Multi-Agent Reinforcement Learning through Allocation Optimization

Nathan Margaglio
University at Buffalo
Department of Computer Science and Engineering
namargag@buffalo.edu

Abstract

Recent advancements in deep learning technology have allowed powerful and complex reinforcement learning strategies to emerge that are capable of solving tasks that weren't close to possible even a decade ago. An interesting, and challenging, task for reinforcement learning takes place in the multi-agent setting. In this paper, we will describe the current state of reinforcement learning and how multi-agent settings are being defined and solved. We will then introduce a multi-agent learning approach that builds off of the concept of allocation optimization.

1 Introduction

Reinforcement learning can be applied in a variety of tasks and scenarios where learning an optimal decision policy is valuable. Contemporary research on the subject has focused on not only improving such algorithms, but to also apply existing algorithms in new contexts and with new challenges. As deep learning technologies become more powerful and robust, the large state and action spaces that made many environments computationally infeasible no longer pose such a problem.

Modern reinforcement learning can be traced to Q-Learning[1], which provided a tabular approach to solving the task of complex environment navigation through self-exploration. This allowed an agent to learn which actions should be taken given a specific state in order to maximize an accumulated reward. Q-Learning, however, is unable to solve high-dimensional problems. Due to the drastic increase in computations that come with the increase in the number of inputs, many reinforcement learning problems were intractable for Q-Learning and other dynamic programming approaches.

In 2015, deep learning was introduced to the Q-Learning approach to produce an algorithm capable of self-learning in complex environments where the input may be given as direct pixel data from Atari games[2]. Since then, many major improvements and variations in deep reinforcement learning have allowed for algorithms that are capable of solving environments as complex as learning dexterous in-hand manipulation through simulated environments that can be applied in real-world robotics[3] to beating the world champion of Go through self-play training only[4].

We are now at the point where we are capable of building a set of self-learning agents capable of cooperating with one another. In 2018, OpenAI introduced OpenAI Five, a team of five neural networks trained to play Dota 2 as a team[5]. Dota 2 has a large e-sports following and is considered to be more computationally difficult to solve than Go. OpenAI was not only able to train a self-learning agent to successfully beat champion players at the game in 1v1 matches, but the OpenAI Five team was able to beat the world champion Dota 2 team in a best two-out-of-three tournament.

With multi-agent environments becoming the focus on reinforcement learning, we are aiming to find alternative approaches to solving such challenges that generalize well while leveraging the unique characteristics of this setting.

2 Background

We first describe the mechanisms used to describe reinforcement learning tasks and algorithms.

2.1 Markov Decision Process

A *Markov decision process* (MDP) consists of an agent and an environment reacting to one another in a loop. The environment's dynamics are characterized by a state space S, and action space A, a transition probability function $P^a_{ss'} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$ and an expected reward function $R^a_s = \mathbb{E}\{r_{t+1} \mid s_t = s, a_t = a\}, \forall s, s' \in S, a \in A$. The agent's decision making procedure is characterized by a policy, $\pi(s, a, \theta) = Pr\{a_t = a \mid s_t = s, \theta\}, \forall s \in S, a \in A$, where $\theta \in \mathbb{R}^l$, for $l << |S|$, is a parameter vector. We assume π is differentiable with respect to its parameter, i.e., that $\frac{\partial \pi(s,a)}{\partial \theta}$ exists[6].

At a high level, a MDP starts at time step $t = 0$, at which point the environment produces the initial state s_0. This state is passed through to the agent, which inputs the state along with every available action into it's policy π_θ. The policy will dictate the agent's action, which will then be passed back to the environment. The environment receives the action, and through the state transition probability function and the expected reward function, produces a state and a reward to be passed back to the agent. This process continues until a terminal state s_t is reached.

2.2 Multi-Agent Environments

When working with multi-agent environments, we will consider the multi-agent extension of Markov decision processes called partially observable Markov games[7]. A Markov game for N agents is defined by a set of states S describing the possible configurations of all agents, a set of actions $A_1, ..., A_N$ and a set of observations $O_1, ..., O_N$ for each agent. To choose actions, each agent i uses a stochastic policy $\pi_theta : O_i \times A_i \to [0, 1]$, which produces the next state according to the state transition function. Each agent i obtains rewards as a function of the state and the agent's action $r_i : S \times A_i \to \mathbb{R}$, and receives a private observation correlated with the state $o_i : S \to O_i$. The initial states are determined by a distribution $\rho : S \to [0, 1]$. Each agent i aims to maximize its own total expect return $R_i = \sum_t^T = 0\gamma^t r_i^t$, where γ is a discount factor and T is the time horizon[8].

