Extending World Models for Multi-Agent Reinforcement Learning in MALMÖ

Valliappa Chockalingam[1], Tegg Taekyong Sung[2], Feryal Behbahani[3],

Rishab Gargeya[4], Amlesh Sivanantham[5], Aleksandra Malysheva[6]

[1]University of Alberta, [2]Kwangwoon University, [3]Imperial College London,
[4]Stanford University, [5]University of Southern California, [6]JetBrains Research
[1]valliapp@ualberta.ca, [2]tegg89@gmail.com, [3]feryal.mp@gmail.com,
[4]rgargeya@stanford.edu, [5]sivanant@usc.edu, [6]malyshevasasha777@gmail.com

Abstract

Recent work in (deep) reinforcement learning has increasingly looked to develop better agents for multi-agent/multi-task scenarios as many successes have already been seen in the usual single-task single-agent setting. In this paper, we propose a solution for a recently released benchmark which tests agents in such scenarios, namely the MARLÖ competition. Following the 2018 Jeju Deep Learning Camp, we consider a combined approach based on various ideas generated during the camp as well as suggestions for building agents from recent research trends, similar to the methodology taken in developing Rainbow (Hessel et al. 2017). These choices include the following: using model-based agents which allows for planning/simulation and reduces computation costs when learning controllers, applying distributional reinforcement learning to reduce losses incurred from using mean estimators, considering curriculum learning for task selection when tasks differ in difficulty, and graph neural networks as an approach to communicate between agents. In this paper, we motivate each of these approaches and discuss a combined approach that we believe will fare well in the competition.

Introduction

Reinforcement Learning (RL) is a subfield of machine learning that studies how agents can learn to maximize total reward while acting in environments that provide minimal supervision through reward signals (Sutton, Barto, and others 1998). Generally, the formulation of RL assumes that the environment is modeled as a Markov Decision Process (MDP), given by the 5-tuple $\langle S, A, R, T, \gamma \rangle$ where S denotes the state space, A the action space, R the reward function, T the transition function, and γ the discount factor which controls how much agents favor short-term vs. long term rewards. The objective is usually to maximize the expected return $\mathbb{E}[G] = \sum_{t=1}^{\infty} \gamma^{t-1} r_t$, where r_t is the reward obtained at time t.

Often the MDP and the state space in particular is very large or complex, and, thus, using traditional table-lookup methods that store information per state can be very expensive or lead to poor generalization performance. This problem is generally tackled with function approximation where the states are first mapped onto a lower dimensional space and RL algorithms then work with the extracted features.

Deep Learning is a method to learn general functions using artificial neural networks and hence is a way towards non-linear function approximation (LeCun, Bengio, and Hinton 2015; Arulkumaran et al. 2017). Thus, combining Reinforcement Learning and Deep Learning (Deep RL) has become popular in recent years for training agents in complex environments. Minecraft, one such environment with a rich and highly complex state space, provides a good testbed to evaluate how various Deep RL agents compare. Moreover, since Minecraft allows for multiple tasks and agents, multi-task multi-agent scenarios can be simulated with ease. The recently proposed MARLÖ competition focuses on such scenarios, and in this paper we propose certain agents as good candidates to solve such scenarios.

Preliminaries

In this section, we outline some necessary background information that will be useful in explaining our proposed method and the motivation for various decisions. We first discuss a particular model-based algorithm that we base our agents on. Then, we consider the general RL landscape, and the recent interest in distributional RL. After that discussion, we talk about some important things to note about the MARLÖ competition in particular. Finally, we discuss NerveNet and Graph Neural Networks which provide for an approach to think about when considering multi-agent communication.

When designing reinforcement learning algorithms, there is a choice between model-free algorithms and model-based algorithms. Model-free approaches only focus on the problem of how to act so as to maximize reward. On the other hand, model based approaches also try to understand how the "world" works by learning the underlying MDP of the environment. A recent such approach that has shown promise and which we propose to use is World Models.

