Modeling Macroeconomic Phenomena with Multi-Agent Reinforcement Learning

Nicholas Hirning
Stanford University
nhirning@stanford.edu

Rory Lipkis
Stanford University
rlipkis@stanford.edu

Andrew Chen
Stanford University
asjchen@stanford.edu

Abstract

While reinforcement learning has been widely used to model gameplay, its application to economics remains limited. In this work, we demonstrate how multi-agent reinforcement learning can be combined with a simple economic model to simulate and analyze macroeconomic phenomena under various conditions. In particular, we show how policy gradient learning algorithms yield qualitatively correct results in simple simulations, and we demonstrate nontrivial behavior in more complex scenarios involving over 30 separate agents.

I. Introduction

In simple models of market economies, a common assumption is that each citizen will act in their own best interest. This selfish behavior plays a major role in determining the efficiency of such markets and leads to many of the phenomena predicted and analyzed by economists. The advantage of this assumption is that once a utility function is chosen for these models, each agent in the economy acts simply to maximize their future utility. In practice, this lends itself naturally to construction via a multi-agent reinforcement learning model.

A simple model taught in introductory economics classes is the circular flow model. In this setup, there are two types of agents in the economy: firms (or businesses), and people (or households). These agents interact in two different markets, the goods market and the labor market. In the goods market, people pay currency to firms in exchange for goods. In the labor market, firms pay people for their labor to produce more goods. This basic model is diagrammed in figure 1. The name comes from the fact that currency flows clockwise in a circle between firms and people.

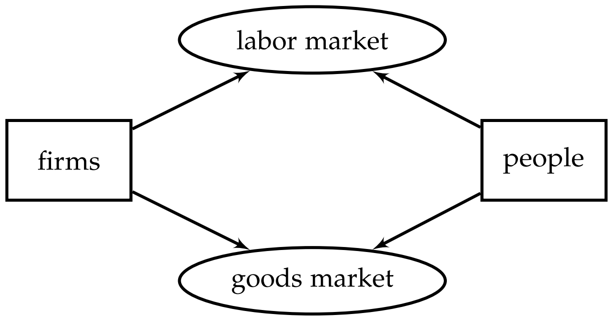

Figure 1: *Basic schematic of the circular flow model, a simple economic model in which society is composed of two types of agents (firms and people) that interact in two distinct markets.*

While the circular flow model does not include many important economic factors, it has the benefit of being simple enough to serve as a starting point for high-level analysis. For more in-depth economic phenomena, one would likely benefit from including more markets and agents; for example, the five-sector economy model includes government, financial markets, and overseas influences. However, as we aim to provide a proof-of-concept for modeling economies with multi-agent reinforcement learning, the circular flow model serves as a simple foundation that can potentially produce nontrivial dynamics.

In the following work, we explore how various economic phenomena can be simulated and analyzed by applying reinforcement learning to the circular flow model. While we are limited by the simplicity of the model, there are many economic metrics that manifest in such a model. For example, the gross domestic product (GDP) is widely touted as a comprehensive measure of a country's economic health. The GDP is defined by the following equation:

$$\text{GDP} = C + G + I + N,$$

where C stands for consumer spending, G for government spending, I for private domestic investments, and N for net exports. In practice, the vast majority of GDP is contained in consumer spending (accounting for roughly 2/3 of the U.S. GDP). While the circular flow model cannot account for net exports or government spending, the total consumer spending and private domestic investments can be approximated by the total quantity of money exchanged in the goods and labor markets.

Another important area of economics studies income and wealth inequality. A common approach to measuring these is to use the Gini coefficient. The Gini coefficient is a measure of the inequality of a distribution p with mean μ. Explicitly, the Gini coefficient is given by

$$G = \frac{1}{2\mu} \iint p(x)p(y)|x-y| \, \mathrm{d}x \, \mathrm{d}y.$$

In a discrete scenario, this becomes the average absolute difference of all pairs of items (e.g., income or absolute wealth) divided by twice the mean.

We aim to explore the applicability of multi-agent reinforcement learning as a means for analyzing macroeconomic phenomena. To this end, we implement the circular flow model and perform basic analyses of several economic metrics, including GDP and the Gini coefficient. As this is a multi-agent problem, agents choose actions and learn simultaneously, making the simulation dynamics quite complex. We also discuss and analyze the effectiveness of two different learning algorithms.

