Single Agent Learning

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

October 11, 2013

Abstract

The aim 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. For completing 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 optimal high-reward policy which yields from the fastest moves of the predator in order to catch the prey.

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.

In this assignment, we have explored the learning methods for estimating value functions and discovering optimal policies.
Monte Carlo methods (MC methods) 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 environment's dynamics, yet can still achieve an optimal behavior. Concerning the disadvantages of these methods, Monte Carlo method is conceptually simple and do not require any model, 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 based on other learned estimates, without waiting for a final outcome like MC:

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

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

During this research, the main focus is to understand better the idea of each model-free method and to highlight some demands like: What are the differences, both theoretical and practical, between all the implemented methods? Which method converges to the most optimal policy faster?

In order to do that, we have structured the report in 4 sections comprising: section 2, where we explain the theoretical background of each method and learn how to implement the pseudo-code; in section 3 we present some of the theoretical differences between the methods and we show the experiments through results and graphics, afterwards in section 4 we discuss about the brief conclusion of this assignment.

2 Methods and Procedures

2.1 Action selection

ϵ-greedy action selection

ϵ-greedy methods are policies, where non greedy actions are taken at random with a probability of ϵ, a small positive number. The remaining bulk of the probability, $1-\epsilon+\frac{\epsilon}{|A(s)|}$ is assigned to the greedy action. Thus, most of the time an action is chosen that has maximal estimated action value (exploitation), yet ϵ allows the discovery of more optimal actions - exploration.

Softmax action selection

The temperature τ in Softmax serves a similar purpose as ϵ in ϵ-greedy - it governs the rate of exploration. High temperatures cause the actions to be all (nearly) equiprobable and low

temperatures cause a greater difference in selection probability. It's distribution is given as:

$$P(a) = \frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}} \tag{2.1}$$

When τ reaches 0 in the limit, the action selection becomes the same as greedy action selection. The difference with ϵ-greedy is that it varies the action probabilities as a graded function of the estimated value, meaning the worst actions are unlikely to be chosen, where as the aforementioned chooses equally among all actions when in exploring mode.This is a good approach to take when the worst actions are very unfavorable.

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 select 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 of 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 2.2, 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.

$$R_t = \sum_{k=0}^{K} \gamma^k r_{t+k+1} \tag{2.2}$$

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.

2.3 Off-Policy Monte Carlo

The off-policy Monte Carlo control method follows the behavior policy while learning and improving the estimation policy. This method requires that the behavior policy should have a

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 ε-soft policy

Repeat forever:
(a) Generate an episode using π
(b) For each pair s,a appearing in the episode:
$R \leftarrow$ return following the first occurrence of s,a
Append R to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(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

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 taken actions 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 behavior.

To compute the return by 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 also under π' which is $p'_i(s)$. Therefore, the Monte Carlo estimation after observing n_s returned from state s is given by:

$$V(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'(s)} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p'(s)}} \tag{2.3}$$

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

$$\frac{p_i(s_t)}{p'_i(s_t)} = \frac{\prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) P_{s_k s_{k+1}}}{\prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) P_{s_k s_{k+1}}} \tag{2.4}$$

$$= \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)} \tag{2.5}$$

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 (2.4) 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 *behavior policy*. An advantage of this method is that the *target policy* may be deterministic whereas the *behavior 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 μ and use it to generate an episode:
    $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T$
  (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 τ 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 attempts to evaluate or improve the policy which is simultaneously used to make decisions.

SARSA ON-POLICY TD CONTROL

Sarsa is an on-policy method of TD learning, whose name (State-Action-Reward-State-Action) is derived from 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. It expresses the transition from one state-action pair to the next and its update rule is as follow:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.6}$$

The algorithm for Sarsa is 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., ε-greedy)
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g., ε-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, then $Q(s', a')$ will be defined as zero.

2.5 Q-LEARNING

Q-learning is a model-free method in the class of Temporal-Difference Learning methods. The main difference with the aforementioned MC methods is that TD methods also bootstrap. They update estimates based on other learned estimates, without waiting for a final outcome.

Q-LEARNING OFF-POLICY TD CONTROL

Q-learning is an off-policy TD control algorithm, unlike Sarsa. It directly approximates the optimal policy Q^* independent from the policy being followed. Its update rule is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.7}$$

The influence of the policy is still in determining which state-action pairs are visited and updated. However, as long as all pairs continue to be updated, which is the minimal requirement to find optimal behavior in the general case, correct convergence shall be reached. Convergence to Q^* with a probability of 1, requires stochastic approximation conditions on the sequence of step-size parameters, i.e. discounted ϵ, α. The algorithm is presented in Figure 2.4 below:

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

Figure 2.4: Q-learning: An off-policy TD control algorithm.

3 Experiments and Results

3.1 Theoretical Differences

The differences that appear while considering the off-policy Monte Carlo and on-policy Monte Carlo methods are related to the theoretical contrast between an off-policy and an on-policy. In the on-policy method, the agent commits to always exploring and tries to find the best policy that still explores. In off-policy method, the agent also explores, but learns a deterministic optimal policy that may be unrelated to the policy followed. If the on-policy method is generally soft (e.g. ϵ-soft policy) and improves the policy that is used to make decision, then the off-policy method is composed of two separated functions (behavior policy and target policy) which can be considered an advantage, because the target policy may be deterministic (e.g. greedy), while the behavior policy can continue to sample all possible actions.
The same differentiation can be applied between Sarsa and Q-learning. Sarsa is useful when you want to optimize a bounded exploratory policy of an agent while exploring - it is on-policy. When it's possible to separate the policies, meaning the policy being learned could be unrelated to the one followed, it might be more advantageous to use Q-learning, which is an off-policy method. Difference between TD and MC methods are that the aforementioned bootstrap; they update estimates based on other learned estimates, without waiting for a final outcome like MC. This lends to a naturally implementation in an on-line, fully incremental fashion.

3.2 Q-LEARNING WITH ϵ-GREEDY: EXPERIMENTS ON α AND γ

Q-LEARNING WITH ϵ-GREEDY: EXPERIMENTS ON α

Recall the update rule for Q-learning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{3.1}$$

We see that both α and γ influence the rate of update ("learning-rate") on the current Q value of a state action pair. While α is overall inclusive and taking the observed next reward into account, γ controls solely the ratio between the current Q value and that of the next state. Taking this into account we will expect to have a slower convergent rate for low values of α and γ. The question remains how much they will influence the convergence to the optimal policy in the long run.

We experiment with the following values for α: 0.1, 0.2, 0.3, 0.4 and 0.5, with the following conditions on the experimental setup:

Parameters	Value
ϵ	0.1
initial Q	15
γ	0.9
N runs	20000

Table 3.1: Experimental setup for testing α

The results from the experiments are displayed below. In order to give a comprehensible graph, we plot the average of batches of a 100 over the 20000. We see that lower values of α give slower convergence as expected, which is clearly illustrated in Figure 3.2 and all of the values converge to average sub 10 capture times after about 1250 runs. Less clear is its influence in the long run, as displayed in Figure 3.3. Capture times are fluctuating between values of 5 and 6, seemingly giving equal performance rates for all values of α in the long run.

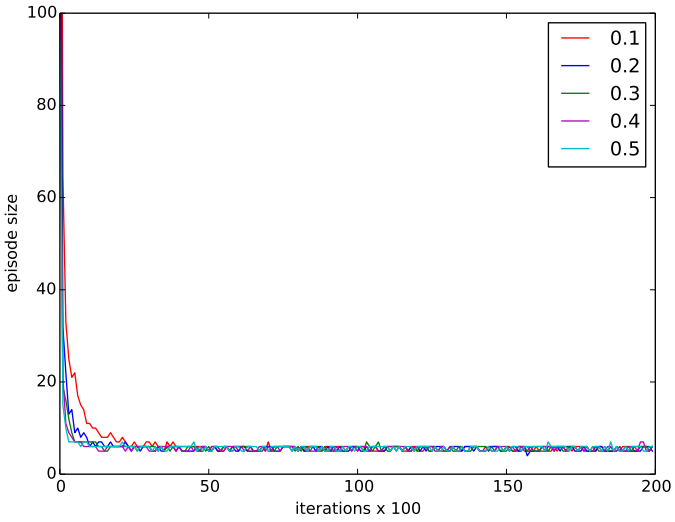

Figure 3.1: Convergence overview for different values of α.

