Indian Institute of Technology, Madras
Chennai, Tamil Nadu

CS6700: Reinforcement Learning, Report on programming assignment 2

Jan-May 2024

Submitted by: Nataraj Das (CS23D404)
&
Shuvrajeet Das (CS23E001)

Here is the GitHub link for all ther codes: `https://github.com/imnataraj/CS6700-PA-2`

0.1 Dueling−DQN

Dueling DQN is an extension of the DQN algorithm, designed to improve learning efficiency by decomposing the Q-value function into two separate streams: one estimating the state value and the other estimating the advantage of each action. The update equation for the dueling network is of 2 forms:

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \frac{1}{|A|} \sum_{a' \in |A|} A(s,a';\theta) \right) \tag{1}$$

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \max_{a' \in |A|} A(s,a';\theta) \right) \tag{2}$$

Equation1 and 2 are the mathematical formulations of the update rule used in the dueling−DQN algorithm.

Type-1 Update Equation:

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \frac{1}{|A|} \sum_{a' \in |A|} A(s,a';\theta) \right)$$

Difference: This equation computes the advantage $A(s,a;\theta)$ relative to the mean advantage over all actions.

Pros:

- By subtracting the mean advantage from the current action's advantage, it helps in stabilizing the training process by reducing variance.
- Provides a clearer separation between the value and advantage functions.

Cons:

- Computationally more expensive due to the need to compute the mean advantage over all actions.
- May not work optimally if the advantages have a skewed distribution.

Type-2 Update Equation:

$$Q(s,a;\theta) = V(s;\theta) + \left(A(s,a;\theta) - \max_{a' \in |A|} A(s,a';\theta) \right)$$

Difference: This equation computes the advantage $A(s,a;\theta)$ relative to the maximum advantage over all actions.

Pros:

- Simpler computation compared to the type-1 equation, as it only requires finding the maximum advantage over all actions.
- Still provides a separation between the value and advantage functions.

Cons:

- Might be prone to higher variance during training, especially if the maximum advantage varies significantly across actions.
- May lead to instability during training if the maximum advantage value is subject to large fluctuations.

Overall Comparison:

- **Type-1** tends to be more stable due to the variance reduction achieved by subtracting the mean advantage. However, it comes with a higher computational cost.
- **Type-2** is simpler computationally but might suffer from higher variance during training, particularly if the maximum advantage varies widely across actions.

The choice between these two update equations often depends on the specific characteristics of the problem domain, computational resources available, and the desired balance between stability and computational efficiency.

0.1.1 In *Acrobot-V1* environment:

Type 1:

Algorithm 1 Dueling DQN Update Algorithm with Type-1 Update Rule

Input: Replay buffer R, main DQN Q_{main}, target DQN Q_{target}, batch size B, discount factor γ
Output: Updated main DQN parameters
while training is not finished **do**
  if $|R| \geq B$ **then**
    Sample a mini-batch of experiences (s, a, r, s', d) from R
    Compute target values:
    $q_{\text{target}} \leftarrow r + \gamma \max_{a'} Q_{\text{target}}(s', a')(1 - d)$
    Compute Q-values:
    $q_{\text{values}} \leftarrow Q_{\text{main}}(s)$
    Compute Q-values for selected actions:
    $q_{\text{action}} \leftarrow q_{\text{values}}[a]$
    Compute advantage:
    $A(s, a) \leftarrow q_{\text{action}} - \frac{1}{|\mathcal{A}|} \sum_{a'} q_{\text{values}}[a']$
    Compute updated Q-values:
    $Q_{\text{main}}(s, a) \leftarrow q_{\text{target}} + A(s, a)$
    Compute loss:
    $\mathcal{L} \leftarrow \frac{1}{B} \sum_i (q_{\text{target}}[i] - Q_{\text{main}}(s, a)[i])^2$
    Compute gradients:
    $\nabla_\theta \mathcal{L} \leftarrow \nabla_\theta \mathcal{L}(q_{\text{target}}, Q_{\text{main}}(s, a))$
    Update main DQN parameters:
    $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
  end if
end while

