CRONE 2018
Conference Proceedings

Batch Reinforcement Learning on a RoboCup Small Size League keepaway strategy learning problem

Franco Ollino[1], **Miguel A. Solis**[2,3*], **Héctor Allende**[1]

***For correspondence:** miguel.solis@innovacionyrobotica.com

[1]Universidad Técnica Federico Santa María; [2]Universidad Católica del Norte; [3]Centro de Innovación y Robótica

Abstract Robotic soccer provides an adversarial scenario where collaborative agents have to execute actions by following a hand-coded or a learned strategy, which in the case of the Small Size League, is given by a centralized decision maker. This work takes advantage of this centralized approach for modelling the keepaway strategy learning problem which is inherently multi-agent, as a single-agent problem, where now each robot forms part of the state of the model. One of the classical reinforcement learning methods is compared with its batch version in terms of amount of time for learning and concluding about updates efficiency based on experiences reusability.

Introduction

When we talk about Batch Reinforcement Learning (BRL), we refer to one of the current line of research in the field of Reinforcement Learning (RL), also concerned about solving sequential decision problems modelled by a Markovian Decision Process (MDP). Given the nature of these problems, as the intuition may suggest, the scope of this type of learning has extended to areas like Robotics applications (***Kober et al., 2013***).

As in the classical approach, with online algorithms, we still focus on teaching an agent how to behave under certain conditions based on punishments or rewards (reinforcement signals) depending on the results of applying a certain action (***Sutton et al., 1998***). Q-learning (***Watkins and Dayan, 1992***) is one of the most popular online algorithms, whose updates are computed in an incremental manner.

BRL approach aims to collect a bunch of experiences and then use them for updating action influences instead of updating the action value function in an incremental way. In this batch framework, algorithms like Experience Replay (ER) or Fitted Q Iteration (FQI) (***Ernst et al., 2005***) can be found.

The Robot Soccer World Cup (RoboCup, (***Kitano et al., 1997***)) is an annual competition whose main objective is far beyond than just playing a robotics soccer game, it presents a natural scenario where RL problems can be found, in addition to several multidisciplinary challenges on its different leagues such as small size league, standard platform league, humanoid league and others. In this problem, a team of cooperative agents have to play a soccer match against another team composed of autonomous agents, noting that a possible objective for a given team could be to keep as far as possible the ball from its own goal area. In order to achieve this objective, many works can be found in the literature, from keepaway strategies using a multi-agent approach (***Stone et al., 2005***) to algorithms focused on training just the goalkeeper, as (***Ahumada et al., 2013***) or (***Celiberto et al., 2007***).

This work, like in (***Ahumada et al., 2013***) and (***Celiberto et al., 2007***) uses a grid for discretizing the state space of the agent and therefore avoiding to deal with a continous state space representation where tabular methods become impractical (***Baird and Klopf, 1993***).

Unlike the above references, most of the works found on literature (***Pietro et al., 2002***; ***Kalyanakrishnan and Stone, 2007***; ***Sawa and Watanabe, 2011***; ***Stone et al., 2005***) generates a state space representation based on angles and distances from the keeper (current learning robot that possess the ball) to every robot on the confined space of interest. Since large (or continuous) state spaces require function approximation, (***Stone et al., 2005***) uses tile-coding for approximating Q-values when implementing and comparing online RL algorithms (Q-learning vs Sarsa(λ)).

Getting closer to our case of interest, assumptions on (***Kalyanakrishnan and Stone, 2007***) allow the agents to communicate with each other in order to share their experiences. They also compare BRL with online RL algorithms, stating that Fitted-Q Iteration and Experience Replay reach a close performance with each other, but they both outperforms the online learning algorithm used in (***Stone et al., 2005***).

This document intends to compare Q-learning and its batch-version using Experience Replay on a simulation of the RoboCup Small Size League, noting that this involves a centralized decision problem, given the setup of the league, reducing the learning problem to a single agent case where each robot plays a fundamental role on the state space representation.

