Homework 5: Reinforcement learning

Abstract

In this homework, a simple reinforcement learning model is implemented. Combinations of the learning parameters (α and ε) and the possible policies (SARSA and Q-learning) are tried, and the results compared.

Theory and Code development

The toy model of reinforcement learning is the following: an agent can move on a grid of size $(X \times Y)$ in order to reach a goal cell, while avoiding possible contacts with the borders or with other obstacles.

The framework is the usual: the action sets A_t include all possible actions the agent can take at time t (it is a subset of the 'total action set' $\mathcal{A}$), the state set $\mathcal{S}$ includes all possible states which the agent can occupy at any time, the reward R_t denotes the payoff for the agent at time t. An agent tries to maximize the *long-term expected reward*

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{k+t+1}, \quad \gamma \in [0, 1) \tag{1}$$

Given a state $s \in \mathcal{S} \equiv \{[X] \times [Y]\}$, one can compute the *expected reward at state* s,

$$v(s) = \mathbb{E}[G_t | S_t = s] = \mathbb{E}[R_{t+1} + \gamma v(s') | S_t = s] \quad \text{[Bellman equation]} \tag{2}$$

where the expectation is computed on the possible *policy* Π that the agent uses. Such policy is the probability with which the agent chooses action a, given that it is in state s:

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s] \tag{3}$$

The Bellman equation then becomes

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] \quad \forall s \in \mathcal{S} \tag{4}$$

Optimizing this function is exponentially difficult; furthermore, the values of $R(a|s)$ and $P(s'|a, s)$ should be known in advance.

An estimate of $v_\pi(s)$, called Q-value, can be computed by first picking a random value and then iteratively updating as follows:

$$Q(a_t, s_t) \leftarrow Q(a_t, s_t) + \alpha(r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{5}$$

This quantity is indexed by both the state s and the action a_t. As such, in the problem at hand it is naturally written as a matrix with shape $(|X \times Y| \times |\mathcal{A}|)$, where $\mathcal{A} = \{Up, Down, Left, Right, Still\}$.

The α parameter has the role of a learning rate; higher values result in larger changes in the matrix Q. An important role is played by the estimate of the best future action, $\max_a Q(s_{t+1}, a)$. This is a so-called *greedy update policy*, and it must not be confused with

the *behavioral policy*, which is the rule by which future actions are chosen. Among these, the most common are the softmax (actions are drawn from the softmaxed Q-values) and the ε-greedy (wp ε select a random action, wp $1-\varepsilon$ choose the greedy, i.e. short-term best, action). The parameters α and ε have to be tuned so that, during the learning, they 'optimize' the tradeoff between exploration of the state-action space and exploitation of the results found up until that point, as to not miss on potential 'good' solutions, while also not wasting time exploring an exponentially large space.

The last possible difference lies in the choice of the update policy. The two main categories are $SARSA$ and Q-learning; while the first updates its beliefs by drawing a move using the behavioral policy (and thus drawing the move from a probability distribution), the latter assumes that the move with the best Q-value will be played. This makes risky play more frequent, as the algorithm does not have to worry about possible future bad moves.

In order to introduce further customization, walls and 'sand patches' were introduced in the environment, together with a history log of the past moves (for future plotting). In the `Environment` class, the following functions were added:

- `add_wall`: given the starting end ending point (which must at least share one coordinate, otherwise the function will not create a wall and return an error), the cells in the middle will become walls. These cells behave like the boundaries: they can not be crossed and give a reward of -1;
- `add_sand`: given a single point, it will become a sand patch. These cells can be crossed, but give a reward of -0.5;
- `get_inits`: returns the point in the grid which are not a wall, sand, or the goal. The agent will start moving from these cells;
- `set_start`: given a point, this function deletes the previous move history and sets the initial point in the given cell. Makes it possible to reuse a same environment with specific starting point;
- `refresh`: removes all obstacles and return the environment to its standard form;
- `plot`: creates a plot of the environment. If the optional parameter 'traj' is set to `True`, the trajectory that the agent took is superimposed to the environment.

In the plots below, walls are represented by black cells, while sand is represented by the sand-coloured cells. Sand patches and walls will be coupled with goal positioning (in indigo in the plots below) in order to probe the various algorithms tendencies. The starting points are plotted in green.

