CSE 510 Reinforcement Learning Spring 2020 Project 3

Using Policy Gradient and Advantage Actor-Critic algorithms to solve OpenAI CartPole

Srisai Karthik Neelamraju
UBID - 50316785
University at Buffalo
Buffalo, NY 14260
neelamra@buffalo.edu

Abstract

Policy gradient methods are used in reinforcement learning (RL) to model and optimize the policy directly, as opposed to value-based methods which aim to approximate the value or action-value functions. The policy is typically modeled with a parametrized function respect to some weights. A neural network is often used for this purpose, since neural networks are universal function approximators. In this project, two RL algorithms: Monte-Carlo Policy Gradient (REINFORCE) and Advantage Actor-Critic (A2C) are implemented for solving the OpenAI Gym environment CartPole. The first algorithm uses the expected return for learning the optimal policy, whereas the second algorithm uses advantages computed by a second network in order to achieve this. After training two agents with these two algorithms, it is observed that the latter has provided better performance than the former, which suffered from the problem of high-variance.

1 Introduction

Policy based methods work by modeling and optimizing the policy directly. They aim to maximize the expected cumulative reward by estimating the gradient repeatedly. They differ from the widely-used value-based methods in reinforcement learning, where learning agents try to learn value functions and improvise policy from them. However, policy-based methods involve no value functions and they instead learn the policy directly. Actor-critic algorithms make use of both the value-based and policy-based algorithms. They involve two sets of parameters – one of which learns a value function and the other learns a policy. The former network's parameters help in optimizing the latter network. In this project, REINFORCE and Advantage Actor-Critic (A2C) algorithms are implemented on CartPole-v0 environment of OpenAI Gym. After training both the agents, the episodic scores and the average scores of the agents during this learning phase are plotted for both the implementations. It is observed that A2C algorithm expectedly performed better than the REINFORCE algorithm, thus demonstrating the effectiveness of the former algorithm.

2 Monte-Carlo Policy Gradient (REINFORCE)

REINFORCE algorithm is a variant of the policy gradient algorithms that finds an unbiased estimate of the gradient without using any value function. For this purpose, it uses a set of parameters to describe its policy as a parametric distribution over the action space. It then improves this policy by updating the network parameters using stochastic gradient descent.

In this algorithm, the actual return at each time step is used as an unbiased sample of the Q-value for an action in a state. The essence of this algorithm is to update the policy network parameters in the direction that favors an action which gives the highest return. Figure 1 demonstrates the algorithm. Since this policy gradient variant is a Monte-Carlo method, there is no learning until the episode is finished. After generating an episode following the initial policy, the return at each time step is calculated. This return is used to update the network parameters θ using the given updated rule. It can be observed that the update is proportional to the return and the gradient of the log probability of the action being chosen.

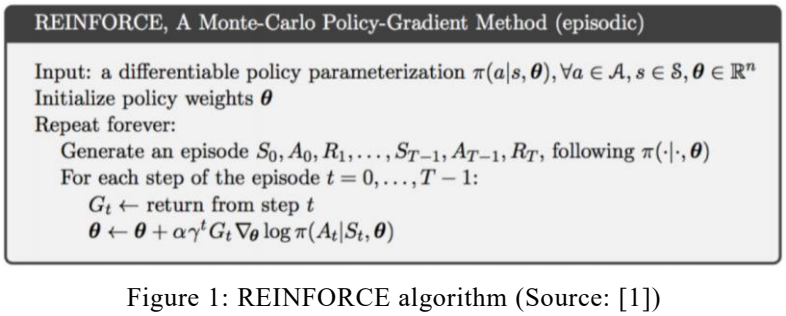


Figure 1: REINFORCE algorithm (Source: [1])

3 Advantage Actor-Critic (A2C)

The aforementioned REINFORCE algorithm suffers from the problem of exhibiting high variance. In order to address this issue, actor-critic methods were proposed in [4]. These methods use both value-based and policy-based methods since they rely on learning both value function and policy. The primary idea of this class of methods is to have a critic that tells the actor how good or bad the chosen action is. In actor-critic methods, there are two sets of parameters that are often referred to as actor network and critic network. The actor performs an action in an environment and the critic computes a value that assists the actor in learning. Later, both the critic and the actor update their network parameters, with the latter's update being in the direction suggested by the critic.

