Single Agent Learning

Duy Chuan Ha, Said Al Faraby, Christos Louizos, Oana Munteanu

October 4, 2013

Abstract

The purpose of this assignment is to focus on the research of a learning scenario in which the agent does not know the transition probabilities, nor the reward structure. In order to complete that, we have implemented and tested various methods, namely Q-learning, On-policy Monte Carlo Control, Off-policy Monte Carlo Control and Sarsa in order to explain their theoretical differences, to compare their results and to conclude which one is the best policy regarding the convergence speed. (Conclusion)

1 INTRODUCTION

In this research, we have considered that the model is unknown and, in this situation, two approaches can be pursued. A model-based approach tries to learn the model explicitly and then use methods like Dynamic Programming to compute the optimal policy with respect to the estimate of the model. On the other hand, a model-free approach concentrates on learning the state value function (Q-value function) directly and obtaining the optimal policy from this estimates.
We shall try to focus on model-free methods for learning in MDPs by making use of:
• Q-learning with ϵ-greedy action selection;
• Q-learning with Softmax action selection;

- On-policy Monte-Carlo Control;
- Off-policy Monte-Carlo Control;
- Sarsa.

TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap). Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to $V(s_t)$ (only then R_t is known), TD methods need wait only until the next time step.

In this assignment, we have considered the learning methods for estimating value functions and discovering optimal policies.
Regarding Monte Carlo methods, they require only experience through sample sequences of states, actions and rewards from on-line or simulated interaction with an environment. They do not require prior knowledge of the environments dynamics, yet can still achieve an optimal behavior. Concerning the disadvantages of these methods, Monte Carlo do not require a model and is conceptually simple, but is not suited for step-by-step incremental computation. Temporal Difference Learning (TD Learning) represents a combination of Monte Carlo and Dynamic Programming ideas. Like MC methods, TD methods do not require a model, while like DP, TD methods bootstrap; they update estimates in part on other learned estimates, without waiting for a final outcome like MC:

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t - V(s_t)] \qquad \text{MC: update target is } R_t \tag{1.1}$$

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} - \gamma V(s_{t+1}) - V(s_t)] \qquad \text{TD: update target is } r_{t+1} + \gamma V_t(s_{t+1}) \tag{1.2}$$

We begin the following report by first explaining the predator-prey environment and defining it's goal in section 1.1. Afterwards, we present in section 2 the methods that were implemented in the aforementioned environment. The results according to each of these methods as well as the comparison between them are displayed in detail in section 3 and the brief conclusion is indicated in section 4.

1.1 Environment

The environment is a 11x11 grid which is toroidal and each state is encoded as the positions of the two agents, predator and prey. The simulator is defined through a while-loop of a hundred runs where the initial position of the predator is (0,0) and that of the prey (5,5). In each transition, the predator will make a move first and then the prey, according to their respective policies. Each run is composed of an episode where the goal of the predator is to capture the prey, which can be interpreted as the end state of this particular episode.

Both the predator and the prey are initialized with a random policy in the beginning where for the predator it is equiprobable to choose any of the possible actions in any state. The prey has a fixed policy and hence it can be modelled as part of the environment. It has 0.8 probability of waiting and 0.2 probability of moving to any of the adjacent squares, unless this square is occupied, hereby changing the probability distribution.

2 METHODS AND PROCEDURES

2.1 Q-LEARNING

Q-LEARNING WITH ϵ-GREEDY ACTION SELECTION

Using ϵ-greedy method underlines the idea of the agent choosing the action that it believes to have the best long-term effect with probability $1-\epsilon$, and it chooses an action uniformly at random. The action is selected independently of the action-value estimates and the highest estimated reward is called the greediest action . This method ensures that if enough trials are done, each action will be tried an infinite number of times, thus ensuring optimal actions are discovered.

Q-LEARNING WITH SOFTMAX ACTION SELECTION

Considering the Softmax Action Selection, we have chosen an action a on the t-th play with the probability that depends on the value of Q and τ, which represents the temperature and is defined as a positive parameter. Therefore, we can assign the distribution in the following way:

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}},$$

A random action is selected with regards to the weight associated with each action, meaning the worst actions are unlikely to be chosen. This is a good approach to take where the worst actions are very unfavourable.