Results

The chosen profile for ε is decreasing in the number of iterations. This choice is dictated by the intuition that the exploration/exploitation tradeoff is best achieved by allowing for a large freedom at the beginning, followed by stability in the latter part of

the learning, as to allow for optimization of the 'chosen strategy'. As a result, with t being the number of the episode, the used profiles are

$$\varepsilon(t) = t^{-1.2}, \quad \alpha(t) = 0.25 \tag{6}$$

Indeed, choosing different profiles with a dependence on the inverse of the number of episodes seem to yield the same overall results. The choice of a power law is due to the relative slowness with which ε decreases, allowing for more exploration. As for α, different choices did not seem to influence the result too much, so a simple constant was chosen.

These profiles were tested with different layouts and different policies. To test the differences, the plotted trajectories started from the same point for all plots. This point was manually chosen because it allowed better testing of the behaviors. Results are collected in the plots in the dedicated section.

The learners display the expected behavior: depending on the environment, with increasing episodes the agent finds better routes toward the goal. In the case of the 'pipe' and 'sand pool' environments (Figs 1, 2,3,4 and 5, 6, 7, 8), this is especially clear: the agent tries different routes, switching from one to another during the learning, so that the same 'idea' is used multiple times. In the end, one choice 'wins' and is used.

Different paradigms also show differences, with softmaxed versions exhibiting a higher tendency towards exploration. Interestingly enough, SARSA seems to better explore the space (indeed, it does not act assuming that the next best move is taken).

Comments

The results are in agreement with the expected outcome. The agent finds the best routes after some exploration-heavy episodes, and refines its choice from there (despite the learning being cut short; one expects that, for higher amounts of episodes, the routes would more or less converge for all paradigms).

The assignment was entertaining and satisfying, especially because of the good accordance between the routes found by the agent and the expected ones.

Plots

Pipe environment

Tests the ability of choosing between a shorter but more costly path and a longer but cheaper one.

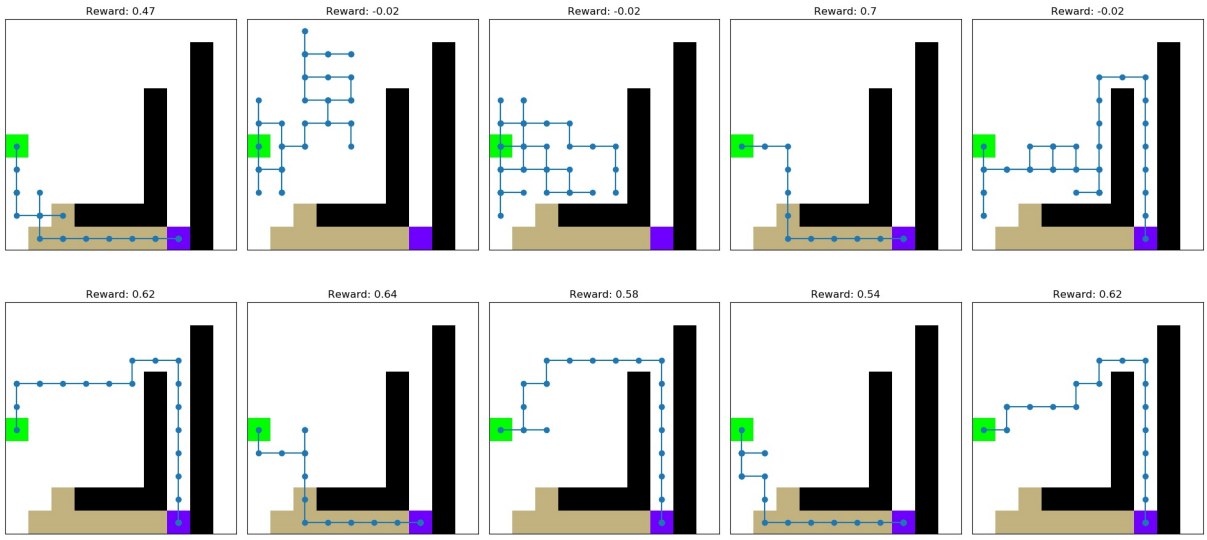

Figure 1: Q-learning, not softmaxed. Plotted every 100 episodes.