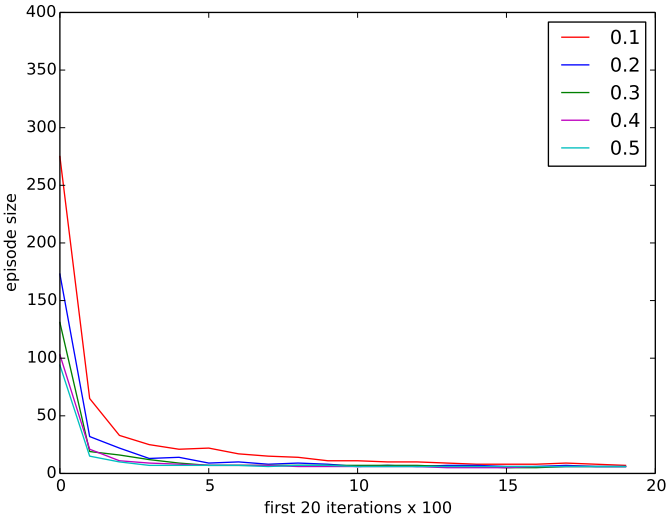

Figure 3.2: Convergence for different values of α in the initial 2000 runs.

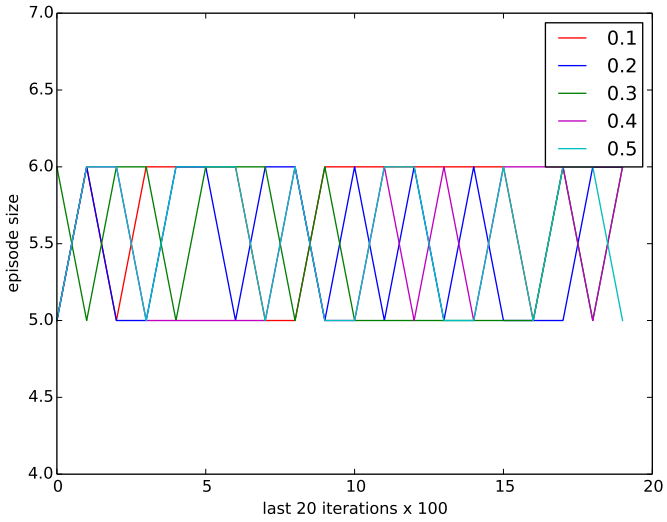

Figure 3.3: Convergence for different values of α in the last 2000 runs.

Since we acquired the optimal policy from value iteration in the first assignment, displayed in Table 3.2, we could plot the performance in terms of a ratio from the optimal policy. We take the argmax of the Q-table after a test run and compare that to the optimal policy. Its ratio will define its optimality:

$$Optimality = \frac{Diff(\text{nr of optimal actions})}{Total(\text{number of actions from the optimal policy})} \tag{3.2}$$

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

Table 3.2: Optimal policy in the reduced state space from Value Iteration given the prey at position (5,5).

The results, plotted as the average of batches of a 100 over the 20000, are displayed in Figure 3.4. For the initial stage of the runs we see that convergence confirm that what was expected

and earlier graphed in the average capture times. Far more interesting is the performance in the long run for different values of α. By comparing directly in the optimal policy space we essentially reduce the noise from the output of the many runs, and thereby reducing the effect of the random element of the prey and the initial start conditions within each run. Since the latter could randomly favor a run, and thereby it's capture time, regardless of the policy following.

In a state space this small, all of the tested policies would converge to similar capture times in the end, caused by the stochastic elements in which each run is initiated. Looking directly at the optimality graph however, we can say that, although its initial convergence is slower, an α value of 0.2 actually performs better in the end. Further tests below would present us, whether this claim withholds or there might be more interesting interactions between the parameters tested.

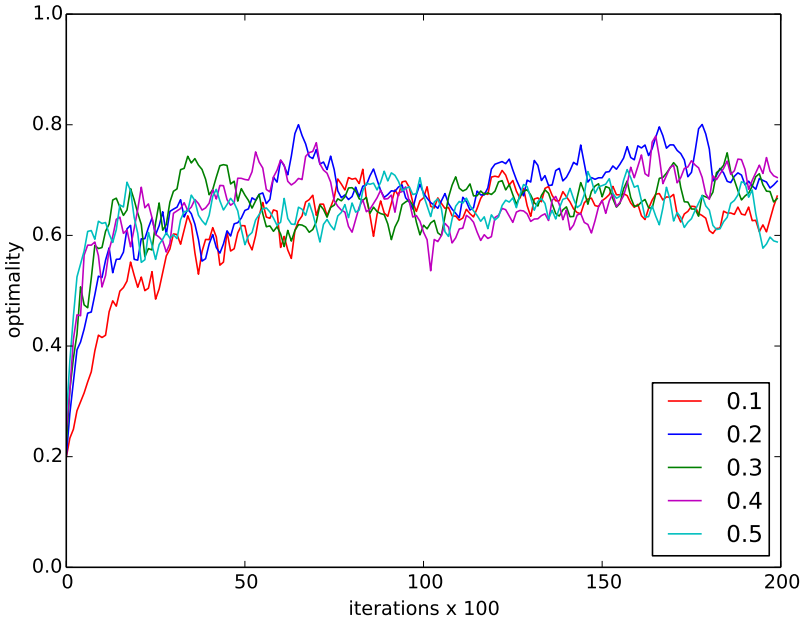

Figure 3.4: Optimality for different values of α as a ratio to the optimal policy.

Q-LEARNING WITH ϵ-GREEDY: EXPERIMENTS ON γ

Initial tests with γ are done with the following values: 0.1, 0.2, 0.5, 0.7 and 0.9, and with the following conditions on the experimental setup:

Parameters	Value
ϵ	0.1
initial Q	15
α	0.2
N runs	20000

Table 3.3: Experimental setup for testing γ

Again, the results are plotted with the average of batches of a 100 over the 20000, which can be assumed for all of the following experiments, unless stated otherwise. The outcome is comparable to that of α; lower values of γ give slower convergence as expected, which is clearly illustrated in Figure 3.6. Interesting is the out-lier start of γ value 0.9, see Figure 3.6. This might be caused by unfavorable stochastic conditions or too greedy initial faulty learning. However, it quickly adapts and makes a sharp dive, overtaking the other values after 200 runs.

All of the values converge to average sub 10 capture times after about 1250 runs, saying very little of the optimality of the policy being learned in the long run. Capture times are eventually bouncing between values of 5 and 6, apart of an out-lier value of 7 for γ value 0.7, which might be due to effects as earlier described, see Figure 3.7.

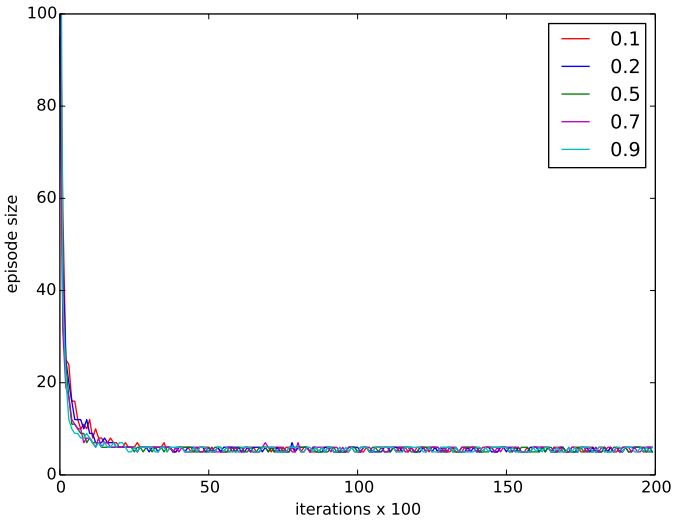

Figure 3.5: Convergence overview for different values of γ.

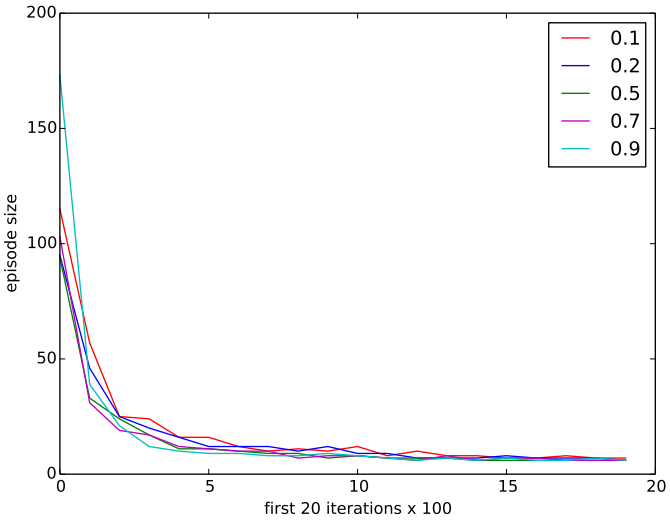

Figure 3.6: Convergence for different values of γ in the initial 2000 runs.

