Autonomous Agents 1: Assignment 2

Tran Cong Nguyen, Lautaro Quiroz, Joost van Doorn, Roger Wechsler

November 28, 2014

1 Introduction

In this assignment we use the same predator-prey environment for the last assignment. In the last assignment we compared different planning algorithms that we used to compute optimal policies for the predator. For those planning algorithms the model of the environment was entirely known during planning. This means, in particular, that for a transition of state s to s' and for an action a the transition probabilities $\mathcal{P}^a_{ss'}$ and the rewards $\mathcal{R}^a_{ss'}$ are entirely known.

In this assignment, however, we use learning algorithms that do not make use of the model. Learning algorithms make use of actual experience, i. e. we let the predator and the prey play their game and observe their steps. Of course, as we are simulating the game, the model is entirely known to us. Our agent, the predator, however, does not know how the environment behaves and thus can only learn while actually interacting with it.

Throughout the following sections we will discuss the theoretical differences of various learning algorithms and compare our expectations against some empirical results that we obtain from a number of simulation experiments.

2 Background

Q-learning, Sarsa and Monte-Carlo Control are the learning algorithms used in this report. The main property of Q-learning and Sarsa is that they are temporal-difference learning algorithms. Temporal-difference algorithms update their state-action values at each step during the episode. This means that they are likely to improve its policy faster than an algorithm not being updated during the episode. There are two types of Monte-Carlo learning algorithms considered in this report, On-Policy Monte-Carlo and Off-Policy Monte-Carlo. These algorithms learn only after one episode is complete. Sarsa and On-Policy Monte-Carlo are both on-policy learning algorithms, whereas Q-learning and Off-policy Monte-Carlo Control are off-policy algorithms. On-policy algorithms follow the policy that it updates while an off-policy algorithm follows a different policy than which it is optimizing.

3 Methodology

We implemented the Q-learning, Sarsa, On-policy Monte-Carlo Control and Off-policy Monte-Carlo Control learning algorithms. We start by analyzing the Q-learning algorithm by varying the learning parameters and measure convergence by the steps it takes for the predator to catch the prey after each episode. We have also implemented softmax and ϵ-greedy action selection and compared their convergence. All plots are generally sampled from 100 runs of the learning algorithm, no smoothing functions are applied to any of the plots.

We compare Q-learning to the other learning algorithms. Using the root-mean-squared-error (RMSE) performance measure we measure the convergence of these algorithms, we measure the number of optimal actions taken and compute the error based on the number of incorrect actions. The number of optimal actions are based on value iterations, where we also consider multiple optimal actions when there are more. Finally we compare all learning algorithms based on the steps to catch the prey after each episode.

4 Results and discussion

4.1 The influence of parameters on learning

Q-learning's parameter ϵ has a direct incidence in the algorithm's resulting policy, as well as in the time it takes to complete the learning process. Based on this value, and the initial value of $Q(s,a)$, the learning algorithm will select whether to sample an action following an exploiting strategy or an exploring one. There is a tradeoff between exploring and exploiting; while exploring can give better policy results, it will take more time to do it.

Figure 1 shows how changing the parameter ϵ affects the convergence speed of the algorithm.

For a stating value $Q(s,a) = 0$, values of $\epsilon = 0$, $\epsilon = 0.3$, $\epsilon = 0.5$, and $\epsilon = 0.9$ corresponds to cases in which the algorithm never takes an exploring action ($\epsilon = 0$), increasing it's probability, to a case in which approximately 72% of the times an exploring (non-greedy) move is selected ($\epsilon = 0.9$).

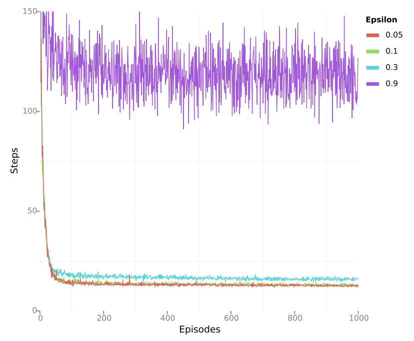


Figure 1: Steps to finish an episode while varying ϵ, with $Q(s,a) = 0$

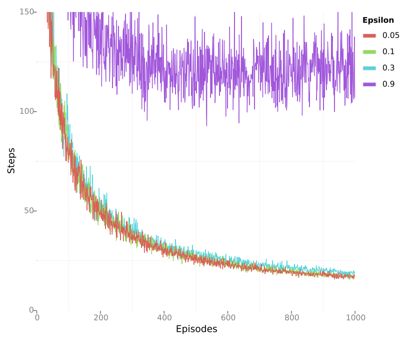


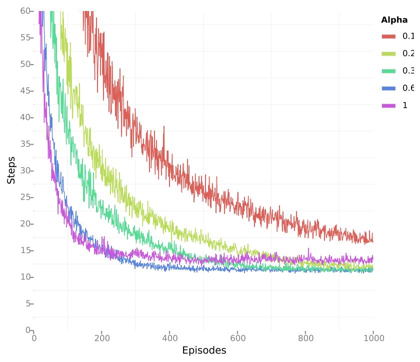
Figure 2: Steps to finish an episode while varying ϵ, with $Q(s,a) = 15$

In general, for any given initial value of $Q(s,a)$, setting $\epsilon = 0$ will make it learn in fewer steps, and it will increase as ϵ increases.

