**Cartpole Revisited**

Justin Starr

Department of STEM

CS 370 – Current/Emerging Trends in CS

Dr. Obafemi Balogun

June 16, 2024

**Cartpole Revisited**

The following is a short analysis of how the cartpole problem can be solved using the REINFORCE algorithm and A2C algorithm. It further explains how policy gradient approaches differ from value-based approaches, such as Q-learning, and how actor-critic approaches differ from value- and policy-based approaches.

The REINFORCE algorithm, short for REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility, is a Monte Carlo variation of the policy gradient algorithm in reinforcement learning (Kang, 2021). “Policy gradient methods update the probability distribution of actions so that actions with higher expected reward have a higher probability value for an observed state” (Yoon, 2018). Essentially, it is an iterative process where the goal is to learn a policy, which maximizes the cumulative future reward that will be received starting from any given time until the terminal time is reached (Yoon, 2018). The REINFORCE algorithm can be used to solve the cartpole problem by first establishing an initial policy (setting up the policy network), then repeating the following steps: execute the current policy in the environment to gather a collection of trajectories, computing the total of the discounted rewards from each state forward, and for each one in the trajectory, calculate the policy gradient by computing the gradient of the log-likelihood for the chosen course of action, and lastly the policy parameters are updated after each episode has completed(GeeksforGeeks, 2024).

In an Actor-Critic model, the actor and critic are two separate networks where the actor takes the state as input and the best action is output, and essentially, by learning the optimal policy (policy-based), controls how the agent behaves (Karagiannakos, 2018). The critic is a function approximator, and its role is to evaluate the environment and the action taken by the actor and output the Q-value (action value), which is essentially the maximum future reward (Karagiannakos, 2018). As time passes, both the actor and critic play a game where they get better in their roles, and the overall architecture learns to play the game more efficiently than if they were to play separately. The Advantage Actor-Critic (A2C) is a specific variant of the Actor-Critic algorithm that uses the advantage function. Q values are comprised of two pieces, a state value function, which is how good it is to be at this state, and an advantage value function, which is how better an action is compared to other actions at a particular state (Karagiannakos, 2018). In A2C, instead of the critic learning the Q values, it learns the advantage values, where evaluations of actions are not only based on how good the action is but it is also based on how much better the action can be (Karagiannakos, 2018). The advantage function tells whether a state is better (the advantage is greater than zero) or worse than expected (the advantage is less than zero), and if it is zero, the actor did not learn from that action (Wang, 2021).

For the cartpole problem, we can implement the A2C algorithm by implementing two networks (the actor and the critic) that will work together to solve the cartpole problem (Wang, 2021). The advantage function will be used to calculate the advantage also known as a prediction error at a high level (Wang, 2021). At each time step, the actor network chooses an action, and the critic network evaluates the Q-value (quality) of a given input state (Wang, 2021). The actor uses the information learned by the critic network (which is learning what states are better or worse), to teach the agent to avoid bad states and seek out good states (Wang, 2021). Together, the actor and the critic learn to balance the pole for longer durations by repeating this learning process (Wang, 2021).

In policy gradient approaches, the objective is to maximize the expected cumulative reward with respect to the policy’s parameters (Yoon, 2018). With policy gradients, the neural network being built directly learns the optimal policy, and instead of learning a function that takes a state as input and outputs Q-values for all possible actions (as in value-based approaches), the function it learns outputs the best action that can be taken from that state (Doshi, 2021). To be more precise, in policy gradient approaches, instead of a single best action, a probability distribution of the actions that can be taken from that state is output, then an action can be taken by sampling from that probability distribution (Doshi, 2021).

Value-based approaches, such as Q-learning, differ from policy gradient in that they are not going to find the optimal policy directly (Doshi, 2021). For a given state, the network learns to output the optimal Q-values, which are then used to determine the optimal policy (Doshi, 2021). Essentially, the higher the q-value, the better the action is (Karagiannakos, 2018). Each approach has its advantages, for example, policy-based approaches are better for environments that are continuous and stochastic, and they have a faster convergence, however, value-based approaches are more sample-efficient and can be steadier (Karagiannakos, 2018).

While value- and policy-based approaches have their differences, actor-critic approaches combine aspects of both value and policy-based approaches (Karagiannakos, 2018). The actor represents the policy, which takes the state in as input and then learns to select the best possible action (Karagiannakos, 2018). It is essentially controlling how the agent acts by learning the optimal policy and it is how it incorporates policy-based approaches. Where it incorporated the value-based approach is with the critic, which evaluates actions by computing the value function (Karagiannakos, 2018). Therefore, value and policy-based approaches are different from each other, whereas actor-critic approaches aim to include both value and policy-based approaches.

