Cooperative Co-Evolutionary Multi-Agent Systems

Tobias Peter[1]

Otto-von-Guericke Universitt, 39106 Magdeburg, Germany,
tobias.peter@st.ovgu.de

Abstract. Cooperative multi-agent systems model how different autonomous agents interact with each other to achieve a shared goal. There are many parameters that the operator of those agents needs to set for the team to act cooperatively and reach the goal. Therefore, methods from the machine-learning domain, are used to explore all possible parameters and find the most optimal ones, automatically. This report will concentrate on research where the teams that have more than two or three agents with different behaviors. At first, we present the history and background of multi-agent systems. After that, we give an overview of the of the problems, which arise when heterogeneous teams and machine-learning are used together. After that, we show existing algorithms, which can cope with these problems.

Keywords: multi-agent systems, machine learning, team learning, co-evolution

1 Introduction

The research field of Artificial Intelligence (AI) [16] about the development of Methods that allows man-made machines to exhibit human-like intelligence. The goal is to create intelligent machines that can entertain or help humans. This term is usually applied to cases when a computer can learn or solve problems on its own, that means cognitive abilities that associate with the human mind. The other thing is that it is a buzzword that is applied to the current cutting edge technology, and constantly shifts as previously challenging task become easy tasks and researchers venture into other areas of research. Current examples of cutting edge AI methods that can learn Go and beat a human champion [18], and autonomous cars, that can drive through a city safely [11]. Such an intelligent machine is usually called an agent. A more formal definition is: an intelligent agent is an autonomous entity which observes its environment through sensors and acts upon the environment using actuators. Agents can be very simple or very complex. In this survey, we will concentrate on Agents of medium complexity which have to work together with other agents and are created by learning from successes and failures.

2 Multiagent Systems

Multi-agent systems (MAS) [17] is a research field about how a set of autonomous agents behave in a shared Environment. The main idea that the agents are deciding themselves what they do and are not controlled by a centralized entity. This can solve problems where control by a centralized entity would not be feasible or needed [4]. In a centralized approach all agents would have to communicate every information with central entity and receive back their instructions. That can be expensive, there could be delays since a lot of communication is needed, everything fails if the communication or central entity breaks. Sometimes such a solution is also simply too complicated and all that is needed are autonomous agents that periodically send status information to a human operator.

In a multi-agent context, the agents have usually only access to local information, but not the state of the whole world or other agents[3]. The rules for every agent, have to be designed in a way so that they can can do local actions based on local information, but still reach a singular, global goal. At least in a fully cooperative or team context, there are also use-cases for a MAS where all agents are competing.

The next sections are an short introduction to machine Learning, which is needed for complex MAS. After that there is a short section about problems about the combination of both topics and the problems that arise.

2.1 Learning

Machine Learning is a well-researched sub-field in Computer Science. Machine Learning is about algorithms that allow computers to extract a model from data, which then can be applied to new and unseen data and still produces the desired results [12, 2]. It is applied in situations where handmade programs are infeasible. Take for example an image recognition task where you have as input a 200x200 gray-scale image, which is, in the computer, represented as 200x200 matrix of real values with 1.0 being white, 0.0 being black, and everything in between being some shade of gray. Now you want to know if it is a picture of a cat or not automatically. Creating a computer program, by hand, to solve this task would be tedious at best, if at all possible.

Contrary to that, you can train a Classifier with a Machine Learning Algorithm. You have to supply the learning algorithm with a lot of pictures which are pre-labeled as 'cat' or 'not-cat'. The Machine Learning algorithm then build, with the help of those pictures, an internal abstract model which can automatically differentiate between 'cat' or 'non-cat' pictures. It automatically recognizes the subtle cues and relationships between single pixels that can be used to identify the cats. After some time the Machine Learning algorithm terminates, and if the training was successful, the classifier can classify pictures which were not used in the training. It can generalize its knowledge to previously unseen pictures.

In Machine Learning three general problem cases are differentiated: Supervised Learning where for each training example the user has to specify the correct output, like in the example above. In Unsupervised Learning the inputs do not need to be specified. Unsupervised Learning concerns itself with finding patterns in the data and is not so useful for learning a user-specified goal. For learning behaviors of agents usually, reward-based Learning or reinforcement Learning is used.