While the REINFORCE algorithm uses return to update the network parameters, A2C uses advantages for this purpose. The advantage value for choosing an action a_t in state s_t is defined by the formula,

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

However, this requires two networks for finding both the Q-values and the V value apart from having the actor network. This is inefficient. In order to have a single network, the relationship between Q and V value can be used [5],

$$Q(s_t, a_t) = \mathbb{E}\left[r_{t+1} + \gamma V(s_{t+1})\right]$$

Subsequently, using w to denote the parameters of the critic network, the above advantage value can be written as,

$$A_w(s_t, a_t) = r_{t+1} + \gamma V_w(s_{t+1}) - V_w(s_t)$$

These advantages are used to update the weights of both the actor and the critic networks. Figure 2 demonstrates the general Actor-Critic algorithm. The term δ can take many forms, giving rise to different actor-critic methods like Q Actor-Critic, TD Actor-Critic and A2C. In this case, $\delta = A_w(s_t, a_t)$. Both the networks are randomly initialized. Since A2C does not follow Monte-Carlo updates, the agent does not need to wait until the end of an episode and it can learn from intermediate rewards. In each state, the agent chooses an action following its current policy and gets a reward from the environment. It is then that the critic network is used to find the advantage (δ) of choosing this action and this value is then used to update the parameters of both the networks using Gradient descent following the given update rules. This process is continued until the network weights result in the the maximum reward.

One-step Actor-Critic (episodic)

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Input: a differentiable state-value parameterization $\hat{v}(s,\mathbf{w}), \forall s \in \mathcal{S}, \mathbf{w} \in \mathbb{R}^m$
Parameters: step sizes $\alpha > 0$, $\beta > 0$

Initialize policy weights $\boldsymbol{\theta}$ and state-value weights $\mathbf{w}$
Repeat forever:
  Initialize S (first state of episode)
  $I \leftarrow 1$
  While S is not terminal:
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \beta \delta \nabla_{\mathbf{w}} \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha I \delta \nabla_{\boldsymbol{\theta}} \log \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Figure 2: Actor-Critic algorithm (Source: [1])

4 Environment

For this project, the OpenAI gym environment CartPole-v0 is used. Cartpole consists of a pole attached to a cart by the means of an un-actuated joint. The cart is placed on a frictionless track and capable of moving along it. The aim is to prevent the pole from falling off the cart by controlling its movement [3]. Figure 3 shows the environment with the cart and the pole. The state in this environment is represented using four real values – cart position, cart velocity, pole angle and pole velocity at tip. Further, there are two possible actions in each state – push cart to the left (action 0) or to the right (action 1). Reward is +1 for every step taken before an episode ends, except for the termination step which gives a reward of 0. In this environment, an episode terminates if either of the following three conditions is satisfied: [6]

i. episode reward is equal to 200
ii. pole angle is more than 12° away from the vertical on either side
iii. cart position is more than 2.4 units away from the center on either side