2.2 ON-POLICY MONTE CARLO

For on-policy Monte Carlo method we have considered the ϵ-greedy policies, meaning that most of the time the chosen action has the maximal estimated action value, but while considering the probability ϵ they instead select an action at random. The on-policy method attempt to estimate the action-value function with respect to the current policy.

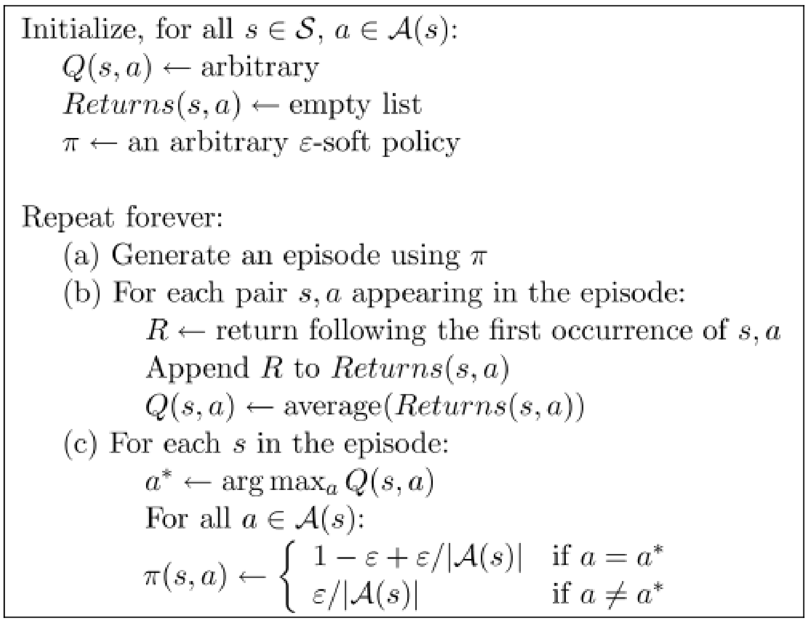
ON-POLICY EVALUATION

On-policy evaluates and improves the policy that is already used by the agent in order to decide which action to pick at a specific state. The policy that is generally used in on-policy control methods is a soft one where the probability is $\pi(s,a) > 0$ for all states and possible actions. The on-policy method implemented in this assignment uses an ϵ-greedy policy, which is an example of an ϵ-soft policy, and this translates into that most of the times the greedy action will be selected, except of some small random probability $\epsilon > 0$ to select a different random action.

On-policy Monte Carlo control behaves like a Generalized Policy Iteration (GPI). The evaluation step is composed by generating random episodes, estimating the action-value function in each one, and then averaging over all of them. First-visit Monte Carlo methods are used in order to estimate the action-value function for each episode derived from the current policy. The returns are computed using the equation 2.1, where t is the state from where we begin to calculate, k is the number of the subsequent states, r is the immediate reward and γ is the discount factor.

$$R_t = \sum_{k=0}^{K} \gamma^k r_{t+k+1} \tag{2.1}$$

After the processing of each episode, the improvement of the policy occurs by getting the action that produces the highest reward in a state, and updates the probabilities in the ϵ-soft policy according to that action, which becomes the new greedy action. The pseudo code is illustrated in Figure 2.1.



```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $Returns(s,a) \leftarrow$ empty list
    $\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy

Repeat forever:
    (a) Generate an episode using $\pi$
    (b) For each pair $s, a$ appearing in the episode:
            $R \leftarrow$ return following the first occurrence of $s, a$
            Append $R$ to $Returns(s,a)$
            $Q(s,a) \leftarrow$ average($Returns(s,a)$)
    (c) For each $s$ in the episode:
            $a^* \leftarrow \arg\max_a Q(s,a)$
            For all $a \in \mathcal{A}(s)$:
            $\pi(s,a) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$
```

Figure 2.1: On-policy Monte Carlo Control Pseudo-code

2.3 Off-Policy Monte Carlo

The off-policy Monte Carlo control method follows the behaviour policy while learning and improving the estimation policy. This method requires that the behavior policy have a non-zero probability of selecting all actions that might be selected by the estimation policy. Separating these two functions, namely behavior policy and estimation policy, represents an advantage in establishing that the behavior policy continues to sample all possible actions, while the estimation policy may be deterministic.

OFF-POLICY EVALUATION

In many cases, V^π or Q^π can be estimated from episodes generated by policy π' as long as all actions taken under π are also taken under π'. Therefore, it is required that $\pi(s,a) > 0$ implies that $\pi'(s,a) > 0$, thus π' must be stochastic. π is *a target policy* because the target of the learning process is to estimate its value function, and π' is *a behavior policy* because it controls the agent and hence generates the behaviour.

To compute the return following the first-visit state s in an i-th episode, the observed return should be weighted by the relative probability of the complete sequence of the episode occurred under π called $p_i(s)$ and π' which is $p'_i(s)$. Therefore, the Monte Carlo estimation after observing n_s returns from state s is given by:

$$V(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'(s)} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'(s)}} \tag{2.2}$$

Then, the relative probability $p_i(s)/p'_i(s)$ is obtained from:

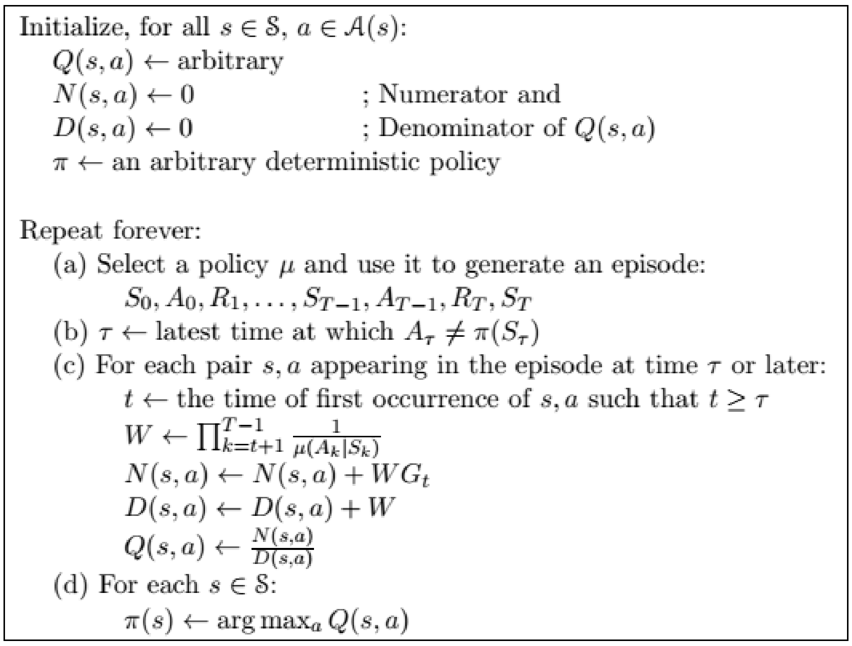
$$\frac{p_i(s_t)}{p'_i(s_t)} = \frac{\prod_{k=t}^{T_i(s)-1} \pi(s_k,a_k) P_{s_k s_{k+1}}}{\prod_{k=t}^{T_i(s)-1} \pi'(s_k,a_k) P_{s_k s_{k+1}}} \tag{2.3}$$

$$= \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k,a_k)}{\pi'(s_k,a_k)} \tag{2.4}$$

where $T_i(s)$ is the termination time of ith episode involving s, and t is the time of first occurrence of state s in the episode. The transition probability in (2.3) is unknown, but it can be eliminated since it belongs to the same environment. Thus, the weight for the return depends only on the two policies.

