Point-v0

1.

$$\begin{aligned}\nabla_\theta log\pi_\theta &= \nabla_\theta(log\frac{1}{2\pi} - \frac{(a-\theta^T\hat{s})^T I(a-\theta^T\hat{s})}{2}) \\ &= -\frac{1}{2}\nabla_\theta(a-\theta^T\hat{s})^T I(a-\theta^T\hat{s})\end{aligned} \tag{1}$$

We know that

$$(a-\theta^T\hat{s})^T I(a-\theta^T\hat{s}) = (a_0-\mu_0)^2 + (a_1-\mu_1)^2 = a_0^2 - 2a_0\mu_0 + \mu_0^2 + a_1^2 - 2a_1\mu_1 + \mu_1^2 \tag{2}$$

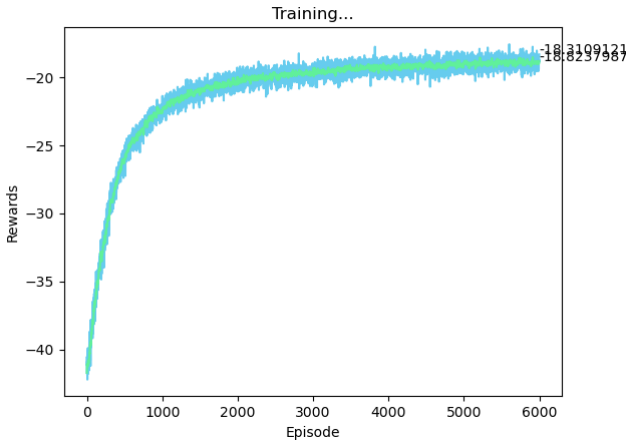
$$\therefore \nabla_\theta(a-\theta^T\hat{s})^T I(a-\theta^T\hat{s}) = -2a_0\nabla_\theta\mu_0 + \nabla_\theta\mu_0^2 - 2a_1\nabla_\theta\mu_1 + \nabla_\theta\mu_1^2 \tag{3}$$

$$\begin{cases} -2a_0\nabla_\theta\mu_0 = \begin{bmatrix} -2a_0s_1 & -2a_0s_2 & -2a_0 \\ 0 & 0 & 0 \end{bmatrix} \\ -2a_1\nabla_\theta\mu_1 = \begin{bmatrix} 0 & 0 & 0 \\ -2a_1s_1 & -2a_1s_2 & -2a_1 \end{bmatrix} \\ \nabla_\theta\mu_0^2 = \begin{bmatrix} 2\theta_{00}s_1^2 + 2\theta_{01}s_1s_2 + 2\theta_{02}s_1 & 2\theta_{01}s_2^2 + 2\theta_{00}s_1s_2 + 2\theta_{02}s_2 & 2\theta_{02} + 2\theta_{00}s_1 + 2\theta_{01}s_2 \\ 0 & 0 & 0 \end{bmatrix} \\ \nabla_\theta\mu_1^2 = \begin{bmatrix} 0 & 0 & 0 \\ 2\theta_{10}s_1^2 + 2\theta_{11}s_1s_2 + 2\theta_{12}s_1 & 2\theta_{11}s_2^2 + 2\theta_{10}s_1s_2 + 2\theta_{12}s_2 & 2\theta_{12} + 2\theta_{10}s_1 + 2\theta_{11}s_2 \end{bmatrix} \end{cases} \tag{4}$$

$$\therefore -\frac{1}{2}\nabla_\theta(a-\theta^T\hat{s})^T I(a-\theta^T\hat{s}) = \begin{bmatrix} -\theta_{00}s_1^2 - \theta_{01}s_1s_2 - \theta_{02}s_1 + a_0s_1 & -\theta_{01}s_2^2 - \theta_{00}s_1s_2 - \theta_{02}s_2 + a_0s_2 & -\theta_{02} - \theta_{00}s_1 - \theta_{01}s_2 + a_0 \\ -\theta_{10}s_1^2 - \theta_{11}s_1s_2 - \theta_{12}s_1 + a_1s_1 & -\theta_{11}s_2^2 - \theta_{10}s_1s_2 - \theta_{12}s_2 + a_1s_2 & -\theta_{12} - \theta_{10}s_1 - \theta_{11}s_2 + a_1 \end{bmatrix} \tag{5}$$

