Winter in Data Science - Hands On Reinforcement Learning

Madhav Gupta(21*D*070043)

***Abstract*— Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence. Rather than directly theorizing about how people or animals learn, we can explore idealized learning situations and evaluate the effectiveness of various learning methods. That is, we can adopt the perspective of an artificial intelligence researcher or engineer. We can explore designs for machines that are effective in solving learning problems of scientific or economic interest, evaluating the designs through mathematical analysis or computational experiments. The approach we explore, called reinforcement learning, is much more focused on goal-directed learning from interaction than are other approaches to machine learning.**

I. INTRODUCTION

Reinforcement learning problems involve learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. In an essential way they are closed-loop problems because the learning system's actions influence its later inputs. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These three characteristics—being closed-loop in an essential way, not having direct instructions as to what actions to take, and where the consequences of actions, including reward signals, play out over extended time periods—are the three most important distinguishing features of reinforcement learning problems.

II.

A. An n-Armed Bandit Problem

In the n-armed bandit problem, each action has an expected or mean reward given that action is selected which is called the value of that action. If we maintain estimates of the action values, then at any time step there is at least one action whose estimated value is greatest. We call this a greedy action. Exploitation is the right thing to do to maximize the expected reward on the one step, but in the long run, exploration may produce a greater total reward, whether it is better to explore or exploit depends in a complex way on the precise values of the estimates, uncertainties, and the number of remaining steps.

B. Action-Value Methods

One way to estimate the action value is by averaging the rewards actually received when the action was selected. If by the t^{th} time step action a has been chosen $N_t(a)$ times prior to t, yielding rewards $R_1, R_2, ..., R_{N_t}(a)$, then its value is estimated to be $\frac{R_1+R_2+...+R_{N_t}(a)}{N_t(a)}$.

Greedy selection method is where we select the action with highest estimated value. In epsilon-greedy method, epsilon refers to the probability of randomly selecting non-greedy actions instead of the greedy one. If we compare the greedy method improved slightly faster than the other methods at the very beginning, but then leveled off at a lower level. The ε-greedy methods eventually perform better because they continue to explore, and to improve their chances of recognizing the optimal action. For example, suppose the reward variance had been larger, say 10 instead of 1. With noisier rewards it takes more exploration to find the optimal action, and ε greedy methods should fare even better relative to the greedy method.

C. Incremental Implementation

Let Q_k denote the estimate for its k^{th} reward, that is, the average of its first k-1 rewards. Given this average and a k^{th} reward for the action, R_k, then the average of all k rewards can be computed by $Q_{k+1} = Q_k + \frac{1}{k}[R_k - Q_k]$.

NewEst. <– OldEst. + StepSize[Traget - OldEst.]

where stepSize in this case is $\frac{1}{k}$. This method reduces computation required to calculate averages.

D. Optimistic Initial Values

The methods used so far are all biased by the initial estimate $Q_1(a)$. For the sample average method bias disappears when all actions are selected at least once, but with constant step-size the bias is permanent. If we

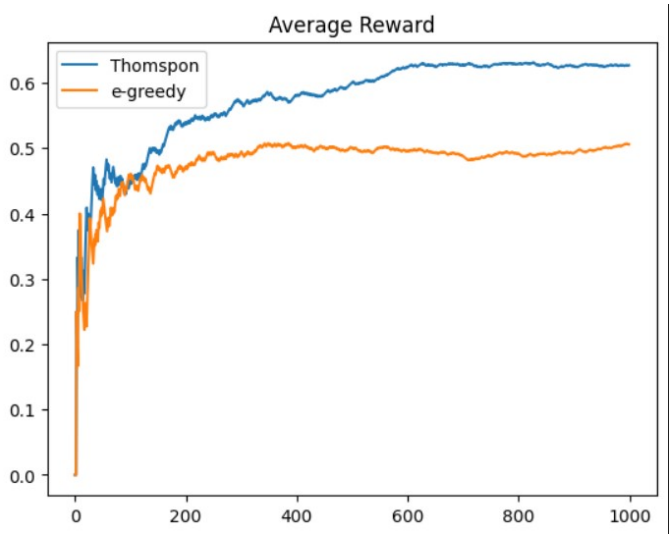

Fig. 1. Comparison of Average rewards for Thompson and e-greedy algorithms .

select a widely optimistic value say +6 in a distribution with mean 0 and variance 1, the reward would be less than starting estimates and system would do fair amount exploration even if greedy actions are selected all the time. It is not well suited for non-stationary problems as its drive for exploration is inherently temporary.