OFF-POLICY MONTE CARLO CONTROL

The Off-Policy Monte Carlo Control uses off-policy evaluation as mentioned before to estimate the action value function, but it improves the *target policy* instead of the *behaviour policy*. An advantage of this method is that the *target policy* may be deterministic whereas the *behaviour policy* remains stochastic in order to keep the chance of exploring all possible actions. The pseudo-code of the whole process of Off-Policy Monte Carlo Control is given by Figure 2.2



```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \leftarrow$ arbitrary
    $N(s,a) \leftarrow 0$                ; Numerator and
    $D(s,a) \leftarrow 0$                ; Denominator of $Q(s,a)$
    $\pi \leftarrow$ an arbitrary deterministic policy

Repeat forever:
    (a) Select a policy $\mu$ and use it to generate an episode:
            $S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T$
    (b) $\tau \leftarrow$ latest time at which $A_\tau \neq \pi(S_\tau)$
    (c) For each pair $s, a$ appearing in the episode at time $\tau$ or later:
            $t \leftarrow$ the time of first occurrence of $s, a$ such that $t \geq \tau$
            $W \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\mu(A_k|S_k)}$
            $N(s,a) \leftarrow N(s,a) + WG_t$
            $D(s,a) \leftarrow D(s,a) + W$
            $Q(s,a) \leftarrow \frac{N(s,a)}{D(s,a)}$
    (d) For each $s \in \mathcal{S}$:
            $\pi(s) \leftarrow \arg\max_a Q(s,a)$
```

Figure 2.2: Off-Policy Monte Carlo Control Pseudo-code

2.4 **SARSA**

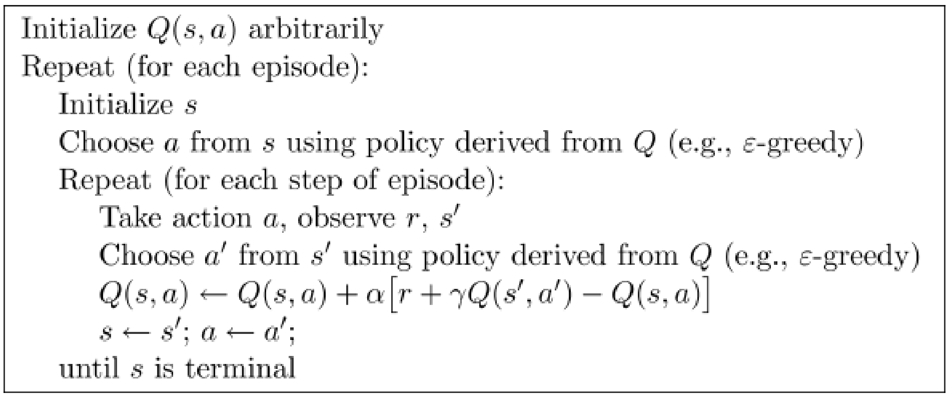
Sarsa is an on-policy method based on model-free action policy estimation. It determines the optimal policy while controlling the MDP with respect to the algorithm which behaves according to the same policy that has been improved.

SARSA ON-POLICY TD CONTROL

Sarsa is an on-policy method of TD learning which applies for every iteration the quintuple $Q(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, where $\mathbf{s_t}, \mathbf{a_t}$ are the original state and action, $\mathbf{r_{t+1}}$ is the reward observed in the following state and $\mathbf{s_{t+1}}, \mathbf{a_{t+1}}$ are the new state-action pair.
Following the explanation of the quintuple from above, its update expression for defining the Sarsa method is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.5}$$

The steps that should be completed for each iteration according to this method are presented in Figure 2.3 below:



```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize $s$
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Repeat (for each step of episode):
        Take action $a$, observe $r$, $s'$
        Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma Q(s',a') - Q(s,a)\big]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until $s$ is terminal
```

Figure 2.3: Sarsa Pseudo-code

As in all on-policy methods, we continually estimate Q^π for the behavior policy π, and at the same time change π toward greediness with respect to Q^π. The update is done after every transition from a nonterminal state s. If s' is terminal, therefore $Q(s',a')$ will be defined as zero. This rule uses every element of the quintuple of events, that make up a transition from one state-action pair to the next one.
Regarding both Q-learning and Sarsa methods, the major difference between them is that the maximum reward for the next state is not necessarily used for updating the Q-values.