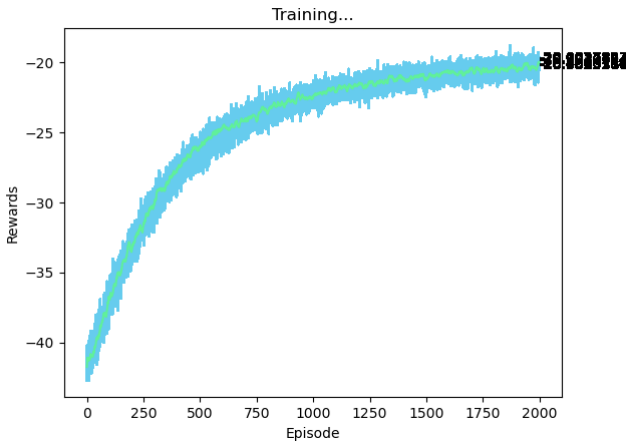
2. As for environment Point-v0, I completed 'update_params' function to update the policy π by different methods, like formula (1), (3) and (4) (bonus method). estimate_net_grad which is another function I should complete is used to compute return, log π from question(1).

The result of policy gradient method (1) in Point-v0 is below:



(1) method with seed 1

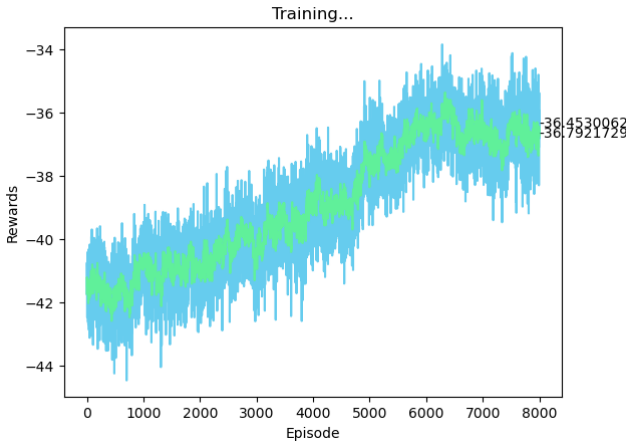
We can observe that the algorithm finally converged to -18. And the algorithm has great learning speed, it is so quick to get the high reward with convergence. However, We can say that the (1) method has a high variance according to the result below when I try different seeds to see what the variance is going on.



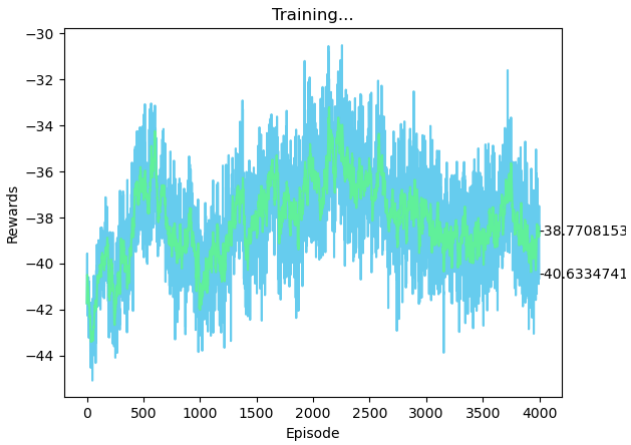
(1) method with different seeds

It is easy to see that the performance of the algorithm is very sensitive to different seeds, though the final results are the same. The theoretical reason is that we only update the policy with return R in (1) method. We know that return R is the sum of discounted trajectory reward. It is reasonable to say that the trajectory agent acted is highly successional, so whether the trajectory is good or not depends on the performance of policy. At the begin, the performance of the policy is uncertain, we can't obtain a good trajectory exactly, so the trajectories which have high variance lead to the high variance of the algorithm.

As for (3) method, it is difficult to converge with the value approximation because of the learning difficulty of the value approximator(critic), like DQN, TD(0) **policy gradient**. This character is proved in my experiments. So here I can't compare (3) method to (1) method.