E. Upper-Confidence-Bound Action Selection

Exploration is needed because the estimates of the action values are uncertain. It would beneficial to select among the non-greedy actions according to their potential for actually being optimal, taking into account both how close their estimates are to being maximal and the uncertainties in those estimates.

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

The square-root term is a measure of the uncertainty or variance in the estimate of a's value. As $N_t(a)$ increases the uncertainty decreases and as t increases uncertainty estimate increases. Parameter c determines the confidence level. As number of steps increases there would be lower selection frequency for actions with a lower value estimate or that have been already selected more times.

III. Markov Decision Problems

Markov decision problems (MDPs) are a mathematical framework for modeling decision-making problems in which the future outcomes are uncertain and depend on the current state and the actions taken. An MDP is defined by a tuple (S, A, P, R, γ) where:

- S is the set of states, which represent the different possible configurations of the system.
- A is the set of actions, which are the different choices that can be made by the decision maker.
- P is the transition probability function, which describes the probability of transitioning from one state to another after taking a specific action.
- R is the reward function, which assigns a real-valued reward to each state-action pair (s, a).
- γ is the discount factor, a value between 0 and 1 that determines the importance of future rewards versus immediate rewards.

A. Policy & Value Function

A policy denoted by π, is a mapping from states to actions. It represents the decision maker's strategy for selecting actions in different states. A policy can be deterministic, where a specific action is chosen for each state, or it can be stochastic, where a probability distribution over actions is defined for each state.

The value function, denoted by V(s) or Q(s,a), is a measure of the expected total reward that can be obtained by following a particular policy starting from a given state or a given state-action pair. The value function captures the long-term implications of the current state and the current policy.

B. State Value and Bellman Equations

A state value is a measure of the long-term expected reward of an agent in a given state. Bellman's equation is a fundamental equation used to update the state values in a dynamic programming algorithm.

$$V(s) = \max(a)\ [R(s,a) + \gamma * V(s')]$$

where max(a) is the maximum expected future reward of taking action a in state s, where a is one of the actions available in state s.

The difference between Value function Q(s,a) and state value V(s) is that in the case of Q function, it maps the state-action pairs to expected long-term rewards, whereas the V function maps only the states.

C. Reward & Transition Functions

The reward for (s,a,s') is taken as a random variable bounded in [-R_{max}, R_{max}], with expectation R(s,a,s') R & T are combined into a single function P{ s',r|s,a} for s' $\in$ S, r$\in$ [$-R_{max}$, R_{max}]. We can also minimize cost rather than maximize reward

D. Episodic Tasks

- Episodic tasks have a special sink/terminal state S_T from which there are no outgoing transitions on rewards.
- From every non-terminal state and for every policy, there is a non-zero probability of reaching the terminal state in a finite number of steps.
- Trajectories or episodes almost surely terminate after a finite number of steps.
- These are useful in many applications like Controlling a helicopter, Winning at chess, Preventing Forest Fires, etc.

E. Banach's fixed-point theorem

- A complete normed vector space i.e. in which every cauchy sequence has a limit in X is called a Banach space.
- Banach's fixed-point theorem is a fundamental result in the theory of metric spaces that states that every contraction mapping on a complete metric space has a unique fixed point.
- Let (X, $\|.\|$) be a Banach space, and let Z: X $\rightarrow$ X be a contraction mapping with contraction factor l. Then: Z has a unique fixed point $x^* \in X$
  For $x^* \in X$, m$\geq$0: $\|Z^m x - x^*\| \leq l^m \|x - x^*\|$