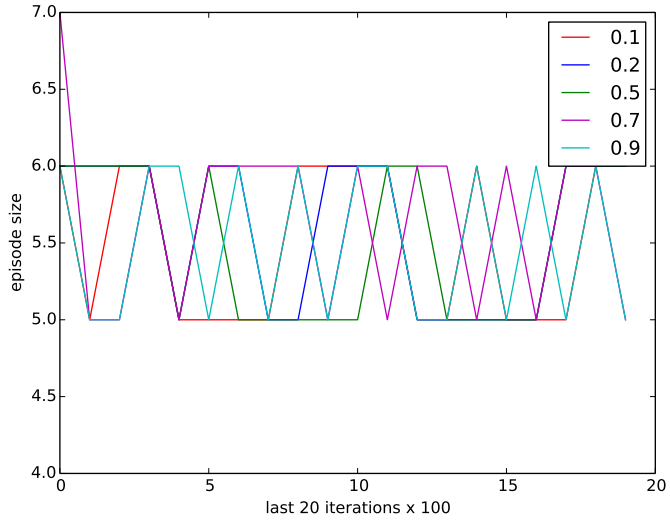

Figure 3.7: Convergence for different values of γ in the last 2000 runs.

If we look directly into policy optimality space, we see that lower values of γ show promising results in the end. Interesting is the climb of γ value 0.9 therein. This might be due to negating effects to the chosen lower value of α of 0.2. Runs with the contrasting value of 0.5 for α are plotted against these promising values of γ to see if this is indeed the case, see Figure 3.9.

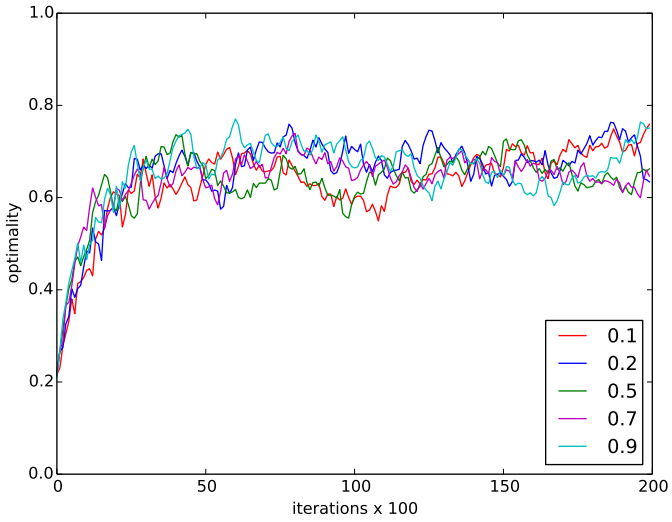

Figure 3.8: Optimality for different values of γ as a ratio to the optimal policy.

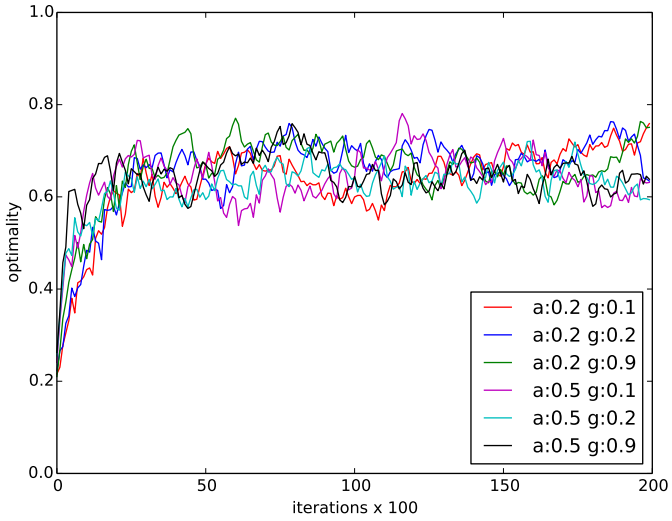

Figure 3.9: Optimality with the interaction of α and γ.

Indeed, higher values of α have a much greater impact as earlier foretold (overall inclusive influence on the update). We see a fast initial climb, but in the long run will impair further convergence and the negating effect of a lower γ has little influence.

3.3 Q-LEARNING WITH ϵ-GREEDY: EXPERIMENTS ON ϵ AND OPTIMISTIC INITIALIZATION

Q-LEARNING WITH ϵ-GREEDY: EXPERIMENTS ON ϵ

We experiment with the following values for ϵ: 0.001, 0.1, 0.2, 0.3, and 0.5, with the following conditions on the experimental setup:

Parameters	Value
α	0.2
initial Q	15
γ	0.1
N runs	20000

Table 3.4: Experimental setup for testing ϵ

ϵ in ϵ-greedy controls the probability in which a random action is selected. Lower values of ϵ will steer its policy to a greedy action selection, while higher values will favor exploration more, thus a faster initial convergence, yet a less optimal policy in the end. With the earlier chosen values of ϵ we shall see if that's reflected in our graphs. From the results below, we see that high values of ϵ show fast initial convergence due to exploration, but give a slow convergence rate in the second stage. They even converge to a less optimal policy in the end. The average capture times are higher overall and its policy is highly fluctuating, as can be seen in Figure 3.13. Without discount of ϵ in the long run the algorithm is always stuck in exploration mode due to the fixed degree of ϵ.

On the other hand the relatively extreme low value of 0.001 for ϵ causes the algorithm to slowly climb and stabilize much more in the long run, giving it the best final average capture time, even though it is not peaking in the ratio of its optimality. The low value gives the algorithm its robustness in the end, not "being tempted" by exploration.

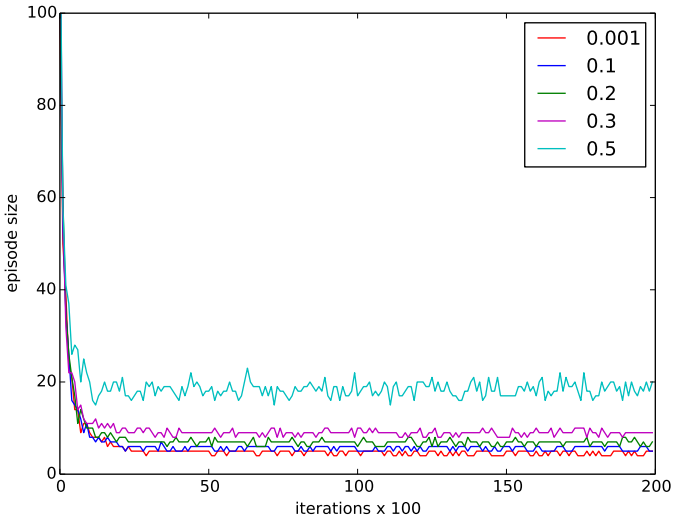

Figure 3.10: Convergence overview for different values of ϵ.

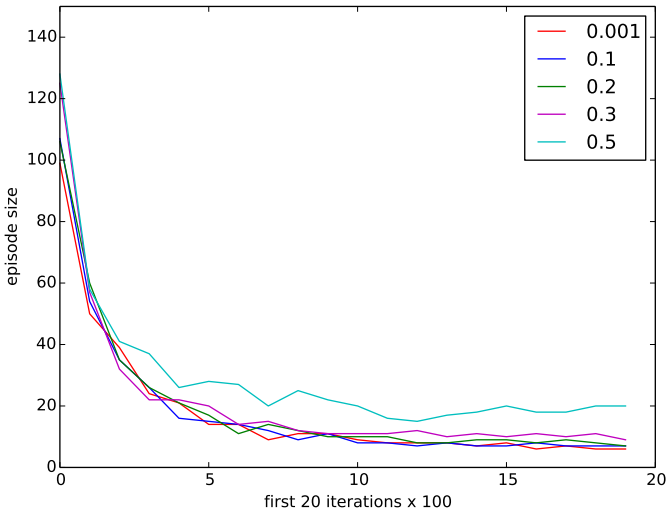

Figure 3.11: Convergence for different values of ϵ in the initial 2000 runs.

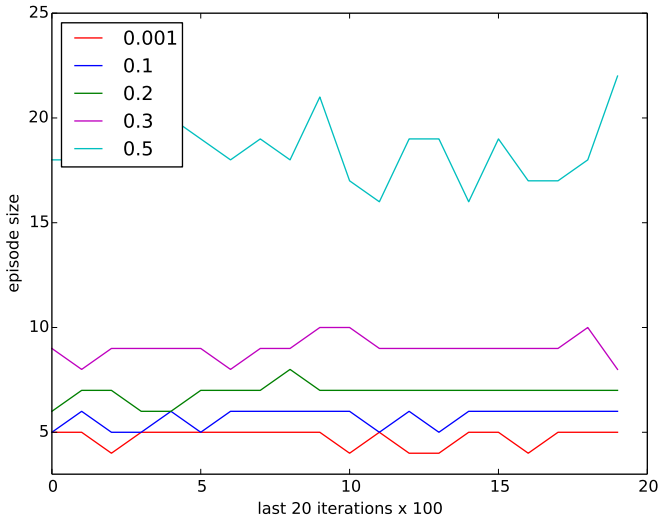

Figure 3.12: Convergence for different values of ϵ in the last 2000 runs.

