Transfer Learning Approach for Autonomous Agents in Collective Games

Ventseslav Shopov, Vanya Markova

Institute of Robotics
Bulgarian Academy of Sciences
Bulgaria
Email: vkshopov@yahoo.com, markovavanya@yahoo.com

***Abstract*—The aim of this study is to present a new approach for Transfer Learning in collective games. This framework is a set of methods for transferring accumulated knowledge. In this way, autonomous agents share their knowledge in order to achieve better performance. The main hypothesis in the study is that the group of agents who exchange knowledge performs better than the same group without Transfer Knowledge, under the same conditions.**

Keywords–autonomous agents; reinforcement learning; transfer learning.

I. INTRODUCTION

The subject of the study is the transfer of knowledge in training and decision-making for autonomous agents. In this study, we look at the environment as collective sequential games. Our goal is to clarify whether Markov Decision Process (MDP) solving methods can be applied to collective games with partially observable goals and partially dynamic environments.

In addition, we raise the question of how effective transfer of knowledge in training and decision-making by autonomous agents in collective games is.

Exploring these issues is important for the development of training with support in general, and in particular for the transfer of knowledge between agents in partially observable and dynamic environments. Knowledge transfer can significantly speed up training and decision-making by autonomous agents. Such research can be found in machine learning, the video game industry, and robotics.

In this study, we build upon a recent method for knowledge transfer, which formulates the sequencing problem as a Markov Decision Process. Recently, various representations that make such knowledge transfer possible for multiple agents in different domains have been explored [1]. In addition, some generalisation of curriculum MDP model have been proposed [2] to handle different kinds of transfer learning algorithms. Another approach formulates the design of a curriculum as a Markov Decision Process, which directly models the accumulation of knowledge as an agent interacts with tasks to produce an agent-specific curriculum [2] such that overall performance or learning speed is improved [3].

There are several studies that introduce methods to generate a curriculum based on task descriptors [4], or by data-driven automated similarity measures [5]. Other methods combine feature-based control in a non-rewarding discrete environment, and imitation learning applied to an ambiguous and unconstrained third party agent [6].

Some recent studies have been performed in regard of creating frameworks for selecting source tasks in the absence of a known model or target task samples based on meta-data [7] or guided by policy sketches. [8]

In our study, we make an effort to allow the application of already developed and tested methods and algorithms to solve MDP in fields such as multi-agent systems and collective games. TL can also accelerate the learning process in various areas of machine learning, the video game and robotics industries. Given certain limitations, it is possible to use solutions that have already been tested, which may lead to a reduction in time of developing new applications.

The main hypothesis of this study is that, subject to certain limitations, it is possible to use classical MDP solving methods for partially observable and dynamic environments. It is also possible to apply knowledge transfer to groups of autonomous agents. Such a transfer leads to acceleration of training and decision-making in collective games.

The article is organised as follows: In Section 2, we briefly look at the theory underlying the proposed solutions, and then we describe the theoretical limitations of our approach and the respective implementation. In Section 3, we experimentally examine the applicability and effectiveness of our approach. In the last part, we describe our findings.

II. METHODS AND MATERIALS

A. Theory

1) Sequential games: We consider sequential games, which are n-player non-zero sum games played on finite trees. Each node of the tree is controlled by either of the players, and the game is played by moving a token along the branches of the tree, from the root node, up to the leaves, which are libelled by a payoff. We also associate a preference relation with each player that indicates how he ranks the payoffs. Let us now formalise the basic notions about these games. The definitions and notations of this section are inspired from [9].

Definition 1. A sequential or extensive form game G is a tuple $(N; A; H; O; d; p; (\prec_i)$ where:

N is a non-empty finite set of players;

A is a non-empty finite set of actions;

H is a finite set of finite sequences of A which is prefix-closed. That is, the empty sequence ϵ is a member of H; and $h = a^1, ..., a^k \in H$ implies that $h^l = a^1, ..., a^l \in H$ for all $l < k$. Each member of $H$is called a node. A node $h = a^1, ..., a^k \in H$ is terminal if $\forall a \in A, a^1, ..., a^k, a \notin H$. The set of terminal nodes is denoted by Z.