II. Literature Review

Despite the applicability of reinforcement learning techniques to economics, there are relatively few examples of such applications in existing literature. One particularly relevant example is a paper by Lozano et al. [1] which focuses on modeling an economy "based on conventions." In this work, the authors apply the SARSA(λ) algorithm in a government agent to search for a good expenditure policy. The authors avoid applying complicated multi-agent reinforcement learning by modeling each firm with a simple algorithm in which firms can copy neighboring firms or (with some low probability) switch to a random policy. The government attempts to increase national income and decrease debt based on some limited knowledge of the existing firms and historical records of income, debt, etc.

One important issue that the authors of [1] grappled with was the vast (and continuous) state space. They applied a multilayer-perceptron (MLP) with a single hidden layer to approximate the Q-function for SARSA(λ). Given the vast array of actions and states within any economy, approximation with an MLP model or similar is common in such scenarios. The authors found some success with their model, with the economy collapsing less frequently over time due to the government's policies.

Another relevant work by Tesauro & Kephart involved investigation of the behavior of reinforcement learning agents in the presence of other adaptive agents for economic purposes [2]. In particular, Tesauro & Kephart consider two selling agents competing in an economic model where each agent applied Q-learning. The state and action space was small enough that the Q-learning could proceed directly. Furthermore, the consumers in the model acted via a simple, greedy rule as opposed to acting as an independent, market-shaping force. Another difference between Tesauro & Kephart and the model proposed here is that the former model does not assume simultaneous price-setting; instead, the two selling agents took turns adjusting prices in response to each other. This turn-based approach is slightly less realistic than the simultaneous setting, but it allowed the authors to frame the problem as a two-player, alternating-turn, arbitrary-sum Markov game. Additionally, the authors spent significant amount of their time computing the Q-function for one of the sellers when the other seller was assumed myopic; they investigated simultaneous Q-learning as well, but noted that no convergence proofs exist for such cases.

While there are relatively few examples of reinforcement learning applied to macroeconomic models, there are quite a few papers with relevant material on applying multi-agent models to various settings involving competition and collaboration. For example, a 2013 study by Bos et. al. found that reinforcement learning alone does not accurately model humans in competitive social environments, but it *does* accurately model this when combined with biased reward representations [3]. In a separate work, Bereby-Meyer and Roth experiment with reinforcement learning in game theoretic situations, such as the repeated prisoner's dilemma, when complicated by noise [4]. The authors conclude that noise slows learning and strongly negatively affects cooperative tendencies [4]. As such, even with simple multi-agent situations involving some randomness, strong cooperation is unlikely.

One study performed by Könönen studied the situation of two competing brokers selling identical products under two different learning methods: modified gradient descent on the values and a policy gradient approach [5]. Könönen found success with both approaches, but found that the system would become intractable as the number of pricing actions increased [5]. To address this, the author suggested applying an MLP or other generalizable model as an extension. The suggestion to combine policy gradient algorithms and a generalizable action-choosing model significanlty influenced our design.

III. Methods

The model consists of an implementation of the circular flow architecture and a reinforcement learning engine. We describe these two components separately.

A. Circular Flow Architecture

The circular flow model uses two distinct types of agents: firms and people. Correspondingly, the simulation defines these two agent types separately. Each agent (firm or human) keeps the following internal state:

- **Money.** Each agent can accumulate and spend money during each timestep.
- **Goods.** Each agent can attain goods (firms create goods through the labor market, and people buy goods through the goods market).

Human agents are additionally each seeded with a constant skill level which governs how many goods a human agent can produce in an hour of work.

On each iteration, the internal state of each agent can change, and this change determines the reward for that agent. Human agents receive a reward equal to the number of goods they consume. Firm agents receive a reward equal to the profit increase in logarithmic space. More precisely, the reward for a firm is

$$r_{\text{firm}} = \log(m + p + \epsilon) - \log(m + \epsilon) = \log\left(1 + \frac{p}{m + \epsilon}\right).$$

Here m is the money of the firm before the update, p is the net profit of the firm for this iteration, and ϵ is a small constant used to prevent infinite negative reward in the case of $m = 0$ or $m = p = 0$. This definition is derived from the idea that the utility of money is logarithmic.

