Retro Learning Week-1 Report

Maloth Vishal Nayak

1 Multi-Armed Bandits Problems

1.1 Definition

The multi-armed bandit problem aims to find a sequence of actions that will maximize the expected total reward over some time period.

1.2 Example

A doctor wants to find the most effective treatment for patients using three possible treatments (actions). Each time a treatment is applied, a different health outcome is observed, quantified by a numerical value, such as the percentage of recovery or a measure of virus presence. This variability occurs due to factors like patient differences and individual reactions to the same treatment.

For each treatment, if enough data is collected, a probability distribution can be constructed to quantify its effectiveness. The outcome value is called the reward, where a higher reward indicates a more effective treatment for the patient.

If the data is known, the most "rewarding" action is the action with the highest mean, as it gives the highest average reward. Since this is not known beforehand, a solution is required.

1.3 Solution to the Multi-Armed Bandit Problem

1.3.1 Terms and Definitions

- Number of arms is denoted by N.
- Action is denoted by a_i.
- Action selected at time step k is denoted by A_k.
- Reward at step k is denoted by R_k.
- The expected or mean reward is $v_i = v(a_i) = E[R_k \mid A_k = a_i]$.

We have to design an algorithm that will estimate the most appropriate actions to maximize the sum of rewards over time. For this purpose, we use the action-value method.

1.3.2 Action-Value Method

This method estimates the values of actions to maximize the rewards.

$$\hat{v}_{i,k} = \hat{v}_k(a_i) = \frac{\sum_{j=1}^{k-1} R_{i,j}}{n_{i,k}}$$

Where $R_{i,j} = 0$ if the action a_i is not selected at time step j, for $j = 1, 2, \ldots, k-1$.

If $n_{i,k}$ is zero, then $\hat{v}_{i,k}$ is set to zero to avoid division by zero issues.

Further simplifying, we get:

$$\hat{v}_k = \hat{v}_{k-1} + \frac{1}{k-1}(R_{k-1} - \hat{v}_{k-1})$$

1.3.3 Greedy Approach

- We can select the action a_j that corresponds to the max value of the action value estimates at the discrete time step k.
- We are greedy and we want to select the action that maximizes the past estimate of the action value. This step is also the exploitation step. We are basically exploiting our past knowledge to gain future rewards.
- One of the issues is that we rely too much on our past experience for making decisions. If our past experience is limited, then our future decisions will be suboptimal.

The appropriate method is to select actions that are not necessarily optimal, and to explore these actions, with some probability ϵ.

1.3.4 ϵ-Greedy Approach

- Select a real number ϵ larger than 0 and smaller than 1.
- Draw a random value p from the uniform distribution on the interval $[0, 1]$.
- If $p > \epsilon$, then select the actions by maximizing the equation above.
- If $p \leq \epsilon$, then randomly select an action. That is, randomly pick any a_j from the set of all possible actions $\{a_1, a_2, \ldots, a_N\}$, and apply it to the system.

1.4 Python Code and Results

You can find the Python code [here](https://github.com/vishalnayak1507/RetroRL.git).

1.4.1 Observations

- **Action values:** `actionValues = np.array([1, 4, 2, 0, 7, 1, -1])`
- The pure greedy approach (red line) does not produce the best results.
- We can observe that for $\epsilon = 0.1$, we obtain the best results. The average reward approaches 6.5, which is very close to 7, one of the true action values for action 5.
- In fact, action 5 should produce the best performance since it has the highest action value.
- We can also track how many times a particular action is being selected by inspecting "howManyTimesParticularArmIsSelected".
- For $\epsilon = 0.1$, we get `[1443, 1433, 1349, 1387, 91501, 1499, 1388]`. Clearly, action 5 is selected the most number of times, so we are very close to the optimal performance with this algorithm.