2.5 Reducing the state space

Regard the current maximum statespace in the form of

$$statespace[predator(x_i, y_j), prey(x_k, y_l)] \tag{2.6}$$

This statespace has size 11^4, spanning all possible positions of predator and prey in the 11×11 grid. Reducing the statespace would mean compressing this table without information loss. Notice the tables of policy and value iteration, table 3.5 and table 3.7, both displaying the values for all possible states by keeping the prey fixed at [5][5]. We can see the same value distribution; and a pattern where the values are proportional to the distance of the predator to the prey. When we look at table 3.1, where the prey is at [3][2], we notice the exact same value distribution, only translated to the current position of the prey.

So a way of reducing the statespace would be by eliminating the representation of the prey within the statespace, by keeping it fixed; we marginalize the conditioning of the prey on the statevalue distribution. We can do this without information loss, by making use of a transform function which translates the actual positions of the prey and predator to an arbitrary one, where the position of the prey has been kept fixed; a statespace where only the position of the predator is variable. In our implementation we make use of one represented by table 3.7.

The transform function hereby is a simple translation in the following form:

$$P'_{pred} = \omega((P'_{prey} - P_{prey}) + P_{pred})) \qquad \begin{cases} P'_{pred} & = \text{Projected predator position} \\ P'_{prey} & = \text{Projected prey postion, [5][5] in our case} \\ P_{prey} & = \text{Actual prey position} \\ P_{pred} & = \text{Actual predator position} \end{cases} \tag{2.7}$$

ω is the wrapper function implemented in our Position class, which takes into consideration that the grid is toroidal. Take table 3.1 as an aid to illustrate this translation, where we pick position [1][5] as P_{pred} having a certain value (6.440 for the actual state value would not yet be known in the reduced space scenario). [3][2], denotes P_{prey}. The value of the state[1][5][3][2] can be looked up by translating this to the calculated reduced space:

$$P'_{pred} = \omega((\begin{bmatrix}5\\5\end{bmatrix} - \begin{bmatrix}3\\2\end{bmatrix}) + \begin{bmatrix}1\\5\end{bmatrix}) = \begin{bmatrix}3\\8\end{bmatrix} \tag{2.8}$$

In the calculated reduced space this projected state[3][8][5][5], which has the same relative position of the predator to the prey as in the actual state, conforms with it's value 6.440, the same as we would in a maximum state space scenario. The reduced state space, effectively sized down to 11^2, results in a lot less memory requirements as well as less computation, which leads us to a faster runtime as seen in table 2.1.

Policy	Normal	Reduced
Policy Evaluation	13.04	0.35
Policy Iteration	28.27	0.28
Value Iteration	4.13	0.27

Table 2.1: Runtime of normal and reduced statespace in seconds

	0	1	2	3	4	5	6	7	8	9	10
0	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237	5.237	5.802
1	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792	5.792	6.436
2	7.839	8.780	10.000	0.000	10.000	8.780	7.839	6.997	6.251	6.251	6.997
3	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792	5.792	6.436
4	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237	5.237	5.802
5	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742	4.742	5.228
6	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291	4.291	4.712
7	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883	3.883	4.291
8	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883	3.883	4.291
9	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291	4.291	4.712
10	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742	4.742	5.228

Table 2.2: Values from value iteration when the prey is at [3][2]

3 Experiments and Results

3.1 Theoretical Differences

3.2 Off-Policy Monte Carlo Control

The purpose of this experiment was to implement Off-Policy Monte Carlo Control algorithm and to observe the results in terms of the target policy, maximum action value function, and also the number of step that the predator take to catch the prey. We also tried several approaches in the behavior policy in order to get a better results. Firstly we set the ϵ value in ϵ-greedy policy as a fix number, then we tried a different approach by making it dynamic by reducing the value after each fix number of iterations. The idea behind this approach was to put more chance in exploration at the beginning and slowly become more deterministic time after time.

Figure 3.1 shows the number of steps the predator took to catch the prey in every iteration governed by the target policy. The ϵ value was set to be fix of 0.1. In order to get a consistent measurement, the episodes were generated from fix start points of the predator and the prey, which were (0,0) and (5,5) respectively. The time that the predator took to catch the prey can be used to measure the performance of Off-Policy Monte Carlo Learning after each iteration, because the target policy uses the Q value immediately after each iteration. Since the target policy was greedy, then it always take best action based on the Q value. The better the Q value, the better target policy's performance should be.