World Models

World Models (Ha and Schmidhuber 2018) looked at learning a compressed spatial and temporal model of an environment in an unsupervised manner and using the learned features to train a very simple and compact policy to solve the task at hand. The authors were able to achieve this using a multi-step training procedure:

1. Collect rollouts from environment (using a random or other simple policy π), i.e., store episode trajectories of the form $s_0, a_0, r_1, s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T, s_T$ where $a_t \sim \pi(s_t)$.
2. Train a variational autoencoder, V, to learn a latent vector representation $z \in \mathbb{R}^m$ given the higher dimensional states collected during step 1.
3. Train a Mixture Density Network-Recurrent Neural Network, M, to model $P(z_{t+1}|z_t, h_t, a_t)$ where h_t is the hidden state of the LSTM used in the MDN-RNN.
4. Train a controller C which learns what action a_t to take using z_t and h_t, like so: $a_t = W_c[z_t, h_t] + b_c$.

While V and M can simply be trained through supervised learning, C can be trained in various ways. However, since the controller portion of the setup is very small (on the order of a few hundred parameters in the experiments used by the authors), training using an evolutionary technique, CMA-ES (Covariance Matrix Adaptation Evolution Strategy (Hansen and Ostermeier 2001)) is possible.

RL Algorithms

When designing RL agents, another common choice to be made is whether the agents should directly learn in policy space (how to act in different states) or learn value functions that define the "goodness" of states or state-action pairs in terms of expected return. In the value-based paradigm, Q-Learning (Watkins and Dayan 1992) and Deep Q-Networks (Mnih et al. 2015) have been popular where, following the Bellman equations, state action pairs, $Q(s_t, a_t)$, are updated towards a target that is generally more accurate, $r_t + \gamma max_a Q(s_{t+1}, a)$. In the policy-based paradigm, policy gradients have been commonly used with some changes to reduce variance. A usual starting point is the Monte-Carlo policy gradient update, REINFORCE (Williams 1992), which performs gradient ascent on the parameters of the policy using sampled returns: $\theta \leftarrow \theta + \nabla_\theta \log \theta_\pi(s_t, a_t) G_t$. Finally, actor-critic methods lie in between the two, where an actor learns and looks to improve a policy and a critic evaluates the policy and, by replacing sampled returns for example, can reduce variance.

Recently, there have been questions about the general approach taken in RL. Namely, the observation is that most prior RL algorithms estimate the average expected return and use these estimates when choosing actions. Distributional RL (Morimura et al. 2010; Bellemare, Dabney, and Munos 2017), has posed an alternative whereby $Z(s_t, a_t)$, the return distribution for state-action pair (s_t, a_t), is updated towards $r_t + \gamma max_a Z(s_{t+1}, a_t)$. Action selection can then be done in an expectation sense or using, for example, risk sensitive approaches where actions with lower variance may be favored.

MARLÖ Competition

One major aspect of the competition is that multiple agents will be involved. With teams, cooperative tasks and competitive tasks become important subsets of tasks to consider. With regard to terminology in fact, the competition specification makes an important distinction between tasks and games by defining a game as a collection of tasks of varying difficulty and level layouts. The competition is structured such that we have access to some publicly available tasks for every game, but some tasks are private and to be used later for evaluation.

Graph neural networks

NerveNet (Wang et al. 2018) is a recently proposed architecture for continuous control where the agent was assumed to have multiple physical parts which can be represented as a graph. Graph Neural Networks (GNNs) were used as they posses *relational inductive biases* and are invariant to node and edge permutations as entities and their relations are represented as sets (Battaglia et al. 2018; Mitchell 1980).

The updates of GNNs begin with propagation of messages to neighboring nodes. The feature vector of the current state for each node u at propagation step 0 is denoted h_u^0. After each node passes messages to neighboring nodes, it aggregates messages it receives from its neighborhood, then updates its feature vector and outputs a vector combining the information from the current and the aggregated messages. A policy network parameterized by the graph can learn what actions to take for each part. This process can be described using the equations below:

$$m_{(u,v)}^t = M_P^v(h_u^t) \text{ where } v \in \mathcal{N}_{out}(u) \quad (1)$$

$$A_v^t = M_A^v(\{m_{(u,v)}^t | u \in N_{in}(v)\}) \quad (2)$$

$$h_v^{t+1} = U^v(h_v^t, A_v^t) \quad (3)$$

$$\hat{y} = O(\{h_v^T \mid v \in G\}) \quad (4)$$

where $\mathcal{N}_{in}(\cdot)$ and $\mathcal{N}_{out}(\cdot)$ represent the in- and out-neighborhoods respectively, $\hat{y}$ represents the output of the GNN and M_P^v, M_A, U^v and O^v may be neural networks and parameter sharing can be applied as necessary. For example, a node u can use the same network M_P for message passing to all nodes on its out-neighborhood $v \in \mathcal{N}_{out}(u)$ or use separate ones for messages to each of these nodes M_P^v and hence the additional superscript v.

