Solving Markov Equilibrium Games by Multi-Agent Deep Reinforcement Learning

Yaolang Zhong*

Abstract

This study delves into the numerical resolution of a critical economic model category, Markov Equilibrium Games, employing a multi-agent deep reinforcement learning algorithm for strategy optimization. Specifically, it focuses on a duopoly context, resembling a Stackelberg game, where two firms engage in sequential decision-making each period. Characterized as model-free learners, these firms initially lack economic theoretical knowledge and develop optimal decision-making strategies solely through mutual interactions in simulations. Their policy functions are defined using neural networks, and training is executed via the Multi-Agent Deep Deterministic Policy Gradient (MA-DDPG) algorithm, a concept from deep reinforcement learning. The study aims to investigate if, under these conditions, the economy can attain an equilibrium where each firm's behavior is optimal. This exploration is set in a linear-quadratic framework, allowing for analytical derivation of the firms' optimal policy functions. The experimental findings indicate a nuanced outcome: the economy occasionally aligns with the analytical equilibrium, but diverges at times. These outcomes provide valuable insights for economists in refining model formulation and applying model-free numerical solutions in economic analysis.

1 Introduction

The economic questions are about how agents reasonably allocate resources to maximize their utilities. The economists build up various models within which they solve for analytical answers to justify how agents should behave optimally. While the mathematical expressions are elegant and straightforward, hence preferable, they are not always reachable, especially when problems and the applied models became more and more complex. Economists have

*Department of Economics, University of Warwick, Coventry CV4 7AL, UK. Email: yaolang.zhong@warwick.ac.uk. Acknowledgments: I am very grateful to my supervisor Mingli Chen for her invaluable guidance and support throughout this work. I am also very thankful to Eric Renault, Herakles Polemarchakis, Roger Farmer and Luis Candelaria for their helpful comments.

made sacrifices to keep the elegance, such as making assumptions that some functions take particular forms to simplify the calculation, or focusing on scenarios around the steady states so the system of equations can be linearized. But all those additional assumptions or ways for solutions reduce the generality and feasibility of the models.

Another issue arises when problems with high dimensional variables are encountered. Even though the economists seek numerical solutions, most of the existing methods have the computational burdens increase exponentially to variable dimensions. The latter issue seems to be more urgent to tackle in the past few years as we entered the so-called 'Big Data Era' where there is a massive amount of data in fields like macroeconomics, industrial organization, finance, etc.

While there have been many works to help in the two mentioned problems in economics, *the curse of mathematical model* and *the curse of dimensionality*, this paper aims to look for help in another fast-developing field under computer science and statistics, the Reinforcement Learning. Reinforcement learning approaches have been the core engines of many Artificial Intelligent achievements in recent years, and well-known applications include Go, Poker, and real-time strategic e-games.

Though unfamiliar to economists, the reinforcement learning approaches are not something completely new. In reinforcement learning problems, agents learn from the interaction with the environment and optimize their behavior to maximize/minimize their overall payoff/cost. In this sense, reinforcement learning shares the same original intuition with economics from optimal control theory. The key difference is that, while in optimal control and economics agents are assumed to have a perfect model characterizing the dynamics of the environment, and can therefore write down and solve a system of equations for optimal policies, in reinforcement learning such an assumption is dropped. The principal theme of reinforcement learning is how the agents gradually optimize their policies through pure trial-and-error (learning) or building up their subjective model (planning). The most typical example is the Dynamic Programming approach, an essential tool in economics and optimal control that solving the system from the Bellman equations but not commonly used in reinforcement learning applications, as discussed in Section 2.

In general cases, the reinforcement learning approaches seeks for numerical solutions of optimal policy, corresponding to the curse of mathematical model problem and it is discussed in Section 2. In high-dimensional continuous space cases, it combines with the function approximation techniques from machine learning literature, and it is hence called deep learning, to approximates the true solutions, corresponding to the curse of dimensionality problem. This is discussed in Section 3.

Besides the potentials of solving the two mentioned problems, I also consider deep reinforcement learning promising in increasing the effectiveness for economists to pin down the model settings when building their models. In particular, to describe and solve a real-world problem, an economist would generally build a primary model, solve it manually to see

whether the agents' optimal behaviors in current settings help demonstrating the underlying problems, and if not then keep modifying the building blocks of the models until the answer is Yes. Implementing deep reinforcement learning approaches would ideally find agents' optimal behaviors and thus the equilibrium of the model much faster than the manual work of economists, especially when the model is complex. In this way the deep reinforcement learning approaches can be the tools for economists in the iterations of model establishment.

There has been a growing literature in recent years applying deep reinforcement learning in the fields of portfolio management, trading strategy and online pricing. While the existing literature is under the single-agent learning framework, this paper is the first work to implement multi-agent deep learning methods to traditional economic problems. Specifically, we implement the MA-DDPG algorithm [Lowe et al., 2017] to solve for a classical duopoly model in economics where a Markov Perfect Equilibrium exists. While the analytical solution based on dynamic programming relies heavily on the assumption of the Linear-Quadratic Regulator, the algorithm can be extended to any general function form in the model.

The rest of this paper is organized as follows. In Section 2 I go through the related literature about the implementation of deep reinforcement learning approaches in finance and economics. In Section 3 I introduce some backgrounds of reinforcement learning. In section Section 4 I briefly go through some state-of-the-art deep learning approaches, including the MA-DDPG algorithm implemented in this paper. In section Section 5 I discuss about my experiment on a duopoly model. In section Section 6 I discuss about some future extensions of my work. Section Section 7 is the conclusion.

2 Literature Review

There have been a few attempts to implement the state-of-art deep reinforcement learning algorithms, which would be introduced in Section 4, in financial investment and portfolio management. For instance, Xiong et al. [2018] used the vanilla DDPG to optimize stock trading strategy in a Dow Jones training environment. Li et al. [2019a] proposed a Adaptive-DDPG algorithm incorporating optimistic or pessimistic deep reinforcement learning that is reflected in the influence of prediction errors. See also Azhikodan et al. [2019] and Jiang and Liang [2017].