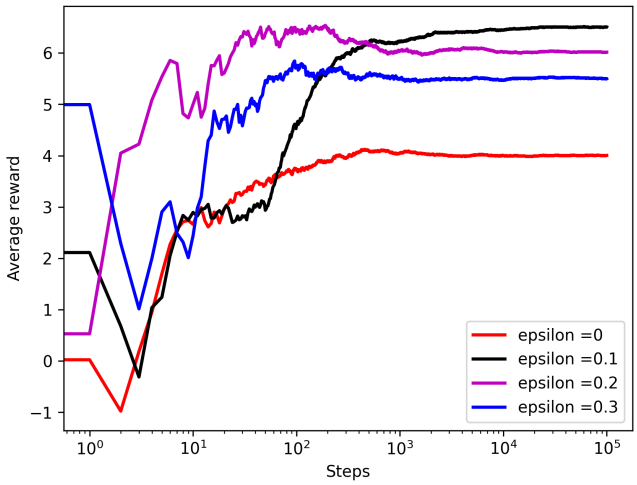

Figure 1: Example Result

2 Cart Pole Control Environment

The cart-pole environment consists of a cart that can move linearly, and a rotating bar or a pole attached to it via a bearing. The control objective is to keep the pole in the vertical position by applying horizontal actions (forces) to the cart. The action space consists of two actions:

- Push the cart left - denoted by zero.
- Push the cart right - denoted by one.

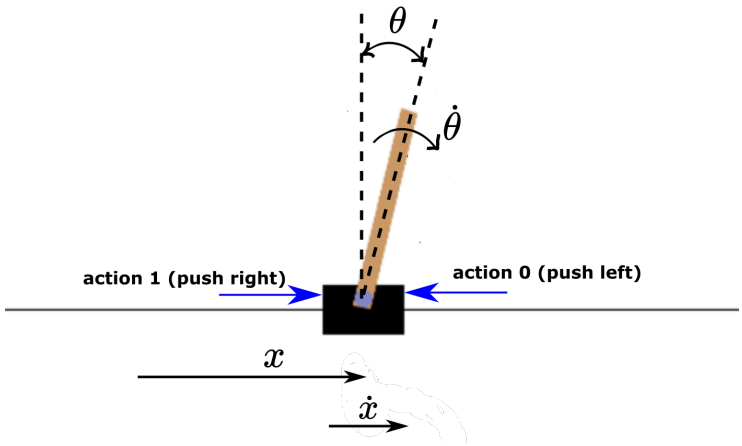

Figure 2: Cart Pole Environment

2.1 Observation States

- Car position is denoted by x. The minimum and maximum values of x are -4.8 and 4.8, respectively.
- Car velocity is denoted by $\dot{x}$. The minimum and maximum values are -4.8 and 4.8, respectively.
- Pole angle is denoted by θ. The minimum and maximum values of θ are -24 degrees and 24 degrees, respectively.
- Pole angular velocity is denoted by $\dot{\theta}$. The minimum and maximum values are $-\infty$ and ∞, respectively.

Initial observations are completely random and the values of states are chosen uniformly from the interval (-0.05, 0.05).

2.1.1 An episode terminates under the following conditions:

- If the absolute pole angle is greater than 12 degrees.
- If the absolute value of cart distance is greater than 2.4.
- If the number of steps in an episode is greater than 500 for version v1 of Cart Pole (200 for version v0).

The reward of +1 is obtained every time a step is taken within an episode.

2.2 Python Code

You can find the Python code [here](https://github.com/vishalnayak1507/RetroRL.git).

- First, we import the necessary libraries.
- Then, we create the environment using:

```
env = gym.make('CartPole-v1', render_mode='human')
```

- After creating the environment, we need to reset it by:

```
state, _ = env.reset()
```

- Next we simulate the environment, first we select the number of episodes and maximal number of time steps within episode.
- We reset the environment and then render the environment.we apply a random action to environment.
- If the return state is terminal state we break the current episode simulation.
- We also use pause in the code in order to ensure the environment is properly animated by :

```
time.sleep().
```

- Finally we close animation window by :

```
env.close().
```

3 Dynamic Programming

: