Homework3 ECE276B - Project Report

Yihan Hu

2019/06/02

1 Hill Climbing Problem

Our goal is to solve the Hill Climbing problem. Our automobile is stuck between two hills and its underpowered engine, even at full throttle, is unable to climb up the steep hill on the right. Our task is to fnd a control policy that takes advantage of the potential energy obtained by swinging between the two hills before driving straight towards the goal to the right. We will consider a simulation environment provided by OpenAI Gym (https://gym.openai.com/) and will compared the differences in on-policy and off-policy model-free planning algorithms. As the state space is continuous we need to apply a disretization to formulate a finite-state MDP with unknown motion model and cost function on which we will use TD policy iteration.

1.1 Problem Formulation

We can define the MDP problem as follow,
1.**state**: $x(t) \in \mathcal{X} = [-x_{min}, x_{max}]$, which can be discretized in algorithm.
2.**control**: $u(t) \in \mathcal{U} = [0(left), 1(not - move), 2(right)]$
3.**motion model**:Unkonwn
4.**cost**: -1.0 for not reaching the goal.(which is also unknow in our case)
5.**Objective function and contraints**:

$$\begin{gathered} V^*(x) = \min_\pi V^\pi(x) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \ell\left(x_t, \pi\left(x_t\right)\right) | x_0 = x\right] \\ \text{s.t. } x_{t+1} \sim unknown \\ x_t \in \mathcal{X} \\ \pi\left(x_t\right) \in \mathcal{U}\left(x_t\right) \end{gathered} \tag{1}$$

Here we are trying to solve a model-free prediction problem, which is to approximate the policy evaluation backup operators $\mathcal{T}_\pi[V]$ and $\mathcal{T}_\pi[Q]$ using sampling instead of computing the expectation exactly. In our case, we are using Temporal-Difference methods. The expected cost can be approximated by a sample average over a single system transition and an estimate of the expected cost at the new state (bootstrapping). In our problem, we do not know our motion model and cost function, by sampling from model, we can estimate Q-function. In this case, the policy improvement step can be implemented model-free(i.e., can compute $min_u Q^\pi(x;u)$ without knowing the motion model p_f or the state cost l. However, we define all sorts of hyperparameters(γ, α, ϵ...) which is also known. Our objective function is to find optimal policy $\pi^* = argmin_\pi Q^\pi(x;\pi(x))$ under the model free-constraint(unknown motion model and stage cost).

1.2 Technical Approach

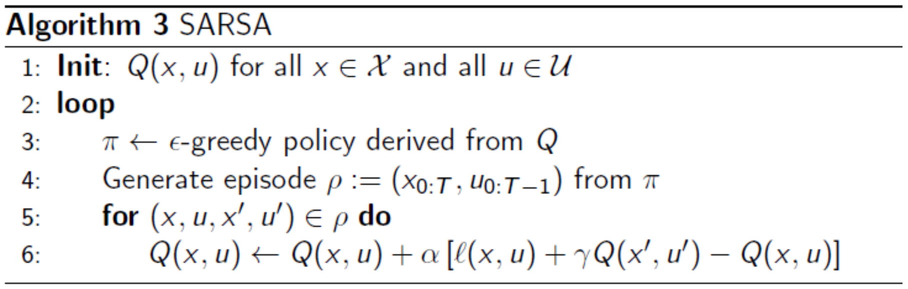
As I have elaborated above, the key ideal of TD model-free prediction is that the expected cost can be approximated by a sample average over a single system transition and an estimate of the expected cost at the new state (bootstrapping). Having sampled from the model, by using running sample average, we can using following equation to update the Q-value:

$$Q(x,u) \leftarrow Q(x,u) + \alpha \left[\ell(x,u) + \gamma Q\left(x', u'\right) - Q(x,u)\right] \tag{2}$$

In the policy improvement step, we compute the current optimal policy based on updated Q-function:

$$\pi'(x) = \underset{u \in \tilde{\mathcal{U}}(x)}{\arg\min} \left[Q(x,u)\right] \tag{3}$$

To ensure that you do not commit to the wrong controls too early and continue exploring the state and control spaces, I randomly initialize the state space of policy. Also, during the exploration, I implement ϵ-Greedy algorithm. Here is the explicit form of the SARSA algorithms:

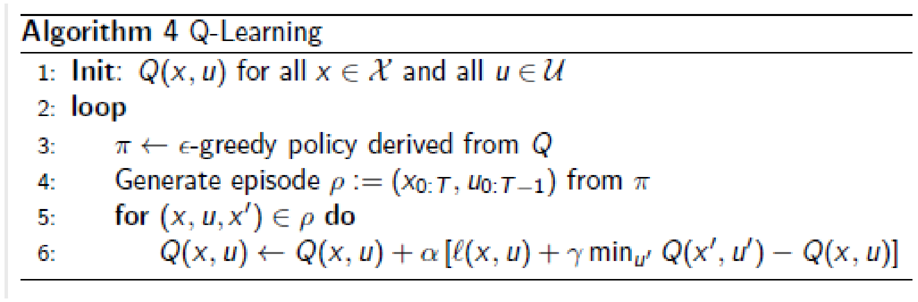


Algorithm 3 SARSA

```
Init: $Q(x,u)$ for all $x \in \mathcal{X}$ and all $u \in \mathcal{U}$
loop
    $\pi \leftarrow \epsilon$-greedy policy derived from $Q$
    Generate episode $\rho := (x_{0:T}, u_{0:T-1})$ from $\pi$
    for $(x, u, x', u') \in \rho$ do
        $Q(x,u) \leftarrow Q(x,u) + \alpha\left[\ell(x,u) + \gamma Q(x',u') - Q(x,u)\right]$
```

Figure 1: SARSA algorithm

It is known that SARSA is on-policy TD training, I also implement an off-policy TD learning(i.e Q-learning). The only difference between two algorithm is the min operation in the update process. Here is the explicit form for Q-learning algorithms:



Algorithm 4 Q-Learning

```
Init: $Q(x,u)$ for all $x \in \mathcal{X}$ and all $u \in \mathcal{U}$
loop
    $\pi \leftarrow \epsilon$-greedy policy derived from $Q$
    Generate episode $\rho := (x_{0:T}, u_{0:T-1})$ from $\pi$
    for $(x, u, x') \in \rho$ do
        $Q(x,u) \leftarrow Q(x,u) + \alpha\left[\ell(x,u) + \gamma \min_{u'} Q(x',u') - Q(x,u)\right]$
```

Figure 2: Q-learning algorithm

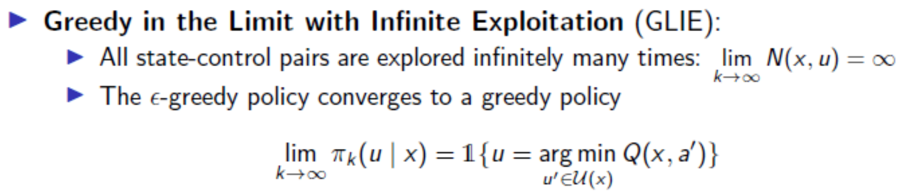
For the necessary conditions for convergence to the optimal action-value function Q^*, we have convergence of Model-free Policy Iteration theorem. Saying that Both MC Policy

Iteration and SARSA converge to the optimal action-value function, $Q(x,u) \to Q^*(x,u)$, as the number of episodes $k \to \infty$ as long as:

1. the sequence of ϵ-greedy policies $\pi_k(u|x)$ is Greedy in the Limit with Infinite Exploitation(GLIE).

2.the sequence of step sizes α_k is Robbins-Monro.

Here is the definition of GLIE:



- **Greedy in the Limit with Infinite Exploitation** (GLIE):
  - All state-control pairs are explored infinitely many times: $\lim_{k\to\infty} N(x,u) = \infty$
  - The ϵ-greedy policy converges to a greedy policy

$$\lim_{k\to\infty} \pi_k(u \mid x) = \mathbb{1}\{u = \arg\min_{u' \in \mathcal{U}(x)} Q(x,a')\}$$

Figure 3: Greedy in the Limit with Infinite Exploitation (GLIE)

1.3 Result

1.3.1 Hyperparameters

In this problem, we have many hyperparameters [learning rate α, forgetting rate γ, meshgrid (hx, hy), maximum sampling time T, ϵ-greedy, on-off policy]. Since some of them do not have significant influence, we only present the analysis among those influential hyperparameters. In the main function, we loop over all sorts of parameters, among all parameters, we find $gamma = 0.8$ and $\alpha = 0.1$ the best. Also, more episode and appropriate exploration will lead to better result.

