RL-Competition: Sarsa(λ) agent

a submission to the competition in the course “Decision making under uncertainty“

John Karlsson
kajohn@student.chalmers.se

Oscar Carlsson
coscar@student.chalmers.se

Oskar Lindgren
oskar.lindgren@chalmers.se

Abstract

We present the results of an Sarsa(λ) based agent with a KL-UCB policy and a maximiser. The agent acts in several different discrete state space environments, two of which we authored and explain here; Tic-tac-toe and Connect Four. Enabling the maximiser proved better in environments with large state spaces. Using KL-UCB did not yield better results than a greedy Sarsa(λ) with $\lambda = 0.2, \alpha = 1$.

1 Introduction

This report details our submission to the Reinforcement Learning competition held in the course “Decision making under uncertainty“ [2]. The competition was to implement a agent that would communicate with RL-Glue [5] and act in environments with discrete state spaces. We implemented an agent that is built upon the ideas of Sarsa(λ) [4] in combination with KL-UCB [3] and we present the results of varying parameters and different approaches to exploration vs exploitation and on how to break ties.

2 Environments

We have implemented two environments representing the board games Tic-tac-toe and Connect Four. The Tic-tac-toe environment is a fully observable Markov Decision Process (MDP) while Connect Four is a Partially Observable MDP (POMDP), both are played against an environment that always plays any available legal move. Illegal moves played by the agent are punished by a negative reward, and the agent gets to try again from the same board state.

	Tic-tac-toe	Connect Four
State space	19683	2188
Action space	9	7
Agent wins	10	1
Illegal move	-10	-1
AI wins	-1	0

Tic-tac-toe The board state is considered as a ternary representation of an integer that defines the state. Each digit describes whether a site at the board is empty (0), occupied by player (1) or by the AI (2). The 9 different actions correspond to playing at one of the 3x3 board game sites. The agent always gets the first move, and wins by getting 3 in a row or when the board is filled.

Connect Four Connect Four is played on a vertical board game, game pieces are dropped down from the top and fills up the board from the bottom. In this environment, the agent are only allowed to observe the board state from the top, playing a piece in one column will hide previous moves in that column from the agent. Similarly to Tic-tac-toe, the agent receives an integer that represents a ternary digit sequence describing the topmost board piece in each out of seven columns. The starting player is randomised, and the game can end in a draw.

3 Agent

Sarsa(λ) The agent is based on the Sarsa(λ) algorithm, which learns a set of action values $Q_t(s,a)$ by the use of eligibility traces [4]. The eligibility trace $Z_t(s,a)$ is initialized to zero for every state-action pair, and for every transition all traces are decayed by a factor $\gamma\lambda$. In addition, the trace of the current state-action pair is incremented by 1. The Q-table is then updated according to:

$$\forall s,a : Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t Z_t(s,a), \qquad (1)$$

where δ_t is the temporal difference defined by $\delta_t = R_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$. The Q-values are initialized to R_{max} for all small state space environments ($|S| < 50$) and to $(R_{min} + R_{max})/2$ otherwise, since an optimistic initialization encourages exploration among the set of actions.

KL-UCB An Upper-Confidence Bound (UCB) policy maps a state s to the action maximizing an upper bound. For Kullback-Leibler UCB, the bound for a given action a is defined by:

$$\max_{q \in]0,1[} \left\{ n_s(a) D_{\mathrm{KL}}\left(\overline{R_s(a)} \middle\| q\right) \le \log(\tau) + c \log\log(\tau) \right\} \tag{2}$$

where $D_{\mathrm{KL}}(P\|Q) = p \log \frac{p}{q} + (1-p) \log \frac{1-p}{1-q}$ is the Kullback-Leibler divergence measure of p and q. [3]

To combine the UCB methods with Sarsa(λ) we let the average reward $\overline{R_s(a)} = Q(s,a)$, and $n_s(a) = \sum_{i=1}^{t} Z_i(s,a)$ be a measure of how often the state-action pair (s,a) has been updated. Further, we let $\tau = \sum_a n_s(a)$ denote a local time for the bandit problem in context s.

Complementing the core mechanism In some environments, the state-action space could be very large compared to the experiment length and a sufficient sampling of the gain of each state-action pair might become too costly to perform. Then it becomes increasingly important how new or poorly sampled situations are approached.

