# Policy Gradient Method Report

The report mainly contains five parts, introduction, problem formulation, technical approach, training results and brief summary.

The key points in the project includes implementation of value iteration, policy evaluation and policy gradient optimization of MDP (Markov Decision Process); the influence of discount factor gamma; the approaches to solve a discrete – action cart – pole balancing task and modify it into continuous space to work on a Pendulum balancing problem.

## I. Introduction

This project is aimed to implement several methods for dealing with the Frozen Lake problem. The whole situation can be represented via a Markov Decision Process (MDP) and a strategy for retrieving the Frisbee can be obtained using value iteration (VI), policy iteration (PI), and policy gradient optimization (PGO). Once learned to retrieve the target Frisbee, we can use the PGO implementation as well as a more sophisticated proximal policy optimization to learn how to balance a pendulum or even play with Atari games.

## II. Problem formulation

### A. *Problem Modeling*

For this project, we are given a certain concrete situation and need to figure out optimal strategy in order to achieve goal. The first important issue is modeling this problem via suitable algorithm or structure. For example, we need to create a suitable structure to represent states of frozen lake, actions to show different kinds of movement. It’s also necessary to come up with an approach to represent policy and value information.

In a word, the proper model will truly do favor to figure out the optimal strategy.

### B. *The understanding of Algorithm*

It’s not easy to figure out the algorithm in this project. The problem we need to deal with is essentially a reinforcement learning problem, what’s more, we need to add some assumption to simply the problem. For another thing, it’s pretty important to figure out various variables, such as states, actions, policy, reward, state value, action value, agent, environment and so on.

In addition, there exists several approaches to obtain optimal strategy and each of them has different properties, therefore, it’s necessary to figure out which one is better and can be generalized for other similar problems.

### C. *More Complicated but Practical Problems*

Once we obtain sophisticated approach to compute optimal strategy, it’s recommended to apply it for dealing with more practical problems, such as the cart – pole problem in discrete space and Pendulum balancing task in continuous space.

It’s not easy since we cannot just use the PGO code in different problems directly. What we need to do is modifying algorithm according to practical problems.

## III. Technical Approach

This section shows some several technical approaches in my code to deal with mentioned problems above.

### A. *Value Iteration (VI)*

Value iteration is one of three basic approaches to compute the optimal strategy or policy for this problem.

The basic idea is, given certain policy, we use backward operation to estimate how good the agent can achieve at certain state. And for backward procedure, similar to the DP, we consider the problem start with simple case, so we start from the last time step which we can consider all possible actions as well as rewards, then based on the computed result to obtain value in previous time steps.

Practically, we need two value functions to achieve VI. One is state value function V which only depends on variable states. The other one is action value function Q which depends on state {s} and action {a}.

The following figure shows the pipeline of VI.

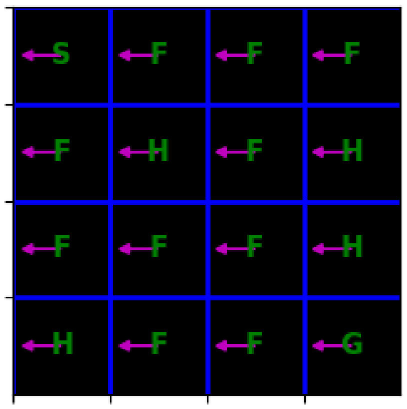
```
input   initial state $b_0$ and horizon $T$
$V_T^*(b) \leftarrow \rho_T(b), \quad \forall b \in \mathcal{B}$
for $t = (T-1)\ldots 0$
    $Q_t^*(b,u) \leftarrow \rho_t(b,u) + \gamma \int \eta(z \mid b,u) V_{t+1}^*(\psi(b,u,z))\,dz, \quad \forall b \in \mathcal{B}, u \in \mathcal{U}$
    $V_t^*(b) \leftarrow \max_{u \in \mathcal{U}} Q_t^*(b,u), \qquad \forall b \in \mathcal{B}$
    $\pi_t^*(b) \leftarrow \arg\max_{u \in \mathcal{U}} Q_t^*(b,u), \qquad \forall b \in \mathcal{B}$
end
return   $\pi_{0:T-1}^*$ and $V_0^*(b_0)$
```

The key equation we need to follow is below one, which represents that in VI, state value only depends on the state without considering action, and then we choose our optimal action based on current update value vector.

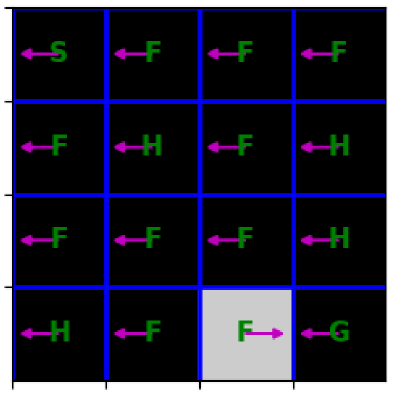
$$V_{K+1}^{\ *}(x) = \max_u \sum_{x'} T(x,u,x')\left[R(x,u,x') + \gamma V_K^*(x')\right]$$

Below, I will show some grid cell figures to illustrate the progress of Value Iteration. The optimal policy of each cell or state is shown by arrows.

The initial state shows a random policy without iteration.