α **and** γ: Our learning rate α and forgetting rate γ are the most influential parameters. Here we fix the other parameters and only tune γ and α. We can see from the plot that gamma rate is not an important factor. However, α are sensitive, it can not be too large or small. We should note that the number of steps are the mean value of 100 runs.

ϵ **decay rate and number of episode** We also tuning the decay rate and number of step. Note that we multiply ϵ by the decay ratio every episode, so 0 decay rate means no epsilon greedy exploration while larger decay means more exploration later on. From the plot we can see that more episode always improve the result, which makes sense because more sampling data. Also, It is clear that appropriate decay rate leads to better result.

Moreover, we fix number of episode to 10000, we can clearly observed that moderate decay rate (multiply 0.95 every 300 episodes) gives the best result, for both on and off policy.

1.3.2 on/off Policy Iteration

From Fig. 1.3.1, we can observed that off policy are better than on policy when less exploration, but worse for higher exploration.

To further explore the difference, I take the moving every of number of steps near 500 episodes. We can see that on policy are better than off policy when converges.

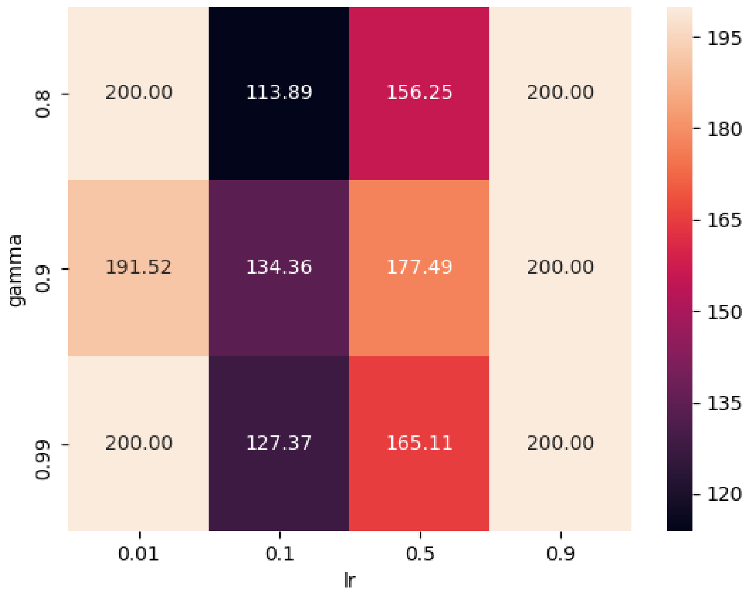


Figure 4: Number of step vs. α and γ

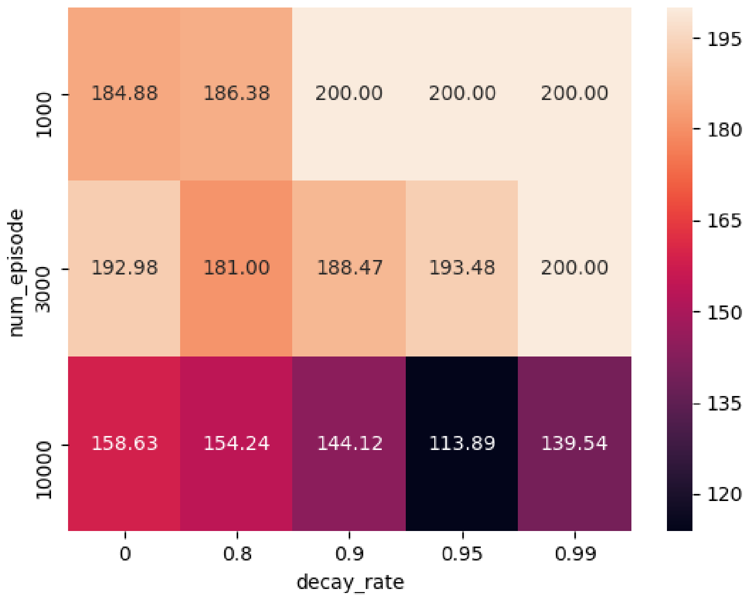


Figure 5: Number of step vs. ϵ decay rate and number of episode.