At each timestep, every agent produces an action that determines how they behave in the two markets. In each market, the designated selling agents give a fixed price and number of units. The buying agents provide a vectorized demand curve which specifies how much the agent is willing to pay for each subsequent item. Thus, each agent provides an action with the following structure:

- **Offered Price.** In the labor market, the human agents set their hourly rate. In the goods market, the firms set their price per good.
- **Offered Units.** In the labor market, the human agents set the number of hours they are willing to work. In the goods market, the firms set the number of goods they are willing to sell.
- **Demand Curve.** The demand curve is a vector. In the labor market, the ith entry of the demand curve corresponds to the price a firm will pay to produce its ith good. In the goods market, the ith entry of the demand curve corresponds to the price a human will pay for its ith good.

The space of all actions is vast. In order to make the problem tractable, the offered price and offered units are discretized into P and U values, respectively. In our simulations, we often set $P \approx 15$ and $U \approx 25$. Similarly, the demand curve was parameterized by families of functions. We experimented with two different families of curves: inverse and linear. The family of inverse curves $y = a/x + b$ is specified by the two parameters a, b (in our simulations we used approximately 36 total possible configurations). The family of linear curves was specified by solely the y-intercept, as the x-intercept was a hyperparameter fixed before training (the x-intercept corresponds to the maximum number of goods that can be purchased or produced by a given human or firm in a single iteration). In practice, the linear family trained faster, but the inverse family led to better results. We use the family of inverse demand curves for all plots presented in this paper.

The state used to produce the action is the money and goods of all firms and the money of all people (concatenated into a single vector). For a system with N firms and M people, the resulting state space has dimension $s_s = 2N + M$. Thus, the reinforcement learning engine (described in the next section) is tasked with learning a mapping from a space of size s_s to a space of size $s_a \approx P \cdot U \cdot F$, where F is the number of possible values for the parameters of the demand curves. While this fully transparent state is not wholly realistic, it significantly simplifies the model and can be justified economically by the increasing availability of information.

In addition to this setup, several additional features were added to enhance the simulations. All human agents' stored money is increased every iteration by an interest rate (generally $< 1\%$). This has the benefit of preventing trivial policies in which humans provide all money to the firms, at which point the firms can no longer directly increase reward. Similarly, every iteration firms' stored money is decreased by fixed operational costs as well as costs that scale with the size of the firm (i.e., $m \mapsto r(m - f)$, where f is a fixed cost and r is a multiplicative decay just below 1). Together, these additions allow for interesting model dynamics for the entirety of the experiment.

B. RL Algorithms

Both the learning algorithms chosen for the simulation were policy gradient methods. This was chosen due to previous literature, and because such methods have stronger convergence guarantees than value-based reinforcement learning methods (though potentially to non-global optima).

In policy gradient methods, instead of approximating the value functions and using a separate exploration strategy for choosing actions, we directly compute the policy. Given a policy model parameterized by θ, we compute probabilities $\pi_\theta(a|s)$. When training this model, we aim to minimize the loss function [7]:

$$J(\theta) = -\sum_{s \in S} P_\pi(s) \sum_{a \in A} \pi_\theta(a|s) Q^\pi(s, a)$$

Intuitively, a low loss means that the agent outputs high probabilities for high Q-valued actions. Policy Gradient Theorem allows us to express the gradient of this loss in a tractable manner [6] [7]:

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

This means that over a trajectory, we simply need to compute the gradients:

$$\left(\sum_{i=t}^{T} \gamma^{i-t} r_t\right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

We can then use standard gradient descent methods to optimize θ.

As discussed previously, the simulation contains large, high-dimensional state spaces. The dimension of a given state vector is $s_s = 2N + M$. Furthermore, every entry can take on a wide range of integral values. Therefore, we use neural network architectures to convert a given state to a vector containing log probabilities for each action. In empirical testing, if s_a is the number of possible actions, we've seen that the following architecture is a simple framework that still yields convergence in our context:

- Input layer with state input, of dimension s_s. State values are unnormalized.
- Fully-connected hidden layer with tanh activation, of dimension $\lfloor \sqrt{s_s s_a} \rfloor$
- Fully-connected output layer with tanh activation, followed by log softmax, of dimension s_a

During training, the learning rate was set to $\alpha = 0.01$.