The remainder of this document is as follows: Section 2 makes a further description for BRL, presenting the algorithms that will be used later. Section 3 makes a brief description of RoboCup Small Size League, and explains how this setup can be used for introducing variations on the approaches found on literature for learning a keepaway strategy, while Section 4 shows the implementation of BRL algorithms on a simulated environment. Finally, Section 5 draw some final conclusions.

Batch Reinforcement Learning

Reinforcement learning (***Sutton et al., 1998***) (RL) tackles the problem of an agent that learns while interacting with the environment, deciding which action a to execute on the current state s of its environment, which transfers the agent to another state s' receiving a reward (reinforcement signal) whose nature would provide a quantification of how desirable was that choice. This problem can be formulated as an MDP (***Sutton et al., 1998***), composed by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ where

- $\mathcal{S}$: denotes the set of all possible states.
- $\mathcal{A}$: is a set of all the actions the agent can execute.
- $\mathcal{T}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is a state transition function, which gives the probability that when the agent is in state s and executes action a, the agent will be transferred to another state s'.
- $\mathcal{R}$: $\mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a scalar (real-valued) reward function.
- π: $\mathcal{S} \rightarrow \mathcal{A}$ denotes the mapping from states to action, describing the policy the agent should take given a certain state.

As mentioned before, the task of the agent is to learn the sequence of actions (therefore the optimal policy, π^*) that leads to maximize the expected sum of all the rewards received in the long-term. This is tackled by maximizing the return R_t, i.e. the discounted sum of rewards that the agent will obtain from time t, given by

$$R_t = \sum_{k=0}^{n-1} \gamma^k r_{t+1+k}, \tag{1}$$

where γ stands for the discount factor, with $0 \leq \gamma < 1$, and r_{t+1} stands for the expected (scalar) reward obtained for executing action a_t in state s_t. Then, two quantifications for the expected return are defined, the value function and action value function, V^π and Q^π respectively.

Value function is defined as the expected return when the agent is on state s_t at time t,

$$V^\pi(s) = E_{\pi_t}\{R_t | s_t = s\}, \tag{2}$$

while the action value function is defined as the expected return when the agent executes a_t on state s_t at time t following policy π,

$$Q^{\pi}(s,a) = E_{\pi_t}\{R_t|(s_t = s, a_t = a)\}. \tag{3}$$

Both functions are clearly related, as

$$V^{\pi}(s) = E_{a|s}\{Q^{\pi}(s,a)\}. \tag{4}$$

A representative method in model-free RL is Q-Learning (***Watkins and Dayan, 1992***), which makes an approximation of the optimal action-value function based on the optimal policy, by making successive updates for estimations of Q, this update would be given by

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

where this approximation, Q, corresponds to the learned action-value function, and α stands for the learning rate.

In order to understand the difference between the incremental update from online algorithms and simultaneous update of batch algorithms, consider two consecutive transitions (s,a,r,s'), (s',a,r,s'') and the classical online Q-learning algorithm. Then, when $Q(s',a)$ is computed using the update rule on (5), this change will not be backpropagated to $Q(s,a)$ nor any of the state-action pairs preceding s', being updated just when those states are visited again.

In the pure batch reinforcement learning approach, the agent does not interact with the environment while the learning phase is taking place. In growing batch reinforcement learning, which most of the modern batch algorithms are based on, the task of collecting transitions and learning from them are alternated for improving the exploration policy.

Algorithm 1 describes the procedural form of a (growing) BRL approach independently of the algorithm used for updating Q-values, as shown on (***Kalyanakrishnan and Stone, 2007***). Note that when the number of forgotten experiences,m, is the same as the size of the size of the batch, i.e. $m = |D|$, experiences are forgotten so growing BRL is reduced to pure BRL, which is not the case of this proposal.