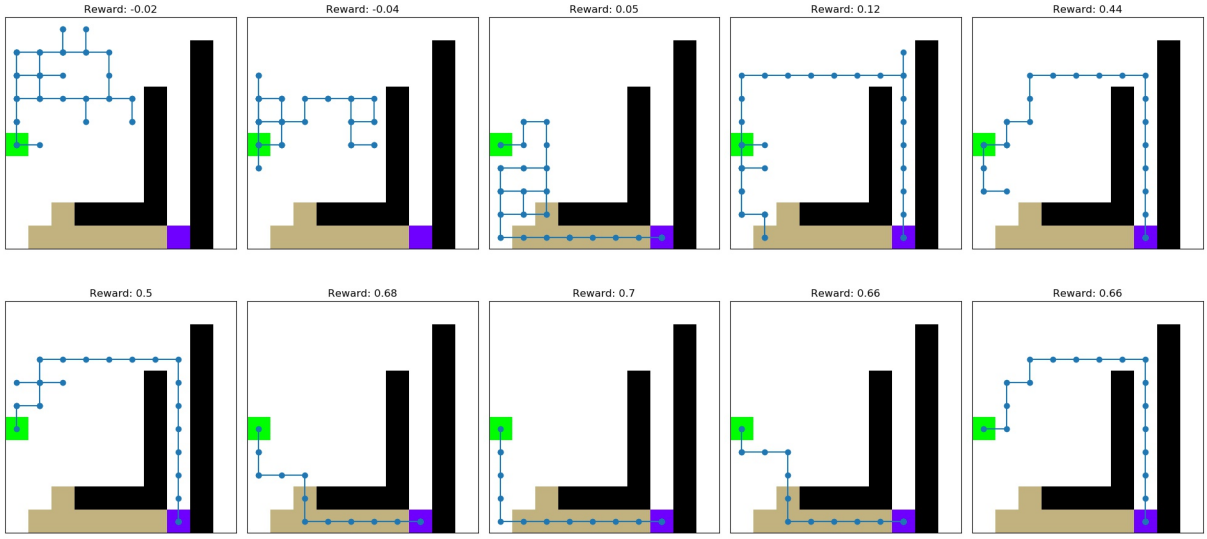

Figure 2: $SARSA$, not softmaxed. Plotted every 100 episodes.

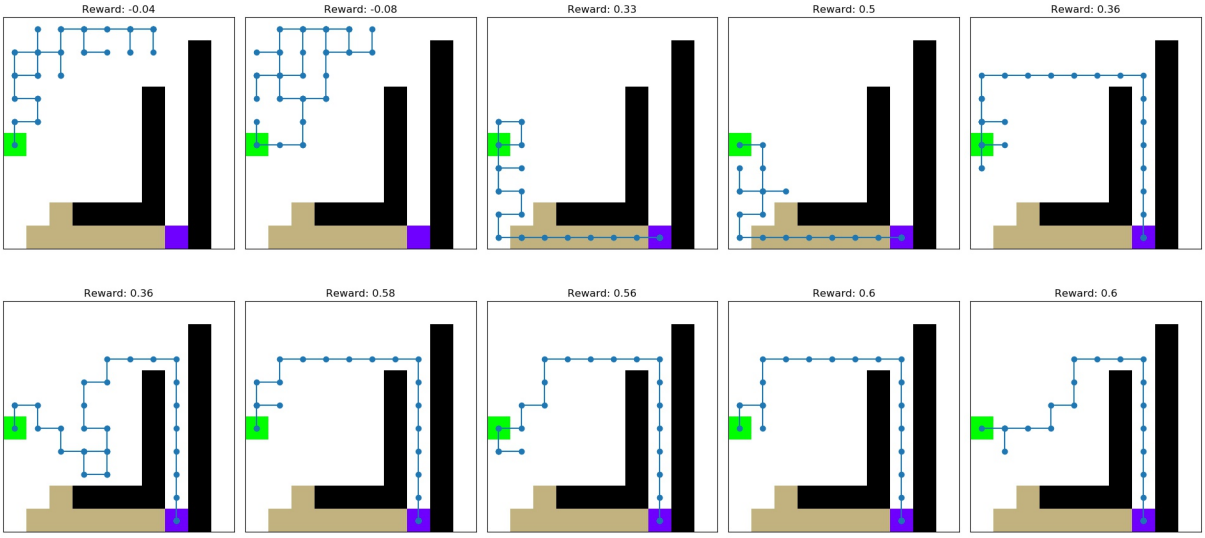

Figure 3: Q-learning, softmaxed. Plotted every 100 episodes.