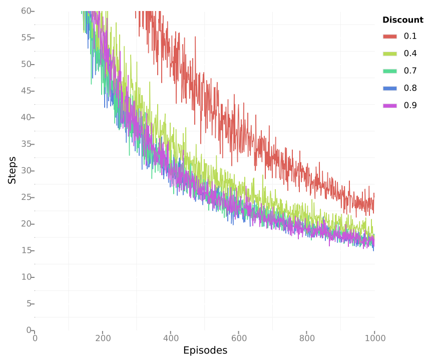
Running the algorithm for an initial value $Q(s,a) = 15$, $\epsilon = 0$, $\epsilon = 0.3$, $\epsilon = 0.5$, and $\epsilon = 0.9$ (see figure 1) forces the agent to choose an exploring action, independently of ϵ (i.e. even when $\epsilon = 0$ the agent still takes exploring actions).

This is due to the fact that we chose an optimistic value of $Q(s, a)$. During the early phase of the algorithm, whichever actions are selected, the return is less than the starting estimates causing the agent to be "disappointed" with its choice and switches to other actions. This result is that all actions are tried multiple times before the values converge, leading to exploration even if ϵ is set to an infinitesimal amount.

The larger the value of ϵ, longer will it take to end an episode during the learning phase.



(a) Learning rate (α)



(b) Discount factor (γ)

Figure 3: Variations of running Q-learning

Figure 3 shows the convergence speed of Q-learning for each episode for variations of both α and γ. Variations in α change the learning rate of Q-learning, a higher learning rate will update the Q values faster, it may become more unstable as it will give a higher weight to more recently observed return values for the state. A learning rate of 1 will only consider the most recent Q value, which clearly converges very fast, as can be seen in Figure 3a, it however does not converge to an optimal policy. Alpha values of 0.6 and lower all seems to converge to the same policy. The discount factor γ affects how next actions are weighted in the Q value of a state. Figure 3b shows how these discount factors affect convergence. Obviously a lower value will not converge as quickly as a high value. Since the predator prey game is an infinite horizon game with only a reward at the end it makes sense to have a high discount factor, such that the long term reward is maintained in the Q values.

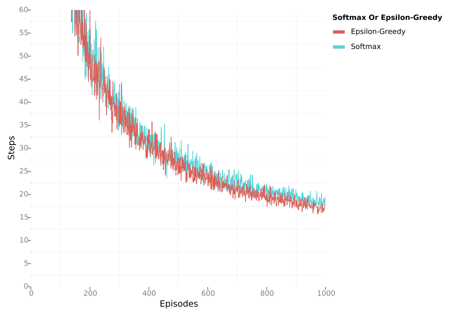


Figure 4: Softmax and Epsilon Greedy compared $\tau = \epsilon = 0.1$

In Figure 4 we compare softmax and ϵ-greedy action selection. The convergence speed does not seems to differ significantly, when looking at this graph. More on this in Figure 7.

The experiments demonstrated in Figure 5 shows the effects of varying the starting Q values on the number of steps needed. As expected with optimistic initialization during the early phase of learning the agent selects exploring actions very frequently which leads to an increased number of steps to finish an episode. Similarly to a high ϵ value this will lead to more exploration, however one key difference in this case is after a sufficient number of episodes, the impact of optimistic initialization eventually fades out. Unlike the case with large ϵ in the ϵ-greedy algorithm, in which suboptimal actions are selected frequently regardless of the amount of experience learned. These extra exploration steps in the beginning are beneficitial for finding the optimal policy in the end, which can be seen in Figure 5 for initial Q value 1, which finds a better policy than initial Q value 0.

