Policy Gradient Update

- Name: Bidit Sadhukhan
- Reg.No: B2230022
- Subject: Reinforcement Learning

Table of Contents

Policy Gradient Update Rule

Cartpole Problem:

The CartPole-v1 problem in the Gym environment is a classic control task that involves balancing a pole on a cart. Here are the key details about this problem:

- **Description**: A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The goal is to balance the pole by applying forces in the left and right direction on the cart.
- **Action Space**: The action is a binary value (0 or 1) indicating the direction of the force applied to the cart: 0 for left and 1 for right.

```
env = gym.make('CartPole-v1')
env.observation_space
Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38], (4,), flo
env.action_space
Discrete(2)
```

- **Observation Space**: The observation consists of four continuous values representing the cart position, cart velocity, pole angle, and pole angular velocity.
- **Rewards**: A reward of +1 is given for every time step the pole is balanced, and the episode ends if the pole angle is greater than ±12° or the cart position is greater than ±2.4.
- **Starting State**: All observations are assigned a uniformly random value in the range (-0.05, 0.05).
- **Episode End**: The episode ends if the pole angle or cart position exceeds certain thresholds, or if the episode length is greater than 500.

Tackling the problem

- Original code by **Janis Klaise.**
- Baseline adjustment and update policy modifications by Bidit Sadhukhan.

These changes have resulted in a significant improvement in the model's performance, as evidenced by Results.

Modifying the discount_rewards function:

The modified code is in here:

```
def discount_rewards(self, rewards, obs):
    # calculate temporally adjusted, discounted rewards with baseline adjustment

        discounted_rewards = np.zeros(len(rewards))
        cumulative_rewards = 0

    # Calculate state values
        state_values = self.get_state_values(obs)

        for i in reversed(range(0, len(rewards))):
            cumulative_rewards = cumulative_rewards * self.γ + rewards[i] - state_values[i]
            discounted_rewards[i] = cumulative_rewards

        return discounted_rewards


    def get_state_values(self, obs):
        # Calculate state values using a linear function
        # obs is a numpy array of observations
        state_values = np.zeros(len(obs))
        for i, ob in enumerate(obs):
            state_values[i] = ob @ self.θ
        return state_values
```

Explanation:

The two functions, `get_state_values` and `discount_rewards`, are essential components of reinforcement learning algorithms, particularly in the context of policy gradient methods.

1. **get_state_values**: This function calculates the state values for a given set of observations. It initializes an array `state_values` with zeros, where the length of this array is the same as the number of observations in the input array `obs`. For each observation, it calculates the state value by taking the dot product of the observation `ob` with the policy parameter vector `self.θ`. This is essentially a linear function of the observation. The final result is an array of state values corresponding to each observation in the input `obs`, which is returned.
2. **discount_rewards**: This function calculates temporally adjusted, discounted rewards with baseline adjustment. It initializes an array `discounted_rewards` with zeros, where the length of this array is the same as the number of rewards in the input array `rewards`. It then calculates the state values for each observation in the input `obs` using the `get_state_values` method. The function iterates over the rewards in reverse order (starting from the last time step) and updates `cumulative_rewards` by discounting the previous cumulative rewards and subtracting the corresponding state value. The adjusted cumulative rewards are then assigned to the `discounted_rewards` array at the corresponding time step. The final result is an array of temporally adjusted, discounted rewards with baseline adjustment, which is returned.