2.2 Reinforcement Learning

Reinforcement Learning (RL) [22] is defined as the process by which an Agent can learn from the repeated interaction with its environment. The agent must have the ability to at least partially observe its environment. A single observation is usually called 'state', as in the state of the environment. The agent also has a set of actions from which he can choose one based his policy. A Policy is a function that maps from a state to an action. It is a bit like the classifier we talked about before the state is the input, then the classifier tries to determine what kind of situation he is in, and then chooses the correct outputs (actions) to achieve his goal. A supervised learning algorithm does not the best choice for this. Since that would mean the user has to know for every situation which actions to choose. That is usually not the case; therefore, RL is used.

The basic idea of RL is that the agent repeatedly visits the training environment where he tries to achieve his goal. Every time he does a series of actions that brings him closer to his goal or he reaches his goal he gets a positive reward. In contrast, every time he does something which pushes him away from his goal or navigates into a situation where he can never reach his goal he gets a negative reward. At first, such an Agent in training would usually behave randomly or is initialized with a basic behavior. Through repeated interactions with the environment the agent would learn in which states he has to do what action to get a reward, and does it more often which will lead to the agent better achieving his goal. That is the origin of the name RL since all that the algorithm does is reinforcing useful behaviors of the agent.

3 Multi-agent Reinforcement Learning

Creating policies for every agent in a MAS by hand is not feasible, except for the simplest systems. Therefore, it is a good idea to use RL to learn policies for all the agents. The use of RL for MAS creates new problems. One of those problems is the very large state space (the set of all possible states a system can be in) which are inherent to MAS. If agent A can be in m states and agent B can also be in m states. The whole MAS can be in $m \cdot m$ states. Generally, a MAS can be in m^n states with m being the number of states each agent can be in and n being the number of agents. That makes the use of traditional RL algorithm which explores the utility of states infeasible. Those algorithm involve estimating value functions that indicate how helpful it is to be in a state or perform an action in a given state. To do that nearly every state has to be visited. Seeing that the state space is very large this is impossible. As a result, many researchers focus on evolutionary algorithms, which explore the space of behaviors directly. This chapter will be discussing evolutionary computation, team composition, and how the concept of policies can be implemented using neural networks.

3.1 Evolutionary Computation

Evolutionary computation are machine learning methods inspired by real-world process of evolution. An evolutionary algorithm (EA) creates an initial set of candidate solutions (individual) and evaluates all of them. The result of the evaluation is called the fitness of the individual and at the end of an evaluation round the most unfit individuals are eliminated from the population. The surviving candidates are used to create new candidate solutions, called the offspring. This is done by selection two or more of the fitter individuals and recombining them or by mutating a single individual (randomly change a small part of it). The offspring is then again evaluated and unfit individuals are discarded. This process is repeated until a fitness goal is reached. In a RL context the individuals are the policies employed by the agents. To evaluate them, they are copied to the corresponding agents and tested in the learning environment. The fitness is usually the summed up rewards collected during the evaluation.

3.2 Team composition

The easiest approach is to use the same policy for all agents. This is called a homogeneous team and simplifies the learning of the agents immensely, because traditional evolutionary algorithms can be used. But homogeneous team learning will never achieve a good performance in scenarios where division of labor is needed. It is also insensitive to agent properties, for example agents with different load capacities or speed.

The other option is to use heterogeneous teams where different agents can have different policies. There can be fully-heterogeneous teams where each agent has a different policy or hybrid teams where groups of agents share the same policy. For heterogeneous teams there are two learning methods. One is team learning which uses a single population for all agents policies. Basically treating it as a big joint policy, this has the advantage that traditional evolutionary methods can be used. The disadvantage is that the search space (the set of all possible solutions) is too big and can not be efficiently explored by current methods for moderately complex problems.

The other method for heterogeneous team learning is concurrent learning, which uses a different population for each policy. Each population is evolved by its own EA and has no influence on the other populations. To reach a cooperative goal, the fitness function of each population rewards the agents for cooperative actions. This approach decomposes the search into different smaller search spaces which can be searched efficiently. The evolutionary variant of this is called cooperative co-evolution (CCEA).

3.3 Policy Implementation

Artificial Neural Networks (ANNs) are general function approximators which have a set of input neurons, hidden neurons, and out- put neurons [10]. The parameters which describe the state are connected to the inputs and the output neurons can be used to determine which actions should be done. For example, in an action-selection network the action corresponding to the highest output neuron activation is used.