ALGORITHM 1

Batch reinforcement learning procedure

```
Initialize Q(s,a) arbitrarily ∀s ∈ S, a ∈ A
Initialize batch of experiences D as an empty set
repeat
  for each episode do
    for each step t on current episode do
      Identify current state s_t
      Choose a suitable action a_t in state s_t using policy derived from Q
      Observe r_{t+1} and s_{t+1} when taking action a_t
      Add experience (s_t, a_t, r_{t+1}, s_{t+1}) on batch D
      s_t ← s_{t+1}
    end for
  end for
  Update Q values
  Forget m experiences from batch D
until action value function convergence is reached
```

Moreover, (***Kalyanakrishnan and Stone, 2007***) states that is better (for their task) to use all the experiences gathered so far. This means that if every batch consists on experiences from 20 episodes, then the first updates of Q estimations will consists on experiences from those 20

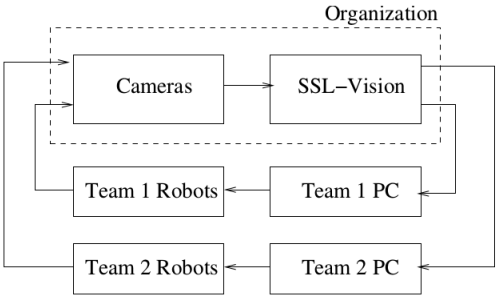

Figure 1. RoboCup SSL scheme

episodes. Then the second time these updates are computed it will consists on experiences from 40 episodes and so on, which represents an extremely memory consuming process.

One of the basic BRL algorithms, Experience Replay (ER) aim to improve the speed of convergence of the action value function by replaying observed transitions repeatedly just as if they were new observations. Algorithm 2 shows the procedural form of this algorithm.

ALGORITHM 2
Experience Replay procedure

1: **for** each training iteration **do**
2: **for** each transition (s_i, a_i, r_i, s_{i+1}) on D **do**
3: Update $Q(s_i, a_i)$ by using

$$Q(s_i, a_i) \leftarrow (1-\alpha)Q(s_i, a_i) + \alpha\left(r_{i+1} + \gamma \max_a Q(s_{i+1}, a)\right)$$

4: **end for**
5: **end for**

It is immediate to note that what this algorithm does, is to compute several times the updates of Q-learning on collected transitions as an offline algorithm would do, thus speeding up the propagation of Q values to preceeding states, but then the system is allowed to collect new transitions for improving those previously computed estimates.

Test domain: RoboCup SSL

RoboCup presents a challenging domain where a team of robots have to play a soccer match against another robotic team, where the particular assumptions on the game varies across the different leagues. This application focuses on the Small Size League (SSL), inspired by the development and research work made by Sysmic Robotics USM (previously known as AIS Soccer) (***Rodenas et al., 2018***), a group of students whose main objective is to compete on this annual event, and also test state-of-the-art computational intelligence techniques on this particular setup.

Figure 1 depicts the scheme of this league, where the current positions of each robot at both teams is given as result of the image proccessing made by SSL-Vision, whose images are acquired through video cameras provided by the organization comittee, located at the top of the soccer field. Then, both teams receive the exact same data to their own decision maker programs, which once an action is chosen informs the actions to take for each robot of its team via a wireless channel.

Although we tackle the problem of finding a keepaway strategy, several challenges arises at the Small Size League in addition to the already mentioned problems like goalkeeper training on (***Ahumada et al., 2013***), learning the opponent strategy as on (***Yasui et al., 2013***),or learning to control the dribbler (***Riedmiller et al., 2008***), noting that the work therein focuses on the Middle Size League. This latter problem also applies to the Small Size League, being specially difficult to

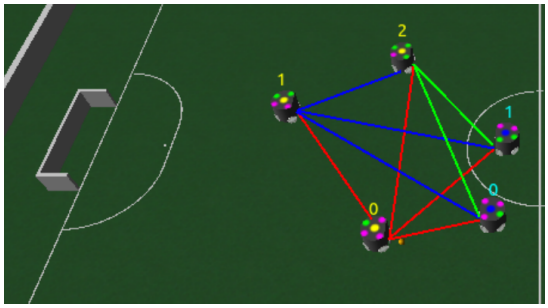

Figure 2. problem setup on GrSim simulator, with 3 keepers and 2 takers

keep possesion of the ball with the dribbler while changing the orientation of the robot.

Implementation and results

Modelling the learning problem

Although we use a slightly different state space representation compared with (***Stone et al., 2005***), by using a centralized decision taking problem given that we have a global vision of the field unlike some other RoboCup leagues. Then, we set the keepaway learning problem to be composed of 3 keepers, robots in charge of keeping the ball as long as possible away from their goal area, and 2 takers, which are robots from the opponent team and whose objective is to take the ball and shoot to the (center of) goal area.