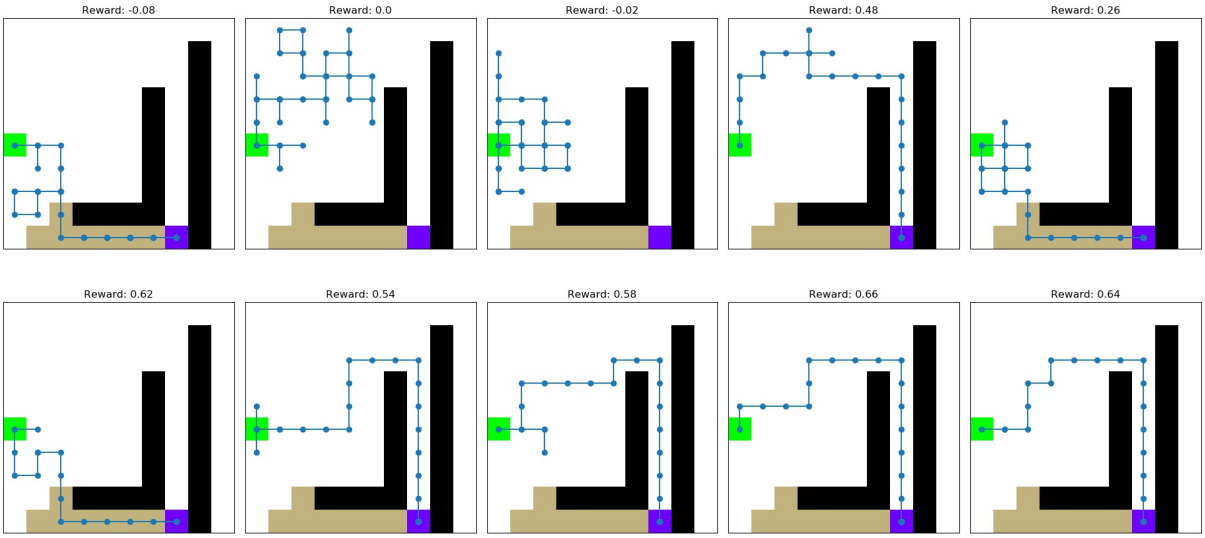

Figure 4: $SARSA$, softmaxed. Plotted every 100 episodes.

Sandpool environment

Tests the ability of circumnavigating/cutting through an obstacle.