The hidden layers apply functions to the outputs of the previous layer which simplify or reduce the data and search for patterns. With the right number of hidden layers and the correct weights ANNs can be an efficient way to implement a policy. If the topology (the layout of the hidden neurons) of a neural network is known and only the connection weights need to be optimized, standard EAs can be used.

If the correct network topology is not known, specialized EAs can also be employed to evolve the topology. NEAT is a popular algorithm for evolving ANNs [21]. There also is an extension of NEAT called HyperNEAT [20] which does not evolve the policy ANN directly evolves a Compositional Pattern Producing Network (CPPN) [19] which in turn generates the network used for the policy. This so-called indirect encoding has the advantages that the genotype is smaller which also reduces the search space.

3.4 Problems and Solutions

CCEA is not without its drawbacks. This chapter shows the important problems and how researchers dealt with them. In general, these problems only play a role if a higher number of agents is used and can be ignored in small systems.

Difficult learning dynamics. In a MAS learning case the fitness of an agent does not only depend on the agent alone. It depends on how good the individual performs with agents from the other populations. It can be seen as very fit if paired with specific collaborators, but very unfit when paired with other collaborators. This leads to policies that are good at cooperating with as many different collaborators as possible. It does not lead to the most optimal policies to solve the problem. [23] showed that certain variants of CCEA gravitate towards those sub-optimal solutions.

Gomes et al. [8] apply novelty-search to CCEA's. Novelty-search prioritizes behavioral diversity and novelty over the fitness function. This seems counter-intuitive at first, but the search for novel behaviors helps to avoid sub-optimal solutions. The novelty search is formulated as an optimization criterion and is combined with the traditional fitness function in a multi-objetive fashion. The authors demonstrate that their approach outperformed traditional CCEA's in multiple task setups.

Panait et al. [15] try to bias the search by assessing the fitness of an individual based on two components: its immediate reward, while interacting with individuals in the population, and a heuristic estimate for the reward it would have received, had it interacted with its optimal collaborators. This heuristic can be estimated in different ways for example the optimal collaborators can be the most successful ones known so far.

Another approach by Panait et al. [14] is to keep an archive of informative and diverse collaborators of past generations. Informative CCEA (iCCEA) tries to build a small archive of collaborators which can represent large parts of older generations. Each archive member helped at least one other individual to receive its highest reward and thus provides information about an interesting part of the search space.

Scalability. This concerns the sensitivity of learning algorithm towards the number of agents it can train and how the policies perform when agents are added to or removed from the system.

Gomes et al. [7] used a specialized CCEA they name Hyb-CCEA which uses the hybrid team composition. Hyb-CCEA monitors the behavioral similarity of the populations: if two populations behave very similar, they are merged into one population. If a divergence of behaviors is observable, the populations are split. The behavioral similarity is measured by Systematically derived behavior characterizations [6]. This adjusts the heterogeneity of system automatically. If the task can be solved with a less heterogeneous team Hyb-CCEA decreases the number of populations and trains the same number of agents with fewer learning processes.

D'Ambrosio et al. [5] show that the HyperNeat method of neuroevolution [20] can be applied to MAS with minimal effort. The new algorithm is simply called multi-agent HyperNEAT. It is based on the idea that in most real-world teams, the position in the the team is associated with a different role. They extend the CPPN (a neural network generator) with the ability to encode conceptual team positions. Now a single CPPN can generate behaviors for a whole team of heterogeneous agents. The CPPN can be trained like a single agent without losing the advantages of heterogeneous teams. This approach is independent of the number of agents to be trained and the authors showed that it can cope with adding or removing agents.

Reinvention. Concurrent learning relies on the assumption that the search space is decomposable into smaller search space which can be optimized independently. Most problems are not that easy to separate and the different agents will need shared skills. Training algorithm which ignore this have to reinvent those shared behaviors for each agent separately. This can be a time consuming process. Hyb-CCEA [7] solves the problem of reinvention since similar behaving agent populations are merged which avoids evolving similar solution in different populations.

Credit Assignement. To train each agent correctly, it needs feedback tailored to his performance. Generally the team can only be evaluated as a whole and a single agent does not get that feedback specific to him. In this case, a team can have some great members and some terrible ones and be as good as a team with all average team members. To reach the globally optimal policies for each agents every agent has to contribute and agents with unused potential can not be tolerated.