2.3 Policy Gradient (PG) Algorithms

Policy gradient methods provide a way for reinforcement learning algorithms to solve stochastic environments, whereas value-based methods (such as Q-Learning) can only solve deterministic environments. In deep Q-Learning, we have our agent learn a function which approximates the future discounted reward given a state and an action. With policy gradient methods, we train our agent's policy directly so that it produces a distribution over actions in which we can sample from.

The gradient for the policy can be written as[6]

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim p^\pi, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)]$$

where p^π is the state distribution. This formulation brings several practical algorithms, notably *actor-critic* algorithms [9] and, by extension, Proximal Policy Optimization [10], which is often used by OpenAI in their reinforcement learning work including OpenAI Five.

3 Optimal Asset Allocation

With our multi-agent scenario defined and our methods to solve them noted, we introduce the idea of optimal asset allocation. At a high level, the task of optimal asset allocation involves some limited resource and multiple assets for which we can distribute the resource. At every time step, a reward is produced based on the distribution, and the environment transitions to the next state based on it as well. This process resembles a Markov decision process, where in place of an action there is this allocation[11].

An example of such an environment in the real-world can be found in financial instrument trading and investing. Here, a portfolio manager acts as the agent and attempts to distribute their finite resources (e.g., cash) throughout a set of assets (e.g., stocks) so as to maximize reward (e.g., profit) while minimizing risk.

Neuneier[12] introduces a basic and controlled version of such an environment which we re-implement (with some minor adjustments) here. We construct an artificial exchange rate such that the state is given by the current exchange rate x and the agent's current position p. We limit the rate to the range $[0.05, 0.95]$ and give the rate an uptrend so that it is likely to move upwards on any given time step. We construct the environments state transition function so that, as the rate approaches it's maximal value, the likelihood of a drastic down tick becomes more likely. An example of the rate over time in such an environment can be found in Figure 1.

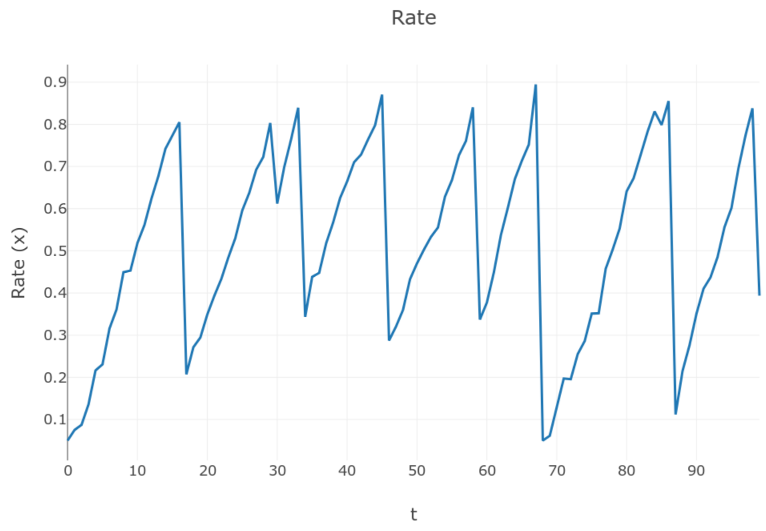

Figure 1: Sample Rate through Time Steps

Our agent can choose one of two actions at any given time step: *long* or *neutral*. If the agent is long, then they will receive a reward proportional to the change in rate from the previous time step to the current time step. If the agent is neutral, they receive a reward of 0. This roughly simulates a trading environment in that the agent will receive a positive reward if they are long while the rate is increasing, but will receive a negative reward if it goes down. Furthermore, the volatile nature of the environments transition probability function provides encouragement to go long when the value of the rate is low and low risk and to go neutral when it is higher and high risk. Figure 2 illustrates the actions taken by an agent taking random decisions in the above example episode as well as the agent's corresponding capital (or accumulated reward starting at 1).

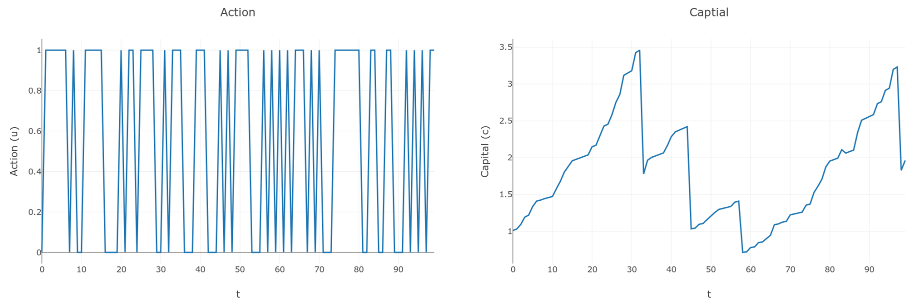

Figure 2: Results for a Random Agent

Next, we train our agent on this environment. In this instance, our agent uses Proximal Policy Optimization (PPO) to learn an optimal policy. We train over 1000 episodes where each episode consists of 100 time steps of our artificial rate exchange. We see that our agent is able to successfully learn over the course of training.