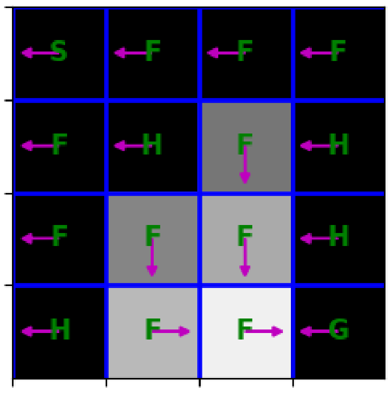


The grid after first iteration shows below, and you can see a obvious backup process which starts from the cell nearby the final state G. And the policy also updates.

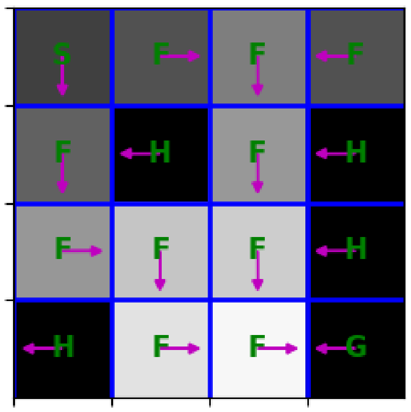


After several iterations, the grid becomes pretty reasonable which shows several trajectories starts from different states.

4th iteration result

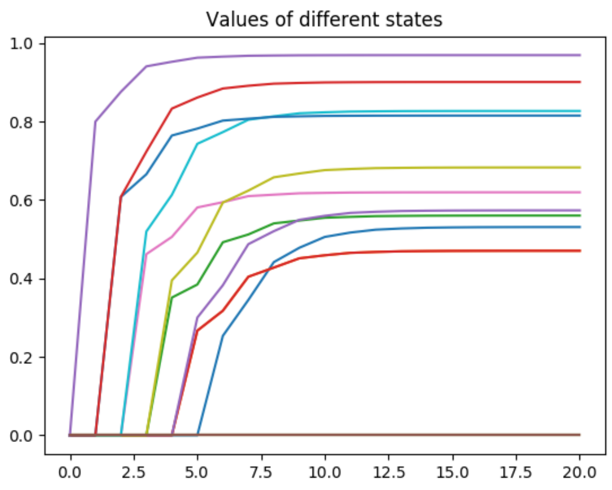


7th iteration result



After some certain times of iteration, the policy will not update anymore which means the algorithm has already converged.

According to the result above, we can verify the accuracy of value iteration via those reasonable trajectories as well as policy. Below, I show one more figure to make the iteration process much clearer.

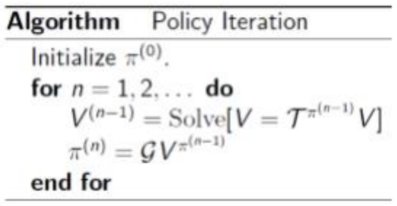


The figure above tells that different states have various state value, and after certain iteration times, their value keep same which shows the convergence state.

## B. *Policy Iteration (PI)*

The basic idea of policy iteration is initializing a policy and then performs two evaluations of state and action value functions on every each iteration time.

The policy iteration contains two steps, policy evaluation and policy improvements. The main part is policy evaluation which contains state value function evaluation and action value function evaluation. Then we update policy via updated action value function. Finally, the algorithm will converge to an optimal policy. Below figure shows the pipeline of the policy iteration.



```
Algorithm  Policy Iteration
  Initialize $\pi^{(0)}$.
  for $n = 1, 2, \ldots$ do
      $V^{(n-1)} = \text{Solve}[V = \mathcal{T}^{\pi^{(n-1)}} V]$
      $\pi^{(n)} = \mathcal{G} V^{\pi^{(n-1)}}$
  end for
```

For the evaluation of state value function V, the basic idea follows the full backup algorithm, the equation shows below.

$$V_{k+1}^{\pi}(x) = \sum_{u} \pi(u \mid x)\left[ r(x,u) + \gamma \sum_{x'} p(x' \mid x,u) V_k^{\pi}(x') \right]$$

Alternatively, the above equation can be solved directly via linear programming $\left(I - \gamma P^{\pi}\right)V^{\pi} = R^{\pi}$.

Once we get the updated state value function, then comes to the evaluation of action value function Q. The algorithm is pretty similar to one in value iteration.

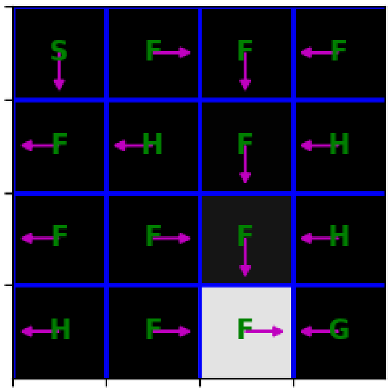
$$Q^{\pi}(x,u) = r(x,u) + \gamma \sum_{x'} p(x' \mid x,u) V^{\pi}(x')$$

For policy improvement section, there is an assumption that the policy $\pi'$ is at least as good as $\pi$ if satisfying the requirement $V^{\pi'}(x) \geq V^{\pi}(x)$ for all states. Therefore, we can improve the current policy via selecting the max value of action state function Q along action dimension.