Methods

We now consider a few different agents as a combination of some of the ideas explained previously. The motivation for these agents and design choices therein will be described in the following section.

The simplest agent we propose is a multi-agent variation of World Models. Firstly, we note that we assume V and M are shared. Alternatively, they can be trained separately for each agent. Regardless, in the explanations that follow, we will only discuss training the controllers. Owing to the small number of parameters in the controllers of world models, we can view them as being "policy embeddings" which dictate how each agent acts in the environment. Assuming that each agent's controller is trained separately, let the parameters of the controllers learned be denoted by C_θ^i for

agent i where C^i_θ corresponds to the controller parameters $[W_c, b_c]$ for agent i's controller C^i. If each agent were to act independently, then it would just forward the latent vector embedding of its current state and hidden state of the LSTM from the MDN-RNN, say $[z^i_t, h^i_t]$, through its controller C^i to get the action it should take. However, in a multi-agent setting, we are often interested in maximizing total reward for a team of agents (particularly so in co-operative settings.) So, we define a meta-controller as a controller which takes in $[z^1_t, h^1_t, ..., z^n_t, h^n_t]$ and outputs $[\pi^1_{a_0}, ..., \pi^1_{a_m}, ..., \pi^n_{a_0}, ..., \pi^n_{a_m}]$ assuming n agents are "under" the teams control, there are m actions and where $\pi^i_{a_j}$ is the probability of agent i taking action a_j.

The next agent we consider is one where the controllers are trained separately, but a Graph Neural Network style approach is used, reducing the number of parameters and encouraging communication of shared knowledge. To do this, we first concatenate the hidden state vectors h^i_t for each agent i, say $H = [h^1_t, ..., h^n_t]$. We then use this matrix to construct a graph G_t. At the next timestep, we can do the same to get a graph G_{t+1}. Now, the graph is to represent the beliefs of how the world works and how the agents relate to each other (which it can do so given the information from the memory module from each agent). So, we can expect that if graphs are learned appropriately, we can infer the graph at time $t+1$, given the graph and the actions taken by each agent $[a^1_t, ..., a^n_t]$ at time t. Thus, we can use supervised learning to train the graph construction. The actual construction process can be done in many ways, but we propose using cosine-distance row- and column-wise to get how entities in the environment and agents relate to each other. With these graphs serving as a proxy for useful information about the environment and agents, these graphs can serve as useful input to each agent's controller.

Past work has looked at action-conditional next state prediction, but has often made the implicit assumption that the environment is deterministic. Since many environments are stochastic and world models itself consists of a parameter τ which controls the environments stochasticity, we can look to previous work in distributional RL to consider a better approach to generating and using next state predictions. Notably, when there are many possible next states, we can integrate over them by the probability to get an "averaged" next state vector. The probability distribution over next states, $P(z_{t+1}|a_t, z_t, h_t)$, is modeled by M in the world models approach. So, instead of sampling from the mixture density model and using a single sample, we can easily sample repeatedly or even in some uniform interval of τ to get multiple possible next states with various probabilities of occurring. Averaging them could give us a more useful z and this is an approach we wish to study. Using all of the sampled z's separately, as in (Buesing et al. 2018) is also possible.

While the evolutionary approach allows us to be free from the problem of needing to define how to use the rewards, we can consider RL approaches as well. Value based approaches help in long term planning and hence we propose using actor-critic methods where the critic can use distributional RL. Given multiple possible next states for the current state (as explained in the previous paragraph), we can even train a more accurate value estimation network for the distribution of expected returns $Z(s, a)$.

Lastly, in a multi-task setting, there may be tasks of varying types and difficulties. Curriculum learning is a common approach as the agent can slowly learn tasks of increasing difficulty without which some tasks may be very hard to solve. We propose having a teacher suggesting tasks using a bandit algorithm, EXP3 (Exponential-weight algorithm for Exploration and Exploitation) that keeps track of how each arm (or task in this case) fares. The general pattern we wish to see here is that as agents show signs of improving in one task (say, in an average reward sense), we select that task more until it is essentially solved and then we move on to another task that is next up in the hierarchy of tasks based on difficulty. Since MARLÖ has the concept of games and tasks, we can use a hierarchical approach whereby we first choose the game and then the task to train on. Alternatively, the teacher can select from all the available tasks directly.