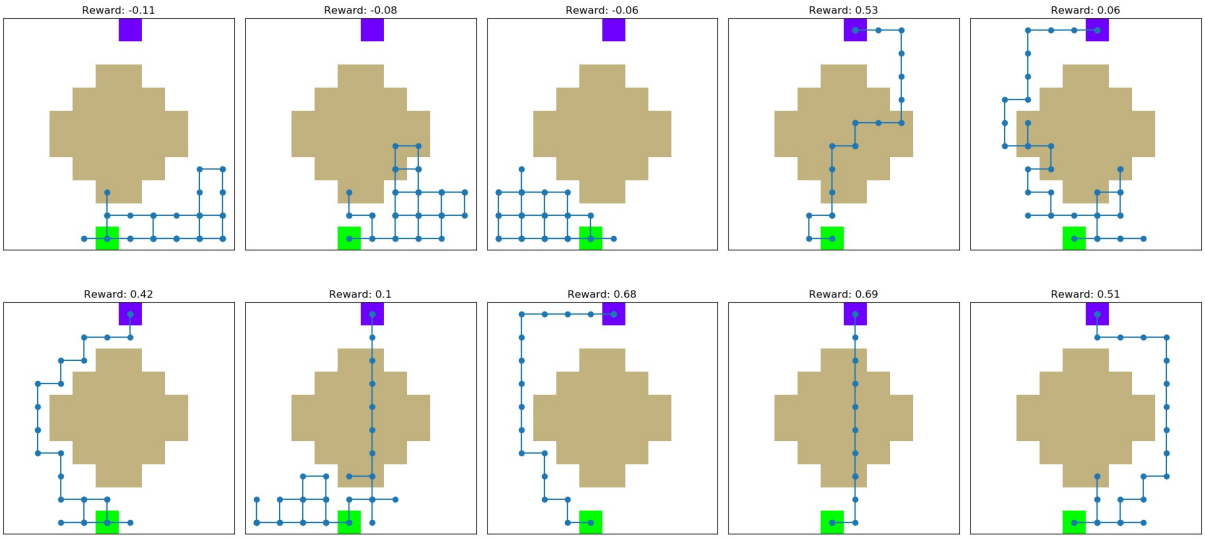

Figure 5: Q-learning, not softmaxed. Plotted every 100 episodes.

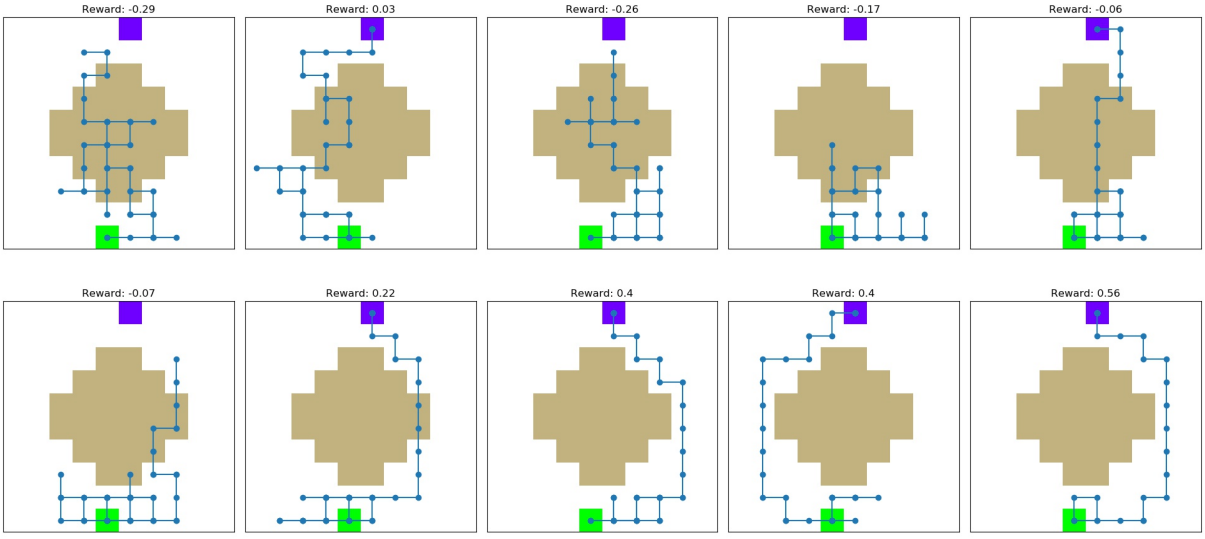

Figure 6: $SARSA$, not softmaxed. Plotted every 100 episodes.