In a word, there are total two main steps in policy iteration, first, policy evaluation will make the value function consistent with the current policy. Then the policy improvement makes the policy greedy with respect to the current value function.

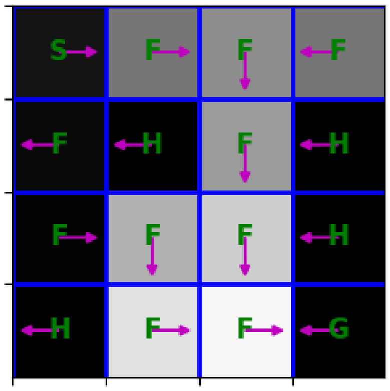
Below, similar to VI part, I will show several key points during the iteration process.

Compared with VI result, the first iteration result of PI shows more information in policy since we take the policy as parameters.

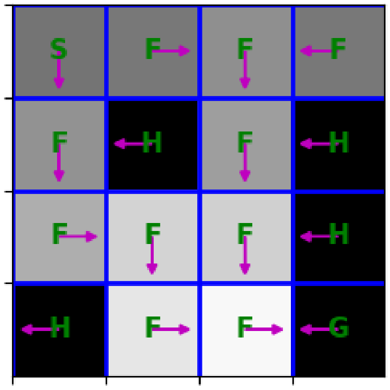


What's more, the speed of convergence of PI is quite faster than VI, the below figure is the result of second iteration time and the policy totally convergence to the optimal state after 4 times iteration while VI takes 8 iteration times.

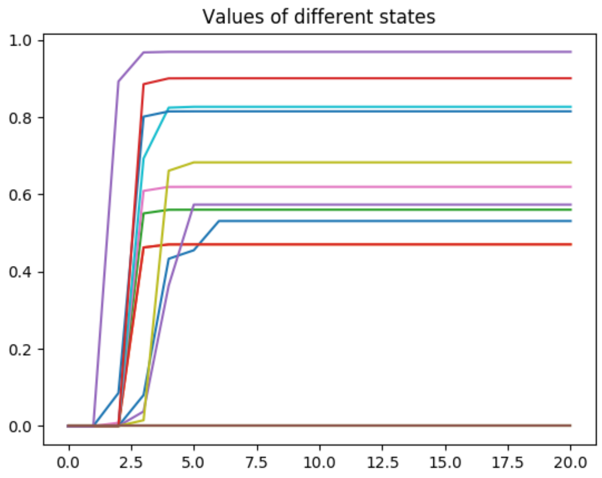
2nd iteration result



5th iteration result



The optimal policy of PI is almost same as VI which also verifies its accuracy. What's more, the speed of PI is quite faster than VI so that it will bring a benefit for larger grid cell case or more complicated problem.



As the figure shows, the convergence speed of PI is faster than VI and final optimal policy is almost same, that is one of advantages of policy iteration.

## C. *Comparison of Value Iteration (VI) and Policy Iteration (PI )*

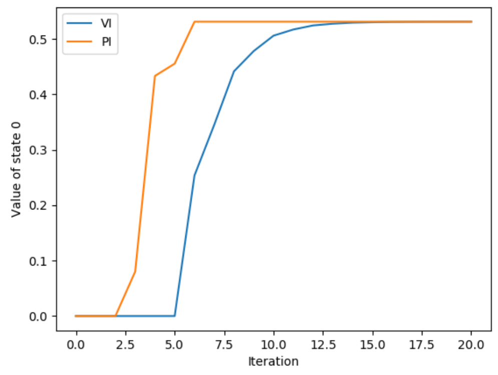
The process of Value Iteration (VI) and Policy Iteration (PI) have some consistent, such as they both need to compute two value functions, and both aim to make the policy converge in order to get the optimal one.

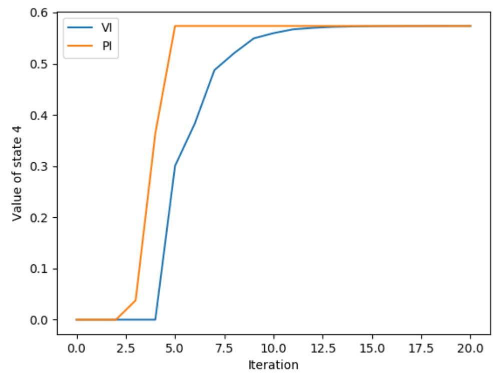
However, there exist some differences between VI and PI.

For Value Iteration, every iteration will update both values and the policy, and the algorithm don't track the policy but tracking the max value over actions implicitly recomputes it, that is why the speed of VI is slower than PI.

For Policy Iteration, we do several passes that update value functions with current optimal policy which is fixed, each pass is fast since the algorithm only considers one action according to the policy instead all of them.

Below figures show the speed difference of VI and PI clearly.





It's obvious that the convergence speed of PI is faster than VI and final optimal result is almost same. Therefore, to some extent, policy iteration performs better than value iteration.

## D. *Policy Gradient Optimization (PGO)*

Besides the above two optimization approaches, policy gradient optimization which we called PGO is also an alternative method.

The basic idea of PGO is learning a parameterized policy that specifies a pdf / pmf over controls u without consulting a value function.