Moreover, as is shown in Fig. 14, by plotting the Q-function and policy respect to episodes, we can find that actually on policy converges a little bit faster than off policy. In my opinion, it is because Q-learning enable Q function to approximates Q^* regardless of the policy being

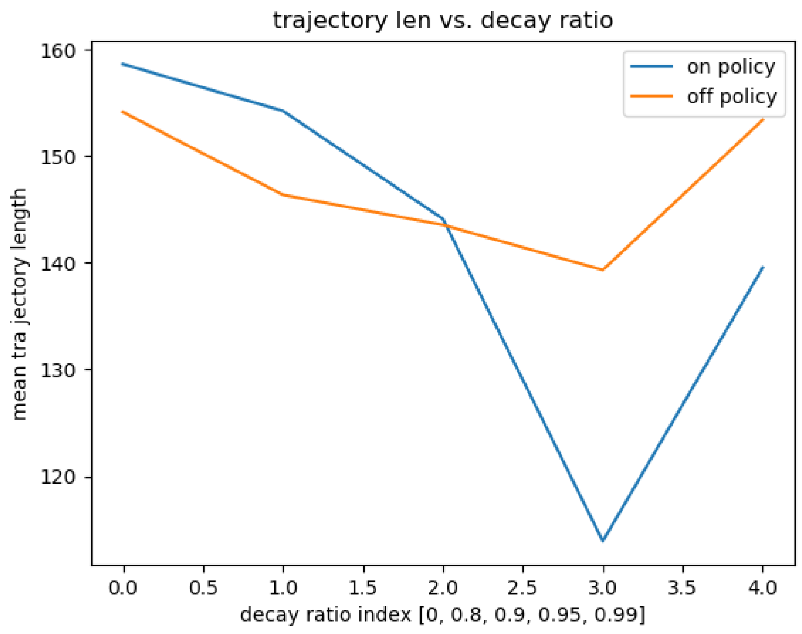


Figure 6: Mean number of step vs. ϵ decay rate.

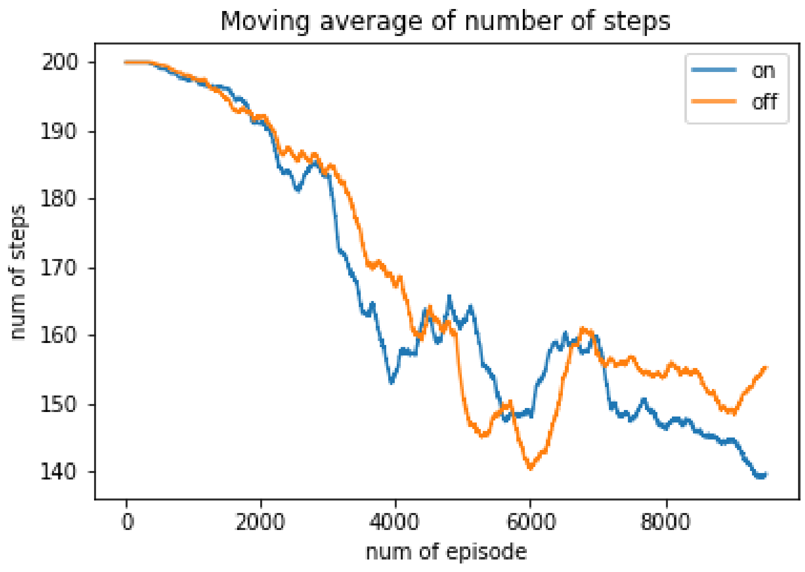


Figure 7: Moving average of number of step vs. episodes for on and off policy. We fix the other hyperparameters, there is no ϵ-greedy in our case.

followed, the min operation inside actually losing the information. However, we gain the generality of data we use, even random walk can be used for training.

1.4 Conclusion

Both on and off policy get reasonable result. However, according to my statistics, on-policy performs better than off-policy training, which makes sense since on-policy directly uses the policy we improved each time. More video and statistics can be in the zip file.

2 Pendulum Problem

In this problem, our task is to solve an optimal control problem in order to balance an inverted pendulum. We will discretize the state and control spaces to formulate a finite-state MDP problem, and then solve it using value iteration and policy iteration.Consider the pendulum state x = $(x1, x2)^T$ where x1 is the angle of the pendulum and x2 is the angular velocity. The continuous-time dynamics are:

$$d\mathbf{x} = f(\mathbf{x}, u)dt + \sigma d\omega \qquad f(\mathbf{x}, u) := \begin{bmatrix} x_2 & x_2 \\ a \sin x_1 - bx_2 + u & \end{bmatrix} \tag{4}$$