(3) method with seed 1



(3) method with seed 2021

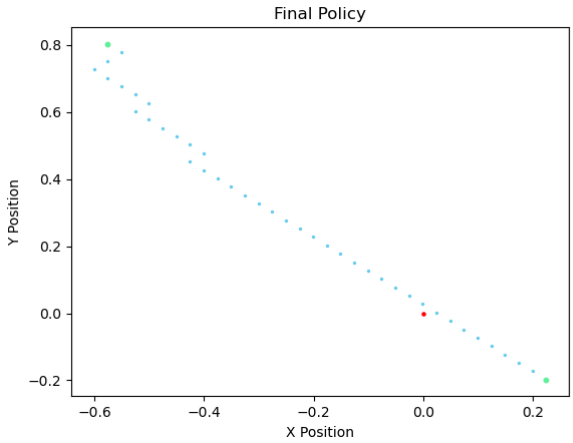
As for the average return after 100 iterations of the algorithm, the result is below:

```
0   Time_sample 0.3971  Time_update 1.6003  train_R_avg -41.72  eval_R_avg -41.67
10  Time_sample 0.4320  Time_update 1.5149  train_R_avg -41.15  eval_R_avg -40.60
20  Time_sample 0.3662  Time_update 1.6666  train_R_avg -40.67  eval_R_avg -41.21
30  Time_sample 0.3697  Time_update 1.5776  train_R_avg -39.92  eval_R_avg -39.32
40  Time_sample 0.3823  Time_update 1.5749  train_R_avg -39.42  eval_R_avg -39.73
50  Time_sample 0.4110  Time_update 1.5577  train_R_avg -38.93  eval_R_avg -38.73
60  Time_sample 0.4451  Time_update 1.6495  train_R_avg -38.51  eval_R_avg -38.17
70  Time_sample 0.4163  Time_update 1.6023  train_R_avg -37.77  eval_R_avg -37.55
80  Time_sample 0.4029  Time_update 1.6862  train_R_avg -38.24  eval_R_avg -38.76
90  Time_sample 0.3844  Time_update 1.5244  train_R_avg -36.94  eval_R_avg -38.69
100 Time_sample 0.4006  Time_update 1.5067  train_R_avg -36.11  eval_R_avg -36.41
```

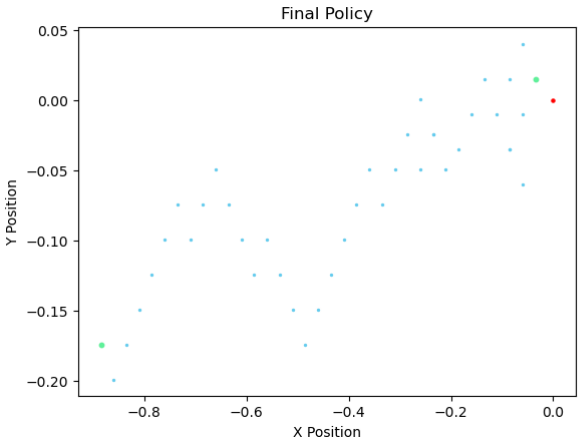
The Average Return After 100 Iterations of The Algorithm

It improves slightly in iteration 100, but we can see that the policy is good enough to get high reward -18 which **is close to the optimal policy** according to the training curve above.

We can observe the final policy from method (1) and (4) (bonus method), we can see the trajectory and conclude that the point tries to reach the origin as fast as possible in 2 axis(x & y). It always choose the closest distance as the target trajectory. We can say that the algorithm finally finds the optimal policy.