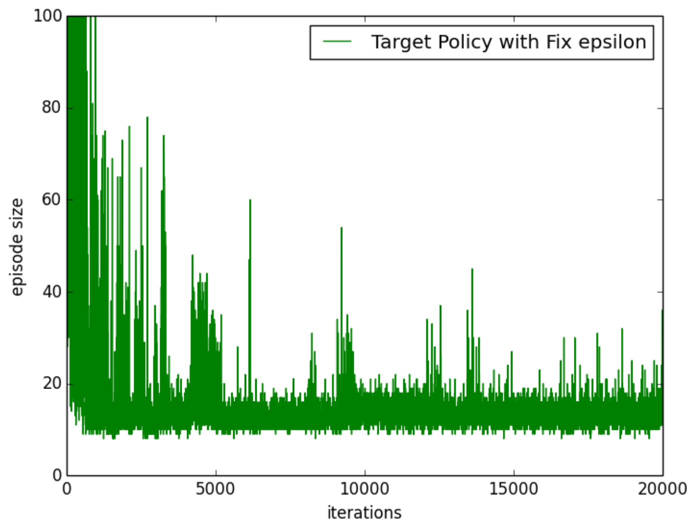


Figure 3.1: Catch time with fix epsilon value

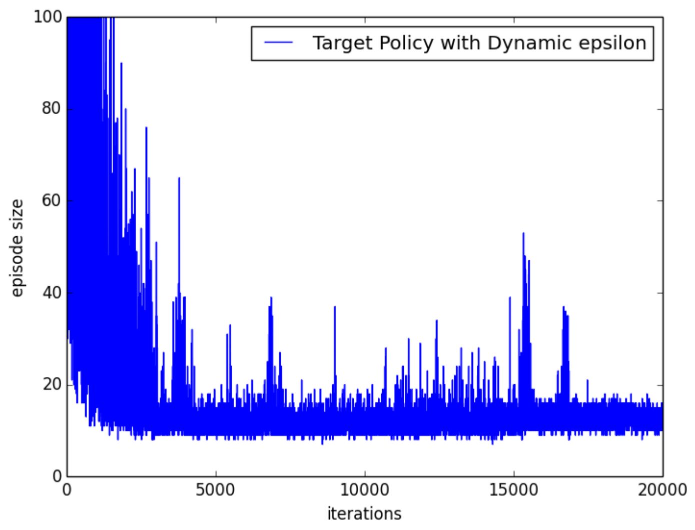


Figure 3.2: Catching time with dynamic epsilon value

Figure 3.3 was generated from the same setting as before except the value of ϵ was set to dynamically changed over iteration. First we set the value to 0.5 then after every 500 iterations it was reduced by multiplying with 0.8. In order to maintain the value not become too small, we set the minimum value to 0.1, because if ϵ is too small then it will become greedy and almost never explore anymore. As a consequence, learning processes will stop because both behavior and target policy always agree on actions taken in the generated episodes.

Even though the difference between the two figures above is not very significant, but we can still see the influence of setting the ϵ value to a fix and to a dynamic value. For fix value approach, the number of time drop quicker than the one from dynamic approach. This is because the fix value approach exploited more even from beginning due to the probability of exploiting much larger than exploring. On the other hand,the dynamic approach give the same probability between exploiting and exploring, so at the beginning it explored more than the fix value approach. Now let us see the last 3000 iterations from the two figures, it can be seen that dynamic value approach shows a slightly better performance, it fluctuates less than that in the other one.