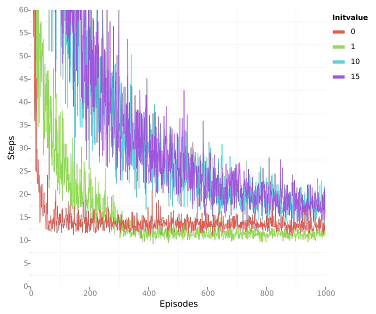


Figure 5: Steps to finish an episode while varying the initial Q values ($\epsilon = 0.1$)

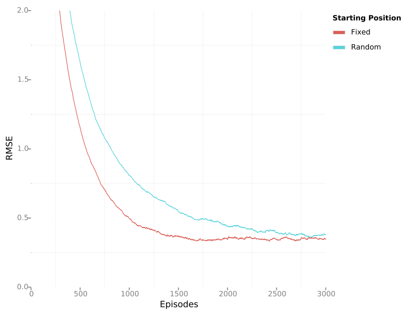


Figure 6: Starting positions for Q-learning

The convergence requirement for these algorithm is that all state-action pairs are updated at least occasionally. This means that the initial state of each episode plays a role in exploration. The two lines in figure 6 depict random initialization, meaning a random initial state is chosen for every new episode, and fixed initialization, in which the same initial state is used for all episodes. The result is averaged over five runs of Q-learning. As we can see, early exploration through random initialization plays a role in expediting the decrease in RMSE between the estimated value-function and the true value function.

4.2 Comparison between algorithms

As long as all state-action pairs continue to be explored, all four algorithms are bound to converge. One advantage of Temporal-Difference learning methods like Q-learning or Sarsa is that they learn from each transition regardless of what subsequent actions are taken. On the other hand, Monte-Carlo methods must wait until an episode has finished to obtain the return, which is needed for learning. Moreover, Monte Carlo can involve discarding exploring moves, which leads to slow learning. Figure 7 shows the RSME between the optimal value-function (found using value iteration algorithm) and the estimate of this true value-function from each algorithm.

From this figure we can see that errors go down as more experience is provided and learned. As expected, Q-Learning (ϵ-greedy and softmax) and Sarsa both learn much faster than Monte-carlo methods due to reasons explained above.

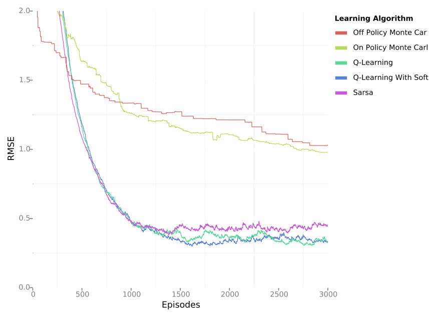


Figure 7: Root Mean Squared Error

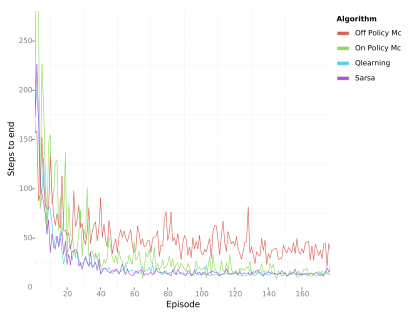


Figure 8: Steps to end episode

Figure 8 demonstrates another comparison between the four algorithms, based on how quickly the agent can capture the prey when experience is presented through the episode one by one. As in the previous case, Q-Learning and Sarsa reduce the RSME significantly faster than Monte-Carlo and as such the number of steps the agent need to make to capture the prey in these two case also reduce faster. An interesting feature is the comparison between On-Policy and Off-Policy Monte-Carlo. Off-policy Monte-Carlo learns after the last nongreedy action of each episode, meaning that if nongreedy are frequent then learning is slow, as depicted in the figure. The initial randomness in the estimation policy means that a large number of episodes is needed for convergence.

5 Conclusion

This report shows a comparison of four different learning algorithms: Q-learning, Sarsa, On-policy Monte Carlo, and Off-policy Monte Carlo. Differences between these algorithms are discussed, and their main singularities presented. On-policy

MC and Off-policy MC are both Monte Carlo learning methods, which can be characterized by needing to end an episode in order to update their policy; while Q-learning and Sarsa are Temporal difference algorithms, which can learn at each time step. On the other hand, Sarsa and On-policy MC are On-policy methods, whereas Q-learning and Off-policy MC are Off-policy algorithms. The main difference resides in the policy they use during the learning process. The former updates the current policy they use to sample and generate episodes, while the latter uses two different policies for this use (a behavioural policy, and a target policy).

Focused on the Q-learning algorithm, an analysis of its parameters is presented. Varying ϵ and the initial value of $Q(s, a)$ has an impact in the exploration-exploitation tradeoff; affecting both, the learning time and the result of the policy. The value of the learning rate α and the discount factor γ affects the time of convergence to an optimal policy, as well as its optimality.

Various scenarios and metrics are analysed. Running the algorithms for a fixed set of parameters, shows that Temporal difference algorithms learn faster, and thus, result in a better policy.