CSE4/510: Reinforcement Learning

Assignment 3 - Policy Gradient

Vinita Venktesh Chappar
University at Buffalo
Buffalo, New York
vchappar@buffalo.edu

Abstract

In this assignment, the REINFORCE algorithm is implemented. This algorithm comes under the Policy gradient methods of Reinforcement learning. Here, the openai gym environment Cartpole is used for applying the algorithm. This algorthm is not much complex in terms of implementations. It does not have many hyper-parameters and hence, it is easier to code and find a balance between gradience calculations for higher rewards. These findings are mentioned in the result analysis.

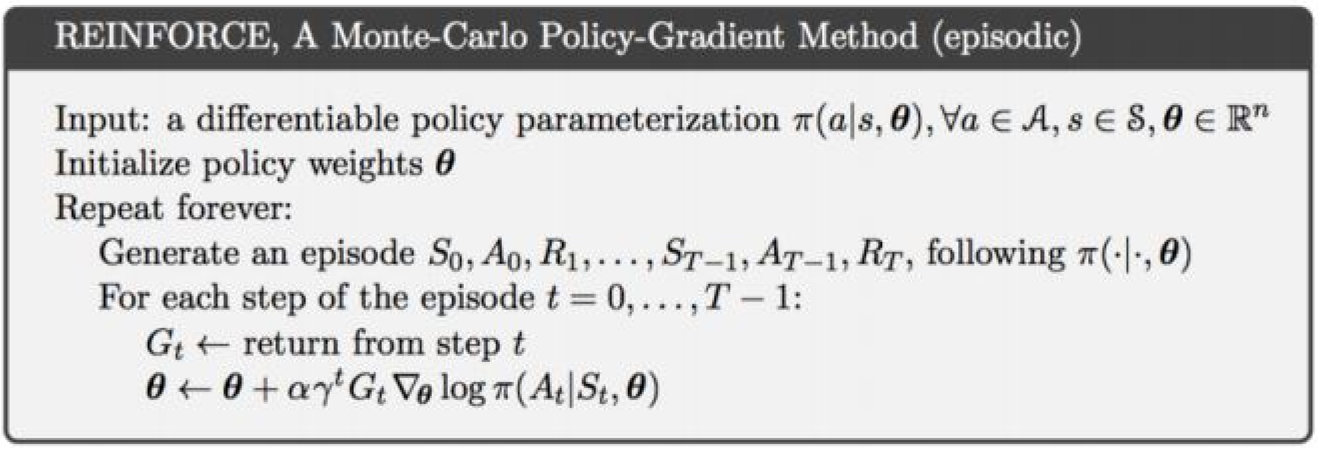
1 REINFORCE algorithm:

Reinforce algortihm is a policy gradient method of reinforcement learning. It is considered as a foundation upon which all the advance policy gradient algorithms are implemented. It is also a model-free RL algorithm. The aim of the algorithm is to maximize the cumulative future reward. Here, the policy is derieved in such a way that in the probability distribution, the action which gives the highest reward will have the highest probability.

Here, a policy parameter theta (θ) is introduced. This will help in optimizing the policy using gradient ascent. The objective funtion J is partially derieved with respect to θ.

This "vanilla" policy gradient algorithm has no bias but high variance. The reinforce algorithm involves the following steps:

1. Generate samples for each episode by selecting most likely next action, given the current policy π
2. For each episode, compute the cumulative reward G_t
3. Update the policy using the discounted cumulative rewards, log probabilities of the actions and learning rate



```
REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
    Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
    For each step of the episode $t = 0, \ldots, T-1$:
        $G_t \leftarrow$ return from step $t$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t,\boldsymbol{\theta})$
```

Figure 1: Pseudo code for REINFORCE algorithm

2 Environment:

For training and testing of this assignment, we are using the openai gym environment "Cartpole-v0". In this game, a pole is attached to a joint on a cart. This cart moves along a frictionless track.[1] The system is controlled by applying a force of -1 or 1 i.e left or right. The goal of the game is to keep the pole in an upright position as long as one can. For every timestep that the pole is upright, agent will recieve a reward of +1. The game ends when the score reaches 200 or the pole drops down below 15 degrees angle.

The cartpoleâĂŹs environment has 4 observations at any given state which contains representations such as angle and position of the cart. Agent can proceed to next state using actions [-1,1].

3 Results and Analysis:

The Reinforce algorithm is the simplest among all the policy gradient algorithm. As such, it does not provide as good results as the A2C or PPO algorithms.