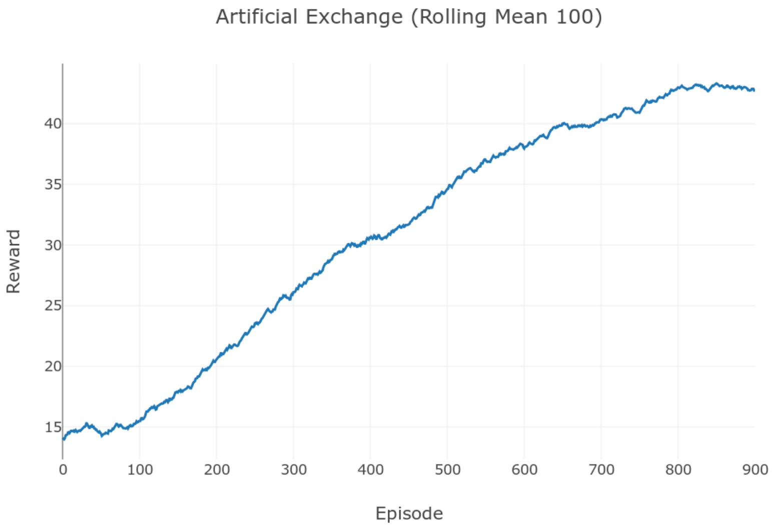

Figure 3: Episode Reward over the course of Training

With our trained agent, we can evaluate it's performance given a new instance of the environment.

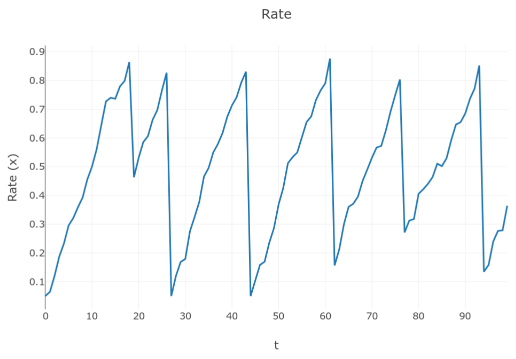

Figure 4: Rate during Trained Sample

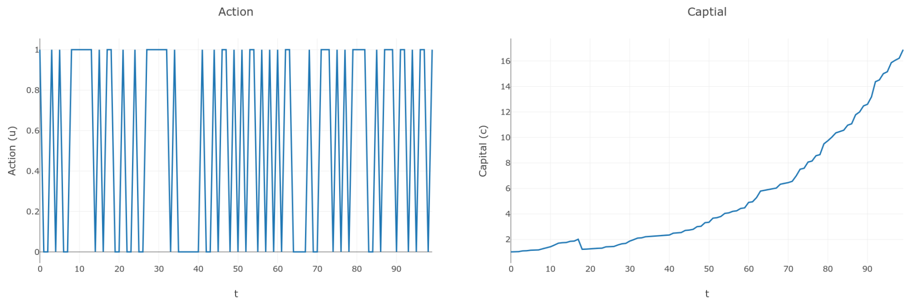

Figure 5: Results for a Trained Agent

The point of this experiment is to demonstrate that our agent is capable of learning the dynamics of an environment where the only action it takes is to either invest or not invest in an asset in order to maximize reward while minimizing risk. This approach can be extended to multiple assets, where the agent can choose to invest in one asset over another. This can be even further extended to invest part of the agent's available capital in one asset and another part in another. Policy gradient approaches have been shown to work well in such environments [13].

4 Multi-Agent Reinforcement Learning through Asset Allocation

We now re-introduce multi-agent concepts into our project and show that an agent capable of learning in an asset optimization setting (such as in the previous section) is able to apply such logic to solve multi-agent environments.