The intuition is, collect a bunch of trajectories first, then make good trajectories more probable as well as good actions according to its discounted reward, and finally push actions towards good ones. However, similar to gradient descent algorithm, if we consider all collected trajectories to compute one single trajectory, the whole procedure will cost so much. Therefore, according to the stochastic gradient descent algorithm, we take each trajectory into account for gradient change, then sum all change up, compute its mean gradient value as simulation expectation operation.

There are two steps in PGO, gradient computation given by each sample trajectory and implement gradient ascent. For gradient computation, the main algorithm shows below,

$$\hat{g}_N(\theta)=\frac{1}{N}\sum_{i=1}^{N}G\left(\tau^{(i)}\right)\log\pi\left(u_t\mid x_t,\theta\right)$$

Since each time we consider only one sample trajectory, so that N = 1.

The trick is the approach of computing advantage reward, 'adv_n' which represents the discounted total future reward - sum of rewards from now to the time horizon with the discount factor. The basic equation shows below,

$$\left[G_0^T\left(x_{0:T},u_{0:T-1}\right),\gamma G_1^T\left(x_{1:T},u_{1:T-1}\right),\gamma^2G_2^T\left(x_{2:T},u_{2:T-1}\right),\ldots,\gamma^kG_k^T\left(x_{k:T},u_{k:T-1}\right),\ldots\right]$$

## E. *Cart – Pole and Pendulum Balancing Problem*

For this problem, we need to apply policy gradient optimization algorithm into the more complicated problem, cart – pole and pendulum balancing problem. In order to make the performance better, we use neural network to training the policy. The main idea is, we need to transfer discrete – space problem into continuous – space.

Basically, for discrete control space, we define the policy as below equation,

$$\pi\left(u\mid x,\theta\right)=\frac{e^{h(x,u,\theta)}}{\sum_a e^{h(x,a,\theta)}}$$

The policy is a softmax function of control preferences $h\left(x,u,\theta\right)$ which can be obtained from a Deep Neural Net or via $h\left(x,u,\theta\right)=\theta^T\phi\left(x,u\right)$.

For continuous one, we define the policy as a Gaussian distribution,

$$\pi\left(u\mid x,\theta\right)=\phi\left(u;\mu\left(x,\theta\right),\sum\left(x,\theta\right)\right)$$

Its corresponding mean $\mu(x,\theta)$, obtained from a Deep Neural Net or via $\mu(x,\theta) \coloneqq \theta_\mu^T\phi\left(x,u\right)$, and $\Sigma\ (x,\theta)$ = $\boldsymbol{diag}(\sigma 2\ (x,\theta))$ with standard deviations $\sigma(x,\theta)$ obtained from Deep Neural Net or via $\sigma\left(x,\theta\right)=\exp\left(\theta_\sigma^T\phi\left(x\right)\right)$ for some state feature vector $\phi\left(x\right)$.

For practical implementation, I try to transfer cart – pole game from discrete space into continuous space. Below shows the basic procedure of space transformation.

a. Take the output of last layer as variable of mean and variance computation.

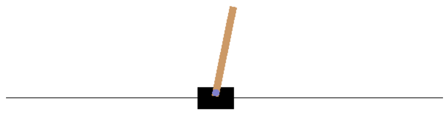
b. Generate a Gaussian distribution by computed mean and variance value, and use this pdf to replace softmax function which is used in discrete space.

c. Sample some action value from given Gaussian pdf and transform into log form.

d. Use new calculated log probability to compute corresponding loss value and entropy for NN training.

However, the code can work and implement iterations, the problem is that the optimal policy might not converge at all since iteration value during the iteration procedure almost keeps same.

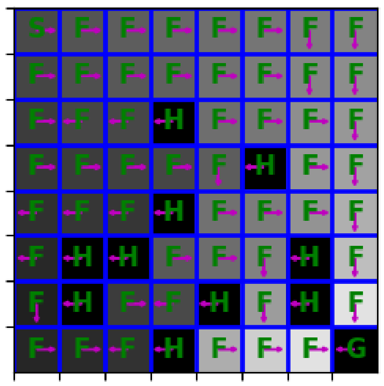
Below shows the figure of cart – pole obtained from my algorithm, the state is always static without significant changing. I have tried my best to figure out that problem, but still cannot handle well.



## F. Discount Factor Issue

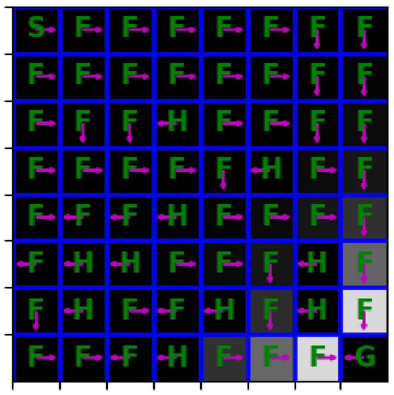
The aim of reinforcement learning or more specific one MDP is to get an optimal policy which can bring the maximum reward in the long run. Therefore, the discounted factor gamma matters a lot. If gamma is smaller, it means the policy attaches more importance on current obtained reward, that is, the policy will be greedy and willing to gaining more rewards as soon as possible. Different discount factor will lead to different policy. In order to make the policy show clear, I tried 8 by 8 grid.