An interesting finding is that the more advanced learning approaches do not necessarily perform better in the investment environment(Li et al. [2019b], Liang et al. [2018]). One reason should be that the dynamic environment in the financial market is too non-stationary and complicated such that the current single-agent deep learning approaches are not enough to capture the insights no matter how they are theoretically more advanced in detailed implementation. This actually calls for multi-agent deep learning framework implemented in this paper in future work.

Some recent works augmenting the deep learning algorithm with theoretical models may provide some inspiration. For example, Yu et al. [2019] trained a trading agent using a novel learning architecture consisting of an infused prediction module, a generative adversarial data augmentation module, and a behavior cloning module. Other applications of deep reinforcement learning closed to economic questions include bidding(Zhao et al. [2018]) and pricing Liu et al. [2019].

3 Background

In this Section I briefly introduce some backgrounds of reinforcement learning approaches to researchers who are not familiar to this topic. I would start from Dynamic Programming, which is classified as the model-based, bootstrapping and expected updating reinforcement learning approach. I then introduce Monte-Carlo, the conversely model-less, no-bootstrapping and sample updating approach, and Temporal Difference, the key novelty in reinforcement learning approach that combines virtues from Dynamic Programming and Monte-Carlo. In next Section I would introduce the state-of-art approaches implementing in continuous or high dimensional cases, which can be considered as the generalized Temporal Difference algorithm with function approximation techniques.

3.1 Markov Decision Processes

Reinforcement learning solves the problems of Markov Decision Processes(MDPs), which operates in the manner that in each period the agent observes state $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$ based on a policy $\pi \in \mathcal{S} \to \mathcal{A}$, then the environment produces a reward r_{t+1} and state transition s_{t+1} as feedback.

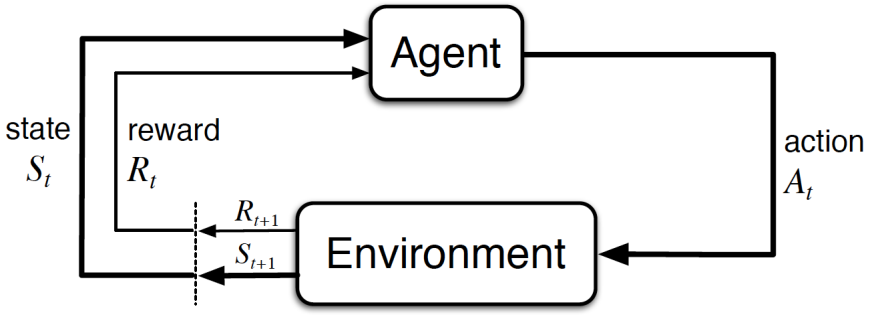


Figure 1: The agent–environment interaction in a Markov decision process

What matters to the agent is the overall return G_t, which is defined as the discounted

sum of rewards. We can then define the value function for state s given a policy π as:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \tag{1}$$

With the value function, an optimal policy π_* with corresponding value function $v_*(s)$ such that $v_*(s) \geq v_\pi(s)$ for all $s \in \mathbb{S}$ and $\pi \neq \pi_*$ is pursued. However, the true value function (1) is generally unknown. Most reinforcement learning algorithms are about how to estimate the value function to obtain an optimal policy. Note that for notation, $V(\cdot)$ refers to the estimate and $v(\cdot)$ refers to the true value.

3.2 Revisit: Dynamic Programming

Let's start with the technique to which economists are mostly familiar. Dynamic Programming(DP) is used with the assumption of a perfect environment model so that given a particular state and an action, the distribution of probable reward and the transition to the next state is explicitly known. In this case, we can use the estimated values of successor states as the backup of the current state. This ideal is illustrated in the backup diagram below, where a white node denotes a state and a black node denotes an action. The value of current state s is equal to the values of all possible next states weighted by the probabilities of actions under a general policy π and the probabilities of transitions given a particular action. If the policy is greedy, then the value of current state is equal to the maximum values among actions.

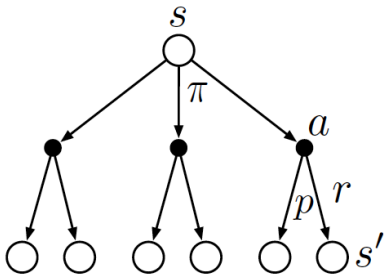


Figure 2: Backup diagram for v_π(Sutton and Barto [2018])

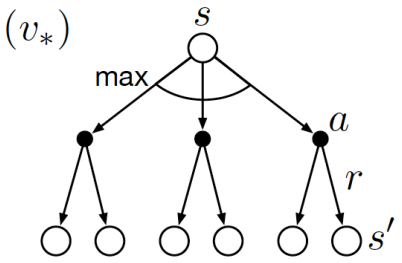


Figure 3: Backup diagram for v_*(Sutton and Barto [2018])

Policy Iteration v.s. Value Iteration

There are generally two basic alternatives to to solve a DP problem and find an optimal policy. The first is policy iteration, whose intuition is straight-forward as that with a

randomly given initial policy, we first iteratively do the policy evaluation

$$V_{(s)} \doteq \sum_a (a|s) \sum_{s',r} p(s',r|s,a)[r+V_{(s')}] \quad (2)$$

to obtain the state value function under the policy, and then do the control process as being one-step ahead greedy:

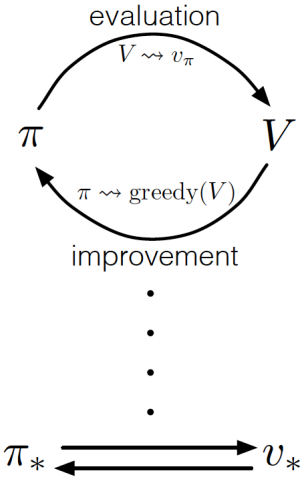
$$(s) \doteq \arg\max_a \sum_{s',r} p(s',r|s,a)[r + V_{(s')}] \quad (3)$$