O is the non-empty set of outcomes, $d : H \setminus Z \rightarrow N$ associates a player with each non-terminal node;

 ISBN: 978-1-61208-712-2

$p : Z \to O$ associates an outcome with each terminal node;

For all $i \in N :\prec_i$ is a binary relation over O, modelling the preferences of player i.

From now on, we fix a sequential game $G = (N, A, H, O, d, p, (\prec_i)_{i \in N})$.

Then, we let $H_i = (h \in H \setminus Z \mid d(h) = i)$ be the set of nodes belonging to player i. A strategy $s_i : H_i \to A$ of player i is a function associating an action with all nodes belonging to player i, s.t. for all

$h \in H_i : hs_i(h) \in H$, i.e., $s_i(h)$ is a legal action from h. Then, a tuple $s = (s_i) \in N$ associating one strategy with each player is called a strategy profile. For all strategy profiles s, we denote by (s) the outcome of s, which is the outcome of the terminal node obtained when all players play according to s. Single-agent Reinforcement Learning (RL) concepts are given first, followed by their extension to the multi-agent case.

2) Markov Decision Process: We formulate the transfer learning problem in sequential decision making domains using the following framework of Markov Decision Process. We use the following definition of MDP as a 5-tuple

$$< S, A, P, R, \gamma > \tag{1}$$

where the set of states, set of actions, transition function and reward function are described. And

$$P : S \times A \to \Pi(S) \tag{2}$$

is a transition function that maps the probability of moving to a new state given an action and the current state,

$$R : S \times A \to R \tag{3}$$

is a reward function. that gives the immediate reward of taking an action in a state.

And

$$\gamma \in [0, 1] \tag{4}$$

is the discount factor. The gradient formula can be written as [10]. So the MDP of the agent is described in (1), where s is the set of states, a is the set of actions, p is the transition function and r is a reward function. The transition function p maps the the probability of moving to a new state given an action and the current states and is shown in (2). The reward functions r that gives the immediate reward of taking an action is described in (3). The discount factor γ is bounded as is shown in 4.

Multi-agent Markov games can be defined by N agents with a set of global or local observations $O_1, ..., O_N$, a set of actions $A_1, ..., A_N$, a set of states S and a state transition function

$$T : S \times A_1 \times A_2 \times ... \times A_N \to S \tag{5}$$

which determines the Markov process. For each agent i, it interacts with the environment by taking actions following its policy $\pi_{Q_i} : A_i \to [0, 1]$ transformed into the next state and gets a reward $r_i : S \times A_i \to R$ to judge the policy's performance. Each agent tries to maximise the accumulated discount return

$$R = \sum t = 0^T \gamma^t r^t \tag{6}$$

and T is the expect time horizon and γ is the discount parameter. In this paper, only local observations are available for all games.

3) Reinforcement Learning: To solve sequential decision-making problems, the agent should learn about the optimal value of each action, defined as the expected amount of future rewards when taking this action and following the optimal policy afterwards. Under a given policy π , the true value of an action a in a state s is

$$Q_\pi(s; a) = E[R_1 + \gamma R_2 + \ldots | S_0 = s; A_0 = a;] \tag{7}$$

where $r \in [0; 1]$ is a discount factor which trades off the importance of immediate and later rewards. The optimal value is then $Q_{\pi*}(s; a) = maxQ(s; a)$. An optimal policy can be easily learned from the optimal values by selecting in every state the highest valued action.

4) Q-Learning: The optimal action values can be derived through Q-learning [11] [12], a form of time learning. The real problems are too large to learn all the action values in all states separately. Instead, we can learn a parametric value $Q(s; a; q_t)$. In this way, Q-learning values update the parameters after taking action A_t at S_t and observe the immediate reward R_{t+1} so that the resulting state S_{t+1} is then

$$q_{t+1} = q_t + \alpha(Y_t^Q - Q(S_t; A_t; q_t))\nabla_{q_t} Q(S_t; A_t; q_t) \tag{8}$$

where q is a scalar value and the target Y_t^Q is defined as

$$Y_t^Q = R_{t+1} + \gamma \max_a Q(S_{t+1}; a; q_t) \tag{9}$$

In order to update the current value $Q(S_t; A_t; q_t)$ towards a target value Y_t^Q the agent applies stochastic gradient descent approach.

