Multi-agent reinforcement learning

Schnebli Zoltan

December 22, 2019

Contents

1 Introduction **1**

2 Reinforcement learning **1**
2.1 Single-agent learning models 2
2.1.1 Markov decision process 2
2.1.2 Partially observable Markov decision process 3
2.2 Multi-agent learning models 4
2.2.1 Markov games 4
2.3 Learning methods 4
2.3.1 Deep reinforcement learning 4
2.3.2 Policy gradient 5

3 Recent research results **5**
3.1 Deep deterministic policy gradient algorithm with generative cooperative policy network 5
3.1.1 Theory 5
3.1.2 Experiments 6
3.2 Learning Hand-Eye Coordination for Robotic Grasping 8
3.2.1 Theory 8
3.2.2 Experiments 8
3.3 Emergent Complexity via Multi-Agent Competition 10
3.3.1 Theory 10
3.3.2 Experiments 10

4 Conclusion **11**

Abstract

The aim of this paper is to present a general overview of the multi-agent systems from the field of reinforcement learning.

First we will present a general formulation of the reinforcement learning problems. Because the multi-agent settings are quite difficult concepts, in the first few chapters we presented the needed models to solve the single-agent problems. After generalizing these models we introduced the Markov Games, which can model every multi-agent systems. Because this setting does not have a specific way to solve, in the last chapter we presented some recent research results, which presented innovative concepts to solve these problems.

1 Introduction

Intelligent agents rarely act in isolation in the real world and often seek to achieve their goals through interaction with other agents. Such interactions give rise to rich and complex behaviors for agents in multi-agent systems. Depending on the agents motivations, interactions could be directed towards achieving a shared goal in a collaborative setting, opposing another agent in a competitive setting, or be a mixture of these in a setting where agents collaborate in teams to compete against other teams. Learning useful representations of the policies of agents based on their interactions is a very challenging step towards the characterization of the agents behavior.

The goal of this thesis is to offer an introduction of the multi-agent systems in the world of reinforcement learning. We will start from a general presentation of the field, before we start to concretize, first, to the easier single-agent systems and after that the multi-agent setting. To show the current state of this sub field we will provide a few recent research results after the main sections.

2 Reinforcement learning

Reinforcement learning is a subsection of machine learning. It's purpose is to teach some software agent to take adequate actions, so that he can maximize a given cumulative reward. In contradiction to the other machine learning methods, in reinforcement learning the learner (in future appearances called *agent*) doesn't gets any advice on how to finish a given task. He has to explore his environment and experience the weight of his decisions to figure out what actions will bring him the most reward. Every agent has a target which he wants to reach optimally. He can perceive his surroundings, can take actions, which changes the environment. The agent doesn't know how well his actions are in the long term, he just gets an instantaneous reward and finds himself in a new state. In order to maximize this reward, the agent needs to collect more and more experience about the environment and his actions. We can observe this cycle in 1, where the agent with some experience takes and action and due to this he gets a reward and more experience Sutton and Barto [2018].

One of the most challenging tasks in this area is, that the agents actions are influencing not only the current reward, but also every state in the *episode* which follows the current one. So the agent has to learn from not just the current reward, but also from the future

ones as well. An episode represents action-state-reward sequence, which ends with a final state, where a final state means that the agent doesn't has a valid action to perform.

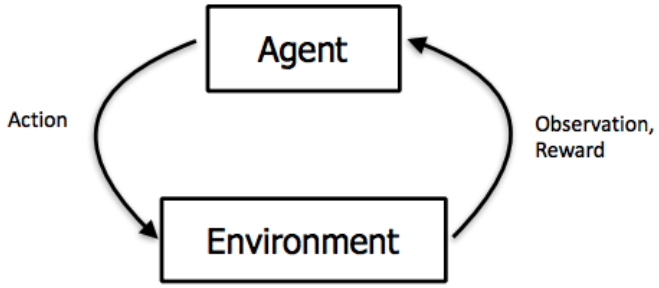

Figure 1: In the picture we can observe, that the agent communicates with environment throughout with his actions. He has at his disposal the state of the environment and he chooses his next actions according to this. Due to this he gets a numerical reward and preceives the changes in the environment.