The second is value iteration, which can be thought as combining policy evaluation and improvement into one operation in each iteration:

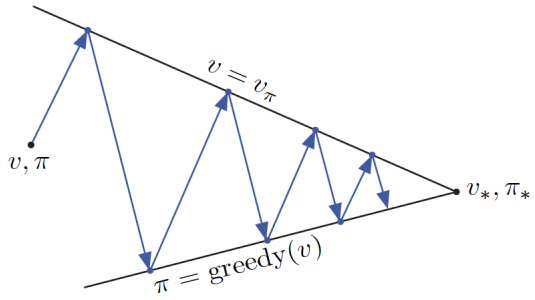
$$V_{k+1}(s) \doteq \max_a \sum_{s',r} p(s',r|s,a)[r + V_k(s')] \quad (4)$$

Generalized Policy Iteration

The generalized intuition behind policy iteration and value iteration is that there are two interacting processes, competing and cooperating with each other, which is termed generalized policy iteration. One is policy evaluation, which makes the value function consistent with the current policy. The other is policy improvement, which makes the policy function greedy to the current value function.



The idea is that no matter how these two forces proceed alternatively since the two goals are not orthogonal, they are driving each other and moving together towards common goals — being stable in the optimal case where the Bellman optimality equation hold.



Q function

The DP approach has three properties, which then connects to three kinds of partition in various approaches in Reinforcement Learning. The first is that DP requires a full model to characterize the environment, so it is called a model-based method. The model-free method then has entirely no assumption about what agents know about the environment. The second is that in DP, we update the current state's estimate by the estimates of state values of other states, which is called bootstrapping. The third is that the update of the estimate of the current state is given by all other potential next states' estimates, weighted by the probability of transition, so it is an expected update. In reality, this is not common, both because we do not have the full transition channel among the states, and because even if we have, it is generally computationally-expensive. On the contrary, most commonly used methods belong to (at least partially) sample updates.

In Reinforcement Learning, most of the time people are more interested in the value-action function, or Q function, $Q(s, a)$, which denote the value of choosing action a in current state s and follow the policy thereafter. The idea is that, in other sample update methods described below, it is not that easy to write down the formula of state value function but the state-action value function.

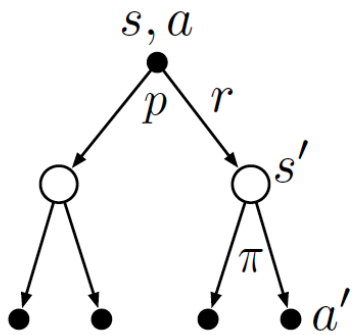


Figure 4: Backup diagram for q_π(Sutton and Barto [2018])

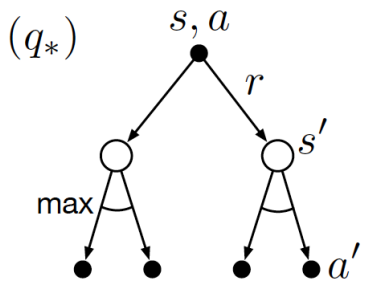


Figure 5: Backup diagram for q_*(Sutton and Barto [2018])

3.3 Monte-Carlo

The Monte-Carlo(MC) method,to the contrast, requires no complete knowledge of the environment, but only experience. Technically, for policy evaluation, in each episode we generate a sequence following policy π: $S_0, A_0, R_1, S_1, A_1 ... S_{T-1}, A_{T-1}, R_T, S_T$, then for each particular action-state pair (S_t, A_t), its sample return is
$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, then we update the Q function:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t)) \tag{5}$$

with some specified learning rate α. Then for the control problem we update the policy to be greedy to the new Q function.

The MC method can be viewed as one that is in the opposed direction to DP. As we said it requires only experience so it is a model-free method; the estimate of return consist complete of real rewards without other estimates, so Monte-Carlo does not bootstrap; finally, it learns from sample returns from episodes so it is a sample update. The diagram below illustrates the differences. Since in each sample experience there is only one sequence of state-action pair, so only the value of next state-action pair can be used as the target to calculate the current pair.

Figure 6: backup diagram MC(Sutton and Barto [2018])

The key to any sample update method is to ensure state space exploration to guarantee the convergence of estimate towards true value asymptotically. It contradicts the nature of policy improvement, in which case being greedy to the value function means prefer some actions that look promising currently. This contradiction is known as the exploration-exploitation trade-off, a persistent problem in reinforcement learning. There are multiple ways to help release the problem, such as starting each episode with a random value-action pair or learning a so-call ϵ-greedy policy, which means always take a random action with a small probability ϵ. We skip the discussion here.

3.4 Temporal-Difference

Temporal-Difference(TD) learning is the most novel and central idea in reinforcement learning, combining both the advantages of DP and MC. TD update estimates partly based

on other estimates so it bootstraps as DP. Bootstrapping generally induces fast learning because the value function is updated during the experience, while in MC, the value function is updated as long as a sample experience is finished so the sample return G_t is formed. also it learn directly from experience so no knowledge is needed for learning as in MC. The most widely used simple one-step TD method of Q function, known as Q-learning(Watkins and Dayan [1992]), make the update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{6}$$

Q-learning is an off-policy learning, which means the policy we are evaluating is not the same as the policy used for exploration in sampling. In Q-learning, we evaluate a greedy policy while using the other policy, such as a $\epsilon-$greedy one for sampling. As we will see, most of the state-of-art algorithms are based on Q-learning, with just additional work for approximation in continuous spaces cases.