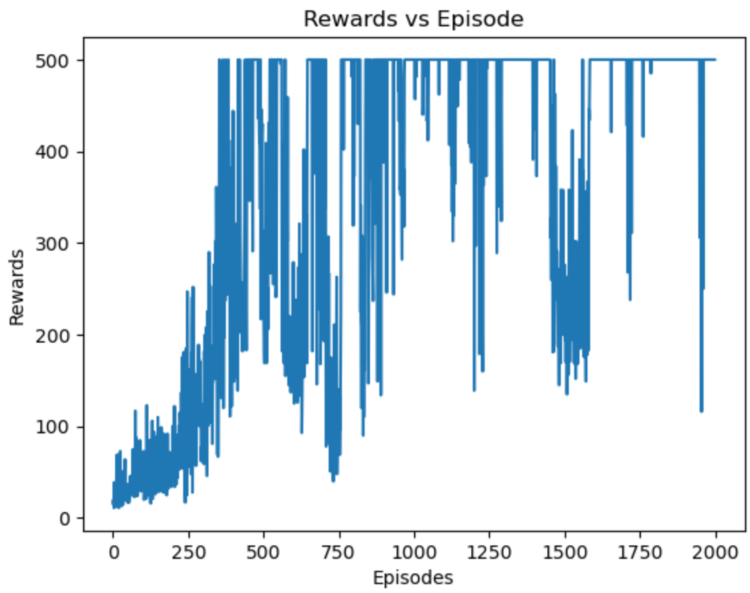
Modifying the update rule:

```
def update(self, rewards, obs, actions):
        # calculate gradients for each action over all observations
        grad_log_p = np.array([self.grad_log_p(ob)[action] for ob,action in zip(obs,actions)])

        assert grad_log_p.shape == (len(obs), 4)

        # calculate temporaly adjusted, discounted rewards
        discounted_rewards = self.discount_rewards(rewards,obs)

        # gradients times rewards
        dot = self.grad_log_p_dot_rewards(grad_log_p, actions, discounted_rewards)

        # gradient ascent on parameters
        self.θ += self.α*dot
```

Explanation:

No changes to the code is made here.

Results:



This is the plot of the original Rewards vs Episode.

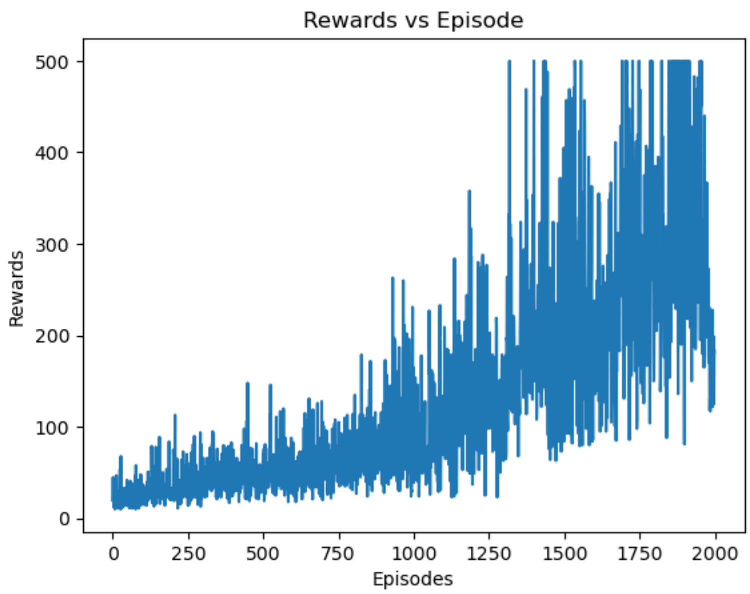
The variance is obtained as

$$Variance : 30796.931847750002$$

Also the bias is

$$Bias : 0$$

In the modified code the Reward vs Episode plot is



The variance is obtained as

$$Variance : 12943.391319000002$$

The bias came out to be

$$Bias : 0$$

Citations:

[1] https://www.janisklaise.com/post/rl-policy-gradients/
[2] https://stackoverflow.com/questions/55779079/discounted-rewards-in-basic-reinforcement-learning
[3] http://minpy.readthedocs.io/en/latest/tutorial/rl_policy_gradient_tutorial/rl_policy_gradient.html
[4] https://github.com/yudhisteer/Reinforcement-Learning-for-Supply-Chain-Management
[5] https://towardsdatascience.com/from-prediction-to-action-how-to-learn-optimal-policies-from-data-part-1-1edbfdcb725d
[6] https://cse.buffalo.edu/~avereshc/rl_fall19/lecture_19_Policy_Gradients_Baselines.pdf
[7] https://github.com/jklaise/personal_website/blob/master/notebooks/rl_policy_gradients.ipynb
[8] https://lilianweng.github.io/posts/2018-04-08-policy-gradient/
[1] https://www.gymlibrary.dev/environments/classic_control/cart_pole/
[2] https://stackoverflow.com/questions/75179713/problem-getting-dqn-to-learn-cartpole-v1-pytorch
[3] https://github.com/openai/gym/issues/2634
[4] https://towardsdatascience.com/how-to-beat-the-cartpole-game-in-5-lines-5ab4e738c93f
[5] https://youtube.com/watch?v=2u1REHeHMrg