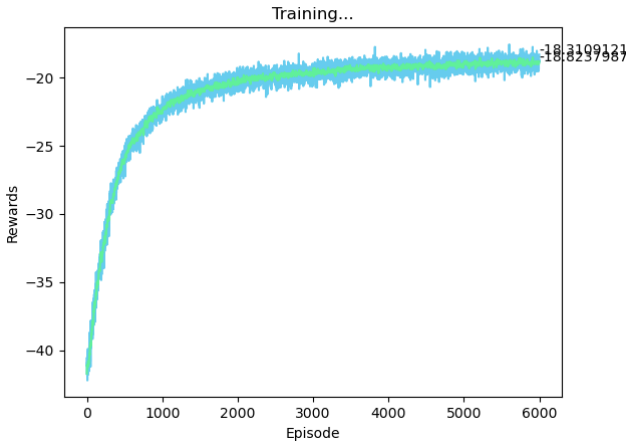


(1) method final policy



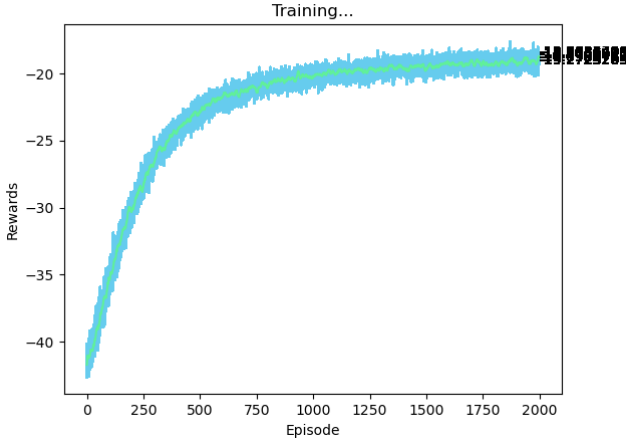
(4) method final policy

3. Bonus:

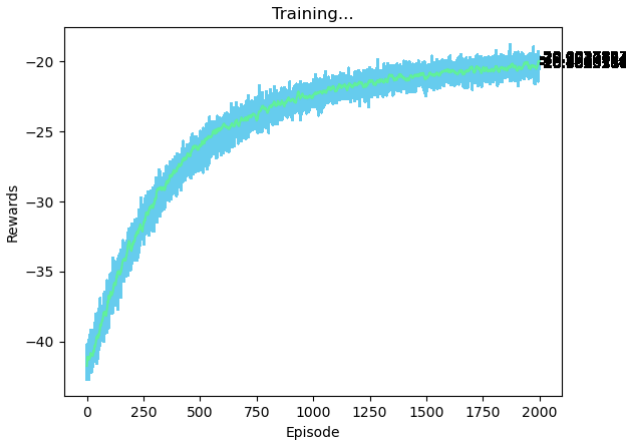


(4) method with seed 1

According to the experiment results of (4) method (minus the expectation of the return R) below, we can say that a baseline will improve the performance of these algorithms. Of course, we need compare to (1) method above.



(4) method with different seeds



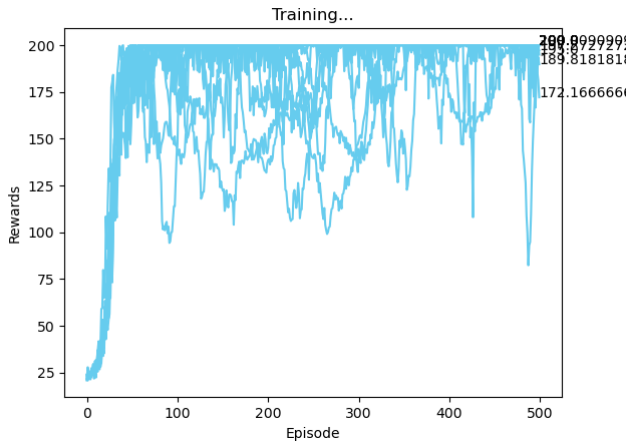
(1) method with different seeds

The results show that the algorithm also has great learning speed, it is so quick to get the high reward with convergence. Meanwhile, the algorithm has a low variance when I try different seeds to see what the variance is going on. The results are similar with different seeds. So the policy in this method with a simply baseline is not sensitive to the different seeds.

CartPole-v0

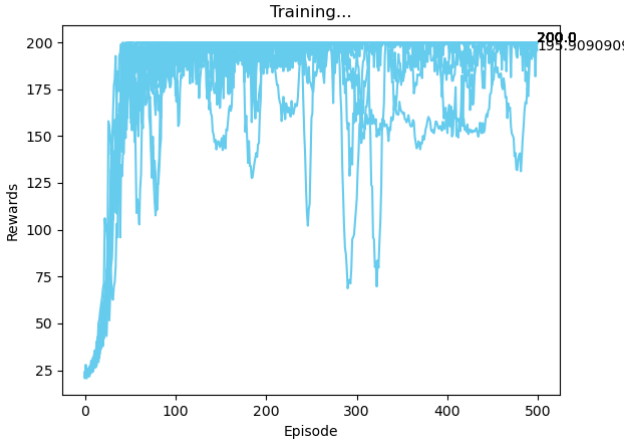
---

1. The curve of (1) policy gradient method is below:



2. (3) policy gradient method converges very slowly. It is difficult to converge with the value approximation because of the learning difficulty of the value approximator(critic), like DQN, TD(0) **policy gradient**. This character is proved in my experiments. So I can't plot its average reward curve.

3. The curve of (4) policy gradient method is below:



4. Summary:

We can observe that the 2 algorithms below and conclude that these algorithm finally converged to 200(the best) quickly with the similar speed. So we can say that the performances of these 2 algorithm are the same. And the algorithms have great learning speed, it is so quick to get the high reward with convergence. However, We can easily compare these 2 algorithm with different seeds and conclude that the (1) method has a high variance according to the result above when I try different seeds to see what the variance is going on. The performance of the algorithm is very sensitive to different seeds, though the final results are the same.

As for (4) method in another place, the results show that the algorithm has a low variance when I try different seeds to see what the variance is going on. The results are similar with different seeds. So the policy in this method with a simply baseline is not sensitive to the different seeds. So it is the effect of the baseline, we can implement a algorithm which has the same performance but low variance. It can stabilize the performance.

And when we implement the policy gradient method, it is good to choose the method which has return R with a baseline (whatever the baseline is, e.g. E(R), state values). It is reasonable not to choose the method with value approximator which converges difficultly.