When this algorithm is used in the stated enviroment the following reward plot has been generated:

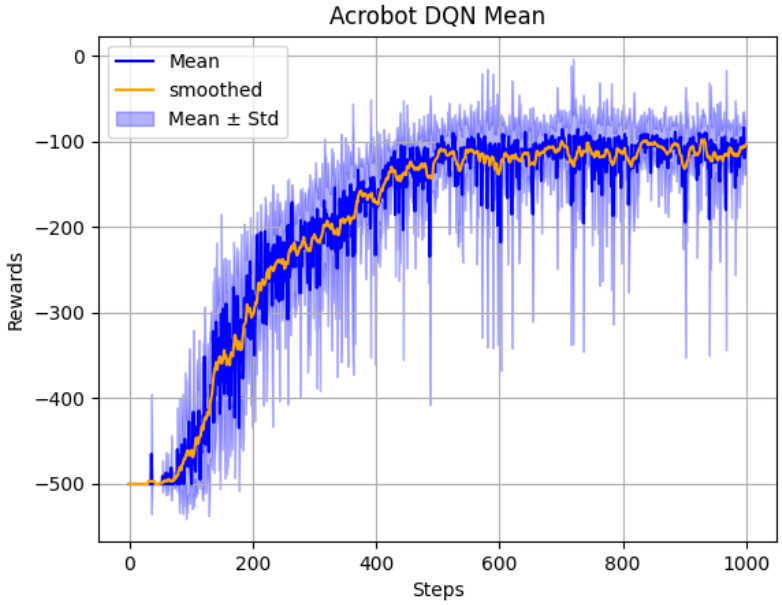

Figure 1: Acrobot_DQN_Mean

Type 2:

Algorithm 2 Dueling DQN Update Algorithm with Type-2 Update Rule

Input: Replay buffer R, main DQN Q_{main}, target DQN Q_{target}, batch size B, discount factor γ
Output: Updated main DQN parameters
while training is not finished **do**
  if $|R| \geq B$ **then**
    Sample a mini-batch of experiences (s, a, r, s', d) from R
    Compute target values:
    $q_{\text{target}} \leftarrow r + \gamma \max_{a'} Q_{\text{target}}(s', a')(1 - d)$
    Compute Q-values:
    $q_{\text{values}} \leftarrow Q_{\text{main}}(s)$
    Compute Q-values for selected actions:
    $q_{\text{action}} \leftarrow q_{\text{values}}[a]$
    Compute advantage:
    $A(s, a) \leftarrow q_{\text{action}} - \max_{a'} q_{\text{values}}[a']$
    Compute updated Q-values:
    $Q_{\text{main}}(s, a) \leftarrow q_{\text{target}} + A(s, a)$
    Compute loss:
    $\mathcal{L} \leftarrow \frac{1}{B} \sum_i (q_{\text{target}}[i] - Q_{\text{main}}(s, a)[i])^2$
    Compute gradients:
    $\nabla_\theta \mathcal{L} \leftarrow \nabla_\theta \mathcal{L}(q_{\text{target}}, Q_{\text{main}}(s, a))$
    Update main DQN parameters:
    $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
  end if
end while

When this algorithm is executed in the stated environment the following reward curve is obtained.