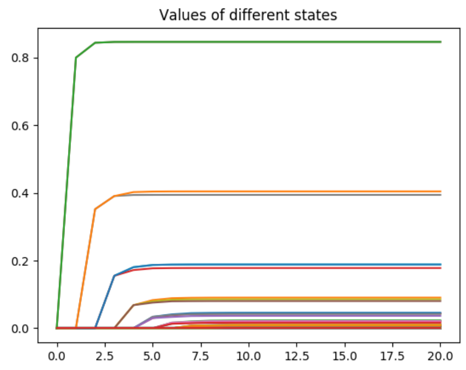
Frist, I try gamma with quite large value which close to 1.



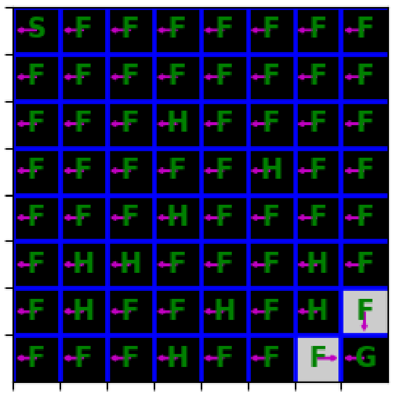
The optimal policy is reasonable, of course, since the discount factor is smaller than 1, so the policy would like to get to the goal place as soon as possible.

When I decrease the gamma value to 0.55, the expected reward for each state will decrease as well, but the policy will be almost same. In the map, the brighter the cell is, means the larger reward it can get.





When I choose an extreme case where gamma equal to 0, which means the feedback reward from the environment will be all same, in that case, the agent cannot do reasonable policy estimation and just perform random actions. For this problem, the action of most cell will keep same after multiple times iteration. Below shows the map after 10 times iteration.

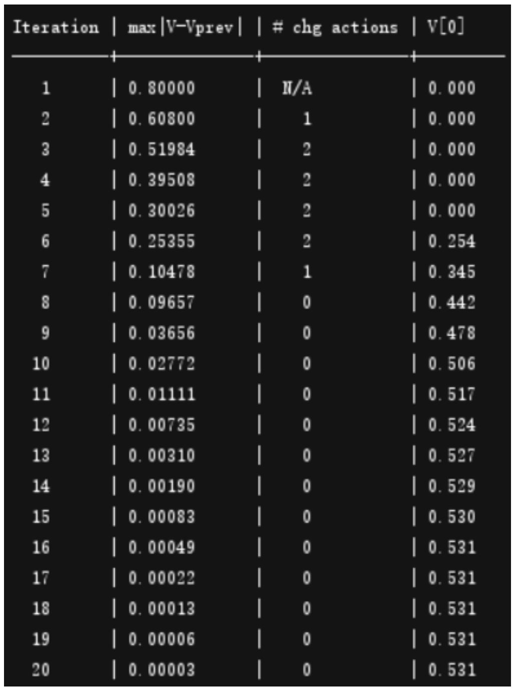


You can find that except two cells nearby the goal position, all other states perform same actions which means they cannot determine its action via feedback reward since I set gamma equal to 0.

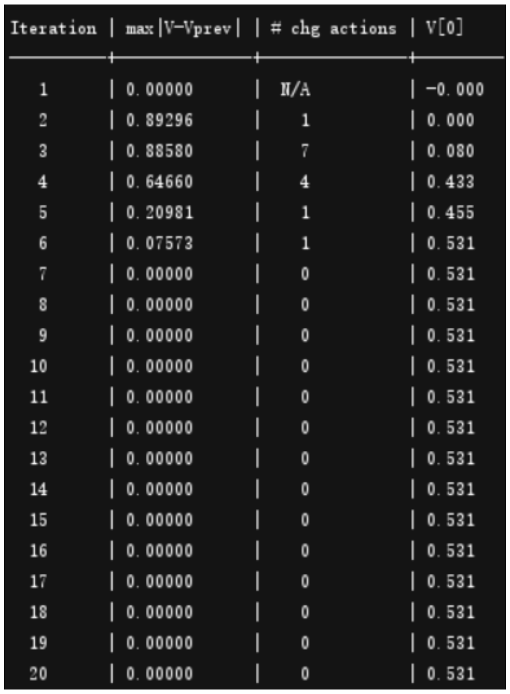
## IV. TEST RESULTS

Below shows the result of convergence process via value iteration, policy iteration, policy gradient optimization. (The label '# chg actions' represents total number of action change at current iteration.)

Value Iteration

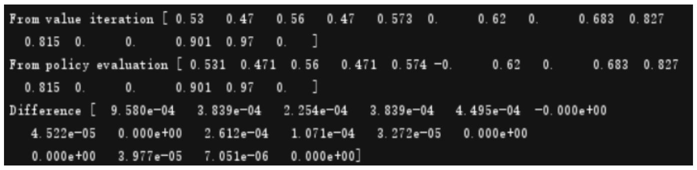

```
Iteration | max|V-Vprev| | # chg actions | V[0]
----------+--------------+---------------+------
    1     | 0.80000      | N/A           | 0.000
    2     | 0.60800      |   1           | 0.000
    3     | 0.51984      |   2           | 0.000
    4     | 0.39508      |   2           | 0.000
    5     | 0.30026      |   2           | 0.000
    6     | 0.25355      |   2           | 0.254
    7     | 0.10478      |   1           | 0.345
    8     | 0.09657      |   0           | 0.442
    9     | 0.03656      |   0           | 0.478
   10     | 0.02772      |   0           | 0.506
   11     | 0.01111      |   0           | 0.517
   12     | 0.00735      |   0           | 0.524
   13     | 0.00310      |   0           | 0.527
   14     | 0.00190      |   0           | 0.529
   15     | 0.00083      |   0           | 0.530
   16     | 0.00049      |   0           | 0.531
   17     | 0.00022      |   0           | 0.531
   18     | 0.00013      |   0           | 0.531
   19     | 0.00006      |   0           | 0.531
   20     | 0.00003      |   0           | 0.531
```