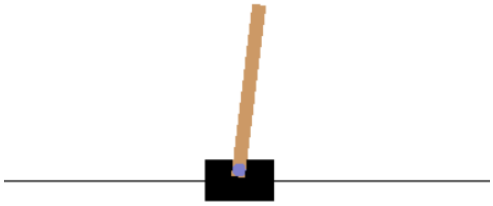


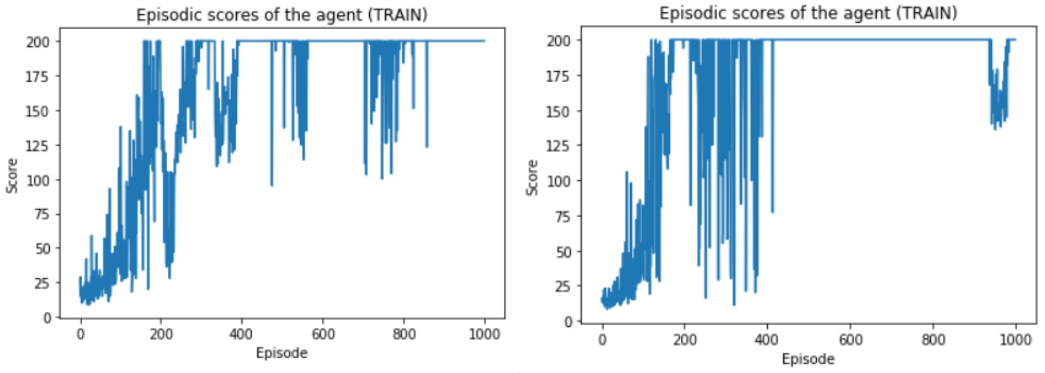
Figure 3: A state in the CartPole-v0 environment of the OpenAI Gym

5 Implementation and Results

In this project, the two RL algorithms – REINFORCE and A2C are implemented on the CartPole-v0 environment. For implementing the first algorithm, a fully connected neural network with two dense layers is employed. The first layer has 10 hidden units and uses 'ReLU' activation, while the second layer has 2 hidden units with 'SoftMax' activation. On the other hand, for implementing the second algorithm, two neural networks are required: one for the actor and one for the critic. However, in order to simplify training the model, the first layer for both these networks is chosen to be common and the output of this layer is then connected to two separate layers corresponding to the actor and the critic. The common layer once again has 10 hidden units and uses 'ReLU' activation. The second layer in the actor network has 2 units with 'SoftMax' activation, whereas it has only 1 unit with linear activation in case of the critic network. In both the algorithm, the optimizer is chosen to be

Adam, with a learning rate of 0.01. Besides, the discount factor is set to 0.99. Deep learning framework PyTorch is used for both the implementations in this project.

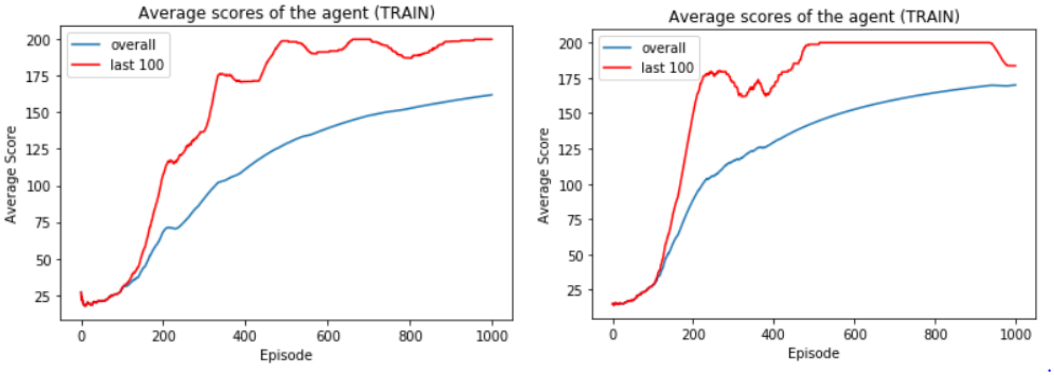
Two agents that use these two algorithms are defined and then trained for 1,000 episodes. The scores obtained by the agents during the training phase are plotted in Figures 4(a) and 4(b). Both the plots display a similar trend during the first 200 episodes, where both the agents have already attained the maximum score of 200.0. During the next 200 episodes, it is observed that there is some variance in the scores for both the agents, with the A2C agent exhibiting it much more than the REINFORCE agent. However, it is after this phase that the former algorithm demonstrates its effectiveness in stabilizing the performance of the agent. The score of the agent remains to be 200.0 for more than 500 episodes, whereas the variance in scores continues for the other agent. After about 900 episodes, there is some drop in the scores for the A2C agent; however, the scores never fall below 125.0.