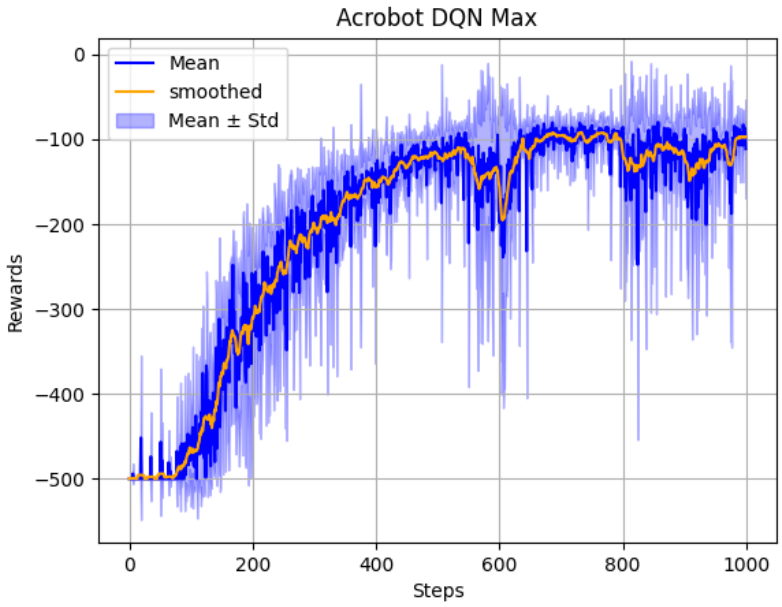

Figure 2: Acrobot_DQN_Max

Comparison of Type 1 and Type 2 from the reward plots:

- Unlike type 2, type 1 exhibits smoother learning curves with less variance in reward plots, due to its variance reduction property. On the other hand, Type 2 exhibits more fluctuating reward plots due to high variance during training.
- Type 1 converges steadily over time, even in environments with high variance and complex dynamics, where as type 2 converges faster, especially if the maximum advantage value provide more accurate guidance.
- Another thing to point out, type 1 was computationally heavy and was taking longer time in comparison with type 2.

0.1.2 In *CartPole* environment:

Type 1:

Algorithm 3 Dueling DQN Update Algorithm with Type-1 Update Rule for Cartpole Environment

Input: Replay buffer R, main DQN Q_{main}, target DQN Q_{target}, batch size B, discount factor γ
Output: Updated main DQN parameters
while training is not finished **do**
  if $|R| \geq B$ **then**
    Sample a mini-batch of experiences (s, a, r, s', d) from R
    Compute target values:
    $q_{\text{target}} \leftarrow r + \gamma \max_{a'} Q_{\text{target}}(s', a')(1 - d)$
    Compute Q-values:
    $q_{\text{values}} \leftarrow Q_{\text{main}}(s)$
    Compute Q-values for selected actions:
    $q_{\text{action}} \leftarrow q_{\text{values}}[a]$
    Compute advantage:
    $A(s, a) \leftarrow q_{\text{action}} - \frac{1}{|\mathcal{A}|} \sum_{a'} q_{\text{values}}[a']$
    Compute updated Q-values:
    $Q_{\text{main}}(s, a) \leftarrow q_{\text{target}} + A(s, a)$
    Compute loss:
    $\mathcal{L} \leftarrow \frac{1}{B} \sum_i (q_{\text{target}}[i] - Q_{\text{main}}(s, a)[i])^2$
    Compute gradients:
    $\nabla_\theta \mathcal{L} \leftarrow \nabla_\theta \mathcal{L}(q_{\text{target}}, Q_{\text{main}}(s, a))$
    Update main DQN parameters:
    $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
  end if
end while

The above algorithm when executed in the stated environment, the following reward plot got generated:

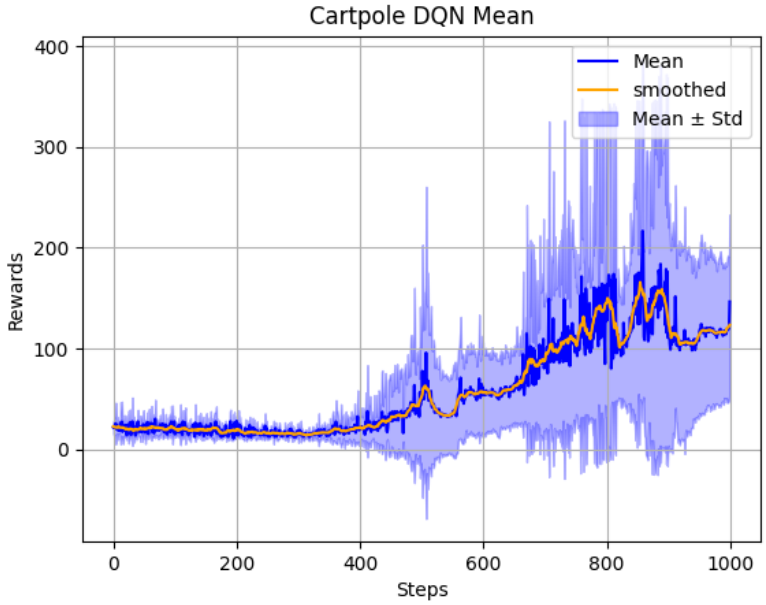

Figure 3: Cartpole_DQN_Mean

Type 2:

Algorithm 4 Dueling DQN Update Algorithm with Type-2 Update Rule for Cartpole Environment

Input: Replay buffer R, main DQN Q_{main}, target DQN Q_{target}, batch size B, discount factor γ
Output: Updated main DQN parameters
while training is not finished **do**
  if $|R| \geq B$ **then**
    Sample a mini-batch of experiences (s, a, r, s', d) from R
    Compute target values:
    $q_{\text{target}} \leftarrow r + \gamma \max_{a'} Q_{\text{target}}(s', a')(1 - d)$
    Compute Q-values:
    $q_{\text{values}} \leftarrow Q_{\text{main}}(s)$
    Compute Q-values for selected actions:
    $q_{\text{action}} \leftarrow q_{\text{values}}[a]$
    Compute advantage:
    $A(s, a) \leftarrow q_{\text{action}} - \max_{a'} q_{\text{values}}[a']$
    Compute updated Q-values:
    $Q_{\text{main}}(s, a) \leftarrow q_{\text{target}} + A(s, a)$
    Compute loss:
    $\mathcal{L} \leftarrow \frac{1}{B} \sum_i (q_{\text{target}}[i] - Q_{\text{main}}(s, a)[i])^2$
    Compute gradients:
    $\nabla_\theta \mathcal{L} \leftarrow \nabla_\theta \mathcal{L}(q_{\text{target}}, Q_{\text{main}}(s, a))$
    Update main DQN parameters:
    $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}$
  end if
end while

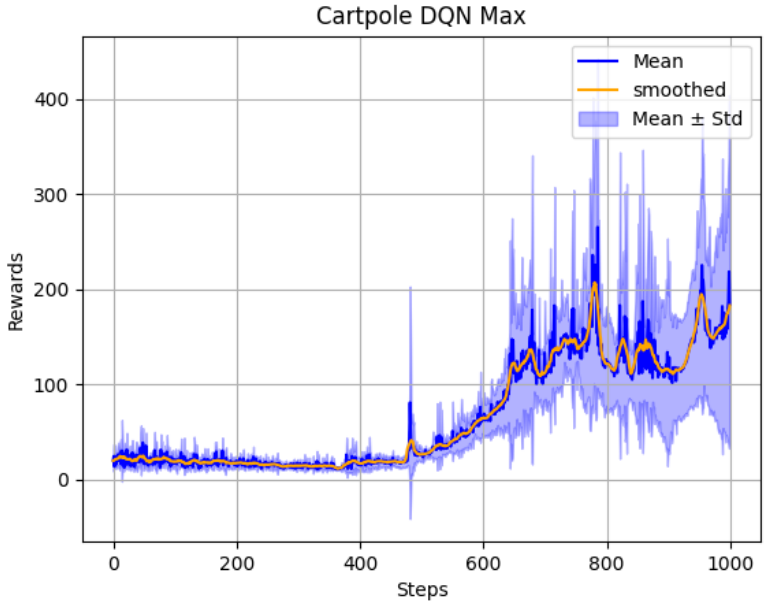

Figure 4: Cartpole_DQN_Max

The following reward plot got generated when the above algorithm got executed in the stated environment:

Comparison of Type 1 and Type 2 from the reward plots:

- **Learning Stability:**
  - **Type-1:** More stable due to variance reduction achieved by subtracting the mean advantage.
  - **Type-2:** Less stable due to potentially higher variance, cause the maximum advantage varies widely across actions.
- **Convergence Speed:**
  - **Type-1:** Convergence is a bit slower due to the additional computation to compute the mean advantage.
  - **Type-2:** Convergence is faster due to simpler computation, but suffers from fluctuations.
- **Overall Performance:**
  - **Type-1:** Provides smoother learning curves and more stable training, hence can be for environments with high variance or complex dynamics.
  - **Type-2:** Converges faster but exhibits more fluctuating behavior during training, potentially requiring more fine-tuning of hyper parameters.

0.2 Monte−Carlo REINFORCE

MC-REINFORCE (Monte Carlo REINFORCE) is a reinforcement learning algorithm used for policy optimization. It belongs to the family of policy gradient methods, which aim to directly optimize the policy parameters of an agent to maximize expected rewards. The key idea behind MC-REINFORCE is to estimate the gradient of the expected return with respect to the policy parameters using Monte Carlo sampling.

Let's break down the key components of the MC-REINFORCE algorithm:

1. **Policy Function**: The policy function, denoted by $\pi(A_t|S_t, \theta)$, specifies the probability of taking action A_t in state S_t given the policy parameters θ. It's essentially the agent's strategy for selecting actions.

2. **Objective Function**: The objective in reinforcement learning is typically to maximize the expected return, also known as the expected cumulative reward. It's denoted by $J(\theta)$, which is the expected value of the sum of rewards obtained by following policy π with parameters θ:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$$

Here, γ is the discount factor, and R_t is the reward obtained at time step t.

3. **Gradient Ascent**: The goal of MC-REINFORCE is to perform gradient ascent on the objective function $J(\theta)$ to update the policy parameters in the direction of higher expected returns.

Now, let's look at the update rules for MC-REINFORCE, both with and without a baseline:

Without Baseline:

$$\theta = \theta + \alpha G_t \nabla_\theta \log \pi(A_t|S_t, \theta)$$

With Baseline:

$$\theta = \theta + \alpha(G_t - V(S_t; \Phi))\nabla_\theta \log \pi(A_t|S_t, \theta)$$

In these equations:

- α is the learning rate, controlling the step size of the updates.
- G_t is the return obtained after taking action A_t in state S_t. It's defined as the sum of discounted rewards from time t onwards.
- $\nabla_\theta \log \pi(A_t|S_t, \theta)$ is the gradient of the log-probability of taking action A_t in state S_t with respect to the policy parameters θ.
- $V(S_t; \Phi)$ is the baseline value function. It's used to reduce the variance of the gradient estimates by providing a baseline for comparison. In this case, the baseline is updated using the TD(0) method.

TD(0) Method for Baseline Update: The TD(0) method (Temporal Difference with a single-step update) is a simple form of temporal difference learning. It updates the baseline value function $V(S_t; \Phi)$ towards the target value, which is the immediate reward plus the estimated value of the next state minus the current estimate.

$$V(S_t; \Phi) = V(S_t; \Phi) + \alpha_v(r_{t+1} + \gamma V(S_{t+1}; \Phi) - V(S_t; \Phi))$$

In this equation:

- r_{t+1} is the immediate reward obtained after taking action A_t in state S_t.
- α_v is the learning rate for updating the baseline value function.

By incorporating a baseline, the update rule adjusts the policy parameters based on the difference between the observed return and the expected return (approximated by the baseline). This helps in reducing the variance of the gradient estimates and can lead to more stable training.