The stage-cost at each state x and control u is:

$$\ell(\mathbf{x}, u) = 1 - \exp(k \cos x_1 - k) + \frac{r}{2}u^2 \tag{5}$$

2.1 Problem Formulation

Since the motion model are given and noise are introduced, this is stochastic continuous-time optimal control problem. In this case, we define:

1.**time**: $t \in [0, T]$

2.**state**: $x(t) \in \mathcal{X} = [-\pi, \pi] \times [-v_{max}, v_{max}], \forall t \in [0, T]$

3.**control**: $u(t) \in \mathcal{U} = [-u_{max}, u_{max}], \forall t \in [0, T]$

4.**motion model**:a stochastic differential equation (SDE):

$$dx = f(x(t), u(t))dt + \sigma d\omega, \qquad x(0) = x_0 \tag{6}$$

where $d\omega$ is Brownian motion noise.

5.**Cost**:

$$\ell(\mathbf{x}, u) = 1 - \exp(k \cos x_1 - k) + \frac{r}{2}u^2 \tag{7}$$

6.**Objective function and constraints**: we should note that In our case is an infinite discounted problem, so T -> ∞. We need to times discounted factorγ in front of the state cost.

To solve the continuous problem we need to discretize our state space as 2D grid. Thus our policy and value function becomes discretized 2D-array. To generate more continuous policy, we use 2d linear/cubic interpolation, which uses linear weight/cubic weight to sample from adjacent points.

$$\begin{aligned}\min_{\pi\in PC^0([0,T],\mathcal{U})} \quad & J^\pi(0,x_0) := \mathbb{E}\left\{ \underbrace{\int_0^T \ell(x(t),\pi(t,x(t)))dt}_{\text{stage cost}} + \underbrace{q(x(T))}_{\text{terminal cost}} \;\middle|\; x(0)=x_0 \right\} \\ \text{s.t.} \quad & dx = f(x(t),\pi(t,x(t)))dt + C(x(t),\pi(t,x(t)))d\omega. \\ & x(t)\in\mathcal{X},\ \pi(t,x(t))\in\mathcal{U}\end{aligned}$$

Figure 8: Objective function and constraints.

2.2 Technical Approach

I discretize the time as dt, angle as $d\theta$ and angular speed as dv_θ. In this case, the stage cost becomes $dt\ell(x,u)$, dynamics becomes $x_{k+1} = x_k + \tau f(x_k, u_k) + \epsilon_k, \quad \epsilon_k \sim \mathcal{N}(0, \tau\Sigma(x_k, u_k))$, and the problem becomes infinite-horizon stochastic optimal control:

$$\begin{gathered} V^*(x) = \min_\pi V^\pi(x) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \ell(x_t, \pi(x_t)) \,|x_0 = x\right] \\ \text{s.t.} \quad x_{t+1} \sim p_f(\cdot|x_t, \pi(x_t)) \\ x_t \in \mathcal{X} \\ \pi(x_t) \in \mathcal{U}(x_t) \end{gathered} \tag{8}$$

Then, value iteration and Policy iteration algorithm can be easily formulated as:

Value Iteration:

$$V_{k+1}(x) = \min_{u\in\mathcal{U}(x)} \left[\ell(x,u) + \gamma \sum_{x\in\mathcal{X}} p(x'|x,u) V_k(x')\right], \quad \forall x \in \mathcal{X} \tag{9}$$

Policy Iteration:

$$V^\pi(x) = \ell(x,\pi(x)) + \gamma \sum_{x'\in\mathcal{X}} p_f(x'|x,\pi(x)) V^\pi(x'), \quad \forall x \in \mathcal{X} \tag{10}$$

$$\pi'(x) = \arg\min_{u\in\mathcal{U}(x)} \left[\ell(x,u) + \gamma \sum_{x'\in\mathcal{X}} p_f(x'|x,u) V^\pi(x')\right], \quad \forall x \in \mathcal{X} \tag{11}$$

Key difference: We can rewrite VI as: The Value Update step of VI is an iterative

2. **Policy Improvement**: Given $V_k(x)$ obtain a stationary policy:

$$\pi(x) = \arg\min_{u\in\mathcal{U}(x)} \left[\ell(x,u) + \gamma \sum_{x'\in\mathcal{X}} p_f(x' \mid x,u) V_k(x')\right], \qquad \forall x \in \mathcal{X}$$

1. **Value Update**: Given $\pi(x)$ and $V_k(x)$, compute

$$V_{k+1}(x) = \ell(x,\pi(x)) + \gamma \sum_{x'\in\mathcal{X}} p_f(x' \mid x,\pi(x)) V_k(x'), \qquad \forall x \in \mathcal{X}$$

Figure 9: Value iteration.

solution to the linear system of equations in the Policy Evaluation Theorem while PI solves Policy Evaluation equation, which is equivalent to running the Value Update step of VI an infinite number of times.

