Reinforcement Learning for Computer Generated Forces using Open-Source Software

Johan Källström
Saab and Linköping University
Linköping, Sweden
Johan.Kallstrom@saabgroup.com

Fredrik Heintz
Linköping University
Linköping, Sweden
Fredrik.Heintz@liu.se

ABSTRACT

The creation of behavior models for computer generated forces (CGF) is a challenging and time-consuming task, which often requires expertise in programming of complex artificial intelligence algorithms. This makes it difficult for a subject matter expert with knowledge about the application domain and the training goals to build relevant scenarios and keep the training system in pace with training needs. In recent years, machine learning has shown promise as a method for building advanced decision-making models for synthetic agents. Such agents have been able to beat human champions in complex games such as poker, Go and StarCraft. There is reason to believe that similar achievements are possible in the domain of military simulation. However, in order to efficiently apply these techniques, it is important to have access to the right tools, as well as knowledge about the capabilities and limitations of algorithms.

This paper discusses efficient applications of deep reinforcement learning, a machine learning technique that allows synthetic agents to learn how to achieve their goals by interacting with their environment. We begin by giving an overview of available open-source frameworks for deep reinforcement learning, as well as libraries with reference implementations of state-of-the art algorithms. We then present an example of how these resources were used to build a reinforcement learning environment for a CGF software intended to support training of fighter pilots. Finally, based on our exploratory experiments in the presented environment, we discuss opportunities and challenges related to the application of reinforcement learning techniques in the domain of air combat training systems, with the aim to efficiently construct high quality behavior models for computer generated forces.

ABOUT THE AUTHORS

Johan Källström is a PhD student at Saab and Linköping University, Sweden. His research focus is deep reinforcement learning for computer generated forces in the context of LVC simulation for training of fighter pilots. He has a background in modeling and simulation for flight training systems at Saab and holds a Master of Science in Applied Physics and Electrical Engineering from Linköping University.

Dr. Fredrik Heintz is an Associate Professor of Computer Science at Linköping University, Sweden. His research focus is artificial intelligence especially autonomous systems, stream reasoning and the intersection between knowledge representation and machine learning. He is the Director of the Graduate School for the Wallenberg AI, Autonomous Systems and Software Program (WASP), the President of the Swedish AI Society and a member of the European Commission High-Level Expert Group on AI. He is also very active in education activities both at the university level and in promoting AI, computer science and computational thinking in primary, secondary and professional education.

Reinforcement Learning for Computer Generated Forces using Open-Source Software

Johan Källström
Saab and Linköping University
Linköping, Sweden
Johan.Kallstrom@saabgroup.com

Fredrik Heintz
Linköping University
Linköping, Sweden
Fredrik.Heintz@liu.se

INTRODUCTION

Pilots of modern fighter aircraft must operate complex aircraft systems, such as weapon, sensor, and electronic warfare systems, to achieve their mission goals. They must also operate in highly complex, contested environments, populated by a large number of allies and adversaries. Providing efficient and effective training solutions for the pilots is challenging. In the Live setting, lack of air space, limited access to systems representative of the enemy's, as well as the need to limit costs impose restrictions on training scenarios. It is often not possible to provide scenarios with the contents and density desired. Live, Virtual and Constructive simulation aims to address these challenges, by moving some contents from the Live setting to the Virtual and Constructive domains. Especially the Constructive domain has high potential, since it can provide vast amounts of synthetic actors at a low operational cost. However, in order to provide a challenging and engaging training environment for the trainee, it is important that the synthetic actors provided by constructive simulations are controlled by realistic behavior models, and require little manual intervention from human scenario operators.

The construction of high-quality behavior models is perhaps the most challenging task in constructing Computer Generated Forces (CGF). This is especially true for the end-users of training systems, who may not have the special skills and experience required to build the models. Instead, they may require help from software engineers, who may not always be available. Transferring domain knowledge from training instructors to engineers is also difficult, and the results may not be as desired until after several iterations. As a result, the development of training contents may lag behind training needs as aircraft systems or the operational environment change. To make it possible for an average user to create simulation contents easily, better tools are required. Instead of demanding that users program the behavior of CGF entities explicitly, it would be desirable to use a declarative approach. It would then be possible to specify the goals and characteristics of synthetic actors using the domain language known by instructors as well as trainees.

In recent years, there have been great advances in the field of Machine Learning. Models built by learning algorithms have reached human or even super-human performance in domains such as image classification, natural language processing and sequential decision-making in games, such as poker, Go and StarCraft. There is hope that these new techniques may fundamentally change how we build intelligent systems. One branch of machine learning is Reinforcement Learning, which is a technique used for developing decision-making policies from interaction with an environment. In this paper, we discuss applications of reinforcement learning in the domain of military simulation. Our own focus is on training systems intended for training of fighter pilots, but the requirements for other use-cases are quite similar. The main goal of this paper is to introduce the reader to the concepts of reinforcement learning, open-source software available to support this technique, and examples of how to apply it in our application domain. We hope that this work can help others in the community get started with experiments.