**Pseudocode for REINFORCE Algorithm for the Cartpole Problem:**

DEFINE and INITIALIZE environment

DEFINE and INITIALIZE the policy network

STORE number of max episodes

STORE number of max steps

INITIALIZE numsteps with an empty array

INITIALIZE avg\_numsteps with an empty array

INITIALIZE all\_rewards with an empty array

# Algorithm

FOR each episode:

ASSIGN state by resetting the environment

INITIALIZE log\_probs with an empty array

INITIALIZE rewards with an empty array

FOR each step in an episode:

Render the environment

STORE action and log\_prob by INVOKING get\_action for the policy network

and PASS the state.

STORE new\_state, reward, and done by INVOKING step function for the environment and PASS the previously stored action.

APPEND log\_probs with log\_prob

APPEND rewards with reward

IF the episode is done:

CALL update\_policy and PASS policy\_net and log\_probs

APPEND numsteps with steps

APPEND avg\_numsteps by calculating the mean of the previous 10

elements in numsteps

APPEND all\_rewards by summiong rewards

PRINT the episode, total reward, average reward, length, and steps

BREAK

END IF done

ASSIGN state to new state

END FOR LOOP

END FOR LOOP

**Pseudocode for A2C algorithm for the cartpole problem:**

DEFINE and INITIALIZE num\_inputs with environment observation\_space.shape[0]

DEFINE and INITIALIZE num\_outputs with environment action\_space.n

DEFINE and INITIALIZE the actor\_critic network

DEFINE and INITIALIZE the actor critic optimizer

INITIALIZE all\_length with empty array

INITIALIZE average\_lengths with empty array

INITIALIZE all\_rewards with empty array

INITIALIZE entropy\_term to zero

# Algorithm

FOR each episode

INITIALIZE log\_probs with empty array

INITIALIZE values with empty array

INITIALIZE rewards with empty array

ASSIGN state by resetting the environment

FOR each step

RETRIEVE value and policy\_dict by CALLING actor\_critic.forward and passing

the state

STORE value

STORE dist

ASSIGN action

ASSIGN log\_prob

ASSIGN entropy

ASSIGN new\_state, reward, and done by taking action in environment

APPEND rewards with reward

APPEND values with value

APPEND log\_probs with log\_prob

INCREASE entropy\_term with entropy

ASSIGN state with new\_state

IF done or steps equals num\_steps – 1

RETRIEVE the quality of the action

APPEND all\_rewards with rewards

APPEND all\_lengths by calculating the mean of the previous 10 elements

in all\_lengths

IF episode modulo 10 equals 0

PRINT the episode

BREAK

END IF

END FOR

END FOR

COMPUTE Q values

UPDATE actor critic

ASSIGN advantage equal to Qvals – values

ASSIGN and COMPUTE actor loss

ASSIGN and COMPUTE critic\_loss

ASSIGN and COMPUTE actor critic loss (ac\_loss)

CALL ac\_optimizer.zero\_grad()

CALL ac\_loss.backward()

CALL ac\_optimizer.step()

**References**

Doshi, K. (2021, January 9). *Reinforcement Learning Explained Visually (Part 6): Policy*

*Gradients, step-by-step*. Medium. <https://towardsdatascience.com/reinforcement-learning-explained-visually-part-6-policy-gradients-step-by-step-f9f448e73754>

GeeksforGeeks. (2024, January 27). *REINFORCE*

*Algorithm*. [https://www.geeksforgeeks.org/reinforce-algorithm/#](https://www.geeksforgeeks.org/reinforce-algorithm/)

Kang, C. (2021, May 12). *REINFORCEMENT on CartPole-v0*. Chan's

Jupyter. <https://goodboychan.github.io/python/reinforcement_learning/pytorch/udacity/2021/05/12/REINFORCE-CartPole.html>

Karagiannakos, S. (2018, November 17). *The idea behind Actor-Critics and how A2C and A3C*

*improve them*. AI Summer. <https://theaisummer.com/Actor_critics/>

Wang, M. (2021, January 22). *Advantage Actor Critic Tutorial:*

*minA2C*. Medium. <https://towardsdatascience.com/advantage-actor-critic-tutorial-mina2c-7a3249962fc8>

Yoon, C. (2018, December 29). *Deriving Policy Gradients and Implementing*

*REINFORCE*. Medium. <https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63>