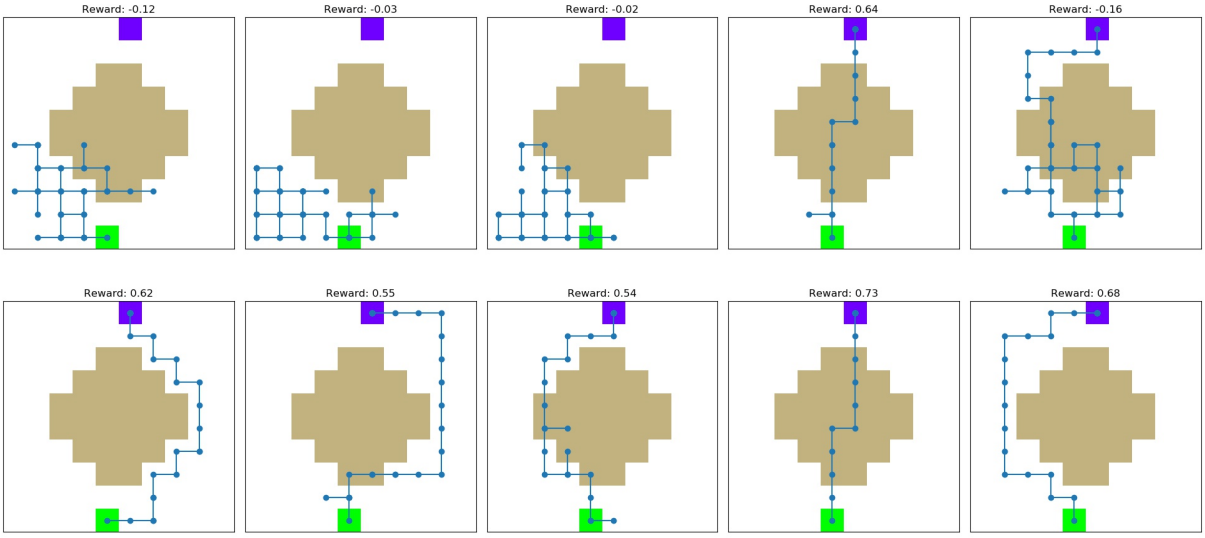

Figure 7: Q-learning, softmaxed. Plotted every 100 episodes.

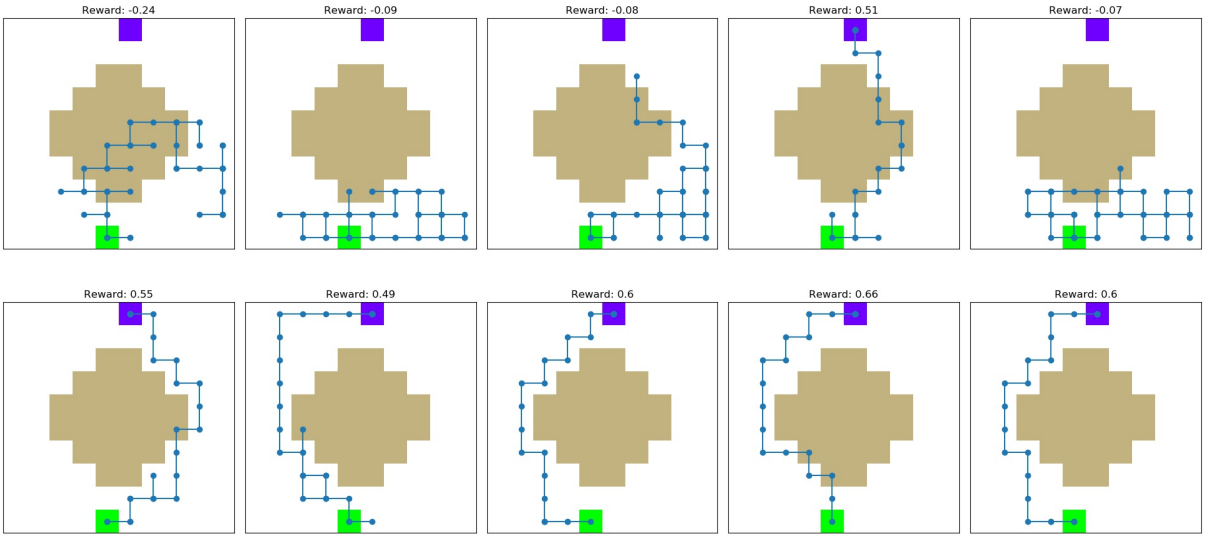

Figure 8: $SARSA$, softmaxed. Plotted every 100 episodes.

Slalom environment

Tests the ability of exploring the environment in order to finds shortcuts.