The remainder of this paper is structured as follows: We first give a short introduction to reinforcement learning and important algorithms developed in recent years, followed by an overview of open-source software used for deep reinforcement learning. We then present an example of how CGF software can be integrated in an infrastructure for deep reinforcement learning, based on our own approach. Finally, we discuss some challenges related to applications of reinforcement learning in the domain of air combat simulation, as well as techniques that could be used to address these challenges.

BACKGROUND

Machine Learning

In the domain of Machine Learning, the aim is to develop algorithms that allow machines to solve problems through inference based on data, instead of explicitly programming these machines to perform a specific task. There are three major paradigms of machine learning: Supervised Learning, Unsupervised Learning and Reinforcement Learning. Supervised Learning techniques are used to build models for classification or regression, based on input-output pairs of data samples. Unsupervised Learning techniques are used to find patterns in data, e.g. clustering of data samples that are similar according to some metric. Reinforcement Learning techniques allow a decision-making agent to learn a policy, i.e. a mechanism for sequential decision-making, by interacting with its environment, so called trial-and-error learning (Sutton & Barto, 2018). All three of these techniques are highly relevant in the military domain, for simulations as well as decision-making components in live systems, and they may also be combined to improve the overall system performance. In this paper, we focus on Reinforcement Learning.

Reinforcement Learning

In reinforcement learning, an agent must explore its environment to learn how to achieve its goals. The agent is guided by a reward signal, which rewards desired behavior, and punishes undesired behavior. The problem is typically modelled using a Markov Decision Process (MDP), which is defined as a tuple (S, A, T, R, γ), where:

- S: Is the set of states of the process
- A: Is the set of actions of the process
- T: Is the (possibly stochastic) transition dynamics of the process, which govern which state the agent will end up in after executing an action
- R: Is the reward function of the process
- γ: Is the discount factor ($\gamma \in [0,1]$), indicating the importance of immediate and future reward respectively

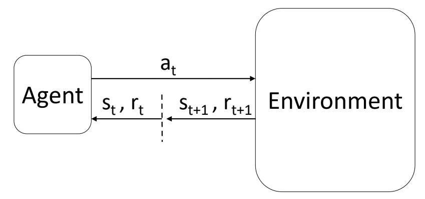

Figure 1: Agent interacting with its environment.

In each step of the MDP, the agent will select and execute an action based on its perception of the current state of the environment, and then observe the resulting new state of the environment, and the reward received, as illustrated in Figure 1. The goal of the agent is to maximize its expected future return R_t when starting in state s:

$$E[R_t | s_0 = s] = V_\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right] \tag{1}$$

Here $V_\pi(s)$ is the state value function, which specifies the expected value of being in state s and then following policy π. Similarly, the value of taking action a in state s and then following policy π is given by the state-action value function $Q_\pi(s, a)$:

$$Q_\pi(s,a) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\right] \tag{2}$$

A unique property of reinforcement learning problems is the necessary trade-off between exploration and exploitation. Exploration means performing exploratory actions to learn more about the environment, and possibly improving the long-term policy. Exploitation means using the current knowledge about the environment, and the policy learned so far, to gather as much reward as possible.

Reinforcement learning algorithms can be divided into value-based and policy-based methods. Value-based methods aim to learn a value function, most commonly the Q function, and then use it to guide the decision-making of the agent. Policy-based methods aim to directly learn a policy, without learning a value function. The two methods can also be combined in actor-critic methods. These methods aim to learn a value function (the critic), which guides updates of the actor's policy. Algorithms can be further divided into on-policy and off-policy methods. On-policy methods learn a policy while using that same policy for exploration, while off-policy methods learn a policy while using a different policy for exploration. An example of an off-policy, value-based reinforcement learning algorithm is Q-learning, which aims to learn the Q function. The update step of the Q-learning algorithm is defined as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\big(r_t + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big) \tag{3}$$

Here $\alpha \in (0,1]$ is the learning rate, which determines the size of the update. Exploration in Q-learning is usually handled through ε-greedy selection of actions. This means that with probability ε, a random action is selected, otherwise the action with highest value according to the current Q function is selected. Often an exploration schedule is used, with ε being large in the beginning of training, and then successively decreased to zero or some small value.