F. Bellman Optimality Operator

The Bellman Optimality Operator is defined as the maximum value of the expected value of the next state and reward, given the current state and action. It helps to find the best possible action to take in a given state, based on the knowledge of the entire MDP.

$$(B^*(F))(s) \stackrel{\text{def}}{=} \max_{a \in A} \sum_{s' \in S} T(s, a, s')R(s,a,s') + \gamma F(s').$$

G. Value Iteration

The algorithm starts with an initial estimate of the value function and then repeatedly applies the Bellman Optimality Operator, which updates the value function based on the expected reward of the next state and the current value function. The process continues until the value function no longer changes, at which point the algorithm has converged to the optimal value function. Once the optimal value function is found, the corresponding optimal policy can be extracted by taking the action that maximizes the value function in each state.

H. Action Value Function

- $Q^\pi(s, a) \stackrel{\text{def}}{=} E[r^0 + \gamma r^1 + \gamma^2 r^2 + \gamma^3 r^3 + ... \mid s^0 = s; a^0 = a; a^t = \pi(s^t) \text{ for } t \geq 1]$.
- $Q^\pi(s, a)$ is the expected long-term reward from starting at state s, taking action a at t=0, and following policy π at t$\geq$1.
- Q^π:S X A $\rightarrow$ R is called the action value function.
- The Q-function can be used to find the optimal policy by selecting the action that has the highest Q-value for each state.

I. Policy Iteration

1) Policy Improvement Theorem: The Policy Improvement Theorem is a fundamental result in the theory of Markov Decision Processes (MDPs) that states that any locally optimal policy can be improved by acting greedily with respect to the current value function. The theorem guarantees that if we have a policy that is at least as good as the current optimal policy, we can make it better by acting greedily with respect to the current value function.
If we know the value function of the current policy, we can improve it by choosing actions that have the highest value in each state. This is done by using the Bellman optimality equation to compute the value function of the improved policy.
$IA(\pi,s) = \{ a \in A : Q_\pi(s,a) > V_\pi(s) \}$
$IS(\pi) = \{ s \in S : \mid IA(\pi,s) \mid \geq 1 \}$
Mathematically it states that:

- If $IS(\pi) = \phi$, then π is optimal, else
- if π' is obtained by policy improvement on π, then $\pi' > \pi$

2) Implication of Policy Improvement Theorem:

- It leads to the convergence of iterative methods: Algorithms such as Policy Iteration and Q-learning use the Policy Improvement Theorem as the foundation. The theorem guarantees that iteratively improving the policy will converge to the optimal policy.
- It can be used to identify the states that need improvement: The set of states where there is at least one action that is better than the current policy, $IS(\pi)$, can be used to identify the states that need improvement.
- It is useful in model-based and model-free methods: The theorem is useful in both model-based and model-free methods, as it can be used to improve the policy regardless of whether the underlying dynamics of the MDP are known or not.

3) Policy Iteration Algorithm:

- Initialize a policy π to an arbitrary policy.

- While there are still improvable states in π, as defined by the IS(π) function, repeat the following steps:
- Apply the Policy Improvement(π) function to the current policy π, resulting in a new policy π'.
- Update the current policy π to the new policy π'.
- Repeat steps 2-4 until the policy π no longer has any improvable states.
- Return the final policy π as the optimal policy for the MDP.

$\pi \leftarrow$ Arbitrary Policy.
While π has improvable states:
$\pi' \leftarrow$ PolicyImprovement(π).
$\pi \leftarrow \pi'$.
Return π.