We construct an environment which facilitates multi-agent learning. We place some number of *mines* as well as some number of *agents* on a grid. At each time step, the agent can choose to move in one direction, which will result in them moving in either 1 or 2 units (selected at random). Each mine has a value associated with it ranging from 0 to 1. At each time step, the mine will gain some small value (e.g., $+0.05$), clipped at 1. If an agent positions itself to be on top of a mine, then the agent will receive a percentage of the mines value at the time step. When this occurs, the mine will lose a portion of its value (e.g., -0.1). This loss is exponential with the number of agents, so that two agents standing on the same mine will result in a loss of 0.3 (instead of just 0.1). The agents receive the position of itself, each mine, and each mine's corresponding value at each time step.

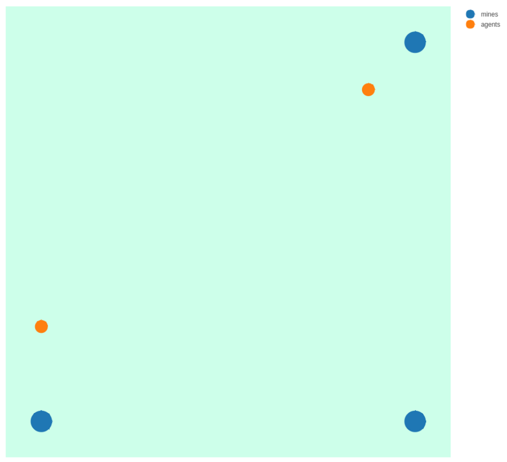

Figure 6: Snapshot of our Multi-Agent Environment

The goal is to have the agent's collectively maximize their reward over time. The environment is designed to encourage coordination between the agents, as a naive approach of simply navigating to the nearest mine or the mine with the highest value is sub-optimal.

We tested this scenario based on heuristic policies, namely evaluating each mine as a function of it's value and distance to the agent and moving towards the mine with the best evaluation. This improves results over the naive approach, but still results in sub-optimal play. In these instances, two agents will find themselves near the same mine, and will thus both navigate to it. When one (or both) of the agents reach the mine and sap it's resources, the agent's will then attempt to move to the next mine with the best evaluation. This is problematic, though, since these agents will be performing under the same policy and will thus attempt to navigate to the same mine. In doing so, the agent's are not leveraging coordination to maximize reward by splitting up and moving between mines so as to not waste one of their times.

We then introduce our allocation learning algorithm. We modify it so that it attempts to allocate the agents as resources to the mines, which act as assets. The allocation replaces the agent's evaluation of the mines, so that their decision making process is left intact while being able to be influences by our main allocation agent. Our main allocation agent is then able to optimize our agent's behavior by coordinating them such that they move themselves to positions that prevent the negative situations found earlier, and thus moving towards an optimal policy.

5 Conclusion

The work presented in this paper acts as a basis for future approaches to solving multi-agent environments. Although many single-agent reinforcement learning algorithms can be successfully adapted to solve multi-agent settings, these approaches fail to capitalize on the unique dynamics found in such a setting. By changing our perspective of a Markov decision process, we can consider other formulations which don't necessarily rely on actions per se.

In the case of asset allocation optimization, it is natural to refer to our agents as resources which we invest into assets to receive a return. In real life, this is the job of managers and coordinators, and this level of abstraction is essential to operating large and complex systems and organizations.

In future work, we wish to explore how different loss functions can be used to maximize reward while reducing profit, such as introducing something like a Sharpe ratio to limit risk while maximizing alpha in a trading environment. Along with this, it is worth exploring the hierarchical nature of such arrangements. In a business setting, for example, management is often layers so as to effectively organize resources from the bottom up. Not only this, but management decisions are made in a hierarchical nature *over time*, such as the case of someone planning budgets for monthly, quarterly, and annual time segments.

Overall, it would seem multi-agent reinforcement learning algorithms have many opportunities to be explored, and asset allocation optimization may prove to be another valuable tool set for the field.

References

[1] Christopher J C H Watkins and Peter Dayan. Q-Learning. Technical report, 1992.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2 2015.

[3] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning Dexterous In-Hand Manipulation. 8 2018.

[4] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 1 2016.

[5] OpenAI. OpenAI Five, 2018.

[6] Richard S Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. Technical report.

[7] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. Technical report.

[8] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. 6 2017.

[9] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning : an introduction*. MIT Press, 1998.

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. 7 2017.

[11] Ralph Neuneier and Ralph Neuneier. Optimal Asset Allocation using Adaptive Dynamic Programming. *IN ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, 8:952–958, 1996.

[12] Ralph Neuneier. Optimal Asset Allocation • uSIng Adaptive Dynamic Programming. Technical report.

[13] Zhipeng Liang, Hao Chen, Junhao Zhu, Kangkang Jiang, and Yanran Li. Adversarial Deep Reinforcement Learning in Portfolio Management. 8 2018.