In reinforcement learning exits a so called exploration-exploitation problem, which is quite unique amongst the other machine learning methods.
Exploration means here that the agent executes a random action in a given state, while *exploitation* means that he takes an actions based on his current knowledge. The contradiction here is, that the agent should maximize his reward based on his knowledge, but he should explore more and more to expand it.
The most popular solution to this problem is the ϵ-*greedy* method. The essence of this method is that the agent tooks an action in which he tries to maximize his reward based on his knowledge with a probability of 1-ϵ and in the remaining ϵ probability he explores his surroundings. This method is often used with a variable ϵ value, so that at the beginning of the episode the value of ϵ is close to 1, but as the episode progresses it decreases close to 0.

2.1 Single-agent learning models

Before we start with multi-agent systems we have to learn how a single-agent system works.

2.1.1 Markov decision process

Reinforcement learning formalizes the interaction of an agent with an environment using a Markov decision process (MDP). This environment has to be composed very strictly and precisely, because otherwise the agent won't be able to learn Hernandez-Leal *et al.* [2018].
An MDP is defined by the 5-tuple $\langle S, A, T_{\cdot}(\cdot,\cdot), R_{\cdot}(\cdot,\cdot), \gamma \rangle$ where:

S and A represents a finite set of states and actions

The transition function T determines the probability of a transition from state s to state s' given action a:

$$\mathcal{T}_a(s, s') = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$$

The reward function R defines the immediate reward that an agent would receive given that the agent executes action a while in state s and it is transitioned to state s':

$$\mathcal{R}_a(s, s') = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$$

$\gamma \in [0, 1]$ represents the discount factor that balances the trade-off between immediate rewards and future rewards. When $\gamma = 0$ the agent only cares about which action will have the largest immediate reward, while $\gamma = 1$ means, that the agent cares about only maximizing the expected sum of the future rewards

MDPs are adequate models to obtain optimal decisions in *single agent environments*. Solving an MDP will yield in a policy $\pi : S \to A$, which is a mapping from states to actions. An optimal policy π^* is the one that maximizes the expected discounted sum of rewards. There are different techniques for solving MDPs assuming a complete description of all its elements.

2.1.2 Partially observable Markov decision process

Partially observable Markov decision processes (POMDP) are a generalization of the regular MDPs. A POMDP models an agent decision process in which it is assumed that the system dynamics are determined by an MDP, but the agent cannot directly observe the underlying state, instead, it must maintain a probability distribution over the set of possible states, based on a set of observations and observation probabilities, and the underlying MDP Smallwood and Sondik [1973].
Therefore a POMDP is composed by a 7-tuple $\langle S, A, T_{\cdot}(\cdot,\cdot), R_{\cdot}(\cdot,\cdot), \gamma, \Omega, O(\cdot,\cdot)\rangle$, where:

S, A, T, R, γ are the same as in section 2.1.1

Ω is the set of all the observations

The O function defines the probability of getting an observation $o \in \Omega$ if the agent executes an action $a \in A$ and enters in the state s':

$$\mathcal{O}_a(o, s') = Pr\{s_{t+1} = s' \mid o_t = o, a_t = a\}$$

At each time stamp the environment is in a given $s \in S$ state, where the agent takes an action $a \in A$. This actions causes the environment to transition into a new s' state with a probability taken from the transition function $\mathcal{T}_a(s, s')$. At the same time the agent receives an observation $o \in \Omega$ which depends on the new state of the environment with probability $\mathcal{O}_a(o, s')$ and also a reward r taken from the reward function $\mathcal{R}_a(s, s')$. After all this, the cycle is repeated until the agent reaches a final state.

Because the agent doesn't knows for sure in which state he currently is, he has to make his choices under a given uncertainty. By interaction with the environment and receiving feedbacks, the agent could update its belief in the true state by modifying the distribution. This property could consequence the agent executing given actions, only to improve its belief of the current state.

2.2 Multi-agent learning models

Multi-agent systems are usually represented by the help of Markov games (MG).