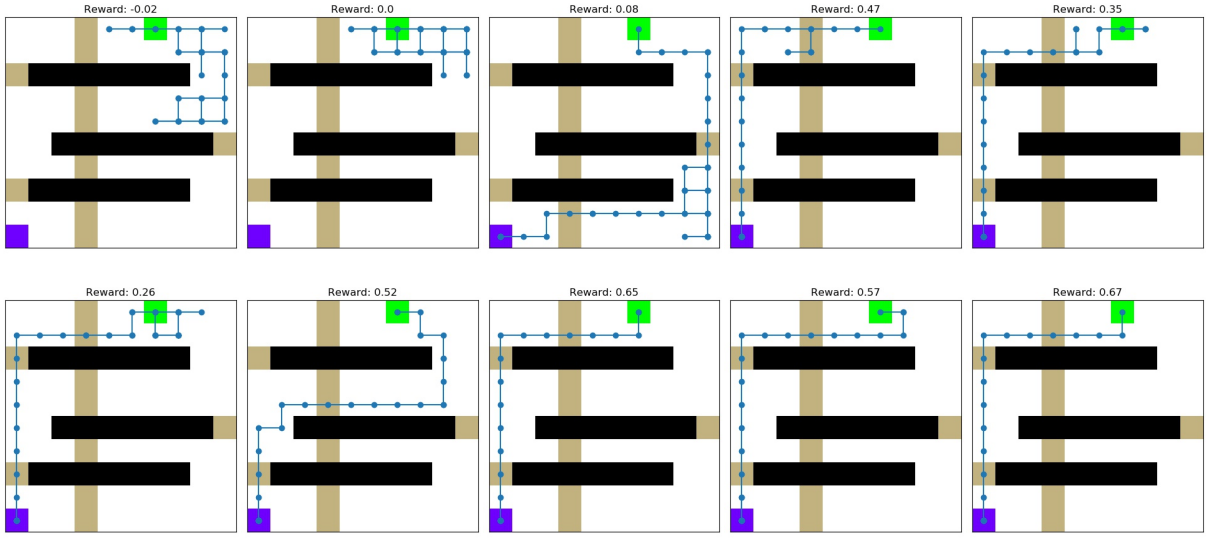

Figure 9: Q-learning, not softmaxed. Plotted every 100 episodes.

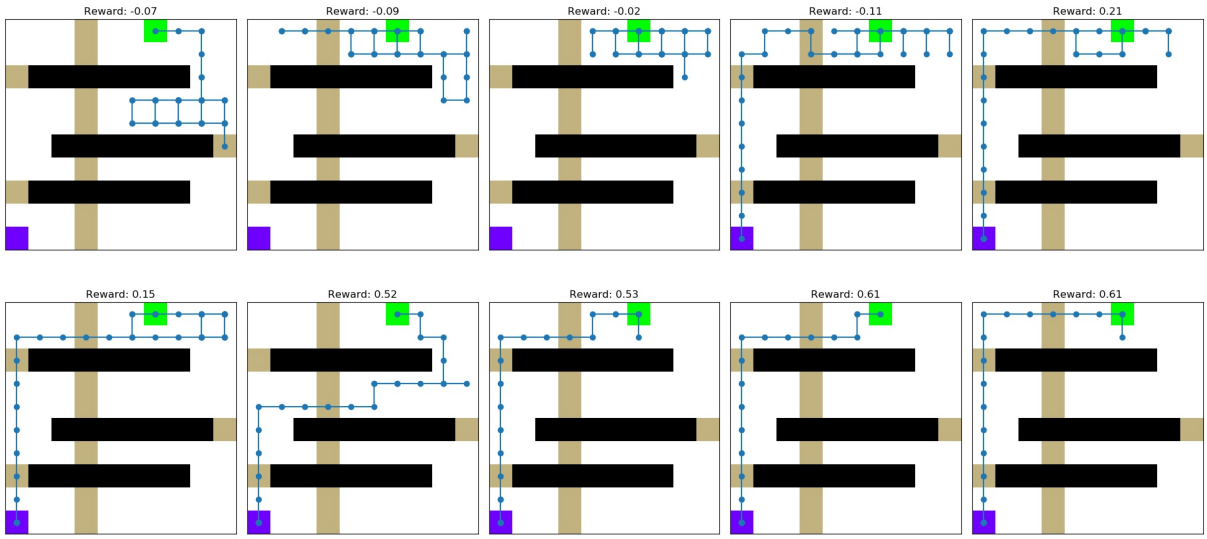

Figure 10: $SARSA$, not softmaxed. Plotted every 100 episodes.