Algorithm 5 MC-REINFORCE Algorithm without Baseline

Input: Replay buffer R, policy function $\pi(\cdot|\cdot, \theta)$, batch size B, discount factor γ
Output: Updated policy parameters θ
while training is not finished **do**
  if $|R| \geq B$ **then**
    Sample a mini-batch of experiences (s, a, r) from R
    Initialize gradient accumulator $\nabla_\theta J(\theta) \leftarrow 0$
    for each experience (s, a, r) in the mini-batch **do**
      Compute the return G_t for time step t:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$

      Compute the log-probability of the action:

$$\log \pi(a|s, \theta)$$

      Accumulate gradients:

$$\nabla_\theta J(\theta) \leftarrow \nabla_\theta J(\theta) + \frac{1}{B} G_t \nabla_\theta \log \pi(a|s, \theta)$$

    end for
    Update policy parameters:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

  end if
end while

0.2.1 In Acrobot−V1 environment

Now the following algorithm has been implemented in order to obtain the following plot.

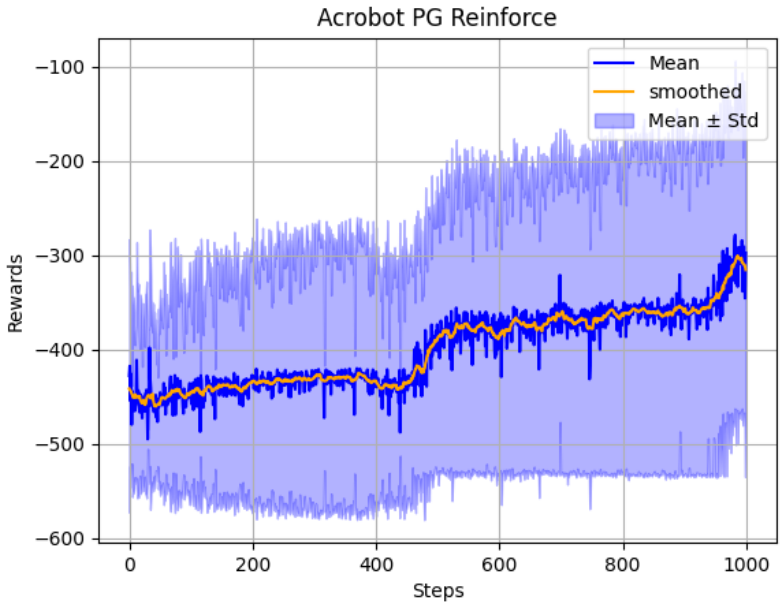

Figure 5: Acrobot_PG_Reinforce

Algorithm 6 MC-REINFORCE Algorithm with Baseline

Input: Replay buffer R, policy function $\pi(\cdot|\cdot, \theta)$, baseline function $V(s; \Phi)$, batch size B, discount factor γ, learning rate α
Output: Updated policy parameters θ
while training is not finished **do**
  if $|R| \geq B$ **then**
    Sample a mini-batch of experiences (s, a, r) from R
    Initialize gradient accumulator $\nabla_\theta J(\theta) \leftarrow 0$
    for each experience (s, a, r) in the mini-batch **do**
      Compute the return G_t for time step t:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$

      Compute the log-probability of the action:

$$\log \pi(a|s, \theta)$$

      Compute the baseline value:

$$b = V(s; \Phi)$$

      Compute the advantage:

$$A(s, a) = G_t - b$$

      Accumulate gradients:

$$\nabla_\theta J(\theta) \leftarrow \nabla_\theta J(\theta) + \frac{1}{B} A(s, a) \nabla_\theta \log \pi(a|s, \theta)$$

    end for
    Update policy parameters:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

    Update the baseline parameters:

$$\Phi \leftarrow \Phi + \beta \nabla_\Phi (G_t - V(s; \Phi))^2$$

  end if
end while

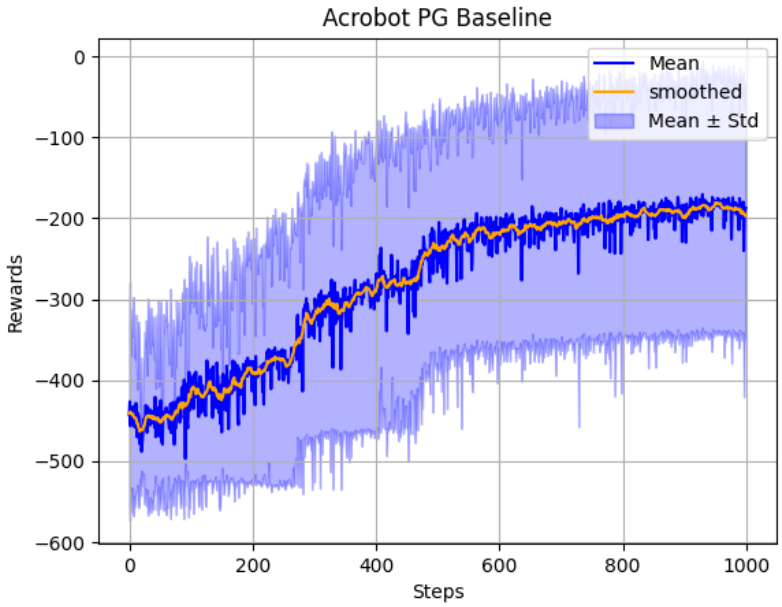

Figure 6: Acrobot_PG_Baseline

Comparison of Type 1 and Type 2 from the reward plots:

1. **Variance in Reward Plot:** Type 2 algorithm exhibits lower variance in the reward plot compared to Type 1 due to the incorporation of a baseline. This results in more stable learning dynamics.

2. **Convergence Speed:** Type 2 algorithm converges faster than Type 1 due to the reduced variance in gradient estimates. Basically it exploits the advantages of the baseline to converge towards a better policy more efficiently.

3. **Robustness:** Type 2 algorithm is more robust to noisy or sparse rewards compared to Type 1. The baseline helps in smoothing out fluctuations in the gradient estimates, making the learning process more resilient.

4. **Dependency on Baseline:** Type 2 algorithm's performance is sensitive to the choice of baseline function and its parameters. A poorly chosen baseline leads to sub optimal performance or even destabilize the learning process.

0.2.2 In CartPole−V1 environment

Algorithm 7 MC-REINFORCE Algorithm without Baseline in Cartpole Environment

Input: Policy parameter θ, learning rate α
Output: Updated policy parameter θ
while training is not finished **do**
    Sample a trajectory $[(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots]$ following policy $\pi(\cdot|\cdot, \theta)$
    for each time step t in the trajectory **do**
        Compute return G_t from time t onwards:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$

        Update policy parameter:

$$\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi(a_t|s_t, \theta)$$

    end for
end while

The following reward curve has been generated by executing the stated algorithm in the stated environment environment.