Also, the Complexity of VI per Iteration is $O\left(|\mathcal{X}|^2|\mathcal{U}|\right)$ and Complexity of PI per Iteration is $O\left(|\mathcal{X}|^2(|\mathcal{X}|+|\mathcal{U}|)\right)$, we can see that PI is more computationally expensive than VI.

2.3 Result

We first tuning the parameter with coarse grid, and fix the hyperparameters for thin grid. Here we choose $\gamma = 0.95, \alpha = 0.1, k = 3, r = 0.1, a = 1, b = 0.1 and dt = 0.1, n_\theta = 180, n_v = 40, n_u = 40$ as our parameters.

We plot the value function over episode between VI(value iteration) and PI(policy iteration). We can find that both algorithm converge to similar result. As we can see in comparison Fig. 2.3, policy iteration converge much more faster over the episode, it can quickly converges to the optimal value in few iterations, which makes sense, because PI does policy evaluation step until it converges while VI does it just once. However, value iteration is more stable, as we can see in Fig. 2.3 and Fig. 2.3. there exists clear artifacts in policy iteration algorithm, but no artifacts in value iteration when noise are small(sigma = 0.01). As noise grows bigger, both algorithm get the exactly the same result.

It is interesting that as noise becomes larger, our value function and optimal policy becomes smoother, which makes sense, because our Gaussian kernel gets larger and more states are weighted averaged during the iteration process.

As for the noise threshold not able to achieve stable equilibrium, it is hard to define what is a equilibrium state. Once we fix the policy, by gradually increasing the noise level, even with a very high noise, our pendulum can still trying to get to the top. Because we apply a relatively small cost for control, so we can even get a balanced vertical state from any state, even start from the bottom. However, if we change the noise level during the training process, we can see that the value function becomes smooth, and the policy gradually degenerates to 0, i.e not applying any control to any state. This makes sense, because the noises are too large that apply any control would be a waste of energy. In this case, we can see from Fig. 2.3 that when sigma exceed 3, we cannot balance our pendulum anymore.

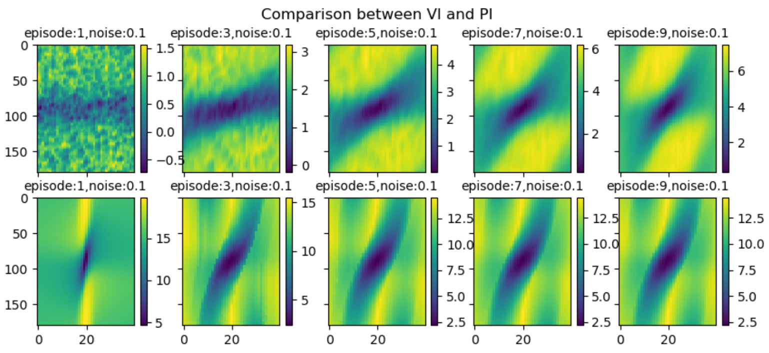


Figure 10: Convergence speed analysis. The first row is Value iteration and second row is Policy iteration.

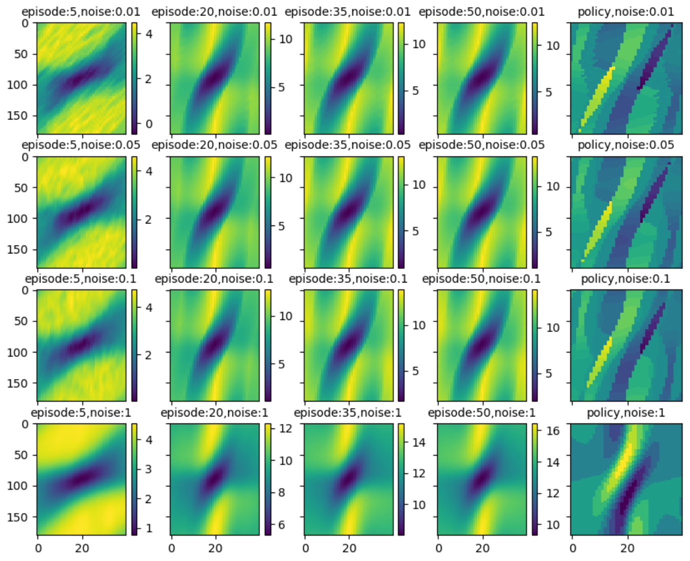


Figure 11: Value iteration algorithm, we choose different episode(from left to right) and different noise (top to bottom). Also, a optimal control policy at the end.