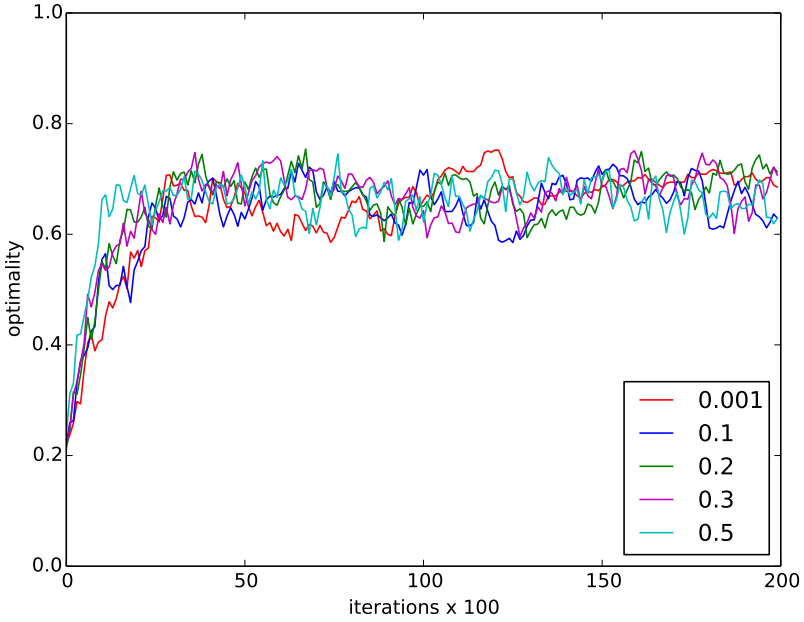

Figure 3.13: Optimality for different values of ϵ as a ratio to the optimal policy

For the optimistic initialization of the Q-table we decided to start with the values derived from the optimal policy acquired from Value Iteration, see Table 3.6. Actions corresponding to those of the optimal policy, see Table 3.2, will get its assigned value, while others are given the value 0. Further tests are done with initialization values of 0, 5 and 10, added to the value of 15 of earlier experiments. Conditions on the experimental setup are as follow:

Parameters	Value
α	0.2
ϵ	0.1
γ	0.1
N runs	20000

Table 3.5: Experimental setup for testing optimistic initialization

	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 3.6: Values from value iteration when the prey is positioned at [5][5]

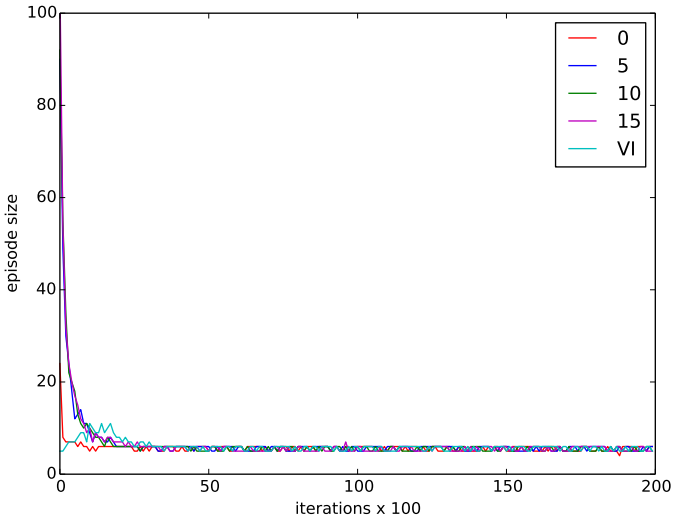

Figure 3.14: Convergence and divergence overview for variable optimistic initialization.

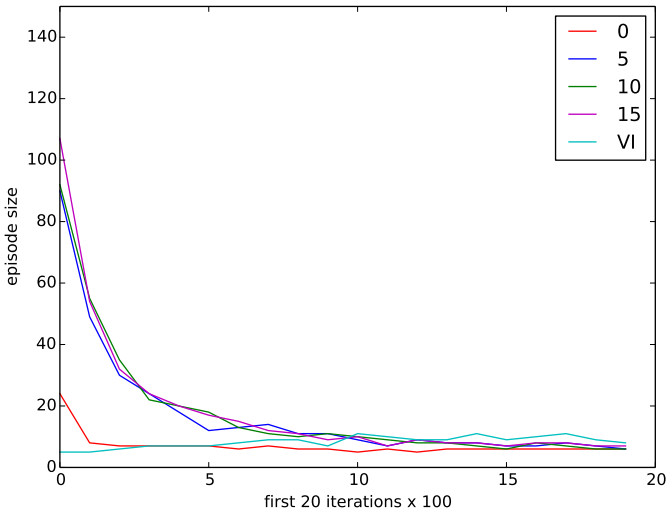

Figure 3.15: Convergence and divergence for variable optimistic initialization in the initial 2000 runs.

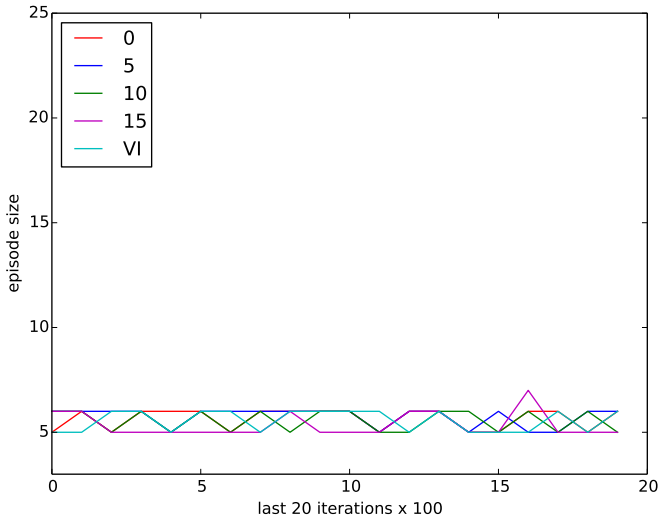

Figure 3.16: Convergence for variable optimistic initialization in the last 2000 runs.

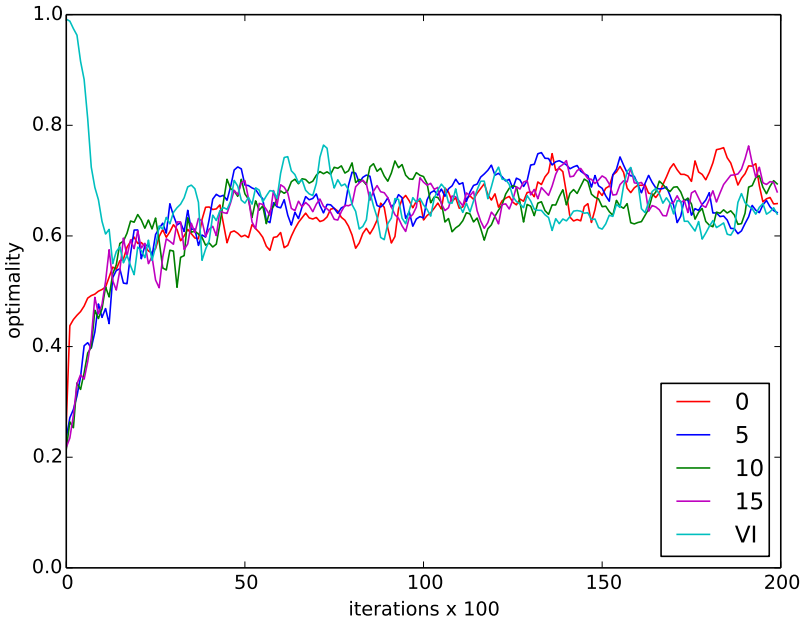

Figure 3.17: Divergence from the optimal policy after optimal initialization in case of VI.

Results for the optimal policy initialization are not surprising, as it confirms our earlier notion that fixed exploration by its parameters will not allow the algorithm to converge back

to its optimal policy. We see a sharp drop in its optimality and climbing initial average capture times, due to its straying from the aforementioned. In the end its performance is comparable to the other initialization values and reaches a certain equilibrium governed by the parameters of α, ϵ and γ.
In the initialization value of 0 it's interesting to see that the relatively large difference from the maximum reward of 10, allows it to converge a lot faster in comparison to the other initialization values, but quickly loses that advantage when the difference in the value of initialization and maximum reward become smaller. It seems that greediness can also be achieved by an initialization of values with a large δ from the reward.