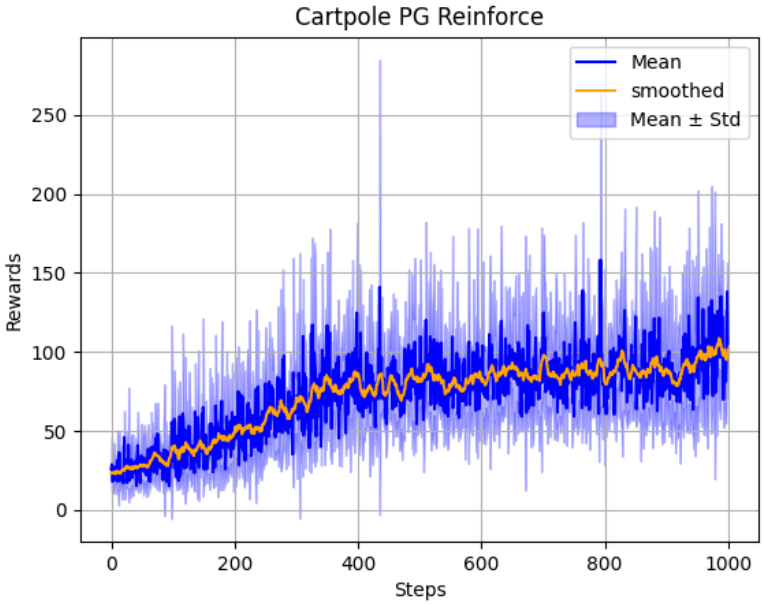

Figure 7: Cartpole_PG_Reinforce

Algorithm 8 MC-REINFORCE Algorithm with Baseline in Cartpole Environment

Input: Policy parameter θ, baseline parameter Φ, learning rate α
Output: Updated policy parameter θ, baseline parameter Φ
while training is not finished **do**
    Sample a trajectory $[(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots]$ following policy $\pi(\cdot|\cdot, \theta)$
    for each time step t in the trajectory **do**
        Compute return G_t from time t onwards:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$

        Compute baseline estimate $V(s_t; \Phi)$
        Update policy parameter:

$$\theta \leftarrow \theta + \alpha(G_t - V(s_t; \Phi)) \nabla_\theta \log \pi(a_t|s_t, \theta)$$

        Update baseline parameter:

$$\Phi \leftarrow \Phi + \beta(G_t - V(s_t; \Phi)) \nabla_\Phi V(s_t; \Phi)$$

    end for
end while

The following reward curve has been generated by executing the stated algorithm in the stated environment environment.

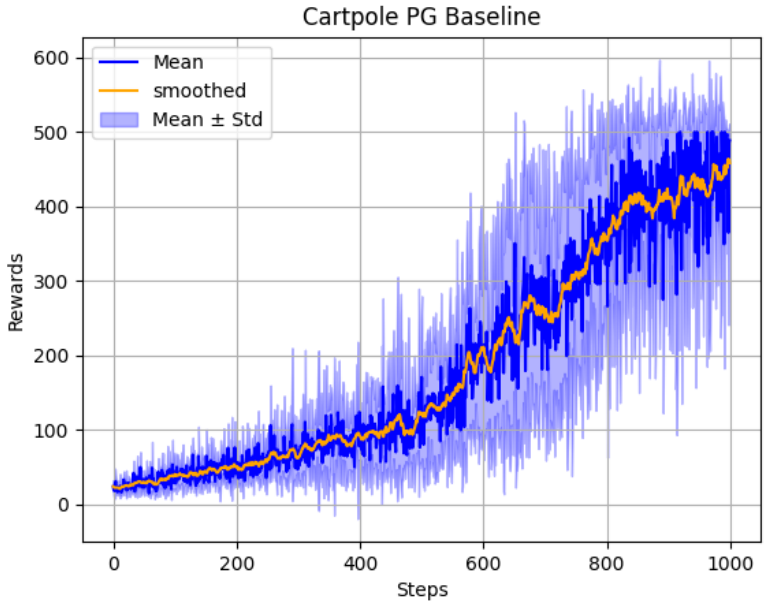

Figure 8: Cartpole_PG_Baseline

Comparison of Type 1 and Type 2 from the reward plots:

- **Performance:** The REINFORCE algorithm with a baseline again performs better in terms of stability and convergence speed compared to the variant without a baseline. This is because the baseline helps reduce the variance of gradient estimates, leading to smoother learning curves and more reliable training.
- **Reward Plot:** When comparing the reward plots of the two algorithms, the REINFORCE algorithm with a baseline shows smoother and more consistent improvement in rewards over time. On the other hand, the variant without a baseline exhibits more erratic behavior, with larger fluctuations in rewards due to higher variance in gradient estimates.
- **Training Efficiency:** The algorithm with a baseline requires fewer training iterations to achieve comparable performance to the variant without a baseline. This is because it makes more efficient use of the sampled trajectories by reducing the impact of noisy gradient estimates.

In summary, while both variants of the Monte Carlo REINFORCE algorithm can learn to solve the Cart-pole environment, the version with a baseline offers better stability and convergence properties, as well as potentially faster training. However, the choice between the two depends on factors such as computational resources, training time constraints, and desired level of performance.

Here is the GitHub link for all the codes that were deployed: `https://github.com/imnataraj/CS6700-PA-2`