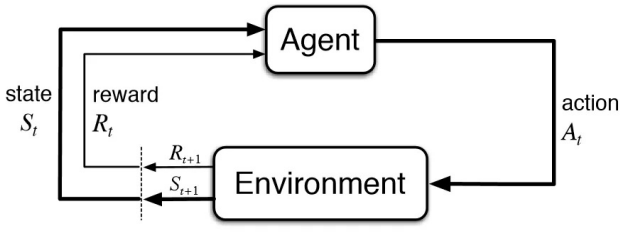

Fig. 3. The agent—environment interaction in Reinforcement Learning

Figure 3 shows the basic structure of the Agent-Environment Interface. The *agent* — the decision-maker — interacts with its *environment* — everything outside it —

through a series of chosen actions and the resulting rewards that the agent tries to maximise over time as it iterates through a number of $tasks$ — one instance of the reinforcement learning problem. This subsequent process leads to changes in the $state$ of the environment.

At each time step, the agent implements a mapping from given states to the probabilities of selecting each possible action. This mapping is known as the agent's $policy$ and is denoted as π_t, where $\pi_t(a|s)$ is the probability that A_t = a if S_t = s. So in technical terms, Reinforcement Learning methods specify how the agent changes its policy as a result of its experience by interacting with its environment's state. The agent's goal is to maximize the total cumulative reward it receives over a long run of time steps.

While talking about expected rewards, another term to consider is the *expected return*, which is generalised as some function of the reward sequence. If the sequence of rewards after time step t is denoted as R_{t+1}, R_{t+2}, R_{t+3}... and so on, the expected return is denoted as

$$G_t = f(R_{t+1}, R_{t+2}, R_{t+3} \ldots)$$

For example, G_t can be denoted as the sum of rewards uptil the final time step T:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots + R_T$$

Reinforcement Learning tasks are generally of two types - episodic and continuing. Episodic tasks generally end in a special $terminal$ state, while continuing tasks go on without a well-defined limit; $T \to \infty$.

A policy can be optimally defined with its State-Value function, which can be expressed in terms of the Bellman Equation for states clubbed under S:

$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r}(s', r|s, a)[r + \gamma v_\pi(s')$, for all $s \in$ S

A policy is optimally compared to other policies by comparing their state values:

$$\pi \geq \pi' \text{ iff } v_\pi(s) \geq v_{\pi'}(s)$$

Thus, the optimal state-value function can be defined as the state-value function yielding the maximum expected rewards for all states. And based on this, the Bellman Optimality function is defined:

$$v_*(s) = \max \sum_{s',r}(s', r|s, a)[r + \gamma v_*(s'), \text{ for } a \in A$$

C. Dynamic Programming

Dynamic Programming refers to a collection of algorithms that can be used to compute optimal policies through an improvised recursive manner, given a Markov decision process (MDP) as the environment. The key idea of dynamic programming, and extendable to reinforcement learning, is the use of value functions to organize and structure the search for good, optimal policies. Two major methods of achieving this are value iteration and policy iteration.

1) Value Iteration: As covered earlier, value iteration implies that a policy reaches its optimal state $v_\pi(s) = v_*(s)$ if and only if for any state s' reachable from s, π achieves the optimal value from state s';

$$v_\pi(s') = v_*(s')$$

2) Policy Iteration: Policy iteration involves directly evaluating a policy and then improving it. To evaluate a policy, Bellman's expectation equation is iteratively applied to evaluate the state value function for all states in an MDP for a given policy. To improve the policy, a greedy action is directly picked based on the true value function:

$$\pi' = \text{greedy } (v_\pi)$$

This way, probabilities are assigned to the actions based on the maximum value of the state-value function of the adjacent states.

D. Prediction and Control Methods

Prediction and Control are two major tasks in reinforcement learning. Prediction involves approximating how well a particular policy performs on the MDP. For a given learning algorithm L, at each time step t, L provides an estimated state value v_t. The prediction problem aims to construct an L such that

$$\lim_{t\to\infty} v_t = v_\pi$$

While in the control problem, the goal is to find the optimal policy for the given MDP. Actions are selected by the learning algorithm L (agent), and the set of next states and rewards are chosen by the MDP (environment). The control problem strives to construct L such that

$$\lim_{H\to\infty} \frac{1}{H}(\sum_{t=0}^{H-1} \Pr[(a^t \sim L(h^t)) \text{ is optimal for } s^T]) = 1$$

E. Algorithms to implement Prediction and Control in MDPs

1) Monte Carlo Methods: Monte Carlo methods are ways of solving the reinforcement learning problem based on averaging sample returns. These methods require only experience—sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.

2) Q-Learning: Q-learning is a values-based learning algorithm which is used to find the optimal action-selection policy using a Q function. The goal is to maximize the value function Q, which in turn helps to maximize the expected reward by selecting the best of all possible actions.

3) SARSA: SARSA (State Action Reward State Action) algorithm is a slight variation of the popular Q-Learning algorithm. While Q-Learning technique is an Off Policy technique and uses the greedy approach to learn the Q-value, SARSA technique, on the other hand, is an On Policy and uses the action performed by the current policy to learn the Q-value.

4) Expected-SARSA: Expected Sarsa is like Q-learning but instead of taking the maximum over next state-action pairs, we use the expected value, taking into account how likely each action is under the current policy.

ACKNOWLEDGMENT

I am grateful to the mentors, Jujhaar and Siddhant, for their expert guidance throughout the course of this project. The structure and sequence of tasks to be completed for this project has been very thoroughly planned and implemented, and I found this project very helpful in learning about a new umbrella topic of Machine Learning. I also thank the Analytics Club, UGAC for continuing to organise Winter in Data Science, which provides a great platform for us to not only pursue but also mentor projects in various fields of Data Science and Machine Learning.

REFERENCES

[1] Sutton, R. S., Bach, F., amp; Barto, A. G. (2018). Reinforcement learning: An introduction. MIT Press Ltd.

[2] Bhatt, S. (2019, April 19). Reinforcement learning 101. Medium. Retrieved January 28, 2023, from https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292

[3] CS 747: Foundations of intelligent and learning agents(autumn 2022). CS 747: Autumn 2022. (n.d.). Retrieved January 29, 2023, from https://www.cse.iitb.ac.in/ shivaram/teaching/old/cs747-a2022/index.html