The key observation is that Sarsa(λ) is capable of quickly distributing the value function over correlated states. However, it relies on that the policy implements efficient heuristics for exploration. We tested a number of different approaches to complement the standard Sarsa(λ) algorithm, both when it comes to exploration vs exploitation, and how to break ties.

3.1 Maximisers

In Sarsa(λ), whenever we reach a new state S, there is no clear differentiation between different actions $a \in A$; statistically, all are equally bad (if the outlook is pessimistic) or good (if we have an optimistic outlook), but no matter how we choose to break the tie in the $Q(S,A)$ matrix there will be some environments where we will benefit, and others where we will lose.

The easiest approach is to just go with the first or last action a in *arg max*$_{a \in A} Q(S,a)$. Just from the example environments, there are cases where this is a good idea, and cases where this is a bad idea. In Tic-tac-toe reverting to a specific order of tries, every time you end up in a new state, is bad in the sense that you will play more illegal moves. But it can also be good because due to how actions are enumerated, the first three of the nine actions could combine into a three in a row. This assuming that the AI doesn't play in that row before. Similarly, in the connect four environment, playing in the same column over and over again results in a win quite often, but if that column is filled, it will similarly cause a lot of illegal moves.

Another method we tried was to assume that in many environments, a state is reminiscent of recently visited states. Even if we lack any information about which action would be beneficial in a specific state, we might have better sampling of the last states we visited, or the states before that. We define a value of an untried action as

$$v(a) = \sum_{i=1}^{t} |A|^{-i} n_{s_i}(a) \sqrt{n_{s_i}(a)} \Big/ \sum_{i=1}^{t} \sqrt{n_{s_i}(a)} \tag{3}$$

It assigns a higher value to actions that were deemed good in recent history, giving more weight to well founded predictions.

4 Results

How we chose the maximiser We tried both to always choose the highest valued action, and selecting an action with probability proportional to (rescaled) value. This method of breaking ties in the Q matrix proved useful in board game environments where the number of states were large. For the smaller environments, it didn't add any benefit and actually caused the agent to perform worse in some examples. It replaced a simple "first best" choice, which turned out to actually be a better decision in some cases, hence we did not chose to use the maximiser in these environments.The softer version of choosing proportional to gain were less successful in all cases.

In the end we chose to only use the maximiser in environments with large state space; e.g. tic-tac-toe and connect four. The maximiser is triggered more often in environments with large state spaces since ties will occur more often. Fig 1 shows the slight increase in cumulative reward with maximiser enabled compared to runs without maximiser.

UCB policies We tested two versions of UCB policies in an attempt to shore up Sarsa(λ)'s unsuitability for bandit problems. The idea was to consider all states as independent bandit problems, where the UCB policies would decide the balance between exploration vs exploitation. The simpler UCB1 does not consider any upper bound on the reward and will never be satisfied and this causes it to perform exploration even when faced with certain victory options. The KL-UCB considers the case where the reward is bounded. Both however,

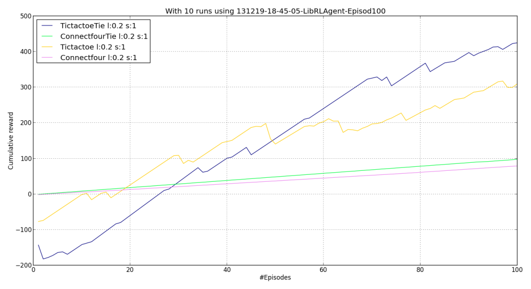

Figure 1: Comparing the use of maximiser on tic-tac-toe and connect four in a experiment with 100 episodes and reward averaged over 10 runs.

assumes long time horizons, where exploration occurs gradually. When faced with the short experiments in the competition, where the optimal approach should be to explore first, and exploit later, they fared worse than simply going for the best option. We believe the meagre success with exploration is also due to that many of the environments have many large penalties and few rewards. Exploring anything but the first found reward is not cost efficient. The fact that all states are considered a new case where exploration should be performed only amplifies the cost of using UCB.

Performance The parameter space for our agent is very large, λ decides how much we diffuse the reward, step size α determines how quick the agent overwrites old information, how we initialise the Q matrix determines if the agent is optimistic or pessimistic when it comes to the degree of exploration. With the UCB policies, we had the option to adjust the tendency to explore with a scalar, we have different maximisers that attempt to exploit structure in the problems. Naturally we didn't have time to sweep all parameters for all environments, but we varied different parameters, keeping most constant and tried to find good parameter values for different types of problems. At initialisation, the agent uses known information of a problem to set the agent parameters, e.g. enables the maximiser for large state spaces, under the assumption that the test environments are good representations of the competition environments.