5) Deep Q Networks: Deep Q networks (DQN) are multi-layered neural networks. These networks, for a given state s, output a vector of action values $Q_{theta}(s; a; q)$, where θ are the parameters of the network. If an action space contains m actions and state space is a n-dimensional vector, the neural network maps R^n to R^m. In addition, in Deep Q Network there is target network [13], with parameters θ^-. This additional network is the same as the original network except that its parameters are copied every τ steps from the online network, so that $\theta_t^- = t$, and are not changed on all other steps. So, the target used by DQN is then

$$Y_t^{DQN} = R_{t+1} + \gamma \max_a Q(S_{t+1}; a; \theta_t) \tag{10}$$

6) Double Q-learning: To prevent overoptimistic value estimation, we can decouple the selection from the evaluation. This is the idea behind Double Q-learning [14]. In the original Double Q-learning algorithm, two value functions are learned by assigning each experience randomly to update one of the two value functions, such that there are two sets of weights, and 0. For each update, one set of weights is used to determine the greedy policy and the other to determine its value. For a clear comparison, we can first untangle the selection and evaluation in Q-learning and rewrite its target 10 as

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, max_a Q(S_{t+1}; a; q_t); q_t) \tag{11}$$

The Double Q-learning error can then be written as

$$Y_t^{DoubleQ} = R_{t+1} + \gamma Q(S_{t+1}, max_a Q(S_{t+1}; a; q_t); q_t) \tag{12}$$

 ISBN: 978-1-61208-712-2

7) Autonomous agents: the autonomous agent in our approach has the following main features: - Autonomy - Reactivity - Proactivity - Communicativeness It is important to emphasise that our agents are autonomous not only during decision-making but also during training. This means that agents are able to learn and adapt to changes in the environment without the need for external training.

B. Implementation

To enable the MDP agent to work in Partially Observed Markov Decision Process (POMDP), its learning algorithm and decision-making algorithm must be expanded. The drawback of this approach is that already trained agents and knowledge gained in the MDP training process can not be reused.

The environment for partially observable dynamic collective games in the MDP environment is important due to two reasons: to be able to apply methods and algorithms developed for classic MDP cases of partially observable dynamic collective games; and it allows to transfer knowledge between agents trained in MDP and POMDP.

In this study, we describe the agent environment as MDP. We are driven by the desire to present the various properties of the autonomous agent so that the agent is compatible with the MDP constraints. Moreover, we strive for a generalised approach to the training of our autonomous agents.

By expanding agents' space, we present POMDP as MDP. Such representation is only possible if the following limitations are met: Partial environmental observability can be eliminated through communication between agents; dynamic changes in the environment are reflected by expanding the transition function.

Our goal is not to expand the environmental model. In addition, environments represented by POMDP can describe significantly complex systems of interactions with those described with classic MDP. Thus, by expanding the agent's state space by adding global states of the medium, the agent is compatible with the classical MDP environments.

But the extension of the MDP notation leads to some drawbacks: such as the need to modify learning algorithms s that the incompatibility of policies resulting from such training has made the transfer of knowledge between MDP trained agents enriched by such trainees POMDP environments. In the case of sequential collective games, describing the environment as partially observable does not necessarily have to be achieved by introducing POMDP. The agents themselves are able to change the environment, but by clearly announcing the changes in the environment, the agents through communication are able to bypass the limit of partial observability.

The imposition of restrictions and the expansion of the state space takes place in two stages:

a set pair $< s_{agent}, s_{env} >$ is created. The so-called pair is used for a generalised representation of agent states in a partially observed collective game environment. By imposing these limitations, we allow the use of MDP solving methods to be applied in the field of collective games with a dynamic environment.