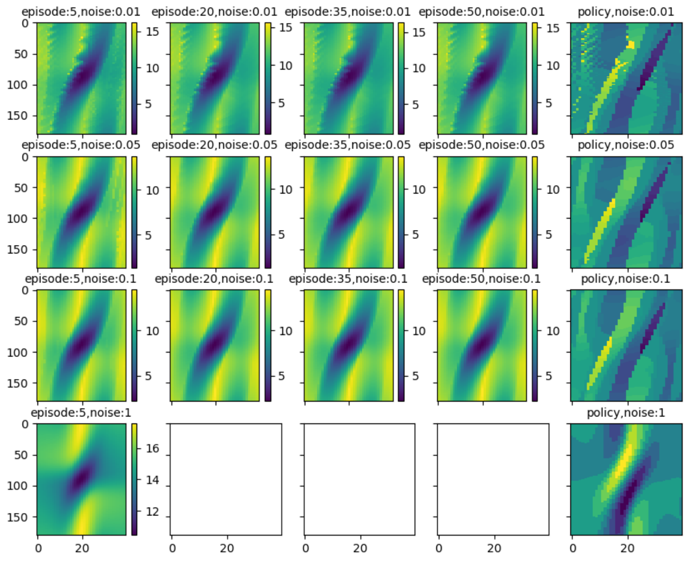


Figure 12: Policy iteration algorithm, we choose different episode(from left to right) and different noise (top to bottom). Also, a optimal control policy at the end. Note that because the algorithm already converged within 15 iteration, thus, there are 3 blanks at the bottom.

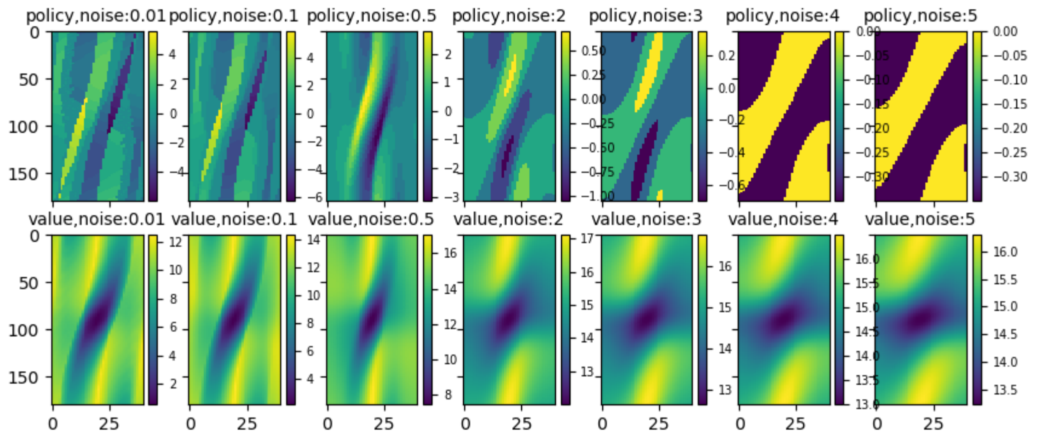
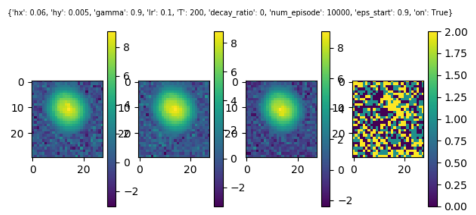


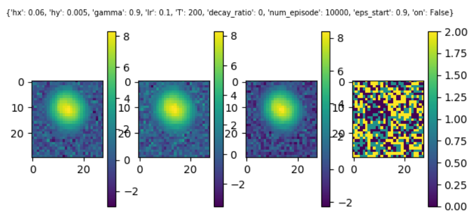
Figure 13: Policy and value function respect to noise.