B.1 REINFORCE Policy Gradient

Perhaps the simplest policy gradient method is the REINFORCE algorithm. This algorithm uses a Monte Carlo approach to collecting training data. The length of the episode T is manually specified, and we generally used $T \sim 100$. The runtime of this algorithm per episode is relatively short. However, often a significant number of episodes are required for the following reasons:

Algorithm 1 REINFORCE Policy Gradient [6]

```
1: procedure REINFORCE
2:     Initialize policy network parameters $\theta$
3:     for $i = 1, 2, \ldots, M$ do
4:         Follow current policy to obtain trajectory
               $\{s_1, a_1, r_1, \ldots, s_T, a_T, r_T\}$.
5:         for $t = 1, 2, \ldots, T$ do
6:             Compute $v_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$.
7:             Update $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$
8:     return $\theta$
```

- Agents only update the parameters once per episode.
- In many contexts, vanilla REINFORCE has been known to suffer from high variance in the losses from episode to episode. [6]
- In our multi-agent problem, the environment dynamics constantly change as every agent responds to each timestep in the markets.

B.2 Actor-Critic

An alternative family of policy gradient methods is the *actor-critic* family. In these algorithms, the actor component computes the log probabilities $\log \pi_\theta(s, a)$ while the critic component approximates the performance of the action. We considered this set of algorithms because in theory, it carries the convergence advantages of general policy gradient methods while reducing the gradient variance [7]. In this scenario, we tested Q Actor-Critic, in which the critic computes the Q-values $Q(s, a)$, as well as Advantage Actor-Critic (A2C), in which the critic computes the advantage $Q(s, a) - V(s)$; the idea of using the advantage vs the Q-value is to compare the quality of the action with a stable baseline, reducing variance. As with REINFORCE, we again used neural networks with one hidden layer for both the actor and critic. Note also that we get to update the parameters after every action via TD learning.

Algorithm 2 Q Actor Critic

```
1: procedure ACTOR_CRITIC
2:     Initialize policy and Q-network parameters $\theta, \omega$
3:     for $i = 1, 2, \ldots, M$ do
4:         Initialize state and action $s_1, a_1 \sim \pi_\theta(a|s_1)$.
5:         for $t = 1, 2, \ldots, T-1$ do
6:             Record reward $r_t$ and next state $s_{t+1}$
7:             Sample the next action $a_{t+1} \sim \log \pi_\theta(a|s_{t+1})$
8:             Update $\theta \leftarrow \theta + \alpha_\theta Q_\omega(s_t, a_t) \nabla_\theta \pi_\theta(s_t, a_t)$
9:             Compute temporal difference error: $\delta_t = r_t + \gamma Q_\omega(s_{t+1}, a_{t+1}) - Q_\omega(s_t, a_t)$
10:            Update $\omega \leftarrow \omega + \alpha_\omega \delta_t \nabla_\omega Q_\omega(s_t, a_t)$
11:    return $\theta, \omega$
```

IV. Results and Data Analysis

We break the results down into two sections: preliminary results and a more complex simulation. The preliminary results are designed to verify that the system is working and producing correct results, while the more complex simulation allows more open-ended exploration of economic concepts.

A. Preliminary Results

To test the convergence of the learning and the ensure that the results were logical, we created a scenario with one firm agent and five human agents. The dynamics discussed below are for a simulation in which all agents utilized the REINFORCE algorithm, all agents used inverse demand curves, the human agent interest rate was set to 1%, the firm operational cost was set to \$10, and the discount was $\gamma = 0.99$. The firm was initialized with \$10000, while the human agents were initialized to have uniform random initial money such that the total money given to human agents was \$5000. The human skill levels (goods per hour) were drawn from $N(1, 0.1)$. The simulation was run for 100 iterations.

Recall that the gradient the REINFORCE algorithm is of the following form:

$$\nabla_\theta J(\theta) = -\sum_{t=1}^{T} \left[\nabla_\theta \log \pi_\theta(s_t, a_t) \left(\sum_{i=t}^{n} \gamma^{i-t} r_i \right) \right].$$

In our training, we designate a proxy for the loss to reflect this gradient:

$$J'(\theta) = -\sum_{t=1}^{T} \left[\log \pi_\theta(s_t, a_t) \left(\sum_{i=t}^{n} \gamma^{i-t} r_i \right) \right].$$

Note that, as the policy converges, the probability of taking a given action a_t from state s_t should approach one, and so the $\log \pi_\theta$ term will approach zero. Therefore, convergence corresponds to the (proxy) losses approaching zero. This occurs in our preliminary simulation, as can be seen in figure 2.