We start with expansion of the state space, where state space is:

$$S = \{s_i\}, i = 1, ..N \tag{13}$$

so we expand space of agent s_{agent} with the space of the environment s_{env}a: so we form a tuple:

$$s_i =< s_{agent}, s_{env} > \tag{14}$$

where s_{agent} as a result of agent's actions and s_{env} as a result of environment changes.

However, to incorporate in natural manner the changes in environment we also have to expand the transition function:

$$< s'_{agent}, s'_{env} >= T(s_i) = T(s_{agent}, s_{env}) \tag{15}$$

so that the agents state is defined as follow:

$$s'_{agent} = Pr[s_{agent}(t+1) = s'_{agent}(t) = s, a_t = a] \tag{16}$$

and the environment changes reflect in environment state:

$$s'_{env} = Te[s_{env}(t+1) = s'_{env}(t) = s, a_t = a] \tag{17}$$

Thus, by expanding the state space and the transition function we map the constrained unobservables and dynamic of the environment into combined state space and extended transition function.

Each agent may have an individual transition function, so different agents can interact in a team, but the degree of knowledge transfer depends on how different the transition function differs between agents. So, to achieve full portability of the methods, as well as knowledge transfer between individual agents the function of the transition of the environment is have to be the same for all agents.

III. Experiments and results

We gather evidence to support the hypothesis that we will speed up the learning process for knowledge transfer. It performs the following experiments: for a given map several combinations of autonomous agents should be generated. These agents should be grouped in three main parts: competitive, cooperative and neutral.

The map is described by its size $n \times n$ and the complexity factor R_c. The map generates random k treasure chests with treasure. The treasure value is 100. Additionally, k traps are generated. These pits can not be set right beside the treasures. If an agent gets into a pit, a -100 prize is generated. As a result of the complexity factor R_c, obstacles are generated. If an agent hits an obstacle, he returns to the starting position. The obstruction generation algorithm does not allow the creation of a closed area. We only issue instances when the number of agents is equal to the number of treasures. A game ends when the agent finds the treasure and takes it, falls in a trap or makes more than n^2 moves. For each move, except the last agency, agent get a small negative reward (for example -1).

Once the agent starts to learn it use a Reinforcement Learning approach. As a base algorithm, we use SARSA. For one agent, we have one pit and one treasure. Solving this problem is trivial.

If we put one agent in map with one treasure then agent quickly learns how to get the treasure. A problem arises when there are two agents and two treasures. Once the first agent reaches his treasure and takes it then in the map will remain an "empty" chest. The second agent, if closer to a treasure already taken, will try to take it, but the chest is now empty. So the second agent will go down to a local minimum and end the game with negative reward.

If the agency is a cooperative then when the first agent gets its treasure it reports to others that treasure is already

 ISBN: 978-1-61208-712-2

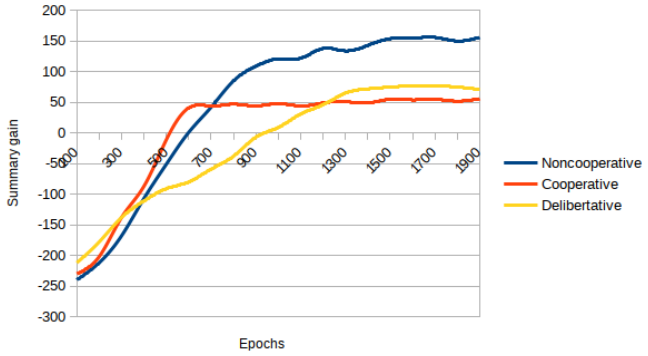

Figure 1. We compare cooperative and competitive strategies in simple one-goal map. In this map there are only one trap and relatively small amount of obstacles.

taken. There are several ways to address this issue, so we need to expand the MDP model. In order to stay in place, the agent should initially ”change” the environment so that the ”changed” environment is in consistency with a policy that will lead to the treasure.

We compare three approaches:

- Non-cooperative game with non-cooperative learning: where the first one has reached treasure ends the game with a 100 prize, and the next may fall to the local minimum.
- Cooperative Game with Deliberative Cooperative Learning: A binary vector for treasure is generated at the coordinator. In practice, the number of states in which there is a permanent effect on all possible treasure states is increased. If only the positive reward of the training process are combined against a sufficiently high level.