2.2.1 Markov games

A *Markov game* extends the general formulation of the partially observable Markov decision processes (POMDP), so it can be used for solving a waste majority of problems. For example, if we assume that we have only one agent, which knows with complete certainty the transitions between the states, the Markov game reduces to an ordinary Markov decision process (MDP) Littman [1994].
In a MG we are given a set of n agent on a state space S with action spaces

$$\mathcal{A} := \{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n\}$$

and observation spaces

$$\mathcal{O} := \{\mathcal{O}_1, \mathcal{O}_2, ..., \mathcal{O}_n\}$$

respectively.

At every time step t, an agent i receives an observation $o_i^{(t)} \in \mathcal{O}_i$ and executes action $a_i^{(t)} \in \mathcal{A}_i$. In a MG, given the global state, each agent takes an independent action to maximize its own accumulated reward while interacting with other agents. Therefore one agents success it dependent on the policies of other agents. The interactions are implying that the accumulated reward of an individual agent and the evolution of the global state are determined by the action of all the agents. Depending on the relationships among the agents rewards, different equilibrium concepts are employed to agents policy. It is generally difficult to derive favorable strategies for a general MG because the optimality concept does not apply here anymore. The biggest change from the standard MDP is, that different agents can receive different rewards for the same state transition.

2.3 Learning methods

In this section, we present the necessary background about the methods, that the presented experiments are using.

2.3.1 Deep reinforcement learning

Alongside the successes, tabular RL methods have major drawbacks: slow learning in large state space, the methods were not general enough and the biggest drawback, that the state representations needed to be hand-specified. Fortunately, these challenges can be addressed by neural networks as function approximators as follows:

$$\mathrm{Q(s,\ a;\ }\theta) \approx \mathrm{Q(s,\ a)}$$

where represents the neural network weights. They help improving the sample efficiency for large state-space problems by generalizing across the states. Second, neural networks can be used to reduce the need for manually designing features to represent state information. However, implementing neural networks to RL problems comes with additional challenges.

2.3.2 Policy gradient

Policy gradient methods differ significantly from the others as they do not suffer from the uncertainty of the environment model. They work by directly computing an estimate of the gradient of policy parameters in order to maximize the expected return using stochastic gradient descent. Such methods are also attractive because continuous states and actions can be dealt with in exactly the same way as discrete ones while, in addition, the learning performance is often increased and convergence at least to a local optimum is guaranteed.

3 Recent research results

Since the agents in multi-agent system lack full knowledge on dynamic environment and other agents strategies, learning an optimal strategy in a multi-agent system is much more challenging than in a single agent system. The target is to let the agents learn to cooperate, coordinate and negotiate between them.
In the following sections we will present some recent approaches to reach these results.

3.1 Deep deterministic policy gradient algorithm with generative cooperative policy network

In the standard deep deterministic policy gradient (DDPG) algorithm, each agent has its own policy network, mapping the agents actions to states, approximated by a Q-network as follows: $a_i = \mu_i(o_i; \theta)$. This algorithm also allows the agents to have individual reward signals: $r_i(s, a_1, ..., a_N)$ from which we derive the decentralized actor network using the individual critic network.

3.1.1 Theory

In Ryu *et al.* [2018] it is proposed a slightly different variant of the algorithm from above: it maintains the decentralized policy network and the individual Q-network, but it also defines an additional policy network, called *generative cooperative policy network* (GCPN) to teach the agents cooperativeness. Each agent i has to maximize its own reward and also the reward of the other agents using the GCPN μ_i^c, where i means the N-agents excluding agent i. Each policy has its own role:

Q-network(individual): this is used for choosing optimal actions during the execution period

Greedy policy network(individual): is trained to maximize the expected return represented as a individual critic network

GCPN: generates action samples to interact with the environment during training period for learning the actor and its Q policy

The difference between the two algorithms are visible in figure 2.
Although with this algorithm the agent learn more cooperative tactics, it also has its drawbacks, because of the additional policies it has an extended training session, so it requires more computation and memory than other algorithms.