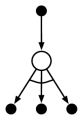


Figure 7: backup diagram Q-learning(Sutton and Barto [2018])

The figure below illustrates all discussed so far, expanding from DP to general reinforcement learning methods. The most naive and straightforward way for reinforcement learning, as in the lower right corner, is exhaustive search, which is surely time-consuming and mostly infeasible. From left to right in the horizontal axis, the width of update increases from using a single sample return as the target to using samples' expectation as the target. From top to bottom in the vertical axis, the depth of update increases from bootstrapping just one-step ahead to infinite-step ahead, which means the target consists of only real returns without any estimates, equivalently no bootstrapping.

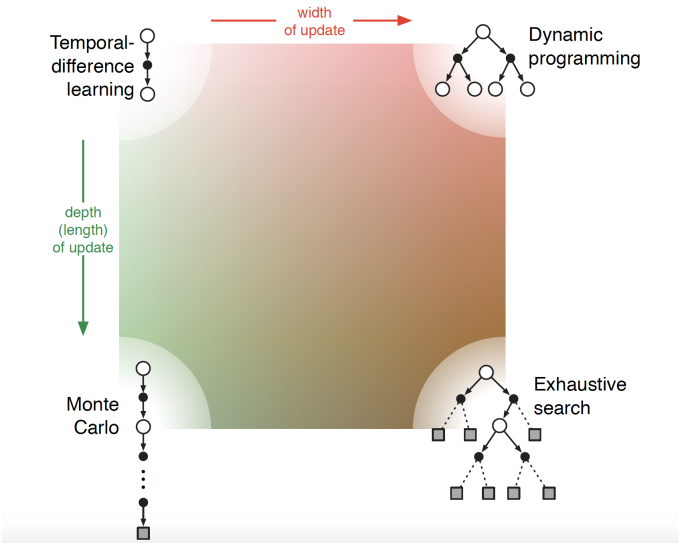


Figure 8: A slice through the space of reinforcement learning methods(Sutton and Barto [2018])

4 Deep Reinforcement Learning

In this section, I continue discussing reinforcement learning approaches from the last part by expanding to state-of-art deep reinforcement learning methods. These methods emerged from the past five years and are the main reason why reinforcement learning prevails in different fields.

4.1 Continuous State Space: Deep Q Network

Whenever there is a continuous or high dimensional state space, it is reasonable to approximate the value function in a parameterized functional form with weight vector $\theta^{\mathbf{Q}} \in \mathbb{R}^d$. The parameters then adjust to fit the true value. It is then connected to supervised learning in machine learning or regression in econometrics, with two questions as what are the features characterizing the state and what is the target value to update the parameters.

While there are various function approximation methods answering the first question, I introduce here the prevailing Artificial Neural Networks(ANNs). A network is a hierarchical system composed of several layers, each of which are made of nodes where computation happens, loosely patterned on a neuron in the human brain that fires when it encounters sufficient stimuli. A node combines input from the data with a set of weights. The input-weight products are summed and then passed through a node's so-called activation function to be non-linearized, as shown below.

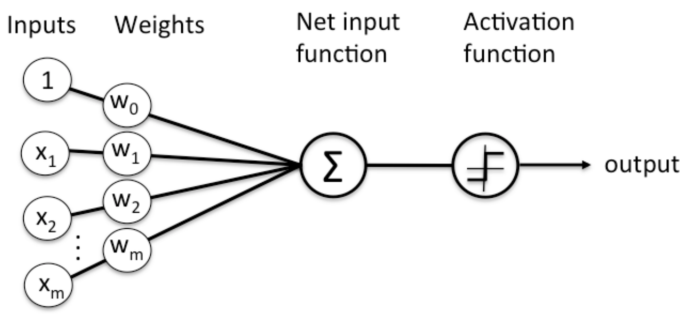


Figure 9: Diagram of a node

Training the hidden layers of an ANN is, therefore, a way to automatically create features appropriate for a given problem so that hierarchical representations can be produced without relying exclusively on hand-crafted features. The latter has been an enduring challenge for artificial intelligence and explains why learning algorithms for ANNs with hidden layers have received so much attention over the years.(Sutton and Barto [2018])

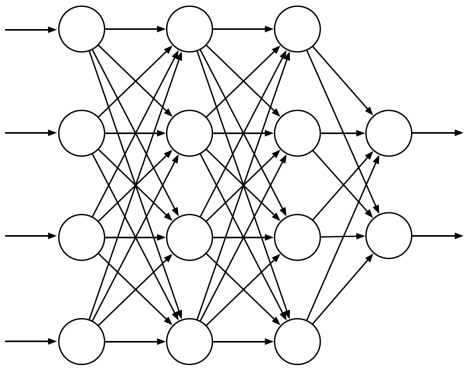


Figure 10: A generic feed-forward ANN with four input units, two output units, and two hidden layers.

The justification of ANNs is that given enough layers of neurons which extracts information from inputs by taking weighted sum and non-linear transformation, the ANNs can approximate any function that satisfies some moderate properties, which is well-known as Universal approximation theorem. A version of this with regard to the ReLU activation function $f(x) = \max\{x, 0\}$ is given by (Lu et al. [2017]):

[Universal approximation theorem] (L1 distance, ReLU activation, arbitrary depth) For any Lebesgue-integrable function $f : \mathbb{R}^n \to \mathbb{R}$ and any $\epsilon > 0$ there exists a fully-connected ReLU network $\mathcal{A}$ with width $d_m \leq n + 4$, such that the function $F_{\mathcal{A}}$ represented by this network satisfies

$$\int_{\mathbb{R}^n} |f(x) - F_{\mathcal{A}}(x)| \, \mathrm{d}x < \epsilon$$

Then all the weights and bias in the network is the parameters to be optimized, for which there is a general method called Stochastic gradient-descent(SGD) that adjusts the parameters after each example by a small amount in the direction that would most reduce the error on that example.

$$\theta_{t+1}^{Q} \doteq \theta_t^{Q} - \frac{1}{2}\alpha\nabla[U_t - \hat{Q}(S_t, A_t; \theta_t^{Q})]^2 \tag{7}$$

where U_t denotes the unknown true value, to estimate which, as for the second question mentioned above, it requires advanced technique generalize from methods briefly discussed in Section 2. For example, the MC algorithm uses the episodic accumulated rewards G_t to estimate the true value. Since the true value is the expected value, the Monte Carlo target is, by definition, an unbiased one, so locally optimal solution is guaranteed.