For simple problems, with low-dimensional state and action spaces, the policy of an agent can be represented by a table, e.g. corresponding to the state-action values of the Q function. However, real-world problems are typically characterized by complex state and action spaces, including continuous states and actions. For these environments the tabular approach to reinforcement learning is not viable. Instead, an approximation of the value function or the policy must be used. The most common type of such a function approximation is a neural network. An early, important work using deep neural networks for function approximation in the context of reinforcement learning was the Deep Q-Network algorithm (DQN), presented by (Mnih et al., 2015). This approach combined a memory of previous experiences in the environment with a target network, updated less frequently than the policy network, to stabilize learning in complex environments. One limitation with DQN is that it does not handle continuous actions. This limitation was addressed by (Silver et al., 2014) and (Lillicrap et al., 2016), who developed the Deterministic Policy Gradient (DPG) and Deep Deterministic Policy Gradient (DDPG) algorithms. These are actor-critic algorithms that can learn deterministic policies for problems with continuous action domains.

One challenge with reinforcement learning is that large amounts of data samples are needed for the agent to explore the environment and find an efficient policy. For complex problems learning times may become very long. One way to reduce the learning time is to use several worker threads in parallel, where agents explore the environment and generate data samples for learning. (Mnih et al., 2016) studied asynchronous gradient descent for training of deep neural networks. They presented asynchronous versions of four reinforcement learning algorithms, including the Asynchronous Advantage Actor-Critic algorithm (A3C). (Schulman et al., 2017) presented another approach for efficient learning, called Proximal Policy Optimization (PPO). This approach makes it possible to perform several updates of the neural network using one collected batch of data.

OPEN-SOURCE SOFTWARE FOR DEEP REINFORCEMENT LEARNING

Programming Frameworks for Deep Reinforcement Learning

The machine learning community has embraced the principles of open-source software. In recent years, several high-quality frameworks have been made available as open-source projects, such as Caffe, Caffe2, Chainer, Microsoft Cognitive Toolkit, MXNet, PyTorch, TensorFlow and Theano. These frameworks provide APIs for defining deep learning models, as well as support for GPU accelerated training of the models. The two most popular frameworks in the reinforcement learning community are TensorFlow and PyTorch. TensorFlow has been around slightly longer, and still has the largest number of users, even though PyTorch has been growing in popularity quickly. Consequently, there is more learning material available for TensorFlow, such as books, online tutorials, and example models, e.g. implementations of deep reinforcement learning agents. It may take some time to get familiar with all the features of a full-fledged deep learning framework, and for this reason simplified APIs have been developed for some of them. The high-level APIs hide some of the low-level details of the computation graph, and allow users to quickly define and train common types of models. One popular API is Keras, which can use the frameworks TensorFlow, Theano and Microsoft Cognitive Toolkit as computation backends. With Keras it is possible to define and train a deep learning model with only a few lines of code. Other examples of high-level interfaces are Skorch for PyTorch and Gluon for MXNet. Built on top of the general-purpose deep learning frameworks, there are frameworks for deep reinforcement learning. These frameworks provide common components that can be used as building blocks for constructing and training agent behavior models. They also provide libraries with reference implementations for many popular algorithms. Some prominent frameworks are DeepMind TRFL, OpenAI Baselines, TensorForce, KerasRL, RayRLLIB, and ChainerRL.

Environments for Deep Reinforcement Learning

In this paper, we assume that the reader would ultimately be interested in using a CGF software to represent the environment in the reinforcement learning problem formulation. However, when doing initial experiments with algorithms, it may be useful to use other, simpler environments, e.g. due to the long simulation times associated with high-fidelity simulation models. One popular open-source library with reinforcement learning environments is the OpenAI Gym. This library contains a set of reinforcement learning benchmarks, and was created to make it easier to compare the performance of reinforcement learning algorithms. All the environments implement the same, generic interface, which specifies properties of the state and action spaces of the environment. This makes it possible to easily evaluate a learning algorithm on several different tasks. For building your own, simple environments, a gridworld simulation engine may be useful. Gridworlds have a discrete state space in the form of a grid, and they are often used to demonstrate reinforcement learning algorithms in literature. There are several open-source gridworld engines available, such as DeepMind's pycolab and Facebook's MazeBaze.

A CGF LEARNING ENVIRONMENT

For our experiments we construct reinforcement learning environments as Python modules that implement the OpenAI Gym interface, since this interface is supported by many of the available implementations of reinforcement learning algorithms. The structure of an environment is illustrated in Figure 2. Most of the environment's functionality is implemented in the EnvironmentCore class. This class communicates with a simulation process, running locally or on a remote computer, through the SimulationInterface, to transfer observations and actions between entities in the simulation and reinforcement learning agents that control them. The SimulationInterface is also used to load simulation scenarios in the CGF software.