Policy Iteration

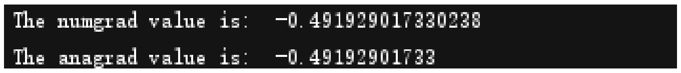

```
Iteration | max|V-Vprev| | # chg actions | V[0]
----------+--------------+---------------+------
    1     | 0.00000      | N/A           | -0.000
    2     | 0.89296      |   1           | 0.000
    3     | 0.88580      |   7           | 0.080
    4     | 0.64660      |   4           | 0.433
    5     | 0.20981      |   1           | 0.455
    6     | 0.07573      |   1           | 0.531
    7     | 0.00000      |   0           | 0.531
    8     | 0.00000      |   0           | 0.531
    9     | 0.00000      |   0           | 0.531
   10     | 0.00000      |   0           | 0.531
   11     | 0.00000      |   0           | 0.531
   12     | 0.00000      |   0           | 0.531
   13     | 0.00000      |   0           | 0.531
   14     | 0.00000      |   0           | 0.531
   15     | 0.00000      |   0           | 0.531
   16     | 0.00000      |   0           | 0.531
   17     | 0.00000      |   0           | 0.531
   18     | 0.00000      |   0           | 0.531
   19     | 0.00000      |   0           | 0.531
   20     | 0.00000      |   0           | 0.531
```

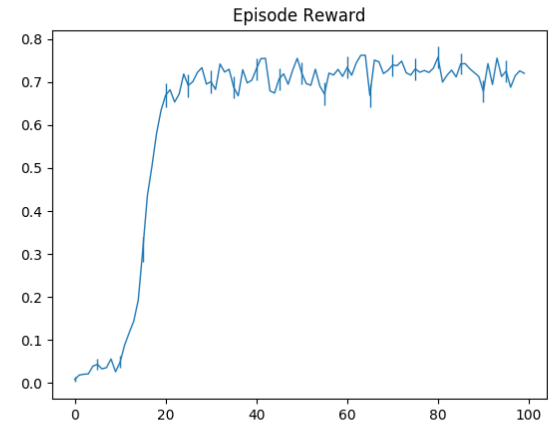
It's obvious that PI converge rapidly than VI.

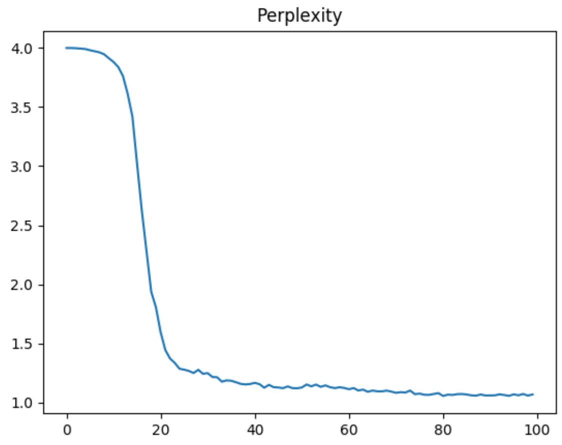
We can also implement a sanity check of two algorithm by comparing two value results and we can find that the difference between them is quite small.

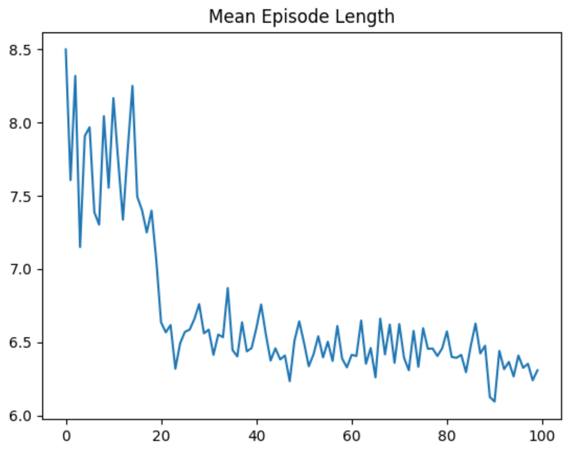

```
From value iteration [ 0.53   0.47   0.56   0.47   0.573  0.     0.62   0.     0.683  0.827
  0.815  0.     0.     0.901  0.97   0.    ]
From policy evaluation [ 0.531  0.471  0.56   0.471  0.574 -0.     0.62   0.     0.683  0.827
  0.815  0.     0.     0.901  0.97   0.    ]
Difference [  9.580e-04   3.839e-04   2.254e-04   3.839e-04   4.495e-04  -0.000e+00
   4.522e-05   0.000e+00   2.612e-04   1.071e-04   3.272e-05   0.000e+00
   0.000e+00   3.977e-05   7.051e-06   0.000e+00]
```