However, the guarantee is not obtained if one wants to take advantage of bootstrapping, which generally induces significantly faster learning. To see this, note that the bootstrapping target, for example, the prototypical TD(0) target as $U_t \doteq R_{t+1}+\gamma\hat{Q}(S_{t+1}, A_{t+1}; \theta_t^{Q})$, have its value depends on the current weight vector θ_t^{Q}. When updating θ_t^{Q} based on the gradients, the target is updated as well. This may result in instability or even divergence.

Another cause of instability when using ANNs or other non-linear approximator roots from the nature of reinforcement learning comparing to machine learning. While in machine learning our observations are independently distributed, in reinforcement learning the sequence of observations, as accumulated in episodic exploration, is correlated. In some cases, the small updates to Q function may significantly change the policy and the data distribution.

The prevailing Deep Q-Network method(Mnih et al. [2015]), besides its sophisticatedly designed network architecture and tuned hyper-parameters, addresses these instabilities with two key ideas. For the first cause, it builds a target network additionally to approximate the target value, whose parameters are then updated periodically or softly with a small learning rate towards the main Q-network. For the second cause, it separates the timing of experiencing and learning from a particular transition, therefore switches from online learning to offline learning, by using a biologically inspired mechanism termed experience replay. Technically, an experienced transition is stored into the experience replay with some capacity instead of being learned intermediately. When learning, a mini-batch of transitions is randomly sampled from the experience replay. Hence the correlation among observations is reduced.

4.2 Continuous Action Space: DDPG

While Deep Q-Network solves problems with high-dimensional observation spaces, it can only handle discrete and low-dimensional action spaces. A natural idea to deal with the

action spaces is to learn a parameterized policy as well. Consider the case of deterministic policy, as in most of the cases of economics, so the policy is a function $\mu : \mathcal{S} \to \mathcal{A}$ parameterized by vector θ^μ. The method to do so is called policy-gradient, or deterministic policy-gradient(DPG) when dealing with deterministic policy, which updates the parameters to maximize some performance measure $J(\theta^\mu)$, so their updates approximate gradient ascent in J:

$$\theta^\mu_{t+1} = \theta^\theta_t + \alpha \widehat{\nabla J(\theta^\mu_t)} \tag{8}$$

where $\widehat{\Delta J(\theta^\mu_t)}$ is a stochastic estimate.

Methods that learn approximations to both policy and value functions are called actor-critic methods, in which 'actor' refers to the learned policy, and 'critic' refers to the learned value function. This DPG algorithm(Silver et al. [2014]) then use the critic approximated by θ^Q as the performance and applies the chain rule to update the actor:

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s|\theta^\mu)}] \\ &= \mathbb{E}[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_t}] \end{aligned} \tag{9}$$

The **Deep Deterministic Policy-Gradient(DDPG)** method (Lillicrap et al. [2015]) modifies the DPG algorithm stated above with neural network function approximator inspired by Deep Q-Network. As in DQN, a replay buffer is used to store and sample transitions (s, a, r, s'). As long as the replay buffer was full, the oldest samples were discarded. At each time step, both the actor and critic are updated by sampling a minibatch uniformly from the buffer. Also, now copies of both the actor and critic are needed as the target networks to obtain $\mu'(s|\theta^{\mu'})$ and $Q'(s, a|\theta^{Q'})$ respectively.

Additionally, to encourage exploration, an exploration policy $\tilde{\mu}$ is constructed by adding noise from some process $\mathcal{N}$ to the actor policy:

$$\tilde{\mu}(s_t) = \mu(s_t|\theta^\mu_t) + \mathcal{N} \tag{10}$$

4.3 Multi-Agent Learning: MA-DDPG

While most of the achievements of reinforcement learning have been in single-agent domains, there are emerging interests in applications involving multiple-agents interactions, cooperative or competitive. A naive method is to apply traditional approaches, such as the DQN and DDPG stated above, into the multi-agents framework, by simply regards other agents as parts of the environment being interacted with. The issue is that as each agent's policy evolves as the training process, the environment becomes non-stationary from the perspective of any single agent. Because of this, the straight-forward use of the replay buffer for training is misleading.

For example, in some time an agent experienced and recorded the transition s, a, r, s' in

the replay buffer. A few steps later it samples this transition from the buffer for learning, but now the environment has been changed and the same state-action pair may lead to a totally different reward and next state since other agents' policies have been updated. In summary, for agent i in a N-agent environment:

$$\mathbf{Pr}(r_i, s_i'|s_i, a_i; \mu_1, ..., \mu_N) \neq \mathbf{Pr}(r_i, s_i'|s_i, a_i; \mu_1', ..., \mu_N') \tag{11}$$

for any $\mu_i \neq \mu_i'$

The **Multi-Agent Deep Deterministic Police Gradient(MA-DDPG)** algorithm(Lowe et al. [2017]) solves the issue by adopting the framework of centralized training with decentralized execution. In particular, it allows agents to use extra information for policy training so long as this information is not used at test time.

Formally speaking, consider a Markov game with N agents. There is a set of states $\mathcal{S}$, a set of actions $\mathcal{A}_1, ..., \mathcal{A}_N$ and a set of observations $\mathcal{O}_1, ..., \mathcal{O}_N$ for each agents. Each agent i use a parameterized deterministic policy $\mu_{\theta_i^\mu} : \mathcal{O}_i \to \mathcal{A}_i$ with the private observation $\mathbf{o}_i : \mathcal{S} \to \mathcal{O}_i$ as input and obtain a reward $r_i : \mathcal{S} \times \mathcal{A}_\rangle \to \mathbb{R}$. The state transition is a function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 ... \times \mathcal{A}_N \to \mathcal{S}$.