Motivation

In this section, we explain the reasoning behind some of our design decisions and choices.

While model-free approaches such as Deep Q-Networks (Mnih et al. 2015) and A3C (Mnih et al. 2016) and their later variants have been popular and shown promising results in the past, a primary reason for their success has been that they are easier to train. Model-based approaches on the other hand are generally more computationally intensive as there are additional steps to model the MDP. However, there are many benefits to learning models. Importantly, running experiments on even simulated environments can be very expensive as we move towards more complex environments. Minecraft is a modern game, built many years past Atari 2600 games, and hence, as one would expect, is a much more computationally intensive environment. Learning models allows for agents to learn without needing to interact with the environment at hand. We can even combine learning from the environment and learning the model with planning using the model following the Dyna architecture, i.e., learning from "dreams."

Now, while using models as a proxy for complex environments is an obvious advantage to using models. There are still many other interesting things to note about using models and model-based learning in general. One primary argument against model-based learning is that getting good models is very difficult and poor models can hinder learning. However, in both World Models and I2A, the authors note that poor models can still in fact help learning. While no solid reason for this is known, we hypothesize that this may be due to some state-aggregation like effects or where the MDP approximated has some strong correlation to the true underlying MDP. In other words, the model is at least able to learn some useful clustering of the states or learn reward and transition functions that are similar to the true reward and transition functions but differ in small ways.

Opponent modeling is another aspect that past research has considered. However, with limited access to opponent states and policies as well partial observability, it can be a very

hard problem. Models allow opponents to just be modeled as part of the environment and thus sidestep the need to incorporate opponent modeling in agents. However, there is still a question of how to setup opponent policies while learning the model. If setting opponent policies randomly does not work, we can use self-play with some noisy version of learned policies.

World models in particular is appealing for a few concrete reasons. Firstly, Minecraft is partially-observable, the hidden state of the LSTM h in the MDN-RNN module could be very useful in making the POMDP more like an MDP. The hidden state encodes information about past observations and actions taken. Hence, h combined with the current state encoding z represents a new state that encodes the history as well.

Secondly, as the tasks in the MARLÖ competition involve multi-agent multi-task scenarios, we need to think about how to train the agents. With world models, we can share the V, the state feature-extraction module, across the agents and games. The memory module M can also be shared with some consideration for how to train it given that agents may have differing roles. The main point however is that world models is structured in such a way that we can share many modules between agents, reducing computational costs and introducing some useful priors. While such sharing of state representations might be possible in model-free algorithms, the learned representations can be quite different given how the architectures are trained (supervised learning vs. RL and hence with different loss functions.)

As a related note, even the structure of the controller, C, is amenable to the multi-agent setting. Since most of the complexity in terms of number of parameters is in the V and M models, C is very small. With multiple agents, we can train a meta-controller with a lot fewer parameters than if we were using separate model-free architectures for each.

Finally, we discuss some motivations behind taking a certain approach to training the controller, C, using RL. Naively using CMA-ES in a multi-task cooperative task setting with each controller being trained separately would lead to an exponential blow-up whereby we need to perform a combinatorial search to find good parameters for each agent in order to maximize "global" reward. Also, notably, while evolutionary algorithms and the controller architecture is suited to policy search, complex tasks often require long term planning and hence value functions are useful. This is the same dichotomy seen in continuous control tasks where policy based methods generally perform much better while in games with some amount of planning necessary, like Atari 2600 games or Go, value based methods have generally done much better. Hence, we believe we should use parameter sharing or metacontrollers (to deal with the exponential blow-up) and Actor-Critic methods given how Minecraft combines the need for impulsive quick-actions and long term planning.