3.4 Q-LEARNING WITH SOFTMAX ACTION SELECTION: EXPERIMENTS ON τ AND COMPARISON WITH ϵ-GREEDY

As in ϵ for ϵ-greedy, τ (temperature) governs the rate of exploration. High temperatures cause the actions to be all (nearly) equiprobable and low temperatures cause a greater difference in selection probability. When τ reaches 0 in the limit, the action selection becomes the same as greedy action selection. The difference with ϵ-greedy is that it varies the action probabilities as a graded function of the estimated value, where as the aforementioned chooses equally among all actions when in exploring mode. With this notion we expect Softmax to "degrade" more gracefully than ϵ-greedy. Experiments on τ are done with values of $0.1, 1\times10^{-2}, 1\times10^{-3}, 1\times10^{-4}$ and 1×10^{-5}, in the following experimental setup:

Parameters	Value
γ	0.1
initial Q	15
α	0.2
N runs	20000

Table 3.7: Experimental setup for testing τ

We compare the results with ϵ-greedy with the same setup as in Table 3.7 and a fixed value of 0.1 for ϵ:

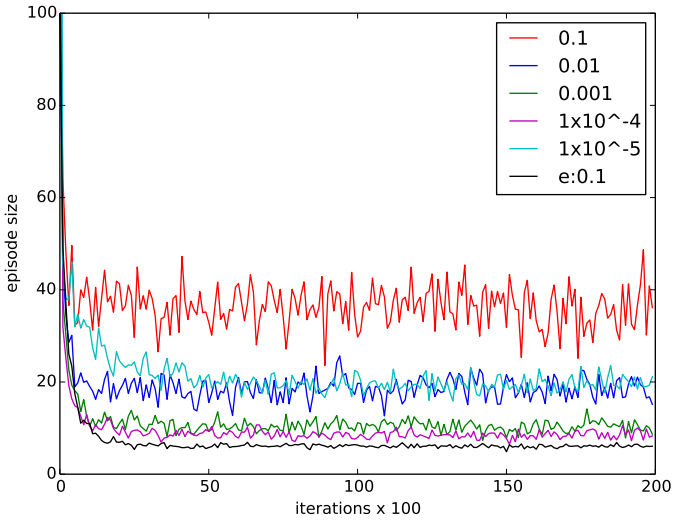

Figure 3.18: Convergence overview for different values of τ in comparison to ϵ-greedy.

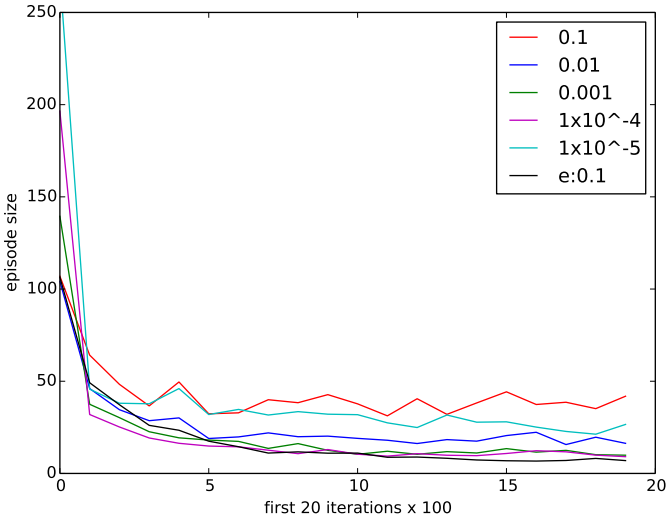

Figure 3.19: Convergence for different values of τ in the initial 2000 runs in comparison to ϵ-greedy.

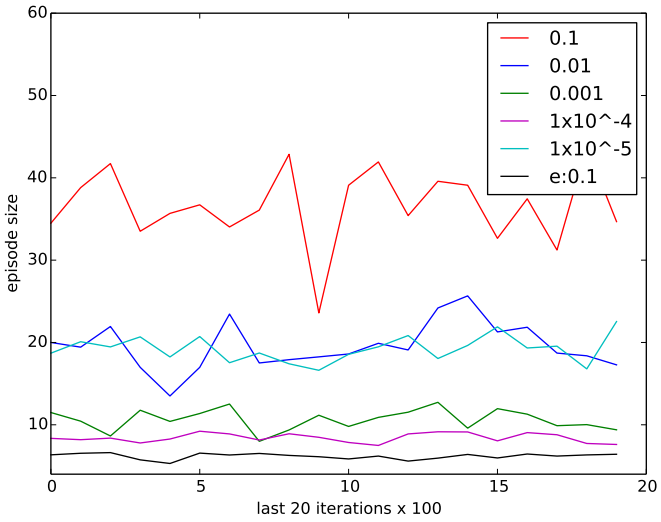

Figure 3.20: Convergence for different values of τ in the last 2000 runs in comparison to ϵ-greedy.

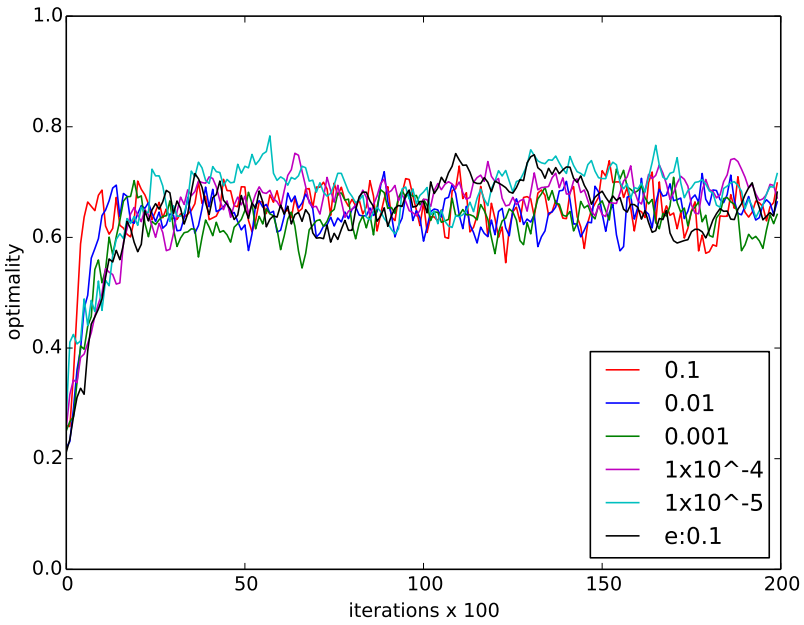

Figure 3.21: Optimality for different values of τ as a ratio to the optimal policy in comparison to ϵ-greedy.

From the results we see that Softmax has similar behavior to ϵ when its parameters are high valued. High temperatures favor exploration and give a very fast climb to optimality, but due to the fixation of the parameters, it's "stuck" in exploration mode and its policy is highly fluctuating with an fluctuating high average capture time as a consequence. As in the case of ϵ-greedy a discounted τ, inverse proportional to the run sequences, would benefit further convergence, as it allows the policy to gradually switch from exploration to exploitation.
In answering the graceful deterioration, we plot different values of τ against ϵ-greedy with aforementioned parameters in the experimental setup after initialization with the optimal policy, see Figure 3.22. We can see that while these values of τ didn't outperform in the average capture times with initialization of 15, see Figure 3.18 and being less greedy, they do show a much more graceful degradation in its optimality:

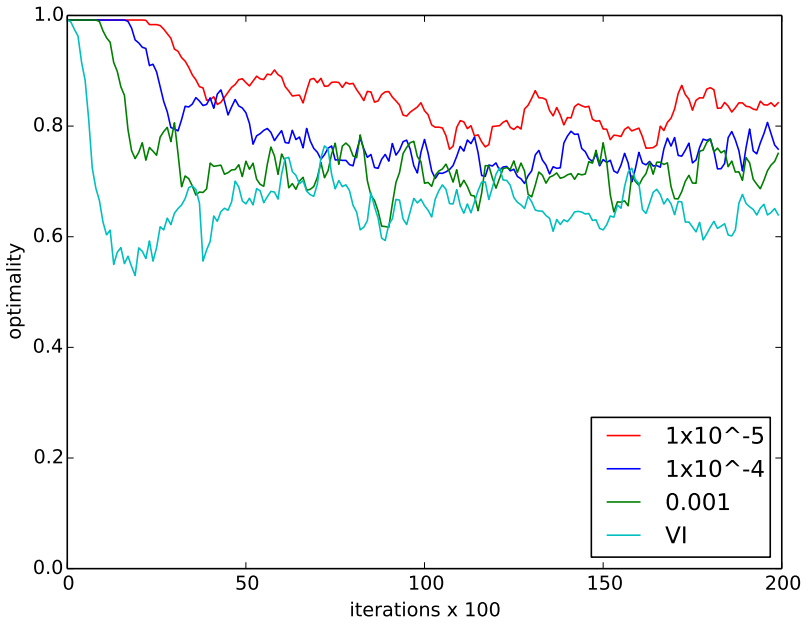

Figure 3.22: Deterioration from the optimal policy for different values of τ in comparison to ϵ-greedy.