As the offense strategy learning for allowing the takers to learn better strategies to effectively score is out of the scope of this work, we fixed their policy in a manner that they are always chasing the ball, and once they got it, just shoot to the goal area.

Figure 2 shows the setup of this problem in the simulated environment where algorithms will be tested, GrSim (***Monajjemi et al., 2011***), which has been very helpful for testing computational intelligence methods before implementing them on the real robots.

The state is composed by distances from every keeper to all the other robots, including takers and other keepers as shown in Figure 2. Also the distance from the ball to every robot is considered, and the angle between a keeper and each taker (with respect to an imaginary horizontal line across the soccer field). In other words, the state s_t at a given time t is composed by

- dist(K_i ,ball),
- dist(T_j ,ball),
- dist(K_i ,K_j), $i \neq j$,
- dist(K_i ,T_j),
- angle(K_i ,T_j),

where K_i stands for the i-th keeper and T_j for the j-th taker. Also, the reader should note that although different state representations could work for a given problem, the angle is neccessary for modelling this problem. Even when assuming that all the robots are always facing the ball, since if just the distance d from the i-th keeper to the j-th taker is used, there would be theoretically infinite points around a circle of radius d and centered on the position of the keeper where the taker could possibly be.

Then, the possible actions to execute by a keeper are

- $hold()$: all keepers remains on their current positions without making any pass nor trying to intercept the ball.
- $pass(K_i, K_j)$: the i-th keeper performs a pass to the j-th keeper, where obviously $i \neq j$ since it would be equivalent to hold() action.
- $intercept(K_1, K_2, ..., K_n)$: send keepers to intercept the ball whenever its respective binary argument is set to 1.

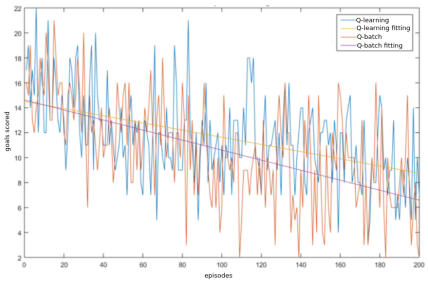

Figure 3. Goals scored by enemy team

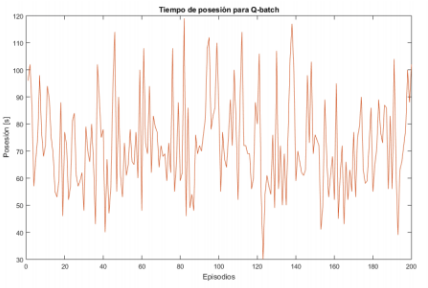

Figure 4. Time of possesion on the ball by keepers

In this case, since there are 3 keepers, $intercept(0,1,0)$ would send 2nd keeper to intercept the ball, while $intercept(1,0,1)$ would send 1st and 3rd keepers to intercept the ball. Note that $intercept(0,0,0)$ is not allowed, since it would be equivalent to $hold()$ action.

Note that unlike the work in (***Stone et al., 2005***), since we have a global vision of the field and thus focused on a centralized decision making problem, we learn a Q value function for the whole system and not one for each keeper. We have identificators for each keeper, so the 1st keeper will be K_1 always, and does not refer to the keeper who is closest to the ball.

Since the final objective of the keepaway learning problem, is to learn to keep as long as possible the ball away from the goal area, then we will reward actions that privileges the ball possesion, and punish actions that leads to lose possesion and punish harder when it leads to a goal scored against the team.

Simulation results

When implementing the algorithms described on Section 2, we used a growing BRL approach where the batch of experiences D contains transitions from 20 episodes, where each one lasts 2 minutes of gameplay (without considering reset time when a goal is scored and robots are re-locating). Then, after updating Q-values estimations all those transitions are discarded, so the size of the Batch always have the data for 20 episodes when entering to the learning phase.

According to rewards obtained through the learning episodes, whose values are set to 5 for keeping possesion on the ball, -5 in case of losing possesion and -50 in case of the enemy team scoring a goal. Then, according to these reinforcement values, Figure 4 shows the evolution of time possesion on the ball.

