IT3708: Project 5 – Reinforcement Learning Using Q-Learning

Author: Petr Zvonicek

a) Implementation

Architecture

The project consists of three fundamental components – Flatland, QLearning and GUI. Flatland is an abstraction of the world, stores the grid and handles movement. QLearning gets an instance of Flatland to be trained.

Main method in QLearning is *learn*, which is basically a realization of the pseudocode listed in the project assignment. The whole learning procedure is iterated several times depending on *iter_num* parameter.

Other key methods are:

- *move*, which calls *flatland* instance that makes movement and returns the state of the cell (empty, food, poison)
- *compute_reward*, which returns reward for given cell state (5 for food, -8 for poison and 5 for final state)
- *best_action*, which returns the best action (left, right, up, down) for given state according to the *q value*.
- *select_action*, which selects either best action or random action
- *update_q*, which updates the *q value*
- *update_eligibility*, which is a realization of the TD(λ) backup scheme

Q[s,a] pairs are stored as a standard Python dictionary. The key is a (s,a) tuple and the value is the weight, by default initialized to zero. The state (s) is a tuple of position and a *Set* of eaten food. The action (a) is an integer value describing direction. The dictionary structure makes it easy to query for $max_a Q[s,a]$. If we want to fetch the maximum value, we will go through the four values – *Q[s, Left]*, *Q[s, Top]*, *Q[s, Right]*, *Q[s, Down]* and select the maximum of these.

Parameters

The used parameters are the following:

- α (learning rate) = 0.2, the computed problem is stochastic, thus the low learning rate is needed
- γ (discount rate) = 0.9, we prefer log-term high rewards to current rewards, thus the discount rate is high
- λ (trace decay) = 0.1, long traces are not desired when food is densily spaced, my experiments showed it has bad impact on performance
- p (probability of random action) = 0.2, the value was chosen by experiment and provided best results

b) Action selection

On each run there is selected either greedy (the best from all possible, exploitation) or random (exploration) action. Although I experimented with dynamic probability in Simulated Annealing

manner (linear and exponential scaling), the best results were always provided by fixed probability of random run 0.2, so I stick with this.

My idea behind this is that we need some exploration in all computation phases. In early phase the exploration is dominant (despite having just 0.2 probability, the weights are 0 for all directions so there is randomly chosen one), but also in late phase some exploration is needed so as not to stay in local maximum.

c) Backup scheme

I've experimented with the both TD(x) and TD(λ) schemes and TD(λ) seems to provide more reliable results. Therefore I used this one. However, even TD(λ) usually provided even worse results than not using any backup scheme at all. I spend a lot of time on tweaking the trace decay factor – λ and eventually the value $\lambda = 1$ did not negativelly influenced the results and perhaps slightly improved the performance. Apart from the standard TD(λ) formulas for Q-learning I'm also resetting all e values to 0 when exploratory (random, non-greedy) step made. Also it was beneficial to adjust the values with backup scheme only on positive reward.

It is, however, still difficult to assess the impact of the backup scheme and it could easily have no impact at all. Due to the randomness, it is not easy to make an objective assessment of that.

I was analyzing the low impact of the backup scheme and I came up with several possible reasons for that. First possibility is a bug in my implementation. I was, however, debugging and tracing the code and the weights were updated as expected. Another reason for that could be the fact that when the food is densely spaced, long traces could improve the path to one food, but that could also lead to missing the other food nearby. I therefore think that long traces (big x or λ) are much more suitable for sparse food placement than for dense placement.