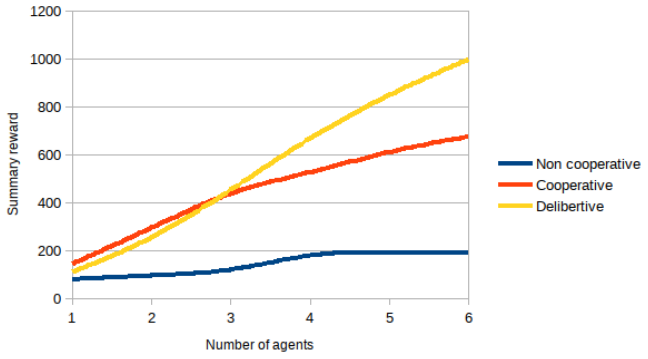

Figure 2. We compare competitive, cooperative and deliberative strategies in complex multi-goal map. In this map there are plenty of traps and relatively big amount of obstacles.

As can be seen in Figure 1, if we have only one prize, then cooperative behaviour has no advantage over competitive behaviour. In Figure 2 one can see that with the growing number of agents in the team, the use of the deliberate approach is better than the rest of the algorithms.

IV. Conclusion

A new approach is proposed to transfer knowledge among agents in collective games.The approach suggested in this article allows knowledge from pre-trained agent for pre-defined environments to be used. Our approach allows to speed up the training of agents. Instead of random values initialisation of utility or quality function, we can take such values from an already trained agent. By expanding the state space and transition function in MDP classes, we allow, subject to certain limitations, that MDP solving methods be applied to partially observable and partially dynamic environments. In addition, it is possible to transfer knowledge into collective applications.

From the results of our research it follows that in the case of only one treasure, complex cooperative interaction has no advantages over competitive approaches. In other words, the use of deliberative techniques in simple systems is over-engineering. And only with the increasing complexity of the choice between individual goals, the cooperative behaviour demonstrates the advantages of the deliberative approach.

References

[1] S. Narvekar and P. Stone, “Learning curriculum policies for reinforcement learning,” arXiv preprint arXiv:1812.00285, 2018.

[2] S. Narvekar, J. Sinapov, and P. Stone, “Autonomous task sequencing for customized curriculum design in reinforcement learning.” in IJCAI, 2017, pp. 2536–2542.

[3] S. Narvekar, J. Sinapov, M. Leonetti, and P. Stone, “Source task creation for curriculum learning,” in Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2016, pp. 566–574.

[4] M. Svetlik et al., “Automatic curriculum graph generation for reinforcement learning agents,” in Thirty-First AAAI Conference on Artificial Intelligence, 2017, pp. 2590–2596.

[5] H. B. Ammar et al., “An automated measure of mdp similarity for transfer in reinforcement learning,” in Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014.

[6] P. Fournier, O. Sigaud, M. Chetouani, and C. Colas, “Clic: Curriculum learning and imitation for feature control in non-rewarding environments,” arXiv preprint arXiv:1901.09720, 2019.

[7] J. Sinapov, S. Narvekar, M. Leonetti, and P. Stone, “Learning inter-task transferability in the absence of target task samples,” in Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 725–733.

[8] J. Andreas, D. Klein, and S. Levine, “Modular multitask reinforcement learning with policy sketches,” in Proceedings of the 34th International Conference on Machine Learning-Volume 70. JMLR. org, 2017, pp. 166–175.

[9] M. J. Osborne et al., An introduction to game theory. Oxford university press New York, 2004, vol. 3, no. 3.

[10] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press Cambridge, 1998, vol. 1, no. 1.

[11] C. J. Watkins and P. Dayan, “Q-learning,” Machine learning, vol. 8, no. 3-4, 1992, pp. 279–292.

[12] C. J. C. H. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, King’s College, Cambridge, 1989.

[13] V. e. a. Mnih, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, 2015, pp. 529–538.

[14] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning.” in AAAI, vol. 16, 2016, pp. 2094–2100.

 ISBN: 978-1-61208-712-2