Communication between the simulation and the environment module is implemented using ZeroMQ, an open-source, lightweight messaging middleware, with bindings for many programming languages, including C++ and Python. ZeroMQ makes it easy to implement several popular messaging patterns, such as request-reply, publish-subscribe and push-pull. Messages are specified using Google protocol buffers, which is a language-neutral and platform-neutral mechanism for serialization of structured data. A simple protocol language is used to create message specifications, which can then be compiled to source code in a wide variety of programming languages, including C++ and Python.

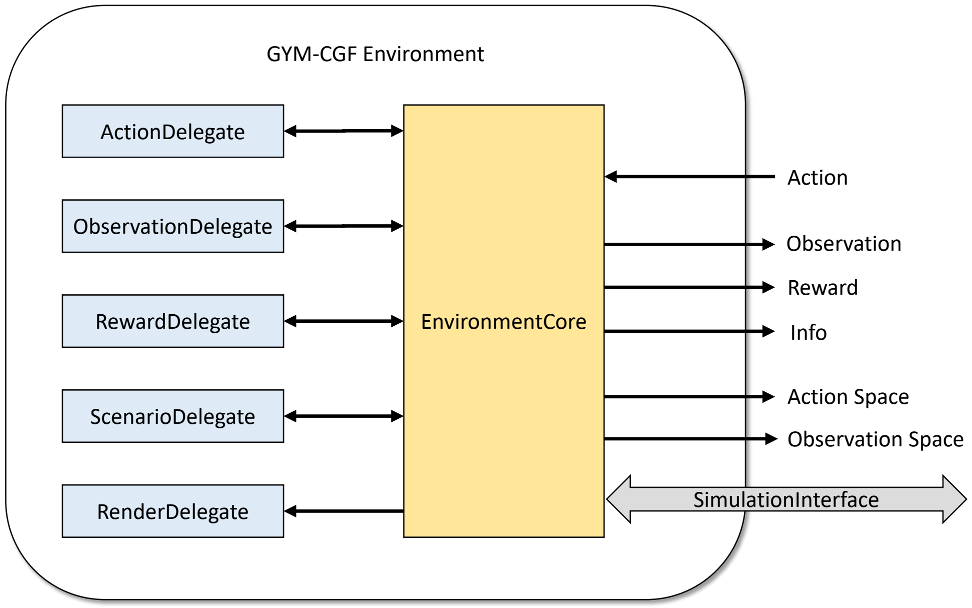

Figure 2: Components of a GYM-CGF environment.

To configure a specific environment, a number of delegate objects are used:

- ActionDelegate: The ActionDelegate specifies the action space of the environment (as one of the space definitions available in OpenAI Gym). During execution, it takes as input an action from this space and converts it to an ActionRequest message, which can then be sent to an entity in the simulation by the EnvironmentCore.
- ObservationDelegate: The ObservationDelegate specifies the observation space of the environment (as one of the space definitions available in OpenAI Gym). During execution, it takes as input a StateUpdate message from an entity in the simulation and converts it to a state observation from the observation space, which can then be presented to the agent.
- RewardDelegate: The RewardDelegate takes as input a state observation and calculates a scalar reward signal, which can then be presented to the agent.
- ScenarioDelegate: The ScenarioDelegate manages the scenario to be simulated, including termination criteria. For each episode during training, the delegate adjusts the scenario contents as needed and generates a SimulationRequest message, which can be sent to the simulation by the EnvironmentCore.
- RenderDelegate: The RenderDelegate renders a view of the current state of the simulated scenario. This can be useful for debugging. We implemented a simple rendering of a map using the Python Matplotlib and Basemap libraries.

DEEP REINFORCEMENT LEARNING IN THE DOMAIN OF AIR COMBAT SIMULATION

In our experimentation with deep reinforcement learning in the domain of air combat simulation, we have identified several challenges, which are typically not present in many of the simpler benchmark environments for reinforcement learning. The state and action spaces are high-dimensional and complex, making it difficult for the agent to learn important state features and suitable strategies for decision-making. For instance, in many scenarios the environment is only partially observable, e.g. due to limitations in sensors or effects of electronic warfare. Furthermore, in most scenarios agents do not act alone, but must instead cooperate with allies while competing with enemies to reach their goals. To handle long-term as well as short-term goals, decision-making at different timescales may be required. Rewards representing the most important goals are typically delayed and sparse, for instance given at the end of the scenario if the agent was victorious, making it difficult to assign credit to the right actions. It is also possible that the goals of agents will vary among runs of the simulation, depending on training needs. For instance, we may want to adjust the difficulty level of the simulation to fit the proficiency of the trainee. Finally, since running high-fidelity simulations is computationally expensive, it is important to make the learning process as sample efficient as possible. In the sections below, we discuss some techniques that can be used to address these challenges.