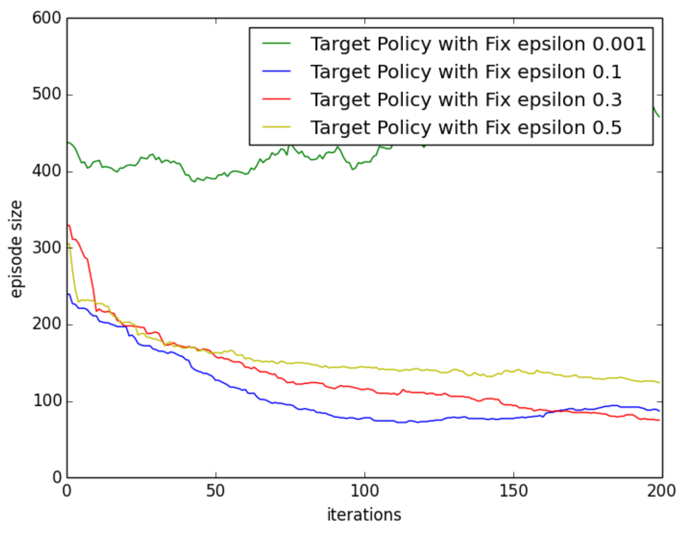


Figure 3.3: Catching time with dynamic epsilon value

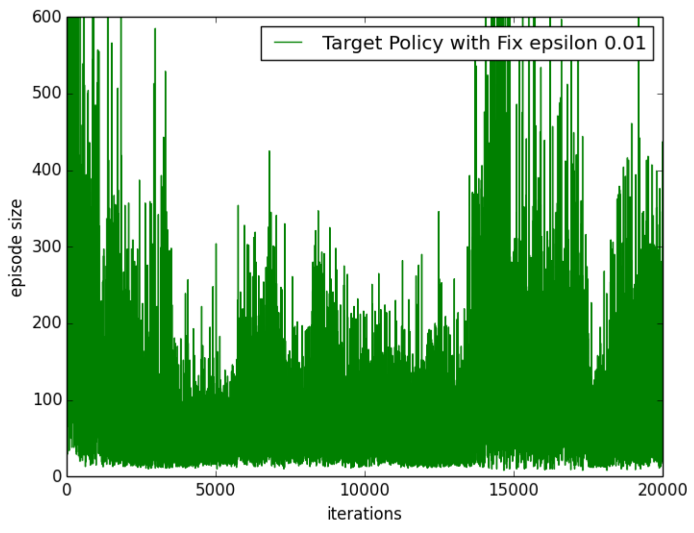


Figure 3.4: Catch time with dynamic epsilon value

	0	1	2	3	4	5	6	7	8	9	10
0	→	↓	↓	→	↓	↓	←	←	←	←	←
1	→	→	→	→	↓	↓	←	↓	←	←	←
2	↓	→	↓	→	↓	↓	↓	↓	←	←	↓
3	↓	↓	→	→	↓	↓	←	←	←	↓	↓
4	→	→	→	→	↓	↓	↓	↓	↓	←	←
5	→	→	→	→	→	P	←	←	←	←	←
6	→	→	↑	→	→	↑	←	←	←	↑	←
7	→	→	→	↑	→	↑	↑	←	←	←	←
8	↑	→	↑	↑	↑	↑	↑	←	←	←	←
9	→	↑	↑	→	↑	↑	↑	←	←	↑	↑
10	↑	↑	→	↑	↑	↑	↑	←	←	←	←

Table 3.1: Target Policy Off-Policy Monte Carlo Control

The goal of the experiment in this algorithm was to evaluate the arbitrary policy, and to measure the number of iterations it took to converge. There were two parameters that should be defined, the discount factor γ and a threshold value for the stopping condition θ. The values of the parameters along with some results are shown in the following table.

	Summary
γ	0.8
θ	1.0E-20
Number of iterations	106

Table 3.2: Policy evaluation results

The number of iterations is the number of sweeps (sweep is a backup operation for all states) until it achieved convergence.

Predator	Prey	$V(s)$
(0,0)	(5,5)	0.005724141401102873
(2,3)	(5,4)	0.18195076385152237
(2,10)	(10,10)	0.18195076385152237
(10,10)	(0,0)	1.1945854778368172

Table 3.3: State values for the following states

In our test environment, the goal of the planning is to make the predator catch the prey as fast as possible. So state-value functions in this case represent how close the predator is from the prey in a given state. The closer the predator is from the prey, the higher the state-value function should be for that state. Table 3.3 shows the state-value functions for four different states after performing iterative policy evaluation with parameters given in table 3.2.

The results make sense because the state value is proportional to the Manhattan distance between the predator and the prey. Intuitively the predator has a higher chance of catching the prey when it's closer.