The MA-DDPG algorithm uses a centralized action-value function $Q_i^\mu(\mathbf{x}, a_1, ..., a_N|\theta_1^Q, ..., \theta_N^Q)$ as critic, which takes as input the actions of all agents $a_1, ..., a_N$ and some state information $\mathbf{x}$. A simplest case would be that $\mathbf{x} = (o_1, ..., o_N)$. A replay buffer $\mathcal{D}$ is used as in the cases of DQN and DDPG, but now it collects the experience of all agent by storing the tuple $\mathbf{x}, \mathbf{x}', a_1, ..., a_N, r_1, ..., r_N$ for each transition. The actor is updated with the gradient:

$$\nabla_{\theta_i^\mu} J(\mu_i) = \mathbb{E}_{\mathbf{x}, a \sim \mathcal{D}}[\nabla_{\theta_i^\mu} \mu_i(a_i|o_i) \Delta_{a_i} Q_i^\mu(\mathbf{x}, a_1, ..., a_N|\theta_1^Q, ..., \theta_N^Q)|_{a_i = \mu_i(o_i)}] \tag{12}$$

The critic is updated with the gradient:

$$\nabla \mathcal{L}(\theta_i^Q) = \nabla \mathbb{E}_{\mathbf{x}, a, r, \mathbf{x}'}[(Q_i^\mu(\mathbf{x}, a_1, ..., a_N|\theta_1^Q, ..., \theta_N^Q) - y)^2] \tag{13}$$

with the target as

$$y = r_i + \gamma Q_i'^\mu(\mathbf{x}', a_1', ..., a_N'|\theta_1'^Q, ..., \theta_N'^Q)|_{a_j' = \mu_j'(o_j)} \tag{14}$$

where $\mu' = \{\mu_{\theta_1'}, ..., \mu_{\theta_N'}\}$ is the set of target policies.

Here, the centralized training is implemented with the implicit assumption that each agent knows globally all that other agents know when training: their observations, actions, rewards, and even the policies. This assumption is reasonable, so long as the agent has only his local information when executing the action based on policy. Such an assumption does not go beyond the rationality assumption in economics. The assumption of knowing others' policies can even be removed by letting each agent infer policies of others from their

actionsLillicrap et al. [2015], but we remain in our experiment for simplicity.

The primary motivation behind MA-DDPG is that if an agent knows the actions taken by all agents then the environment is stationary from its perspective even the policies are changing:

$$\begin{aligned}\mathbf{Pr}(r_i, s_i'|s_i, a_1, ..., a_N; \mu_1, ..., \mu_N) &= \mathbf{Pr}(r_i, s_i'|s_i, a_1, ..., a_N) \\ &= \mathbf{Pr}(r_i, s_i'|s_i, a_1, ..., a_N; \mu_1', ..., \mu_N')\end{aligned} \tag{15}$$

for any $\mu_i \neq \mu_i'$.

5 Experiment

In this section, I implement the MA-DDPG algorithm to solve a Markov Perfect Equilibrium(MPE) game. The MPE is used to describe settings where multiple decision-makers interact non-cooperatively over time. Particularly, in each time step, all agents have common observations over the state and the transition of which is affected by all agents' actions. Each agent thus has to take into account others' policies to maximize its payoff. The MPE describes a situation where no agent wishes to change its policy, taking as given others' policies. The MPE game has been studied widely in economics in past years. Well known examples include topics in IO such as choice of price, output, location or capacity for firms(Ericson and Pakes [1995]; Ryan [2012]; Doraszelski and Satterthwaite [2010]) and topics in environmental economics like rate of extraction from a shared natural resource(Levhari and Mirman [1980]; Van Long [2011]).

In the following part, I first introduce the duopoly model aimed to solve, with a specified Linear-Quadratic form assumption, so an analytical solution existsSargent and Stachurski [2015]. Details of the analytical solution are in the Appendix. While the experiment tackles a benchmark low-dimensional problem as an exercise, the intuition is not trivial, and the MA-DDPG algorithm is superior mainly in two aspects of generalization:

- While the analytical solution relies heavies on the Linear-Quadratic form assumption, the MA-DDPG algorithm can be implemented without such an assumption.
- For other MPE games with a different number of heterogeneous agents and economic settings, the MA-DDPG algorithm can be easily implemented with changes of just the network architecture and hyper-parameters.

I then compare the performance of our numerical solution to the analytical solution and show that the numerical solution has a satisfactory approximation.

5.1 The Duopoly Model

Consider the case where two firms, 1 and 2, are the only producers of a particular goods. The demand is governed by a linear inverse demand function:

$$p = a_0 - a_1(q_1 + q_2) \tag{16}$$

with $a_0 > 0, a_1 > 0$

Both firms face the quadratic adjustment cost function with the same parameter $\gamma > 0$, so if firm i's current production is q_i and it aims to produce $\hat{q}_i$ in next period, its payoff is:

$$\pi_i = pq_i - \gamma(\hat{q}_i - q_i)^2 \tag{17}$$

The objective of the firm i is to maximize

$$\sum_{t=0}^{\infty} \beta^t \pi_{it} \tag{18}$$

Firm i chooses a decision rule that sets next period quantity $\hat{q}_i$ as a function f_i of the current state (q_i, q_{-i}). Given f_{-i}, the Bellman equation of firm i is

$$v_i(q_i, q_{-i}) = \max_{\hat{q}_i} \{\pi_i(q_i, q_{-i}, \hat{q}_i) + \beta v_i(\hat{q}_i, f_{-i}(q_{-i}, q_i))\} \tag{19}$$

It is not straightforward to solve the Bellman equation since it includes agent i's expectation on the other agent's strategy.