Exploration with Intrinsic Motivation

In environments that require long sequences of actions to reach states that provide the agent with a reward, such as military simulation scenarios, algorithms that use simple schemes for exploration, e.g. DQN with ε-greedy exploration, may be unable to find an effective policy. For instance, standard DQN agents struggle to find the way out of the first room in Montezuma's Revenge, a video game from the Arcade Learning Environment (Bellemare et al., 2013). In this room, illustrated in Figure 3, the agent must collect a key, and then climb to the top of the screen to exit one of the doors. The first reward is not given until the key has been collected. One way to address this type of challenge is to augment the environment's reward signal with a reward that is intrinsic to the agent. For instance, (Bellemare et al., 2016) proposed to add an exploration bonus based on events that provide information gain, which greatly improved DQN's performance on Montezuma's Revenge. Similarly, (Jaderberg et al., 2017) used pseudo-rewards awarded to agents for performing auxiliary tasks to promote learning in the absence of extrinsic rewards. The auxiliary tasks were related to control and reward prediction, and helped train the policy and value functions of the agent. The approach also allowed the agent to learn important aspects of the main task, and improved the performance of an A3C-based agent on several challenging environments.

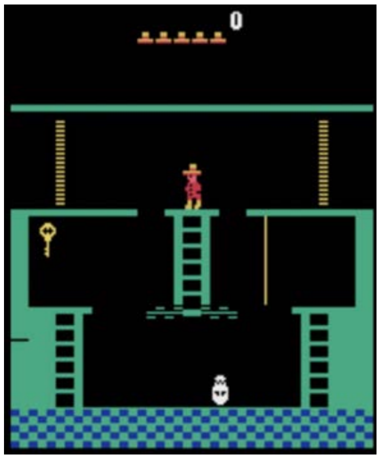

Figure 3: Montezuma's Revenge.

Reinforcement Learning with Transfer Learning and Curriculum Learning

To efficiently learn complicated tasks, it may be beneficial to break them down into a sequence of subtasks. This approach is adopted in curriculum learning, where the agent is faced with increasingly difficult tasks. With a properly designed curriculum, the policy learned in one task can be transferred to the next, and thus speed-up the overall learning. Ideally, the curriculum should be automatically generated, e.g. by observing the performance of the agent for each subtask. One form of curriculum is to successively change the simulated scenario, to make it more challenging for the agent. An example is given in Figure 4, where the agent should navigate to a target, and additional hostile air defense systems are added in each lesson in the curriculum. Another approach, suggested by (Florensa et al., 2017), is to adjust the distribution of start states of the agent, making it start increasingly far from the goal state. An example of this approach is illustrated in Figure 5, where the distribution of the agent's start position is changed in each lesson, successively moving the agent further from the target, and thus increasing the difficulty of the task.

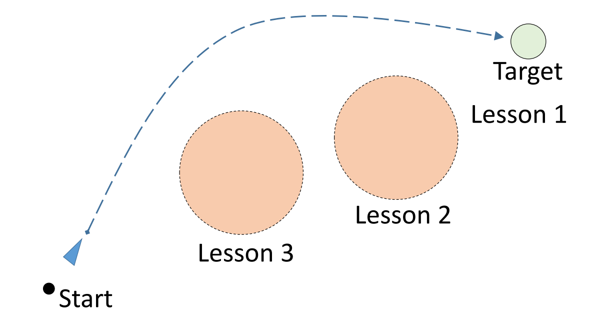

Figure 4: Example of curriculum learning with increasing difficulty in the scenario.

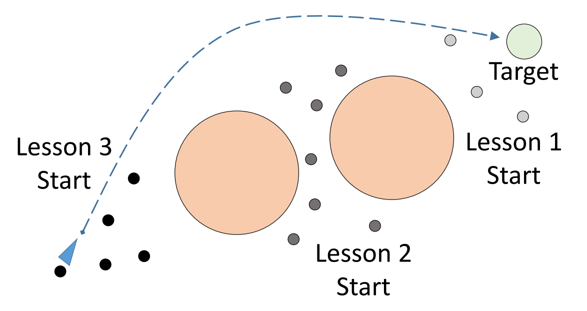

Figure 5: Example of curriculum learning with changes in start distribution.

A different, innovative approach was proposed by (Andrychowicz et al., 2017), who studied how to handle sparse or even binary rewards in multi-goal reinforcement learning. In multi-goal reinforcement learning, the input to the policy is not only the current state of the environment, but also a goal that the agent should reach. In the air combat domain the goal could be a specification of a target that a synthetic pilot should attack and destroy. In the proposed algorithm, Hindsight Experience Replay (HER), experiences are stored in a replay memory, not only with the actual goal specified in the episode, but also with a set of additional goals, e.g. possible goal states actually achieved by the agent. The policy is then trained by sampling from the replay memory, resulting in an implicit curriculum, since simple as well as difficult goals are experienced in the replay phase. The approach was evaluated with good results on robotic arm manipulation tasks.