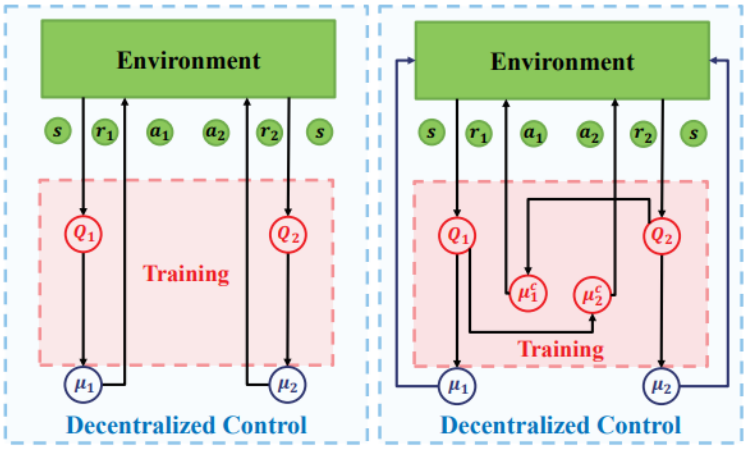

Figure 2: In the left you can observe the general deep deterministic policy gradient algorithm and in the right the generative cooperative policy network variant.

3.1.2 Experiments

To evaluate the performance of the proposed algorithm, first, they used on the Predator-pray environment, where several predators gain more *reward*, if they cooperate with each other to catch a pray. Figure 3 illustrates the environment with two obstacles, three predators and one pray. Each agent should cooperate, because the pray is faster than the predators.

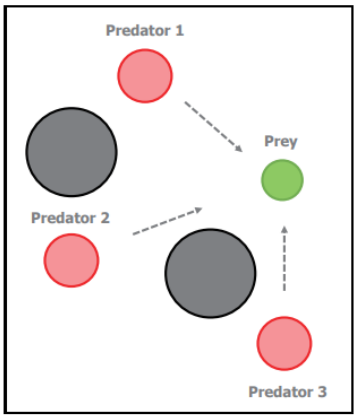

Figure 3: Predator-pray environment in OpenAI Gym.

They compared the performances of the following four learning algorithms:

CF: a multi-agent actor-critic algorithm designed for a team game with a shared return for all agents. Each agent who tries to maximize its shared return through the global Q-network directly induces the learning of collaborativeness behavior among the agents.

FDMARL: a multi-agent actor-critic algorithm designed for a cooperative game with individual return for each agent. The algorithm requires each agent to infer the global return through parameter sharing and uses the inferred global return to derive the decentralized policy.

DDPG: a multi-agent actor-critic algorithm designed for general non-cooperative game. This algorithm requires each agent to learn other players policies and use the trained policies to approximate its own Q-network. In addition, the individual policy is also trained to interact with the individually updated Q-network.

DDPG-GCPN1: the proposed algorithm with the randomly selected GCPNs in sample-generating.

DDPG-GCPN2: the proposed algorithm with GCPN ensembles in sample-generating.

They tried these algorithm with 2 kind of reward functions. The first one gives every agent a reward of 10, if they catch the pray, and another one which gives only individual rewards.
In Figure 4 there is presented the normalized score and catching rate of the predators with each algorithm for the shared (a) and individual (b) case. In the shared reward case (the left figure), each algorithm has similar score although the score of CF is slightly lower. They observed that the shared reward led to weak cooperation in training period, preventing the agents from reaching equilibrium. In the individual case (the right figure) the results tend to be very different. In particular, the proposed variant performs better than the other. That is because every agent gets an individual reward, than, it tries to perform other actions based on the other agents policy, leading to an more optimal training and equilibrium. Although the other two algorithms were able also to learn cooperativeness, but they couldn't achieve equilibrium in cooperativeness.