2.4 Conclusion

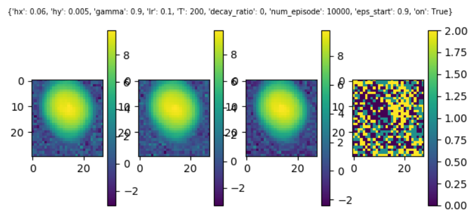
We can see that both VI and PI get good result. More specifically, PI converges faster(even within few steps) with noise, however, VI is more robust than VI, since it can always get a convergence result, while PI might not. As for the final policy and value, both algorithm can converge to the same result. Moreover, noise will smooth the value function and policy might degenerate to 0 if noise is too big.


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': True}

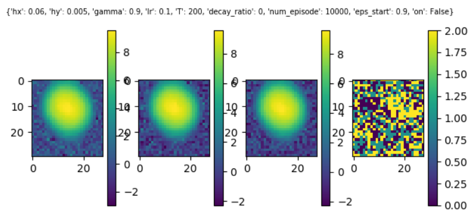
(a) 300 on


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': False}

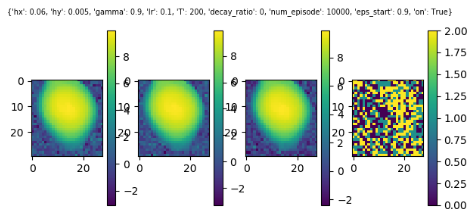
(b) 300 off


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': True}

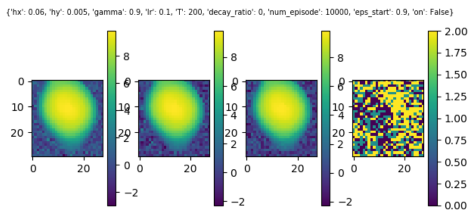
(c) 900 on


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': False}

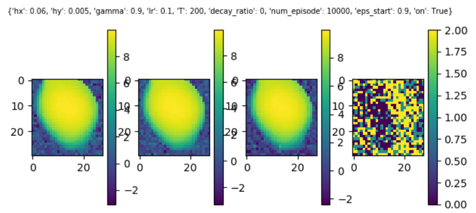
(d) 900 off


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': True}

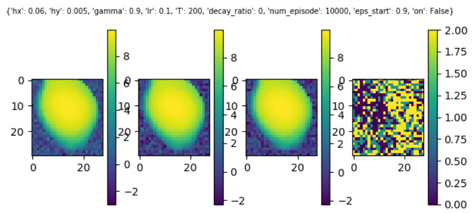
(e) 1500 on


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': False}

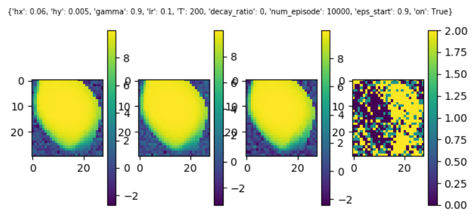
(f) 1500 off


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': True}

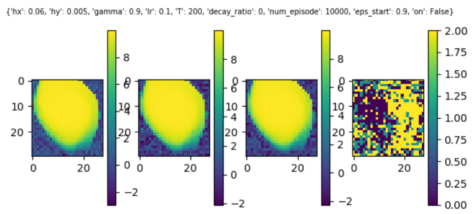
(g) 2400 on


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': False}

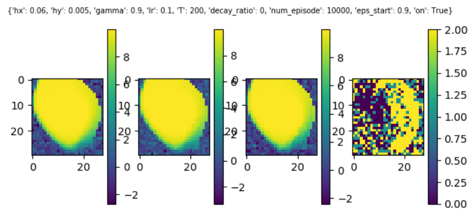
(h) 2400 off


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': True}

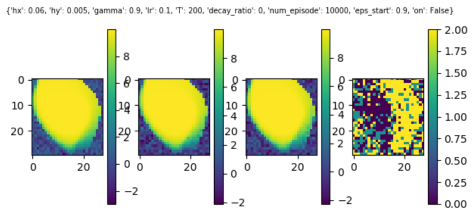
(i) 5700 on


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': False}

(j) 5700 off


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': True}

(k) 9900 on


{'hx': 0.06, 'hy': 0.005, 'gamma': 0.9, 'lr': 0.1, 'T': 200, 'decay_ratio': 0, 'num_episode': 10000, 'eps_start': 0.9, 'on': False}

(l) 9900 off

Figure 14: The evolution process of Q-function of policy for on and off policy