Below are the scenarios observed:

3.0.1 Scenario 1:

Hyper Parameters
Learning rate = 0.001
Discount rate = 0.99
Number of episodes = 1500

Result Graph:

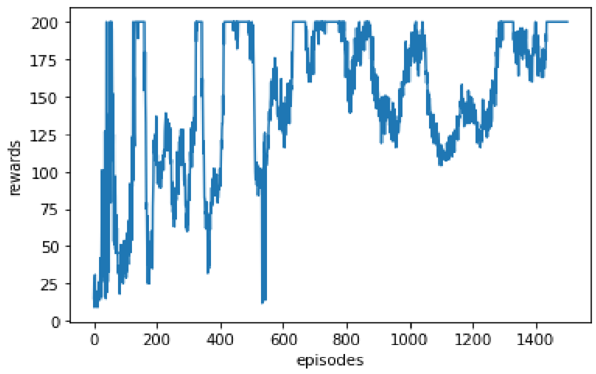


Figure 2: Episodes vs Rewards per episode

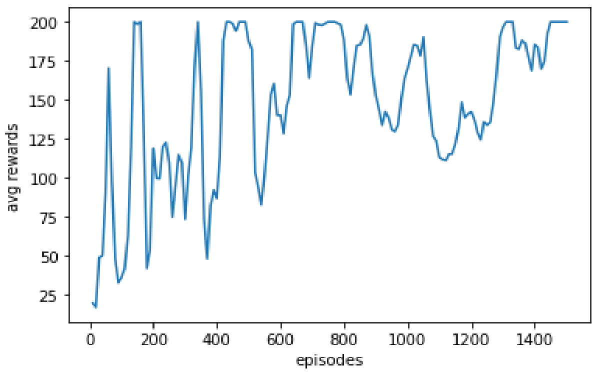


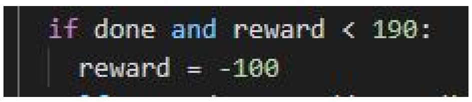
Figure 3: Episodes vs Average rewards

Problem: As you can see from the above graph, the agent manages to learn quickly and achieves maximum reward in merely 170 episodes. However, the agent does not learn to maintain that reward. It becomes greedy and will prefer one action over the other highly. This can make the policy deviate highly. This is the most severe drawback of the reinforce algorithm - High Variance.

Analysis: We can still tweak some hyper parameters to gain greater rewards using reinforce algorithm. In the second scenario, we added a reward based change in our algorithm. If the agent does not give us a result above the reward threshold, we give it a penalty.

3.0.2 Scenario 2:

Hyper Parameters
Learning rate = 0.01
Discount rate = 0.99
Number of episodes = 1500
Additional change: reward penalty if current episode reward < reward-threshold.



```
if done and reward < 190:
    reward = -100
```

Following are the graphs we got after these changes:

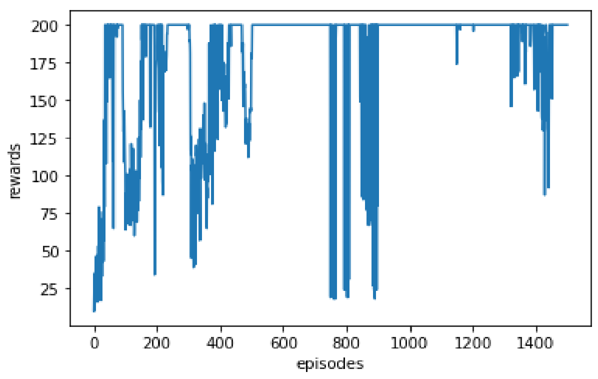


Figure 4: Episodes vs Rewards per episode

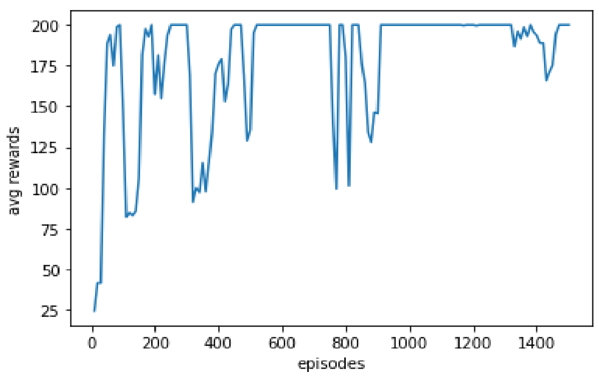


Figure 5: Episodes vs Average rewards

Analysis: As we can see, this small change did manage to get us some significant results. The algorithm has low variance now as compared to the former scenario.

4 References

[1] OpenAI Gym environments documentation, "https://gym.openai.com/ envs/CartPole-v1/"

[2] Class slides, "https://piazza.com/class_profile/get_resource/kafhgep7q3l1yr/kbsdykvsopx4yu"

[3] Deterministic Policy Gradient Algorithms, "http://proceedings.mlr.press/v32/silver14.pdf"

[4] Policy Gradient Methods for Reinforcement Learning with Function Approximation, https://papers.nips.cc/paper/1713-policy-gradient-methods-for-reinforcement-learning-with-function-approximation.pdf