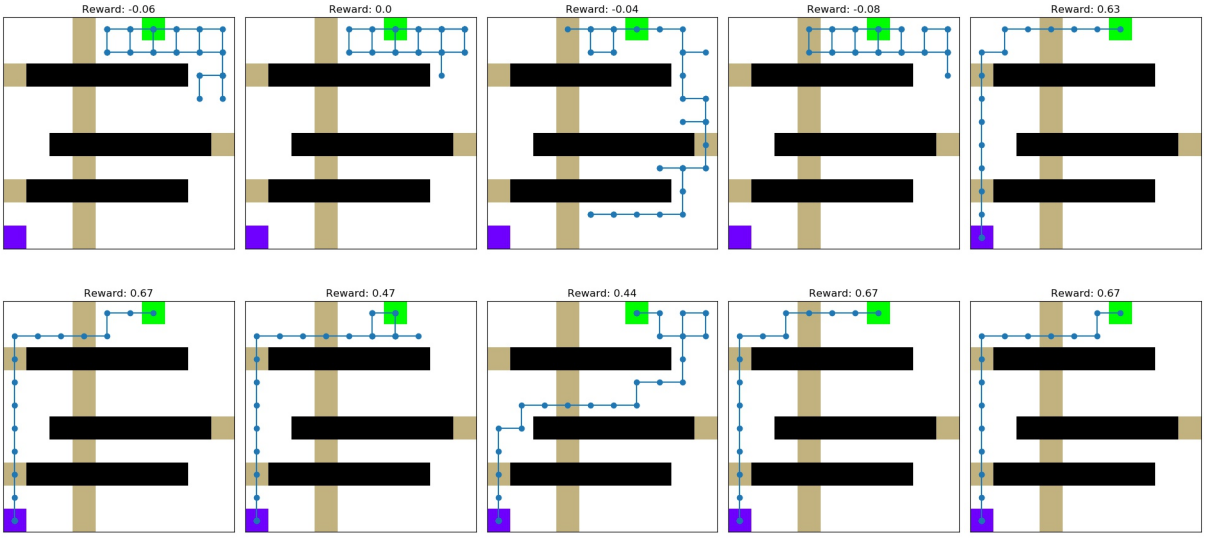

Figure 11: Q-learning, softmaxed. Plotted every 100 episodes.

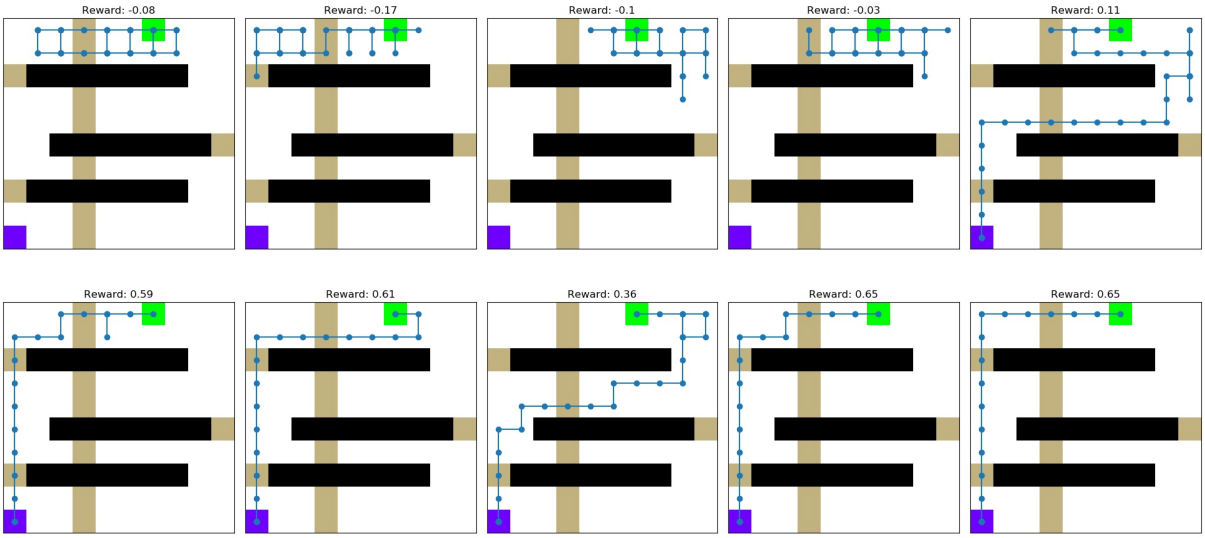

Figure 12: $SARSA$, softmaxed. Plotted every 100 episodes.