3.5 SARSA: A COMPARISON WITH Q-LEARNING IN ϵ-GREEDY AND SOFTMAX ACTION SELECTION

The main difference between Sarsa and Q-learning is that Sarsa being an on-policy method, actually follows the policy being taken and updates its value in respect to that transition. Q-learning, however is directly approximating the optimal policy, regardless of the policy being followed. With this notion we expect Sarsa to converge slower than Q-learning. Experiments will show it that's reflected by the plotted comparisons between Q-learning and Sarsa, conditioned on Softmax and ϵ-greedy action selection. The experimental setup is as follow:

Parameters	Value
γ	0.1
initial Q	15
α	0.2
ϵ	0.1
τ	1×10^{-4}
N runs	20000

Table 3.8: Experimental setup for Sarsa and Q-learning comparisons.

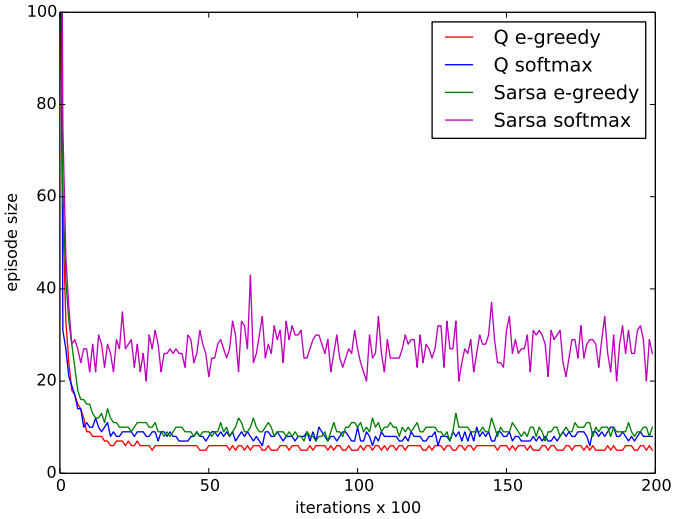

Figure 3.23: Convergence overview for Sarsa and Q-learning in ϵ-greedy and Softmax action selection.

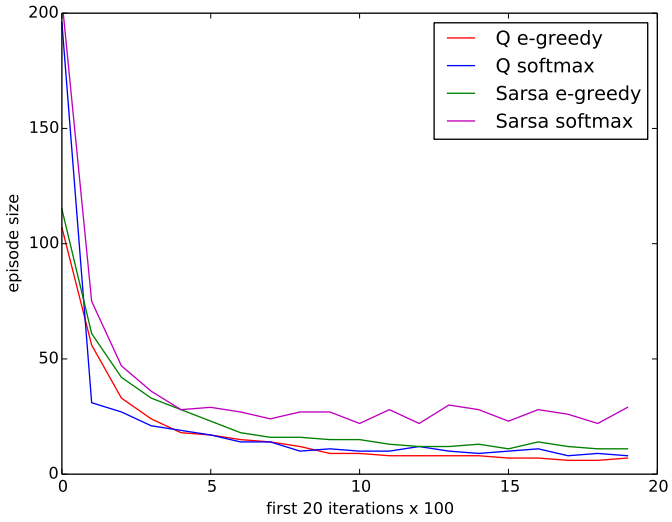

Figure 3.24: Convergence for Sarsa and Q-learning in ϵ-greedy and Softmax action selection in the initial 2000 runs.

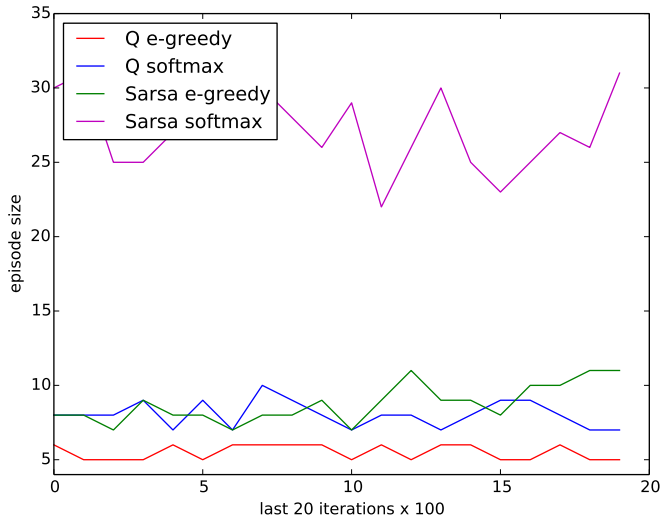

Figure 3.25: Convergence for Sarsa and Q-learning in ϵ-greedy and Softmax action selection in the last 2000 runs.

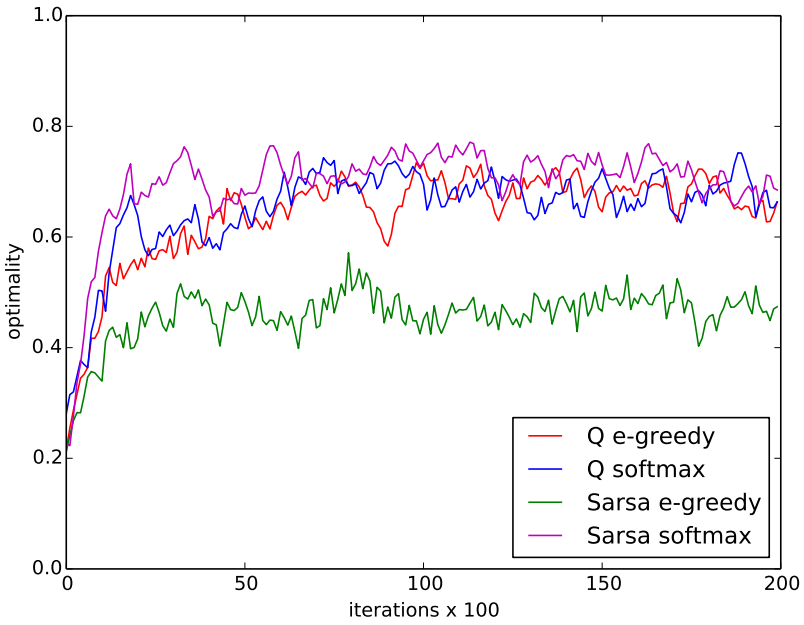

Figure 3.26: Optimality for for Sarsa and Q-learning in ϵ-greedy and Softmax action selection as a ratio to the optimal policy.

From the results we see that Sarsa is indeed converging slower than its counterpart in Q-learning. Softmax action selection with Sarsa seems to increase its tendency to explore. We see highly fluctuating high average run times, comparable to Q-learning with large valued ϵ with ϵ-greedy or even with that of large temperature τ with Softmax, see earlier Figures, while its value here is actually small, 1×10^{-4}. This exploration mode does favor a fast convergence to the optimal policy, yet it is not exploiting it, being hindered by its tendency to explore.

3.6 On-Policy Monte Carlo Control

By performing this experiment the goal was to see the performance of the On-policy Monte Carlo control in the predator prey environment. The maximum number of control iterations was fixed at 20000 for every run with different parameters. Additionally the final outcome of the optimal policy was recorded as well as the final values of the function $Q(s,a)$. The metric used in order to determine the performance between different approaches has the size of the episode that was generated each time in the algorithm. A smaller size, between $[1,15]$, shows that the predator catches the prey faster, whereas a bigger size denotes the opposite. Different values for the ϵ parameter for the ϵ-soft policy were tested, since ϵ directly influences the amount of exploring and exploiting the agent performs. Additionally, we also experimented on runs with optimistic initial values, namely 15, for the state-action function $Q(s,a)$, since it also influences the amount of exploration. Only fixed ϵ values were tested with optimistic initial values of $Q(s,a)$ since they performed better than the adaptive ones, as we will see in

the following section. The discount rate γ for the returns, was fixed at 0.8.