5.2 Numerical Solution: MA-DDPG

Inspired by OpenAI Gym, an online platform for researchers to test the performance of their new algorithms, in implementation I wrap the economic model into a packaging environment characterized by a_0, a_1, γ, β and number of firms n, which in this experiment is 2. The OpenAI Gym consists of various games wrapped into APIs with actions as inputs and new states and rewards as outputs.

The environment can be initialized with two randomly selected productions such that the initial price would not be negative, or with two specified initial productions for testing. In each step of a game, the MA-DDPG algorithm would solve for each agent its action v_1, v_2 given the state (q_1, q_2). Then the environment outputs rewards (r_1, r_2) and new states $(q_1' \doteq q_1 + v_1, q_2' \doteq q_2 + v_2)$ as feedback. The game would stop and restart as long as the price or any production is below zero. Below is the detailed algorithm for the MA-DDPG(Lowe et al. [2017]) implementation.

Algorithm 1 Multi-Agent Deep Deterministic Policy Gradient for N agents(Lowe et al. [2017])

for episode = 1 to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    For each agent i, select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t$ with respect to the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}_j, a_j, r_j, \mathbf{x}'_j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma Q_i^{\mu'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)|_{a'_k = \mu'_k(o_k^j)}$
      Update critic by minimizing the loss:
      $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\mu}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:
      $\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i}\mu_i(o_i^j)\nabla_{a_i} Q_i^{\mu}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)|_{a_i = \mu_i(o_i^j)}$
    end for
    Update target network parameters for each agent i:
    $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
  end for
end for

Because in this game the MPE is not the case of achieving highest accumulated rewards for each agent, but rather the monopoly solution, I do not use the rewards as indicator of performance. For testing, I specify a case with initialized productions $q_1 = 1, q_2 = 1$ and compare how the price and sum of productions would evolve to the case of analytical solution:

While the approximation is not that perfect, I think there are two reasons:

- The duopoly model is low-dimensional and in simple functional form, so the neural network approximator is easily over-identified and thus unstable.
- I am using a vanilla algorithm. In practice, more sophisticated techniques are used to increase and stabilize the performance, such as Batch Normalization, Penalty, and methods for tuning the hyper-parameters.

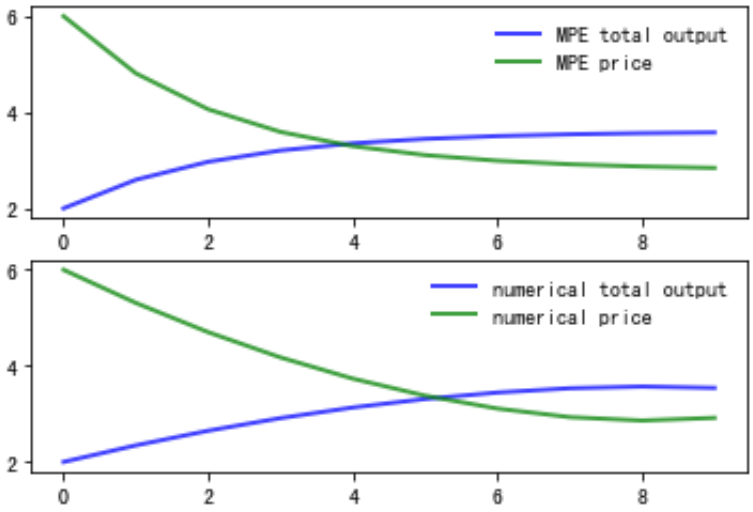


Figure 11: performance of MA-DDPG algorithm with 2 hidden layers of 4 neurons each for both the actor and critic. The learning rate of both actor and critic is 1×10^{-5} and the learning rate of target networks soft updating is 1×10^{-2}

.

6 Future Extensions

While this paper is the first work to implement a multi-agent deep learning algorithm to solve traditional economics questions, there are several extensions that I think are worth for explorations in future works.

- As I analyzed in section 5, more sophisticated pre-training or network architecture designing techniques would surely increase and stabilize approximation performance. There is abundant literature on deep learning that can be referred to.
- Implement the algorithm to economic models with higher dimensional variables or higher complexity where analytical solutions are infeasible, in which case the advantages of our new methods are better shown.
- Go deeper into the multi-agent learning algorithms to solve economic games in different forms. For instance, games with sequential moves, or games with partially observed state variables.
- Combine the learning algorithms with data-driven models to implement the algorithms with agents training from real-world data. Learning from reality and solving real problems is certainly the goal every economist is pursuing.

7 Conclusion

This paper introduces a new approach from the fast-developing reinforcement learning literature, the multi-agent deep deterministic policy gradient (MA-DDPG) algorithms from Reinforcement Learning literature, to solve numerically one of the cornerstone problems in economics, the Markov Equilibrium games. I first argue that traditional approaches in solving economic problems suffer from the curse of the mathematical model and the curse of dimensionality. Since the reinforcement learning problems share the same goal with economic problems as maximizing agents' overall payoffs or minimizing the overall costs, I infer various approximation techniques in the former may help solve the latter's mentioned problems. Additionally, I consider our new approach a promising method to help increase the effectiveness of economic model establishment.

I then briefly discuss the backgrounds, including Dynamic Programming, Monte-Carlo, and Temporal Difference to the state-of-art deep reinforcement learning approaches, including Deep Q-Network, DDPG, and MA-DDPG, step-by-step to reach the approach implemented in this paper. My experiment shows that the MA-DDPG algorithm can approximate the optimal performance in the testing case quite well, though not perfect. Then I go further to discuss the reasons and look into the potential future extensions.

References

Akhil Raj Azhikodan, Anvitha GK Bhat, and Mamatha V Jadhav. Stock trading bot using deep reinforcement learning. In *Innovations in Computer Science and Engineering*, pages 41–49. Springer, 2019.

Ulrich Doraszelski and Mark Satterthwaite. Computable markov-perfect industry dynamics. *The RAND Journal of Economics*, 41(2):215–243, 2010.

Richard Ericson and Ariel Pakes. Markov-perfect industry dynamics: A framework for empirical work. *The Review of economic studies*, 62(1):53–82, 1995.