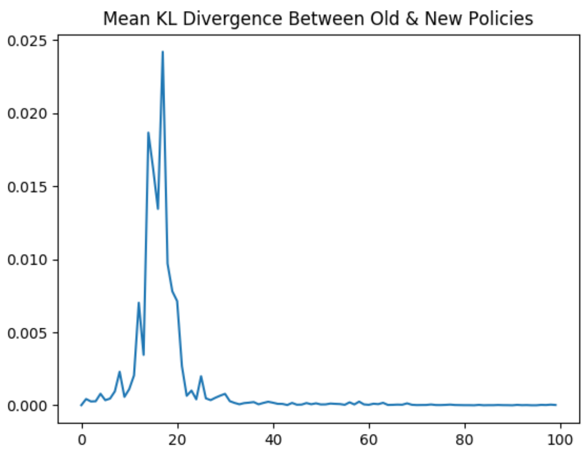
PGO

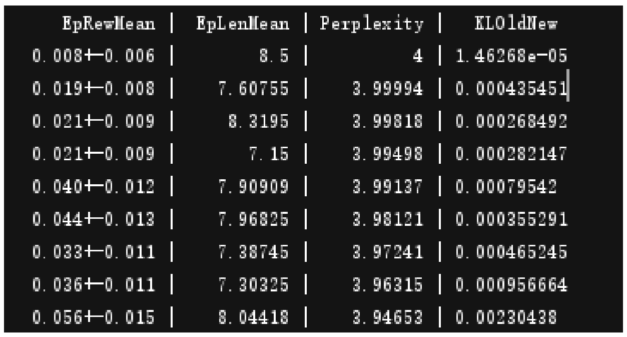

```
The numgrad value is:  -0.491929017330238
The anagrad value is:  -0.49192901733
```





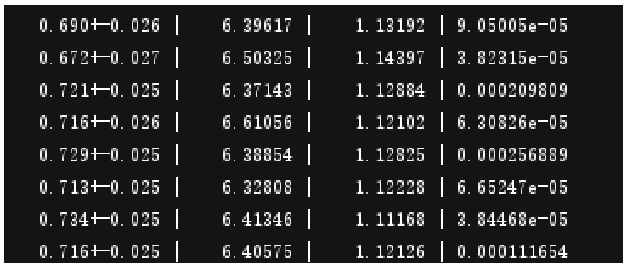






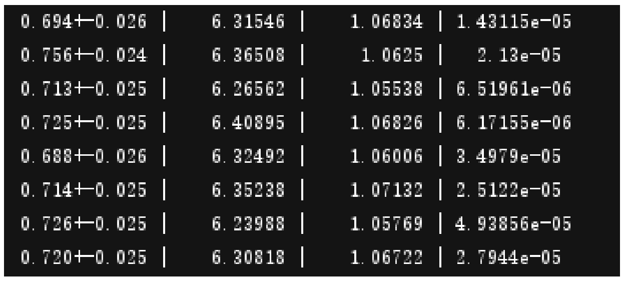
```
 EpRewMean  | EpLenMean | Perplexity |  KLOldNew
0.008+-0.006 |       8.5 |          4 | 1.46268e-05
0.019+-0.008 |   7.60755 |    3.99994 | 0.000435451
0.021+-0.009 |    8.3195 |    3.99818 | 0.000268492
0.021+-0.009 |      7.15 |    3.99498 | 0.000282147
0.040+-0.012 |   7.90909 |    3.99137 | 0.00079542
0.044+-0.013 |   7.96825 |    3.98121 | 0.000355291
0.033+-0.011 |   7.38745 |    3.97241 | 0.000465245
0.036+-0.011 |   7.30325 |    3.96315 | 0.000956664
0.056+-0.015 |   8.04418 |    3.94653 | 0.00230438
```

……….



```
0.690+-0.026 |   6.39617 |    1.13192 | 9.05005e-05
0.672+-0.027 |   6.50325 |    1.14397 | 3.82315e-05
0.721+-0.025 |   6.37143 |    1.12884 | 0.000209809
0.716+-0.026 |   6.61056 |    1.12102 | 6.30826e-05
0.729+-0.025 |   6.38854 |    1.12825 | 0.000256889
0.713+-0.025 |   6.32808 |    1.12228 | 6.65247e-05
0.734+-0.025 |   6.41346 |    1.11168 | 3.84468e-05
0.716+-0.025 |   6.40575 |    1.12126 | 0.000111654
```

………..

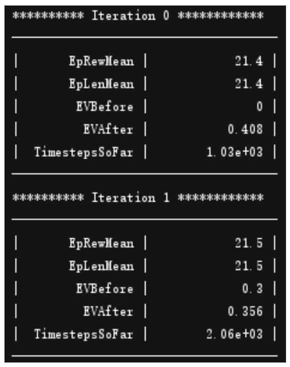


```
0.694+-0.026 |   6.31546 |    1.06834 | 1.43115e-05
0.756+-0.024 |   6.36508 |     1.0625 |   2.13e-05
0.713+-0.025 |   6.26562 |    1.05538 | 6.51961e-06
0.725+-0.025 |   6.40895 |    1.06826 | 6.17155e-06
0.688+-0.026 |   6.32492 |    1.06006 | 3.4979e-05
0.714+-0.025 |   6.35238 |    1.07132 | 2.5122e-05
0.726+-0.025 |   6.23988 |    1.05769 | 4.93856e-05
0.720+-0.025 |   6.30818 |    1.06722 | 2.7944e-05
```