Experiment with different values of ϵ

The values that we tested were fixed ϵ at 0.7, 0.5, 0.3, and 0.1, as well as adaptive ϵ that decreases by 20% every 100 iterations, with different initial value. The results are shown in figure 3.27 below. The total number of iterations is 20000, where we take the average episode size every 100 episodes, in order for the visualization to be clearer. The value of ϵ directly

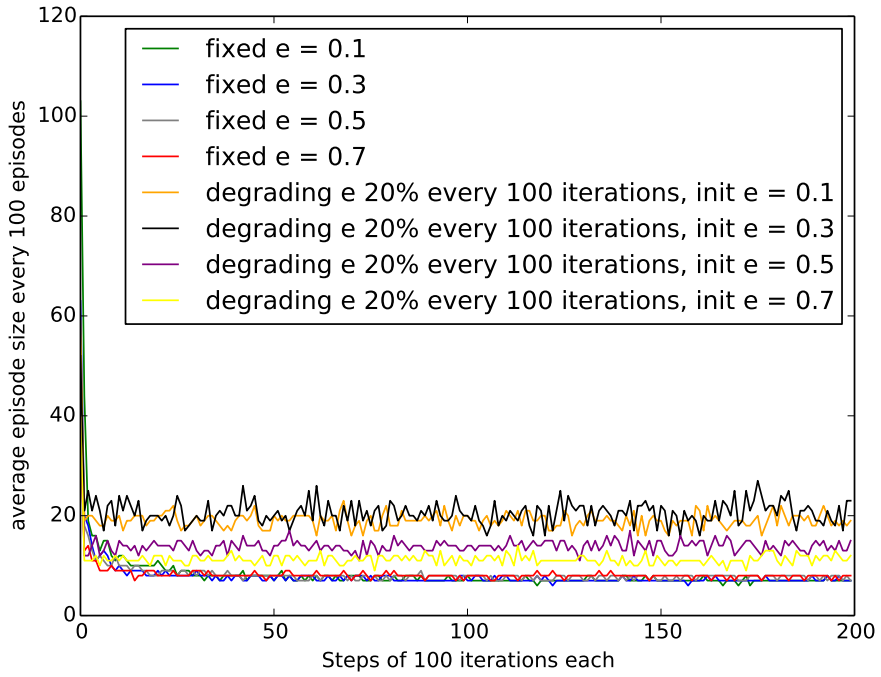

Figure 3.27: On Policy MC catching time with different epsilon value

influences the amount of time it takes for the agent to find a good estimate for the $Q(s,a)$ value. Fixed ϵ performs better than adaptive ϵ and this is expected since the degree of exploration is minimized as the number of iterations increases. The effect becomes worser when the initial value of ϵ is turning lower, since the degree of exploration is limited quite early, hence the agent cannot explore and get a better estimate for the Q value. However higher values allow more exploring, thus getting better results. Fixed ϵ achieves higher performance since the ratio of exploration/exploitation is consistently kept during all the control iterations, essentially making the agent explore even in later iterations of the control loop. We can see from 3.27 that higher values of ϵ achieve a sharper decrease in the episode size in the beginning, since there is a bigger episode size which derives from the high exploration probability, but they perform worse in later iterations since they don't exploit as much as lower values of ϵ. The overal best choice seems to be a fixed ϵ of 0.1 since it might converge slower to a better policy,

but it produces more consistent results in the later iterations, due to the increased rate of exploitation.

Experiment with optimistic initial value of $Q(s,a)$

Having an optimistic initial value of Q just forces the agent to explore in the begining which creates episodes with an extremely big size, which results in a sharper drop after that since the Q values have a reasonable value again. This approach didn't provide better results though since the estimated values were not obtained through episodes of a reasonable size, hence they were not very accurate. The results can be seen in figure 3.28 where we performed an average of the episode size for every 100 iterations again, in order to produce a clearer visualization.

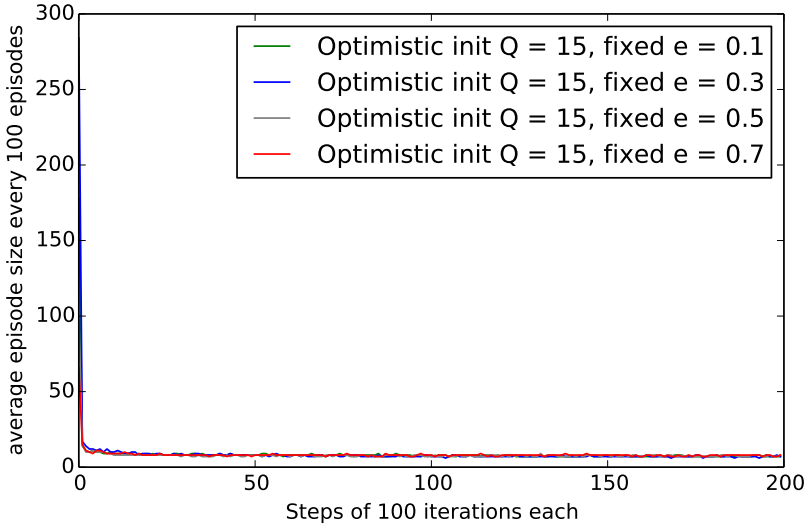

Figure 3.28: On Policy MC catching time with optimistic initial Q value

Convergence

As we have seen in figure 3.27 the algorithm seems to converge in an optimal policy in about $100 * 100 = 10000$ iterations, which explains the consistent low episode size and faster catch times for the predator. The optimal policy table derived with the best choice of epsilon is shown in 3.9 and the $Q(s,a)$ values that the policy was derived from are show in table 3.10. The values are reasonable since they follow the pattern which implies that the further away the predator is from the prey, then the lower the value will be.

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

Table 3.9: Target Policy Off-Policy Monte Carlo Control

	0	1	2	3	4	5	6	7	8	9	10
0	1.013	1.466	1.659	2.305	2.934	3.626	2.935	2.368	1.861	1.576	1.245
1	1.386	1.898	2.229	2.922	3.681	4.589	3.731	2.906	2.340	1.752	1.549
2	1.939	2.421	2.946	3.666	4.599	5.904	4.588	3.657	3.028	1.864	1.855
3	1.832	2.494	3.138	4.734	5.880	7.545	5.882	4.669	3.737	2.593	2.038
4	2.922	3.805	4.873	6.069	7.489	10.000	7.551	6.054	4.814	3.744	1.985
5	1.785	2.977	5.805	7.533	10.000	0.000	10.000	6.836	5.345	2.817	1.902
6	1.237	2.008	4.417	5.889	6.917	10.000	7.489	6.030	2.870	2.188	1.518
7	2.354	3.050	3.710	4.843	5.984	7.452	5.705	4.581	2.119	2.756	1.972
8	1.382	1.929	2.768	1.074	4.633	5.838	4.490	3.654	2.716	2.216	1.758
9	1.157	1.827	1.079	2.040	1.196	4.588	2.791	1.472	2.377	1.804	1.402
10	0.676	1.172	1.493	1.845	2.257	3.569	2.727	2.102	1.182	0.966	0.943

Table 3.10: Optimal state value function for each state given prey at (5,5)

3.7 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 steps the predator take to catch the prey. We tried two different approaches of setting the ϵ of the behavior policy, fixed value and dynamic value which will be explained in the next part. Different fixed ϵ value of ϵ-greedy policy also were tested to show how this parameter, which represents the probability of exploring and exploiting, affect the learning processes of action value function. We organize this subsection starting from presenting the results of using different approach of setting ϵ value of behavior policy, then we show the result of using several different fixed values of ϵ, and finally we will show the convergence of the target policy.

EXPERIMENT WITH DIFFERENT APPROACHES OF SETTING ϵ VALUE

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 reduced the chance thus it started to exploit more time after time, but also maintained it to be stochastic in order to keep the learning process run.
Figure 3.29 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.

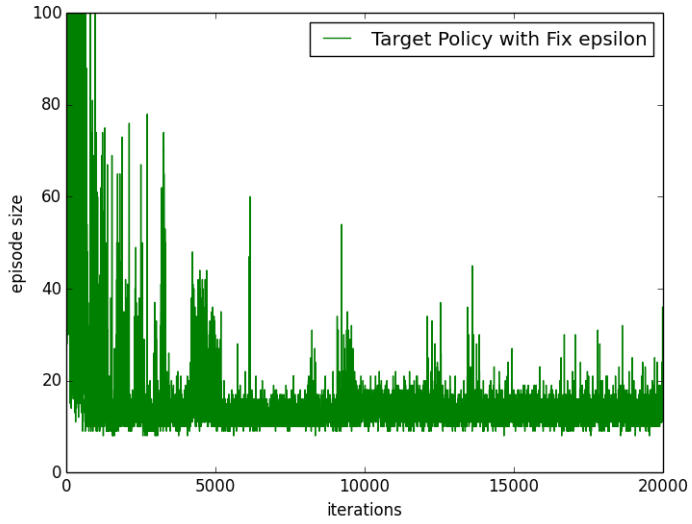

Figure 3.29: Catching time with fix epsilon value

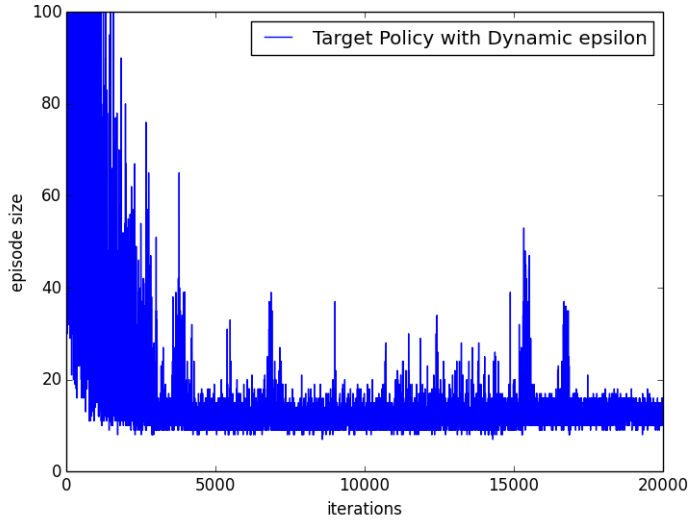

Figure 3.30: Catching time with dynamic epsilon value

Figure 3.30 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 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 figure. Since the difference is not significant then we can not claim which one is better. The general statement is that they both can converge with anough number of iterations.