The performance of our agent is visualized in Figure 3. It shows how the agents average performance over 65 runs in a 100 episode experiment against all seven competition environments. Maximiser is off, no UCB and Q is initialised to the maximum reward, i.e. optimistic explorative. Tic-tac-toe benefit greatly from $\lambda = 0$, mines perform slightly better with λ around 0.2. After the competition we noticed that the chain environment performed noticeably better for high λ. The other environments do

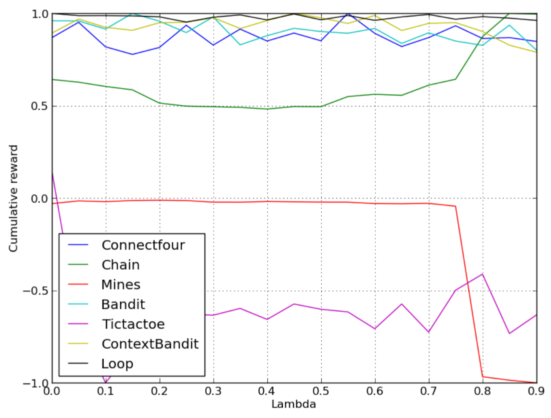

Figure 2: How the cumulative reward changes with the different values of λ.

not depend strongly enough on λ to warrant any specific value. Since we lacked sufficient chains data, we decided on $\lambda = 0.2$ for unknown environments.

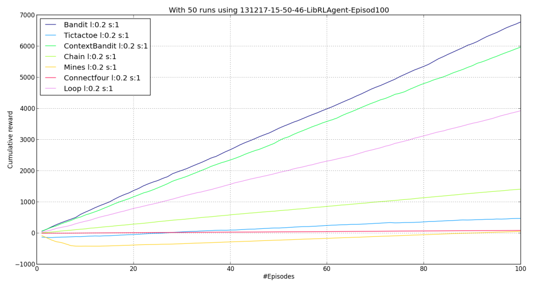

Figure 3: Agents cumulative reward averaged over 50 runs in a experiment with 100 episodes

The yellow curve represents the performance on the mines environment and at around 10 episodes it is clear that the agent has learned the environment. The performance on Connect four, represented as the red curve in Fig 3, has a much lower maximum reward than the other environments, which explains why it might look like the agent performs worse in this environment, but the agent does indeed learn this environment too.

Fig 4 shows how our agent gradually finds a shorter (but not the shortest) path to the goal in the mines environment. Red coloured paths are early in the experiment, green are later. In this case, from the 18th episode and onwards, the agent takes the same path every time.

We also noticed that Sarsa(λ) in its original formulation will decide on the next action before it updates the Q matrix. In the specific case where penalties are incurred for illegal moves, and the agents are forced to try

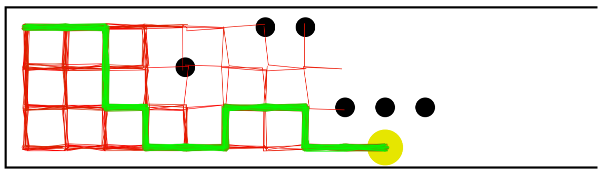

Figure 4: Example of how the agent learns a short path from start to goal (yellow). The agent traverse the world 100 times, early attempts are coloured red, later green. Hitting a mine (black) costs a penalty. Some noise is added to the paths to make it easier to distinguish them.

again, this causes the agent to perform every illegal move twice. We decided to stick with the original, suboptimal version, since checking policy twice at each time step is inelegant and we wanted to avoid overtraining.

All code, plots and anything other related to this project is publicly available via [1].

References

[1] Carlsson, O., Karlsson, J., and Lindgren, O. RL-competition. https://github.com/Oscarlsson/RL-competition, 2013.

[2] Dimitrakakis, C. Decision making under uncertainty, 2013.

[3] Garivier, A., and Cappé, O. The KL-UCB algorithm for bounded stochastic bandits and beyond. *arXiv preprint arXiv:1102.2490* (2011).

[4] Sutton, R. S., and Barto, A. G. *Introduction to Reinforcement Learning*, 1st ed. MIT Press, Cambridge, MA, USA, 1998.

[5] Tanner, B., and White, A. RL-Glue : Language-independent software for reinforcement-learning experiments. *Journal of Machine Learning Research 10* (September 2009), 2133–2136.