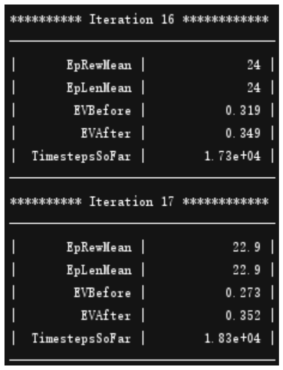
The result above looks reasonable. For the first argument, EpRewMean, this represents the mean reward given by current optimal trajectory. The average tendency of EpRewMean is increasing which means the policy is converging to a local optimal position, and that is what PGO does for this problem.

For the perplexity, this represents a measure of the distribution over the action preferences for a given state. At the start, the value should be 4 since the policy is randomly which means 4 possible actions are equally preferred. Then with iteration times increasing, this value should be close to 1 to show the deterministic or near – deterministic policy.

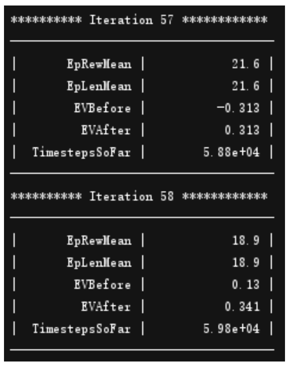
Cart – Pole Problem



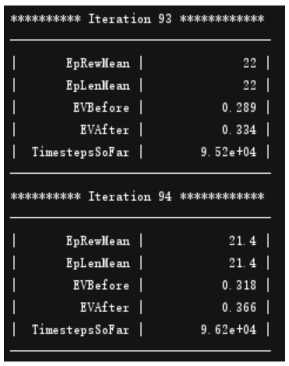
```
********** Iteration 0 ************
------------------------------------
|       EpRewMean |          21.4 |
|       EpLenMean |          21.4 |
|        EVBefore |             0 |
|         EVAfter |         0.408 |
|  TimestepsSoFar |      1.03e+03 |
------------------------------------
********** Iteration 1 ************
------------------------------------
|       EpRewMean |          21.5 |
|       EpLenMean |          21.5 |
|        EVBefore |           0.3 |
|         EVAfter |         0.356 |
|  TimestepsSoFar |      2.06e+03 |
------------------------------------
```



```
********** Iteration 16 ************
------------------------------------
|       EpRewMean |            24 |
|       EpLenMean |            24 |
|        EVBefore |         0.319 |
|         EVAfter |         0.349 |
|  TimestepsSoFar |      1.73e+04 |
------------------------------------
********** Iteration 17 ************
------------------------------------
|       EpRewMean |          22.9 |
|       EpLenMean |          22.9 |
|        EVBefore |         0.273 |
|         EVAfter |         0.352 |
|  TimestepsSoFar |      1.83e+04 |
------------------------------------
```



```
********** Iteration 57 ************
------------------------------------
|       EpRewMean |          21.6 |
|       EpLenMean |          21.6 |
|        EVBefore |        -0.313 |
|         EVAfter |         0.313 |
|  TimestepsSoFar |      5.88e+04 |
------------------------------------
********** Iteration 58 ************
------------------------------------
|       EpRewMean |          18.9 |
|       EpLenMean |          18.9 |
|        EVBefore |          0.13 |
|         EVAfter |         0.341 |
|  TimestepsSoFar |      5.98e+04 |
------------------------------------
```



```
********** Iteration 93 ************
------------------------------------
|       EpRewMean |            22 |
|       EpLenMean |            22 |
|        EVBefore |         0.289 |
|         EVAfter |         0.334 |
|  TimestepsSoFar |      9.52e+04 |
------------------------------------
********** Iteration 94 ************
------------------------------------
|       EpRewMean |          21.4 |
|       EpLenMean |          21.4 |
|        EVBefore |         0.318 |
|         EVAfter |         0.366 |
|  TimestepsSoFar |      9.62e+04 |
------------------------------------
```

The above two tables show the result of iteration, you can find the mean reward is not increasing anymore which means the NN is not updating. The problem might be my loss function which cannot compute correct error.

## V. Brief Summary

Reinforcement learning is wide used in modern learning realm. The basic idea is make the agent generate optimal policy according to state and reward obtained from the environment. Markov Decision Process is a special case of Reinforcement learning that the system has the Markov property assumption which means current state and reward is independent from all other states.

In a word, the aim of RL and MDP is to find optimal policy which can bring the agent the max reward in a long term. For MDP, there are three basic approaches to optimize the policy, value iteration (VI), policy evaluation (PI) and policy gradient optimization (PGO).

After completing the implementation, PI shows the fasted convergence speed which PGO is the slowest one. However, PGO is the more sophisticated one which can be applied into multiple complicated RL and MDP problems.

For another thing, it's quite different to deal with control problems in discrete and continuous space. For discrete problem, it's necessary to use softmax function to generate our policy model, while for continuous one, rather than using softmax function, we need to build a continuous Gaussian distribution to represent policy. In addition, the deep neural network helps a lot in MDP and PGO training.