Potential-Based Reward Shaping

In environments with sparse rewards, learning may be extremely slow. One possible way to speed up learning is to add additional components to the reward signal, to push the agent in the right direction. This is called reward shaping, and affects the exploration of the agent, and possibly the time it takes to converge to a policy. For example, the *Q*-learning update step with reward shaping *F*(*s*, *s'*) is defined as:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_t + F(s, s') + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{4}$$

One possible risk with reward shaping is that a bias may be introduced in the policy, i.e. the agent is prevented from finding an optimal policy for the original problem. It has been shown that the only reward shaping that is guaranteed to not have this effect is potential-based reward shaping (Ng et al., 1999). Potential-based reward shaping defines the shaping reward as the difference in potential of successive states:

$$F(s, s') = \gamma\phi(s') - \phi(s) \tag{5}$$

Here $\phi(s)$ is the potential of a given state, and γ is the discount factor of the MDP.

As an example of potential-based reward shaping, consider the simple scenario presented in Figure 6, where we would like the agent to navigate from the start position to the target. Due to the distance between the start position and the goal state, it may be difficult for the agent to find the target if a sparse reward signal is used. If we instead define a potential based on the distance to the target, the agent can be given continuous feedback regarding its progress towards its goal.

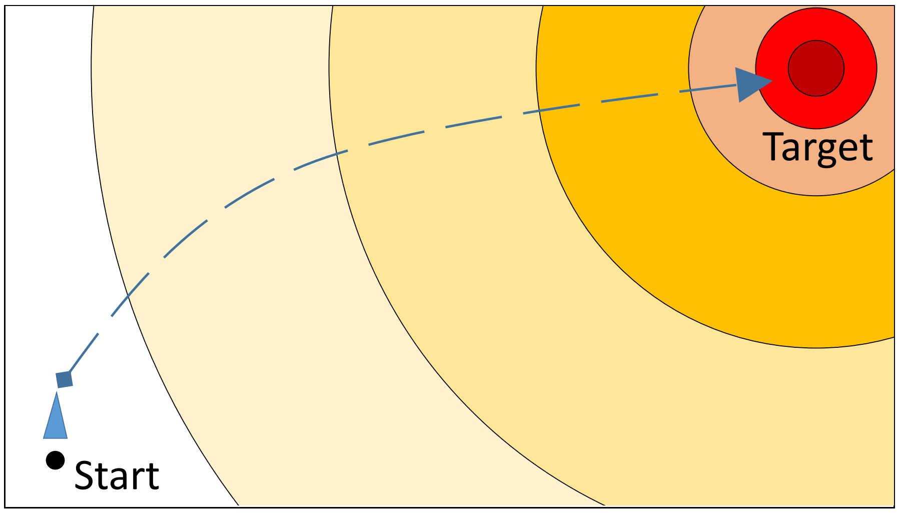

Figure 6: Potential-based reward shaping for a navigation task.

It is worth noting that though potential-based reward shaping is guaranteed to not introduce a bias in the policy, there is no guarantee that it will help the training to converge faster. When applied in the high-dimensional state space of the air combat domain there can also be some counter-intuitive effects if the discount factor γ is less than one. If we are using a negative potential, and the agent is far from the goal, the agent may receive a positive reward for moving in the wrong direction. Similarly, when using a positive potential, when the agent is far from the goal it may receive a negative reward for moving towards the goal. The actual reward received is also affected by the resolution of the agent, i.e. the size of the time–step used. If the resolution is too high, the agent may not be able to reach the level of advancement towards the goal required by the discount factor in each step.

Hierarchical Reinforcement Learning

Compared to many of the popular reinforcement learning benchmarks, air combat scenarios typically evolve over quite long time-horizons, with actions being performed at different timescales. It then becomes challenging for the agent to explore the effects of its complete set of actions when trying to achieve its long-term goals. In hierarchical reinforcement learning temporal abstractions, such as macro-operators, sub-tasks and modes, are introduced to simplify the problem. In addition to the possible improvements in learning performance, it may also be more natural to discuss end-user needs in terms of more abstract actions, e.g. when doing requirements elicitation.