It can be seen from Figure 3, where the line represents the mean through 10 reproductions of the learning task, that batch version of Q-learning using Experience Replay achieve better performance compared with its classical online version on a smaller amount of time. However, it is expected that after several learning episodes more, batch version would learn faster but they both achieve the same results at last.

Despite the efficiency on the use of collected transitions of the learning agent, speed of convergence for both algorithms is directly affected by the number of possible states obtained from

the chosen state space representation. Then, as the discretization grid becomes thinner, the state space becomes larger and tabular methods become slower and even impractical for a continuous state space representation, so function approximation methods are needed.

Conclusions

As expected because of data reusability of experiences gathered so far, Experience Replay learn faster in terms of defending the goal area, and this is mainly due to its synchrony nature and a better use of collected experience on the interaction process between the agent and its environment for this task. Obtained results shows the benefits of re-using data efficiently and in an inherently multi-agent problem tackled from a single agent learning task given the centralized setup of this league. Future work may include a more in-depth analysis including other update rules and strategies in Batch Reinforcement Learning methods, as well as field testing in other leagues, and considering a continuous state space representation using function approximators such as artificial neural networks or a fuzzy representation of states.

References

Ahumada GA, Nettle CJ, Solis MA. Accelerating Q-Learning through Kalman Filter Estimations Applied in a RoboCup SSL Simulation. In: *Robotics Symposium and Competition (LARS/LARC), 2013 Latin American* IEEE; 2013. p. 112–117.

Baird LC, Klopf AH. Reinforcement learning with high-dimensional, continuous actions. Wright Laboratory, Wright-Patterson Air Force Base, Tech Rep WL-TR-93-1147. 1993; .

Celiberto LA, Ribeiro CH, Costa AH, Bianchi RA. Heuristic reinforcement learning applied to robocup simulation agents. In: *Robot Soccer World Cup* Springer; 2007. p. 220–227.

Ernst D, Geurts P, Wehenkel L. Tree-based batch mode reinforcement learning. Journal of Machine Learning Research. 2005; 6(Apr):503–556.

Kalyanakrishnan S, Stone P. Batch reinforcement learning in a complex domain. In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems* ACM; 2007. p. 94.

Kitano H, Asada M, Kuniyoshi Y, Noda I, Osawa E. Robocup: The robot world cup initiative. In: *Proceedings of the first international conference on Autonomous agents* ACM; 1997. p. 340–347.

Kober J, Bagnell JA, Peters J. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research. 2013; 32(11):1238–1274.

Monajjemi V, Koochakzadeh A, Ghidary SS. grsim–robocup small size robot soccer simulator. In: *Robot Soccer World Cup* Springer; 2011. p. 450–460.

Pietro AD, While L, Barone L. Learning in RoboCup keepaway using evolutionary algorithms. In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* Morgan Kaufmann Publishers Inc.; 2002. p. 1065–1072.

Riedmiller M, Hafner R, Lange S, Lauer M. Learning to dribble on a real robot by success and failure. In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on* IEEE; 2008. p. 2207–2208.

Rodenas T, Alfaro R, Reyes P, Pandolfa D, Pinto F, Aubel M, Yanes P, Barrera T, Kim SH, Castillo S. AIS Team Description Paper. . 2018; .

Sawa T, Watanabe T. Learning of keepaway task for RoboCup soccer agent based on Fuzzy Q-Learning. In: *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on* IEEE; 2011. p. 250–256.

Stone P, Sutton RS, Kuhlmann G. Reinforcement learning for robocup soccer keepaway. Adaptive Behavior. 2005; 13(3):165–188.

Sutton RS, Barto AG, Bach F, et al. Reinforcement learning: An introduction. MIT press; 1998.

Watkins CJ, Dayan P. Q-learning. Machine learning. 1992; 8(3-4):279–292.

Yasui K, Kobayashi K, Murakami K, Naruse T. Analyzing and learning an opponent's strategies in the RoboCup small size league. In: *Robot Soccer World Cup* Springer; 2013. p. 159–170.