Recently, we have also seen that deep RL seems to benefit from learning distributions even when acting in an expected sense. While it is still unclear why, one reasonable argument for this is that there seems to be some bias induced in the deep RL setting when collapsing distributions to their means. And, indeed, recent work has shown that distributional RL doesn't seem to be of much help in the linear function approximation case (Lyle and Bellemare 2018). Given that we pushed for using actor-critic above, a natural choice for the critic is then to see how predicting return distributions could affect performance, as done in Rainbow (Hessel et al. 2017). Additionally, given the Actor-Critic approach and that we have access to a simulator as well, counterfactual policy gradients (Foerster et al. 2017) can also be used.

Related work

There have been multiple approaches recently in the area of multi-task multi-agent scenarios and, particularly in the Minecraft platform, Malmö, such as (Xiong et al. 2018). They propose the novel model *HogRider* to solve the Pig Chase task. The objective of the task is to chase randomly spawned pigs in a 9×9 grid with collaboration. Since the pigs are moving after the task is initialized, the environment is non-stationary, and hence can be viewed as a multi-agent team-based scenario.

The multi-agent problem of learning from a single scalar reward signal has generally been quite difficult to approach but has recently been looked at. Value Decomposition Networks (Sunehag et al. 2017) look to decompose the team value function into agent value functions based and see that the method fares better when weight sharing and information channels are used. Again, this is in fact a strong argument for using a model-based approach whereby many of the feature representations are shared across agents.

A recent retro game contest held by OpenAI has tackled various scenarios similar to the MARLÖ contest (Gray and Brown 2018) and has further brought attention to the need for adaptive exploration and meta-learning in RL.

Discussion and Future Work

We believe this study is particularly useful and well-motivated because most of the challenges faced by agents evaluated in a multi-task/multi-agent settings are with understanding of the interactions between agents and their respective policies. Our approach attempts to alleviate these difficulties with a Graph Neural Network and Policy Embedding approach. We further can address issues of uncertainty and model interpretability using distributional RL. Finally, we utilize curriculum learning to address agents' need to balance performance on multiple tasks of varying difficulties, intelligently deciding which tasks to train on and alleviating issues with catastrophic forgetting .

The immediate next steps are to continue implementations of the above ideas into an agent and evaluate its performance in the MARLÖ competition compared to other common benchmark agents. We also wish to look into how to construct simulated environments with various types of opponents and do ablation tests to see which of the ideas detailed above lead to the most improvement. We will keep up-to-date our progress and thoughts here at this webpage: https://sites.google.com/view/rainbowmarlo/.

References

Arulkumaran, K.; Deisenroth, M. P.; Brundage, M.; and Bharath, A. A. 2017. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*.

Battaglia, P. W.; Hamrick, J. B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Bellemare, M. G.; Dabney, W.; and Munos, R. 2017. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*.

Buesing, L.; Weber, T.; Racaniere, S.; Eslami, S.; Rezende, D.; Reichert, D. P.; Viola, F.; Besse, F.; Gregor, K.; Hassabis, D.; et al. 2018. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*.

Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; and Whiteson, S. 2017. Counterfactual Multi-Agent Policy Gradients. *ArXiv e-prints*.

Gray, J., and Brown, T. 2018. Retro contest.

Ha, D., and Schmidhuber, J. 2018. World models. *arXiv preprint arXiv:1803.10122*.

Hansen, N., and Ostermeier, A. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9(2):159–195.

Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2017. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*.

LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *nature* 521(7553):436.

Lyle, C., and Bellemare, M. G. 2018. As expected? an analysis of distributional reinforcement learning.

Mitchell, T. M. 1980. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ. New Jersey.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

Morimura, T.; Sugiyama, M.; Kashima, H.; Hachiya, H.; and Tanaka, T. 2010. Nonparametric return distribution approximation for reinforcement learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 799–806.

Sunehag, P.; Lever, G.; Gruslys, A.; Czarnecki, W. M.; Zambaldi, V.; Jaderberg, M.; Lanctot, M.; Sonnerat, N.; Leibo, J. Z.; Tuyls, K.; and Graepel, T. 2017. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *ArXiv e-prints*.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Reinforcement learning: An introduction*. MIT press.

Wang, T.; Liao, R.; Ba, J.; and Fidler, S. 2018. Nervenet: Learning structured policy with graph neural networks.

Watkins, C. J., and Dayan, P. 1992. Q-learning. *Machine learning* 8(3-4):279–292.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Xiong, Y.; Chen, H.; Zhao, M.; and An, B. 2018. Hogrider: Champion agent of microsoft malmo collaborative ai challenge.