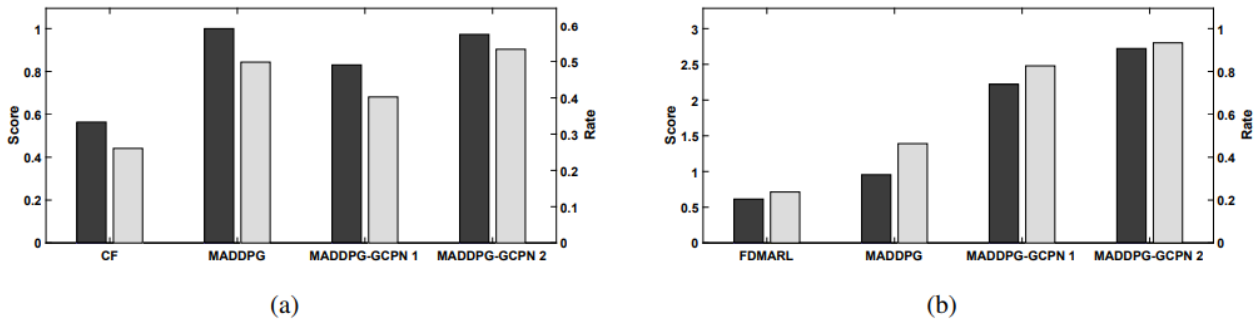

Figure 4: Normalized score (dark) and catching rate (light) in the shared (a) (left) and individual (b) (right) reward. cases with each algorithm.

3.2 Learning Hand-Eye Coordination for Robotic Grasping

In contrast to humans and animals, robots does not have a fast feed-back loop while interacting with objects. For this reason the living beings can execute complex task, such as extracting an object from a collection without any advance planning. They rely just on they touch and vision. To mimic this behavior scientist seeked to incorporate complex sensors into the feedback controllers. But to achieve this they had to specify the features by hand and it required manual or automatic calibration to determine the precise geometric relationship between the camera and the robots end-effector.

3.2.1 Theory

In Levine *et al.* [2018] it is proposed a learning-based approach to learn hand-eye coordination. The approach is data-driven and it is goal centric: it learns to servo a robotic gripper with the help of training directly from image pixels to function the gripper motion. It continuously recalculates the motor commands, adjusts the the perturbation and the grasp from the input of the integrated sensors in order to maiximize the success rate.
This method consists of two components:

- grasp success predicator: this is a deep convolutional neural network (CNN), which determines the likelihood of producing a successful grasp

- continuous servoing mechanism: this uses also a CNN to update the motor commands by continuously updating the path to a successfull grasp

3.2.2 Experiments

The goal of the experiments was to answear two questions. First, that the continuous servoing how significantly improves the accuracy and success rate and second, how well the proposed model works compared with other approaches:

open-loop: this method observes the scene prior to the grasp, extracts image patches, chooses the patch with the highest probability of a successful grasp, and then uses camera calibration to move the gripper to that location. (it does not uses continuous calibration)

random base-line method

hand-engineered grasping system: it uses depth images and heuristic positioning of the fingers to make a grasp

Comparing the algorithms beforehand, the proposed method should work faster than the other, because it does not require knowledge of the camera to hand calibration or depth images. The experiments were evaluated with two protocols. In the first protocol, the objects were placed into a bin in front of the robot, and it was allowed to grasp objects for 100 attempts, placing any grasped object back into the bin after each attempt. Grasping with replacement tests the ability of the system to pick up objects in cluttered settings, but it also allows the robot to repeatedly pick up easy objects. To address this shortcoming of the replacement condition, they also tested each system without replacement 4 times and noted the success rates on the first 10, 20, and 30 grasp attempts.

In Figure 5 we can see the failure rates for each tested method. We can clearly see, that the proposed methods success rate exceeds the other methods. In the case without the replacement, this method could empty the bin in 30 graspings. Compared to the hand-engineered method who struggled to perceive the depth of the objects, this method worked fine without hardware modification.

without replacement	first 10 ($N = 40$)	first 20 ($N = 80$)	first 30 ($N = 120$)
random	67.5%	70.0%	72.5%
hand-designed	32.5%	35.0%	50.8%
open loop	27.5%	38.7%	33.7%
our method	**10.0%**	**17.5%**	**17.5%**
with replacement	failure rate ($N = 100$)		
random	69%		
hand-designed	35%		
open loop	43%		
our method	**20%**		

Figure 5: Failure rates for each tested method

3.3 Emergent Complexity via Multi-Agent Competition

Normally, reinforcement learning algorithms can train agents to solve complex and interesting tasks. Usually the complexity of the agent is closely related to the complexity of the environment. This means, that highly capable agents need complex environment for training.