J. General case of policies

- In principle, an agent can follow a policy λ that maps every possible history s_0, a_0, r_0, s_1, a_1, r_1,..., s_t for $t \geq 0$ to a probability distribution over A.
- Let Λ be the set of such policies λ (which are in general non-Markovian, non-stationary, and stochastic).
- Considering only Π, the set of all policies $\pi : S \rightarrow A$ (which are Markovian, stationary, and deterministic). We can Observe that $\Pi \subset \Lambda$.
- From the previous learnings we already know that $\pi^* \in \Lambda$ such that for all $\pi \in \Lambda$, $\pi^* \geq \pi$.
- But there does not exist $\lambda \in \Lambda \setminus \Pi$ such that $\neg (\pi^* \geq \lambda)$

K. History-dependent & stochastic policies

- History refers to the sequence of states and actions that an agent has encountered up to a certain point.
- Stochasticity refers to the fact that the state of the environment and the outcomes of actions may be uncertain or unpredictable.
- History and Stochasticity can interact in complex ways to affect the decision-making process.
- Non-Markovian policies, which take history into account, may be more effective in highly stochastic environments. On the other hand, Markovian policies, which only consider the current state, may be more efficient in environments with low stochasticity.
- History and stochasticity can help if the agent is unable to sense state perfectly. Such a situation arises in an abstraction called the Partially Observable MDP (POMDP).
- Optimal policies for the finite horizon reward setting are in general non-stationary (time-dependent).
- Optimal strategies in many types of multi-player games are in general stochastic ("mixed") because the next state depends on all the players' actions, but each player chooses only their own.

IV. Reinforcement learning problem: Prediction and Control

- A learning algorithm L is a mapping from the set of all histories to the set of all (probability distributions over) arms.
- Actions are selected by the learning algorithm (agent); next states and rewards by the MDP (environment).
- The control problem is about finding the best policy that maximizes the long-term cumulative reward.
- The prediction problem is about estimating the value function, which is used to evaluate the quality of a policy.
- The control problem is typically solved using techniques such as Q-learning, SARSA, and actor-critic methods, which are model-free methods that do not require knowledge of the dynamics of the environment.
- The prediction problem is typically solved using techniques such as dynamic programming and Monte Carlo methods, which are model-based methods that require knowledge of the dynamics of the environment.

A. Some Assumptions

1) Irreducibility:

- An MDP is considered to be irreducible if it is possible to reach any state in the system from any other state.
- There are no "absorbing states" or "communication classes" that can't be left once entered. This means that the system is "fully connected" because there is a non-zero probability of transitioning from any state to any other state.
- The irreducibility assumption is often used to analyze the behavior of Markov chains, which are a special type of MDP where the agent can't change the environment.

2) Aperiodicity:

- For $s \in S$, $t \geq 1$, let X(s,t) be the set of all states s such that there is a non-zero probability

of reaching s' in exactly t steps by starting at s and following π.
- For s $\in$ S, let Y(s) be the set of all t $\geq$ 1 such that s $\in$ X(s,t); let p(s) = gcd(Y(s)).
- MDP is aperiodic under π if for all s $\in$ S: p(s) = 1.

3) Ergodicity:

- An MDP that is irreducible and aperiodic is called an ergodic MDP.
- Ergodicity refers to the property that a system eventually reaches a steady state distribution of states, regardless of the initial state.
- Ergodicity is important in MDPs because it allows for the use of methods such as value iteration and policy iteration to find the optimal policy.

V. CONCLUSIONS

Reinforcement Learning (RL) is a powerful tool for solving a wide range of problems in artificial intelligence, including the multi-armed bandit problem. The experiments showed that the agent was able to learn the optimal action to take over time, by trying different actions and learning from the rewards it received in the multi-arm bandit problem. This highlights the effectiveness of RL in solving problems where the reward function is not known in advance and needs to be learned through trial and error. In this project, we also saw the use of value-based methods, specifically value iteration and policy iteration, for solving RL problems. Value iteration is a dynamic programming algorithm that iteratively updates the value function of an agent's policy, while policy iteration is a method that interleaves policy evaluation and improvement steps. Overall, Reinforcement Learning is a promising field with many exciting opportunities for future research and development.

ACKNOWLEDGMENT

The author would like to thank, his mentors for this project Jujhaar Singh and Siddhant Midha for helping him throughout the project

REFERENCES

[1] Reinforcement Learning: An Introduction. Second edition, in progress. Richard S. Sutton and Andrew G. Barto c 2014, 2015. A Bradford Book. The MIT Press.

[2] CS 747: Foundations of Intelligent and Learning Agents Lecture Slides