Zhengyao Jiang and Jinjun Liang. Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent Systems Conference (IntelliSys)*, pages 905–913. IEEE, 2017.

David Levhari and Leonard J Mirman. The great fish war: an example using a dynamic cournot-nash solution. *The Bell Journal of Economics*, pages 322–334, 1980.

Xinyi Li, Yinchuan Li, Yuancheng Zhan, and Xiao-Yang Liu. Optimistic bull or pessimistic bear: adaptive deep reinforcement learning for stock portfolio allocation. *arXiv preprint arXiv:1907.01503*, 2019a.

Yuming Li, Pin Ni, and Victor Chang. An empirical research on the investment strategy of stock market based on deep reinforcement learning model. In *COMPLEXIS*, pages 52–58, 2019b.

Zhipeng Liang, Hao Chen, Junhao Zhu, Kangkang Jiang, and Yanran Li. Adversarial deep reinforcement learning in portfolio management. *arXiv preprint arXiv:1808.09940*, 2018.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Jiaxi Liu, Yidong Zhang, Xiaoqing Wang, Yuming Deng, and Xingyu Wu. Dynamic pricing on e-commerce platform with deep reinforcement learning. *arXiv preprint arXiv:1912.02572*, 2019.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pages 6379–6390, 2017.

Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in neural information processing systems*, pages 6231–6239, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Stephen P Ryan. The costs of environmental regulation in a concentrated industry. *Econometrica*, 80(3):1019–1061, 2012.

Thomas Sargent and John Stachurski. Quantitative economics with python. Technical report, Technical report, Lecture Notes, 2015.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

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

Ngo Van Long. Dynamic games in the economics of natural resources: a survey. *Dynamic Games and Applications*, 1(1):115–148, 2011.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522*, 2018.

Pengqian Yu, Joon Sern Lee, Ilya Kulyatin, Zekun Shi, and Sakyasingha Dasgupta. Model-based deep reinforcement learning for dynamic portfolio optimization. *arXiv preprint arXiv:1901.08740*, 2019.

Jun Zhao, Guang Qiu, Ziyu Guan, Wei Zhao, and Xiaofei He. Deep reinforcement learning for sponsored search real-time bidding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1021–1030, 2018.

Appendix

Analytical Computation: The LQ Framework

Here we briefly introduce how the analytical solution is conducted(Sargent and Stachurski [2015]). Since the the motion of dynamics is in linear form and the payoff function of agents are in quadratic form, it is the LQ regulator problem. Generally, in a Linear-Quadratic dynamic game, the agent i takes $\{u_{-it}\}$ as given and minimizes

$$\sum_{t=0}^{\infty} \beta^t \left\{ x_t' R_i x_t + u_{it}' Q_i u_{it} + u_{-it}' S_i u_{-it} + 2 x_t' W_i u_{it} + 2 u_{-it}' M_i u_{it} \right\} \tag{20}$$

while the state evolves according to

$$x_{t+1} = A x_t + B_1 u_{1t} + B_2 u_{2t} \tag{21}$$

Here x_t is the state vector and u_t is the control vector. In LQ framework the agent's optimal decision rule is linear to the state: $u_{it} = -F_{it} x_t$.

Taking $u_{2t} = -F_{2t} x_t$ as given from the perspective of firm 1, its problem becomes minimizing:

$$\sum_{t=0}^{\infty} \beta^t \left\{ x_t' \Pi_{1t} x_t + u_{1t}' Q_1 u_{1t} + 2 u_{1t}' \Gamma_{1t} x_t \right\} \tag{22}$$

s.t.

$$x_{t+1} = \Lambda_{1t} x_t + B_1 u_{1t} \tag{23}$$

where
$\Lambda_{it} := A - B_{-i} F_{-it}$
$\Pi_{it} := R_i + F_{-it}' S_i F_{-it}$
$\Gamma_{it} := W_i' - M_i' F_{-it}$
Decision rules that solve this problem are

$$F_{1t} = (Q_1 + \beta B_1' P_{1t+1} B_1)^{-1} (\beta B_1' P_{1t+1} \Lambda_{1t} + \Gamma_{1t}) \tag{24}$$

where P_{1t} solves the matrix Riccati difference equation

$$\begin{aligned} P_{1t} = \Pi_{1t} - (\beta B_1' P_{1t+1} \Lambda_{1t} + \Gamma_{1t})' (Q_1 + \beta B_1' P_{1t+1} B_1)^{-1} \\ \times (\beta B_1' P_{1t+1} \Lambda_{1t} + \Gamma_{1t}) + \beta \Lambda_{1t}' P_{1t+1} \Lambda_{1t} \end{aligned} \tag{25}$$

and the solution of firm 2 takes the symmetric form.

In our duopoly model, the state and control variable are:

$$x_t := \begin{bmatrix} 1 \\ q_{1t} \\ q_{2t} \end{bmatrix} \quad \text{and} \quad u_{it} := q_{i,t+1} - q_{it}, \quad i = 1, 2$$

For the one-period payoff function, we have

$$x_t' R_i x_t + u_{it}' Q_i u_{it}$$

where $Q_1 = Q_2 = \gamma$,

$$R_1 := \begin{bmatrix} 0 & -\frac{a_0}{2} & 0 \\ -\frac{a_0}{2} & a_1 & \frac{a_1}{2} \\ 0 & \frac{a_1}{2} & 0 \end{bmatrix} \quad \text{and} \quad R_2 := \begin{bmatrix} 0 & 0 & -\frac{a_0}{2} \\ 0 & 0 & \frac{a_1}{2} \\ -\frac{a_0}{2} & \frac{a_1}{2} & a_1 \end{bmatrix}$$

The motion of the dynamics is x_t is $x_{t+1} = Ax_t + B_1 u_{1t} + B_2 u_{2t}$ where

$$A := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad B_1 := \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad B_2 := \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$