3.3 Policy Iteration

For all the experiments with policy iteration theta was defined as $\theta = 1.0E^{-20}$, in order to perform a thorough sanity check, as it was suggested. As far as convergence is concerned it is presented in table 3.4, and as we can see for larger values of γ convergence is slower since more iterations are needed because Δ is smaller and it takes longer to reach θ.

γ	Evaluation runs	Improvement runs
0.1	100	8
0.5	227	7
0.7	323	8
0.9	763	10

Table 3.4: Convergence in iterations for different γ

Following a similar notion as in policy evaluation we present the state-value table in 3.5 for the states that are composed from the prey being at position prey(5,5). As in the case of policy evaluation the state values are proportional to the distance between the predator and the prey. The closer to the prey the higher the value. As we can also see the position of the prey is defined as the goal to be reached, hence the value will be 0 since there aren't any successor states and the episode ends.

	0	1	2	3	4	5	6	7	8	9	10
0	3.883	4.291	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883
1	4.291	4.712	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291
2	4.742	5.228	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742
3	5.237	5.802	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237
4	5.792	6.436	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792
5	6.251	6.997	7.839	8.780	10.000	0.000	10.000	8.780	7.839	6.997	6.251
6	5.792	6.436	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792
7	5.237	5.802	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237
8	4.742	5.228	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742
9	4.291	4.712	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291
10	3.883	4.291	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883

Table 3.5: Values from policy iteration when the prey is at [5][5]

3.4 Value Iteration

As in policy iteration, similar tables, namely 3.6, 3.7, for value iteration are presented, with $theta = 1.0E^{-20}$.

γ	Nr. of iterations
0.1	18
0.5	25
0.7	27
0.9	29

Table 3.6: Convergence in iterations for different γ

	0	1	2	3	4	5	6	7	8	9	10
0	3.883	4.291	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883
1	4.291	4.712	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291
2	4.742	5.228	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742
3	5.237	5.802	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237
4	5.792	6.436	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792
5	6.251	6.997	7.839	8.780	10.000	0.000	10.000	8.780	7.839	6.997	6.251
6	5.792	6.436	7.148	7.936	8.780	10.000	8.780	7.936	7.148	6.436	5.792
7	5.237	5.802	6.440	7.148	7.936	8.780	7.936	7.148	6.440	5.802	5.237
8	4.742	5.228	5.802	6.440	7.148	7.839	7.148	6.440	5.802	5.228	4.742
9	4.291	4.712	5.228	5.802	6.436	6.997	6.436	5.802	5.228	4.712	4.291
10	3.883	4.291	4.742	5.237	5.792	6.251	5.792	5.237	4.742	4.291	3.883

Table 3.7: Values from value iteration when the prey is at [5][5]

3.5 Comparison of Policy and Value iteration

We can see that tables 3.5 and 3.7 have the same values, and that can be explained because both methods converge to the optimal policy. The only difference can be the number of iterations it takes to converge to that optimal policy. Value iteration has only one backup operation for each state and reduces the policy evaluation to one step, whereas policy iteration performs multiple evaluations until it reaches the improvement part. That's why it will converge much faster than policy iteration which can be shown in graph 3.5.

Figure 3.5: Comparative graph of convergence for policy and value iteration

4 Conclusion

To conclude we presented the algorithms of policy evaluation, policy iteration, value iteration as well as their results on a predator-prey environment which consists of a toroidal grid. Through the presentation of those results we gained better insights on the advantages and disadvantages of using a specific algorithm. In our assignment value iteration converges faster to the optimal policy than policy iteration, and seems to be better in that regard. However maybe in some specific problems we need a decoupling of policy evaluation from improvement, because further evaluation is not needed. This can only be done with policy iteration.

Additionally, we reduced the statespace from 11^4 states to 11^2 states and we can actually reduce it even more, and we plan to try it for assignment 2, ie a quarter of that same statespace by employing mirroring. The idea came from the visualization of the distribution of values of the states around the projected prey. It is not as straightforward as the transformation used in the initial reduction because state actions need to be mirrored as well.

References

[1] Richard S. Sutton, Andrew G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, A Bradford Book, 1998.