One popular approach to hierarchical reinforcement learning is the options framework (Sutton et al., 1999), which provides a formalism for temporally extended actions. An option $\omega \in \Omega$ is a tuple $(I_\omega, \pi_\omega, \beta_\omega)$, where:

- I_ω: Is the initiation set, specifying the states from which the option can start
- π_ω: Is the intra-option policy
- β_ω: Is the termination condition of the option, specifying the probability of terminating in a state

Possible examples of options related to the scenario in Figure 5 are “move to target”, “inspect target”, and “deliver weapon”. Instead of working with primitive actions such as low-level guidance, sensor and weapon commands directly, one can instead learn a policy over options π_Ω. In intra-option value learning (Sutton et al., 1999), intra-option policies and termination conditions of options are learned. The option-critic architecture makes it possible to learn options end-to-end (Bacon et al., 2017), including intra-option policies, termination conditions and the policy over options.

Multi-Agent Reinforcement Learning

For some time, reinforcement learning research focused on single-agent problems. However, since many real-world problems are multi-agent by nature, interest in multi-agent algorithms has increased in recent years. Such algorithms are highly relevant in the air combat domain, since pilots typically carry out their missions in teams. In the multi-agent setting, there are two major approaches for decision-making: the centralized and the decentralized approach. In the centralized approach, one agent controls multiple entities in the scenario. This is simple in the sense that normal, single-agent reinforcement learning algorithms can be used. However, this approach does not scale well, since the number of actions grows exponentially in the number of controlled entities. The decentralized approach has better scalability. In this approach, each agent controls one entity, which acts on its own, based on its own observations. One problem with decentralized multi-agent reinforcement learning is that, from the point of view of a single agent, the environment dynamics, which include other agents, may become non-stationary when several agents learn concurrently.

One approach for learning decentralized policies in multi-agent environments is by using centralized learning and decentralized execution. In this approach, a centralized critic uses additional information, e.g. the state observations and actions of all agents, to guide the learning of decentralized policies for individual actors. During execution each actor takes actions according to its own, local observations. There are several recent examples of algorithms that take this approach. (Lowe et al., 2017) extended the DDPG algorithm to mixed cooperative-competitive multi-agent environments (MADDPG), using a centralized critic for each agent. The algorithm supports continuous actions and communication among agents. (Foerster et al., 2018) proposed Counterfactual Multi-Agent (COMA) policy gradients for cooperative multi-agent environments, using a centralized critic and decentralized actors. A novel way of determining a single agent's contribution to the joint Q function was used to learn the decentralized policies. In the QMIX algorithm (Rashid et al., 2018), a centralized mixing network is used to estimate the joint action-values of a team of agents, based on the action-values of individual agents, in such a way that consistency between centralized and decentralized policies is guaranteed.

One way to facilitate multi-agent reinforcement learning is to use predictive models of other agents' long-term goals and immediate actions. Such models reduce the complexity of the problem that the model-free reinforcement learning algorithms must solve, and can support decision-making in cooperative as well as competitive scenarios. The models can be handcrafted by experts with domain knowledge, or built with data-driven methods, such as supervised or unsupervised learning.

Multi-Objective Reinforcement Learning

In many real-world problems, multiple possibly conflicting objectives must be considered when making decisions. For instance, in the air combat domain we may consider multiple enemy targets to attack, multiple high-value assets to protect, resource consumption and safety. The priorities among these objectives will define the characteristics of the agent. In training scenarios, we may want to vary the agents' characteristics over time, to fit the training needs of the students. Multi-objective reinforcement learning (MORL) is an extension to the standard reinforcement learning paradigm, which explicitly deals with the problem of prioritizing among a set of objectives. The following three scenarios, presented by (Roijers et al., 2013), can be used to motivate multi-objective algorithms:

- The unknown priorities scenario: The priorities among objectives are not known at training time, so it is desired to delay the selection of a policy until the execution phase, e.g. to find a suitable synthetic opponent for a trainee
- The decision support scenario: Defining the priorities among objectives may be difficult for the end-user, e.g. instructors and trainees that use a training system, and we therefore want to present a set of policies to support selection
- The known priorities scenario: The priorities among objectives are known at training time, but using multi-objective methods makes solving the problem easier, e.g. when the user utility function is non-linear in the objectives

In multi-objective reinforcement learning, instead of the standard MDP, a multi-objective MDP (MOMDP) is used. The MOMDP provides a vector-valued reward signal, with each element in the vector representing the reward for one of the objectives. The value function $\boldsymbol{V}_{\pi}(s)$ is also vector-valued, and to determine the overall value of a policy a scalarization function is used to transform the vector into a scalar.