3.3.1 Theory

In Bansal *et al.* [2017] it is proposed that in a competitive multi-agent environment, agents trained with self-play can achieve behaviors much more complex than the environment itself. This work introduces several competitive multi-agent environments where agents compete in a 3D world with simulated physics. The trained agents learn a wide variety of complex and interesting skills, even though the environment themselves are relatively simple. The skills include behaviors such as running, blocking, ducking, tackling, fooling opponents, kicking, and defending using both arms and legs. The mentioned environments can be viewed in Figure 6 and the reward functions used in those are as follows:

> Run to Goal: the first agent reaching the goal gets +1000, while the other -1000. If no one reaches the goal both are getting -1000
>
> You Shall Not Pass: if the blockers successfully blocks the other and stands at the end it gets +1000, otherwise 0, while the other agent -1000. If it fails in blocking it gets -1000 and the other get +1000.
>
> Sumo: the first agent knocking out the other agent gets +1000, while the other -1000. If no one reaches the goal both are getting -1000
>
> Kick and Defend: successful kick or defend results in +1000 and -1000 for the other.

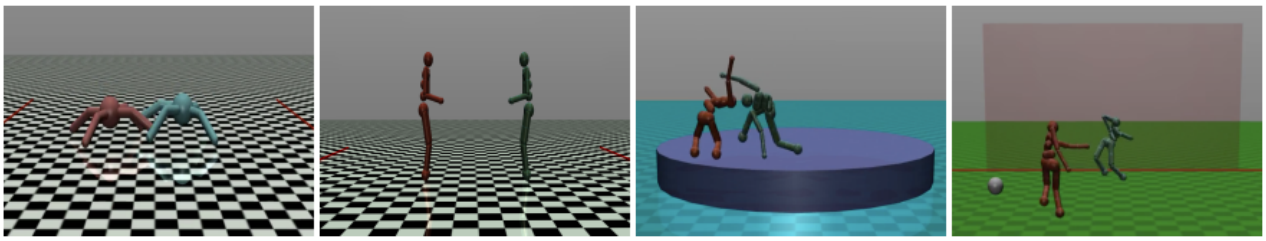

Figure 6: Illustrations of the environments: Run to Goal, You Shall Not Pass, Sumo and Kick and Defend

3.3.2 Experiments

For training the agents they tried two training frameworks: exploration curriculum and opponent sampling. The exploration curriculum uses a dense reward at every step in the beginning phase of the training to allow agents to learn basic motor skills, like walking forward or being able to stand, which would increase the probability of random actions

from the agent yielding a positive reward. This reward is called exploration reward and it is decreased slowly as the episodes are passing. In the opponent sampling framework all agents are learning simultaneously in pairs. In the paper, they observed that training against random old opponents are much more efficient than against the latest available one.

The agents learned numerous interesting behaviors. For example in Run-to-Goal the agents demonstrated blocking behaviors. Humanoids first tried to avoid each other so that they could bump into each other with force and try to recover from the impact. On You-Shall-Not-Pass the humanoid learned to block by raising the hand while the other learned to duck under it. On Sumo the humanoids learned to knock down their opponent using their head, or charging towards them at the beginning whereas the opponent tried to fool it by stepping out of the opponents way at the edge of the ring. On kick-and-defend the kicker learned to kick the ball high and tired to avoid the defender. The kicker also learned a fooling behavior, where it moved left and right quickly on close to the ball to fool the defender. The defender learned to defend on the motion using its legs and hands to obstruct the ball.

4 Conclusion

In this work, we presented the basics of reinforcement learning and the models to solve both single- and multi-agent systems. This is followed by a section which contains recent researches in this domain. Because currently these systems does not have a specific way to solve these problems everybody seeks to be the first to find a general solution. For this reason there are numerous ongoing researches in this field. Our aim was to make a general introduction to the multi-agent system because with solving this problem we could automatize even more work in the fabrics and save human resources.

References

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. Is multiagent deep reinforcement learning the answer or the question? a brief survey. *arXiv preprint arXiv:1810.05587*, 2018.

Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier, 1994.

H. Ryu, H. Shin, and J. Park. Multi-Agent Actor-Critic with Generative Cooperative Policy Network. *ArXiv e-prints*, October 2018.

Richard D Smallwood and Edward J Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations research*, 21(5):1071–1088, 1973.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.