EXPERIMENT WITH DIFFERENT VALUES OF ϵ

In this part we made some experiments by changing the fix value of ϵ. Since the epsilon represents the chance of exploring and exploiting in learning, then we will make a conclusion related to that matter.

The values that we tested was 0.5, 0.3, 0.1, and 0.001, and the result is shown by figure 3.31 below. The number of iterations is 10000, to make a clearer figure, we devided it into 200 parts, and averaged each part, so 1 iteration in the figure is average of 50 iterations.

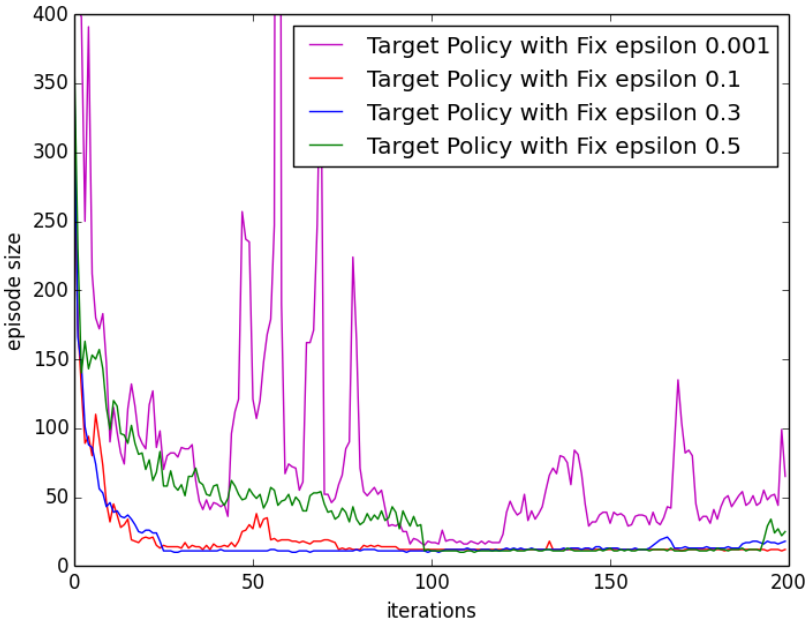

Figure 3.31: Catching time with different fixed epsilon value

The figure above tells us that in our test case, ϵ value of 0.1 and 0.3 give the best performance, while 0.5 result shows a slower convergence than the previous values and 0.001 become the worst. This result confirm that exploration and exploitation have a significant role in learning process. Setting the exploitation probability value too small makes the model hard to learn, it could be trapped to exploit wrong actions, where setting the value too high means small chance to exploiting and then leads to a late convergence.

CONVERGENCE

In this section we will show the convergence of the Off-Policy Monte Carlo Control method. Table 3.11 and table 3.12 show the optimal value function and the optimal target policy's actions respectively. Even though this method use sampling to estimate the target policy, but with enough number of iteration the convergence to the optimal value function can be reach. From several experiments, we know that as long as the behavior policy is stochastic and has reasonable ϵ value, like what we have explained in the previous part, then it will converge. The results below were generated from 100.000 iterations with fixed ϵ 0.3.

	0	1	2	3	4	5	6	7	8	9	10
0	1.282	1.501	2.068	2.514	3.149	3.673	2.949	2.488	2.028	1.626	1.324
1	1.540	2.055	2.469	3.208	3.974	4.694	3.951	3.185	2.573	2.006	1.791
2	1.896	2.691	3.198	3.987	4.939	6.054	4.958	3.999	3.201	2.576	1.965
3	2.622	3.233	3.985	4.943	6.150	7.611	6.198	4.971	4.027	3.191	2.395
4	3.058	4.004	4.960	6.156	7.609	10.000	7.631	6.192	4.712	3.837	2.962
5	3.763	4.738	6.005	7.633	10.000	0.000	10.000	7.616	6.026	4.729	3.721
6	3.207	3.967	4.928	6.159	7.620	10.000	7.597	5.981	4.852	3.699	3.007
7	2.587	3.240	4.007	4.987	6.011	7.624	6.164	4.887	3.912	3.144	2.540
8	2.129	2.202	3.174	4.005	4.830	6.037	4.749	3.781	3.073	2.494	2.073
9	1.695	2.091	2.624	3.240	3.747	4.752	3.845	2.947	2.564	2.021	1.511
10	1.199	1.706	1.964	2.349	2.925	3.733	3.038	2.451	1.971	1.715	1.378

Table 3.11: Optimal state value function for each state given prey at (5,5)

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

Table 3.12: Optimal target policy's actions for each state given the prey at (5,5)

3.8 Comparison of all approaches to the learning problem

Regarding on and off policy Monte Carlo control, in the case of the best parameters ϵ for both of the algorithms, the On-policy Monte Carlo performs better. It converges faster to an optimal policy that produces faster catch times for the predator and manages to keep them consistent in the later episodes. On the contrary Off-policy seems to converge slower and has some fluctuations on the catch times which was derived from the fact that the optimization of the target policy was done through a different behaviour policy. Furthermore, in the off-policy method, state-action pairs that will be used to estimate are only pairs that occur after the last different action between the two policies. So it is likely to use fewer samples than those in on-policy with the same number of iterations. The average catch times for the last 100 episodes are shown in table 3.13 and show the fact that on-policy Monte Carlo produces a better policy.

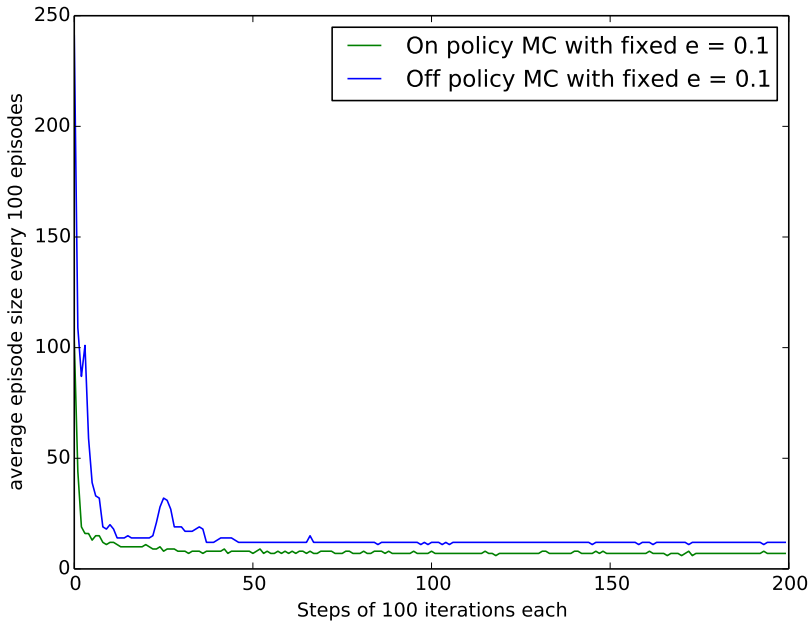

Figure 3.32: Catching times for best choice of ϵ for On and Off policy MC

Method	Average
Q-Learning	5
Sarsa	11
On-policy MC	7
Off-policy MC	12

Table 3.13: Average catching time for all algorithms at convergence

4 CONCLUSION

In this research, we have explored the problem of the single agent learning by using different methods that highlight this problem. Monte Carlo methods as well as Temporal-Difference methods have been used, namely On and Off policy Monte Carlo control, Q-Learning and Sarsa.
Regarding the on-policy Monte Carlo Control method, we conclude that the optimal policy can be reached quite fast, since the convergence happens in a relative few episodes, and finally the average catching time is low, which is a good indication of an optimal policy.
Even though it seems to converge slower than the on-policy, but in most cases, with the stochastic behavior policy and enough number of iterations, the off-policy Monte Carlo method will converge to the optimal policy. In our test case, 0.1 and 0.3 of epsilon value performed best.

The advantage of off-policy over on-policy is that the policy that we want to estimate can be greedy, so if we use this policy in the middle of learning, it exactly represents the condition of the value function at that time.

REFERENCES

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