Tumer et al. [1] define two properties a good fitness function for each concurrent learner. It has to be aligned with the global fitness function of the whole MAS that means an increase in the agents fitness function also improves the global fitness function. It also has to be sensitive to the agent it evaluates and not the other agents. If these properties are not fulfilled the learning goal will be reached slowly or not at all. They show difference evaluation functions can be used to that effect. Those functions try to compute how the system would have performed if one agent had been removed.

There are also knowledge-based and trust-based multi-agent credit assignement methods[9, 13]. They are using computational models of knowledge and trust, respectively, to divide the reward among the agents. They employ a critic agent (not a part of the MAS) who keeps track of knowledge and trust values for each agent.

4 Discussion

This report gave an overview how machine learning can be used to train each agent in a MAS. After some general introductions to the related research fields we explained how to apply RL to MAS. Specifically, we looked at co-evolutionary methods which can cope with the large state and search space, typically encountered when dealing with MAS. We discussed the problems which arise with the use co-evolutionary methods and how they can be handled.

References

1. Agogino, A., Tumer, K.: Efficient Evaluation Functions for Evolving Coordination. Evolutionary Computation **16**(2) (6 2008) 257–288
2. Alpaydn, E.: Introduction to machine learning. Volume 1107. (2014)
3. Balaji, P.G., Srinivasan, D.: An introduction to multi-agent systems. Studies in Computational Intelligence **310** (2010) 1–27
4. Bergenti, F., Vargiu, E.: Multi-agent systems in the industry three notable cases in Italy. In: CEUR Workshop Proceedings. Volume 621. (2010)
5. D'Ambrosio, D.B., Stanley, K.O.: Scalable multiagent learning through indirect encoding of policy geometry. Evolutionary Intelligence **6**(1) (2013) 1–26
6. Gomes, J., Mariano, P., Christensen, A.: Systematic Derivation of Behaviour Characterisations in Evolutionary Robotics. In: Artificial Life 14: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems, The MIT Press (7 2014) 212–219
7. Gomes, J., Mariano, P., Christensen, A.L.: Cooperative Coevolution of Partially Heterogeneous Multiagent Systems. International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015) (Aamas) (2015) 297–305

8. Gomes, J., Mariano, P., Christensen, A.L.: Novelty-Driven Cooperative Coevolution. Evolutionary computation (12 2015) 1–33
9. Harati, A., Ahmadabadi, M.N., Araabi, B.N.: Knowledge-Based Multiagent Credit Assignment: A Study on Task Type and Critic Information. IEEE Systems Journal **1**(1) (9 2007) 55–67
10. Kriesel, D.: A Brief Introduction to Neural Networks. Retrieved August (2005) 244
11. Litman, T.: Autonomous Vehicle Implementation Predictions: Implications for Transport Planning. Transportation Research Board Annual Meeting **42**(January 2014) (2014) 36–42
12. Mitchell, T.M.: Machine Learning. Number 1. (1997)
13. Nazari, S., Shiri, M.E.: Trust-Based Multiagent Credit Assignment (TMCA). Springer International Publishing (2016) 335–349
14. Panait, L., Luke, S., Harrison, J.F.: Archive-based Cooperative Coevolutionary Algorithms. Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06 (2006) 345–352
15. Panait, L., Luke, S., Wiegand, R.P.: Biasing coevolutionary search for optimal multiagent behaviors. IEEE Transactions on Evolutionary Computation **10**(6) (2006) 629–644
16. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Upper Saddle River: Prentice hall (2010)
17. Shoham, Y., Leyton-brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. ReVision **54**(1-4) (2009) 513 p.
18. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587) (2016) 484–489
19. Stanley, K.O.: Compositional Pattern Producing Networks: A Novel Abstraction of Development. (2007)
20. Stanley, K.O., D 'ambrosio, D., Gauci, J.: A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks. Artificial Life journal **15**(2) (2009)
21. Stanley, K.O., Miikkulainen, R.: Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation **10**(2) (6 2002) 99–127
22. Sutton, R.S., Barto, a.G.: Reinforcement learning: an introduction. IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council **9**(5) (1998) 1054
23. Wiegand, R.P.: An analysis of cooperative coevolutionary algorithms (2004)