For example, the value function could be scalarized by calculating the weighted sum of its elements:

$$V_\pi^{\boldsymbol{w}}(s) = f(\boldsymbol{V}_\pi(s), \boldsymbol{w}) = \sum_{i=1}^{n} v_i w_i \tag{6}$$

$$\sum_{i=1}^{n} w_i = 1 \tag{7}$$

To solve the MOMDP we would like to find optimal policies for all possible user preferences among objectives. For complex problems, this may not be possible, but instead an approximation must be used. When using deep reinforcement learning we can choose between learning a set of policies, e.g. an approximation of the Convex Coverage Set (CCS) of the policy space, as proposed by (Mossalam et al., 2016), or training a single neural network to approximate multiple policies, by conditioning the network on user preferences, as proposed by e.g. (Abels et al., 2018). A suitable policy, matching the current user preferences, can then be selected at runtime. In our previous work, we studied how this approach could be used to build agent-based simulations with tunable dynamics (Källström & Heintz, 2019).

CONCLUSION

In this paper, we discussed reinforcement learning for computer generated forces, in the context of air combat simulation intended for training of fighter pilots. We presented an overview of open-source software tools, which can help establish an infrastructure for reinforcement learning, and quickly get started developing CGF behavior models. We also described how simulation software can be integrated with these tools. Finally, we discussed some of the challenges that we face when trying to apply reinforcement learning in the air combat domain, such as high-dimensional state and action spaces, partial observability, and delayed and sparse rewards, as well as possible ways to address these challenges. In spite of several great achievements, state-of-the-art deep reinforcement learning algorithms still have some shortcomings. Further algorithm development could help realize the true potential of reinforcement learning in the context of military simulation.

For instance, due to the poor sample efficiency of reinforcement learning algorithms, large amounts of data are currently required to train behavior models. Consequently, powerful computation resources are required to generate these data, and a considerable amount of time may be required to train the models. State-of-the art achievements using e.g. real-time strategy games as environments for training agents often require thousands of CPU cores to produce results in a reasonable amount of time. The poor sample efficiency also limits the possibilities for doing online learning, e.g. in an operational training system. If algorithms become more efficient, adaptive training systems could be constructed more easily.

It is also important to consider usability aspects. Policies represented by deep neural networks suffer from a lack of transparency. This is problematic in many application domains, including the domain of military training. For instance, in a debriefing situation it is important to be able to explain the actions of synthetic actors. The current way of configuring the learning algorithms and the environments used for training of agents may also not be transparent for the normal end-user of training systems. The interface between instructors, operators and trainees on one hand, and the learning algorithm on the other hand is an important area for further research. Another important research question is how to adapt training curriculums, and the organizations that implement them, when intelligent, synthetic agents become prevalent.

Interest in reinforcement learning has grown tremendously in the last few years and is still growing. The field is advancing rapidly, and the research community is moving away from studying toy problems, with simple environments and single agents, to study problems that are more challenging, such as complex real-time strategy games. In these new domains, algorithms must deal with multiple agents, who may be cooperating as well as competing, partial observability and decision-making over long time-horizons. Thus, they have similar characteristics as military scenarios. It can therefore be assumed that algorithms that fit the needs of military simulations will be developed, and that these algorithms can help build the next-generation training systems.

ACKNOWLEDGEMENTS

This work was partially supported by the Swedish Governmental Agency for Innovation Systems (NFFP7/2017-04885), and the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

Abels, A., Roijers, D. M., Lenaerts, T., Nowé, A., & Steckelmacher, D. (2018). Dynamic Weights in Multi-Objective Deep Reinforcement Learning. *arXiv preprint arXiv:1809.07803*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., ... & Zaremba, W. (2017). Hindsight experience replay. In *Advances in Neural Information Processing Systems* (pp. 5048-5058).

Bacon, P. L., Harb, J., & Precup, D. (2017). The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research, 47*, 253-279.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., & Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems* (pp. 1471-1479).

Florensa, C., Held, D., Wulfmeier, M., Zhang, M., & Abbeel, P. (2017). Reverse curriculum generation for reinforcement learning. In *Conference on Robot Learning (CoRL).*

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018). Counterfactual multi-agent policy gradients. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations (ICLR).*

Källström, J., & Heintz, F. (2019). Tunable Dynamics in Agent-Based Simulation using Multi-Objective Reinforcement Learning. In *Adaptive and Learning Agents (ALA) workshop at AAMAS*, Vol.19.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2016). Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations (ICLR)*.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, O. P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems* (pp. 6379-6390).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *Thirty-third International conference on machine learning* (pp. 1928-1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529.

Mossalam, H., Assael, Y. M., Roijers, D. M., & Whiteson, S. (2016). Multi-objective deep reinforcement learning. *arXiv preprint arXiv:1610.02707*.

Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML* (Vol. 99, pp. 278-287).

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J., & Whiteson, S. (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Thirty-fifth International Conference on Machine Learning*.

Roijers, D. M., Vamplew, P., Whiteson, S., & Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48, 67-113.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *Thirty-first International Conference on Machine Learning*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence, 112*(1-2), 181-211.