Autonomous Agents 1
Assignment 2

By Group 4: Gieske, Gornishka, Koster, Loor

November 21, 2014

Introduction

This report contains the analyzation of an implementation of single agent learning using reinforcement learning. The agent does not know the transition probabilities nor the rewards. There are two ways to solve this problem: the agent can learn the model and perform model based planning or the agent can perform model-free learning and learn a high-reward policy. In this case, model-free learning is implemented and analyzed.

There are several ways of model-free learning. Just to name a few, on-policy Monte Carlo, off-policy Monte Carle, Sarsa and Q-Learning come to mind. The focus of this analyzation lies on Q-Learning. For comparison, Sarsa was also implemented. When selecting actions, the performance of ϵ-greedy and softmax action selection are also analyzed and compared to one another.

Theory

Temporal difference learning methods

Before explaining what Q-Learning is and does, it is important to take a look at temporal difference methods. Namely, because Q-Learning is a temporal difference method. Temporal difference learning, in this case, combines Monte Carlo methods and Dynamic Programming methods. This leads to very powerful and valuable techniques which can be used in model-free planning. Dynamic Programming exploits the Bellman-equation, but require the model. Monte Carlo methods do not require the model, but do not exploit the Bellman-equation either. Temporal difference methods do not require the model, but do exploit the Bellman-equation. This has lead to temporal difference methods becoming core algorithms in reinforcement learning.

Q-Learning

This is an off-policy temporal difference learning method.
Watkins' Q-Learning algorithm according to [1]:

Initialize Q(s,a) arbitrarily and e(s, a) = 0 for all s,a
Repeat (for each episode):
    Initialize s,a
    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)
        $a^* \leftarrow \arg\max_b$ Q(s',b) (if a' ties for the max, then $a^* \leftarrow$ a')
        $\delta \leftarrow$ r + γ Q(s',a^*) - Q(s,a)
        e(s,a) $\leftarrow$ e(s,a) + 1
        For all s,a:
            Q(s,a) $\leftarrow$ Q(s,a) + $\alpha\delta$ e(s,a)
            If a' = a^*
                e(s,a) $\leftarrow \gamma\lambda$ e(s,a)
            else
                e(s,a) $\leftarrow$ 0
        s $\leftarrow$ s'; a $\leftarrow$ a'
    until s is terminal

Sarsa

This is an on-policy temporal difference learning method.
From [1]:

Initialize Q(s,a) arbitrarily and e(s, a) = 0 for all s,a
Repeat (for each episode):
    Initialize s,a
    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)
        $\delta \leftarrow$ r + γ Q(s',a') - Q(s,a)
        e(s,a) $\leftarrow$ e(s,a) + 1
        For all s,a:
            Q(s,a) $\leftarrow$ Q(s,a) + $\alpha\delta$ e(s,a)
            e(s,a) $\leftarrow \gamma\lambda$ e(s,a)
        s $\leftarrow$ s'; a $\leftarrow$ a'
    until s is terminal

Action selection methods

In order to select which action to choose according to a given policy, a tradeoff between exploration and exploitation must take place. This tradeoff is important when performing reinforcement learning as the rewards must be maximized, but exploration may lead to finding higher rewards. There are several action selection methods which can be used to select actions. The two techniques analyzed in this report are ϵ-greedy and softmax action selection.

An ϵ-greedy policy mainly exploits the known states to maximize the immediate reward. At times, this policy will explore new states. The times when this policy explores new states is probabilistically determined. This oftentimes leads to desired behaviour. However, it is possible for the agent to stand beside the goal state when the ϵ-greedy policy turns to explore a new state. This can lead to undesirable behaviour, especially in tasks where the worst actions have catastrophic consequences (e.g. falling off a cliff).

Softmax action selection offers a solution to the problem presented by ϵ-greedy policies. The greedy action is still assigned the highest probability, but all other probabilities are ranked and weighted according to their values [1]. There are several ways of implementing the sofftmax action selection method. This implementation uses the most common implementation, which is the Boltzmann-distribution:

$$\frac{exp[Q_t(a)/\tau]}{\sum_{b=1} exp[Q_t(b)/\tau]}$$

Where τ is a positive parameter known as the temperature. Lower temperatures lead to a greater difference in selection probability for actions that differ in their value estimates [1].

It is unclear whether ϵ-greedy action selection is better than softmax action selection. The performance of either method may depend on the task at hand and human factors. Experimenting with both methods will lead to more insight in both algorithms.

Implementation

The implementation consists of the following files:

Agents_new
: This file contains implementions of the Agent class, the Prey class and the Predator class. Both the predator and the prey inherit functions of the Agent class. The Agent class contains functions any agent needs, such as a set of actions, a policy and other functions. As the predator is the agent is the agent this implementation focuses on, the predator class contains more functions than the predator class.

Helpers
: This file contains many helper functions. These functions aid in computation and decision making, but cannot (and need not) be part of a specific class.

Other_objects
: This file contains the Policy and Environment classes. The environment of the game as well as the rules are implemented in the Environment class. The Policy class contains the implementation of Q-Learning, Sarsa, ϵ-greedy, softmax action selection and more functions that help in determining and optimizing a policy as well as choosing an action of this policy.

Newstate
: This file contains the Game class as well as a demonstration function. The Game class instantiates the game, initialized the predator and the prey and assigns policies to these. The game is run N times and the result is printed. The demonstration function also performs Q-Learning, Sarsa and Monte Carlo. It also uses ϵ-greedy and softmax action selection. The results are printed in the command line and graphs are used for analyzation.

Analysis

Q-Learning

Sarsa

ϵ-greedy vs. softmax

Conclusion

Files attached

- newstate.py
- agents_new.py
- other_objects.py
- helpers.py ...

Sources

1 Barto and Sutton (http://webdocs.cs.ualberta.ca/ sutton/book/the-book.html) ...