(a) (b)

Figure 4: Comparing episodic scores of the agent for 1,000 training episodes on CartPole using (a) REINFORCE and (b) A2C algorithms

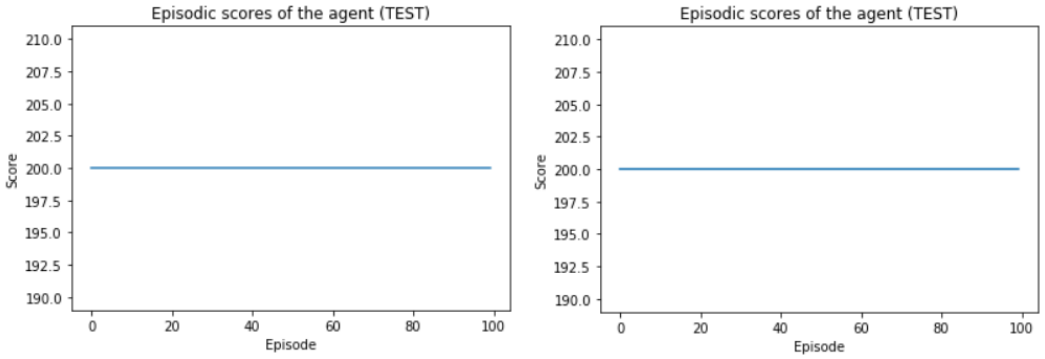
Further, Figures 5(a) and 5(b) display the average scores of the agent during their training. Specifically, the overall average score and the last 100-episode average score are plotted. The overall average reward plots show a gradual increase in the scores, implying that the agents are being trained appropriately. Besides, the last 100-episode average score plot also follows a similar trend. However, these average scores in the case of REINFORCE agent show an increase and then a drop trend throughout the learning phase. On the other hand, the scores in the case of A2C agent follow a much more stable trend. This indicates the better performance of the A2C policy gradient algorithm as against the REINFORCE algorithm.



(a) (b)

Figure 5: Comparing average scores of the agent per episode for 1,000 training episodes of CartPole using (a) REINFORCE and (b) A2C algorithms

Finally, the two agents are tested for 100 episodes. These scores are plotted in Figures 6(a) and 6(b). It is observed that both the agents obtained the maximum score of 200.0 in each of the testing episodes, suggesting that both the algorithms enabled the agents to learn the environment thoroughly.



(a) (b)

Figure 6: Comparing episodic scores of the agent for 100 testing episodes on CartPole using (a) REINFORCE and (b) A2C algorithms

6 Summary

In this project, two reinforcement learning algorithms – REINFORCE and Advantage Actor-Critic algorithms, were implemented with PyTorch deep learning framework in Python. The OpenAI Gym environment CartPole-v0 was chosen as the environment for applying these algorithms. For parametrizing the policy, a fully-connected 2-layer dense neural network architecture was opted for in both the algorithms. Two agents, where one agent used the REINFORCE algorithm for learning whereas the other used A2C, were separately trained on the environment for 1,000 episodes. The episodic and average scores of both the agents were observed. Besides, the recent 100-episode average score was also observed in both cases. Later, the scores of the learned agents on the environment were observed for 100 testing episodes. Both the algorithms performed similarly while testing, getting the maximum score of 200.0 on each of the test runs. However, it was primarily inferred from the training phase that the A2C algorithm gave much better performance than REINFORCE in terms of the variance in the obtained scores. The former algorithm converged with much more stability than the latter, which showed high variance in the agent's behavior.

References

[1] Richard S. Sutton and Andrew G. Barto, "Reinforcement learning: An introduction", Second Edition
[2] CSE 510 Introduction to Reinforcement Learning Slides by Alina Vereshchaka
[3] https://github.com/openai/gym/wiki/CartPole-v0
[4] Vijay R., and John N. Tsitsiklis. "Actor-critic algorithms." Advances in neural information processing systems. 2000.
[5] https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f
[6] Acknowledgement: Few parts of this project report are based on my CSE 510 RL Projects 1 and 2