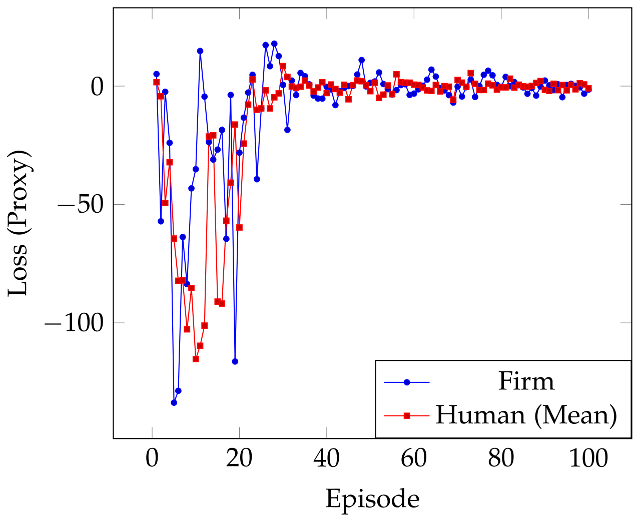

Figure 2: *The agents' losses converge to zero over time. This plot illustrates the loss of firm agents vs. human agents in a simulation with one firm agent and five human agents. The loss for the human agents is the average of all five human agents' losses. This simulation uses the REINFORCE learning algorithm, and indicates that the algorithm is succcessfully converging to a policy within 100 episodes.*

While this indicates that agents are able to converge to policies in this multi-agent environment, it does not necessarily indicate that the learned behavior leads to positive rewards. To investigate this, we can look at how the firm's profit changes over time and the actions that the firm is taking. As there is a single firm and multiple people, we might expect that the firm has more of an ability to monopolize the market, pushing the price of goods higher. As such, we expect the price at which the firm sells goods to rise. In figure 3, we see that the firm's profit is generally increasing over time from an initial average profit of approximately \$23 per iteration to an average profit of up to \$50 per iteration. The noisiness in the increase in profit may be due to the complex, changing environment (due to the changing actions of the human agents), or the fact that the reward is actually the future-discounted logarithmic profit.

Figure 4 illustrates the frequency at which the firm chooses a given price in the first and last episodes. In particular, the firm generally learns to select higher prices as it has a monopoly on the goods market and the five human agents are competing for the firm's offerings. Note that this precise pattern is not wholly reproducible; on subsequent runs, the firm agent will display similar behavior but the precise actions chosen will vary.

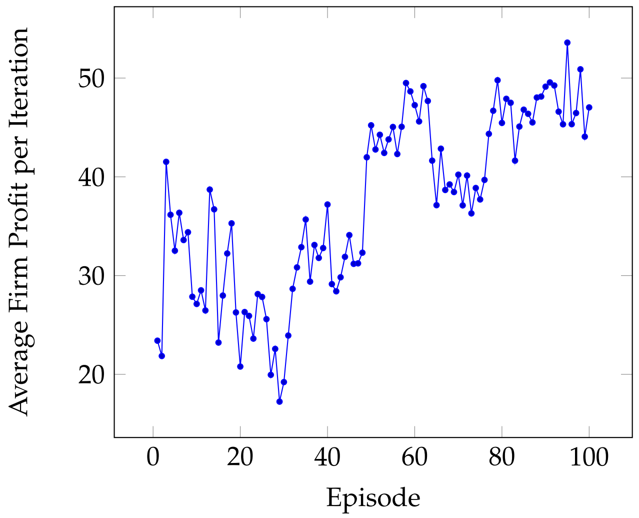

Figure 3: *Firm profit increases over time. This plot shows the average profit the firm gains per iteration as a function of episode. The simulation shown uses a single firm agent and five human agents, all using the REINFORCE learning algorithm. This indicates that the REINFORCE learning algorithm is yielding higher reward and better policies over time.*

We also attempted to run both Q Actor-Critic and A2C algorithms for the firm in the same monopoly situation as before (one firm with five people). However, while the policy networks would consistently converge (see Figure 5), the critic networks would rarely converge, about 10% of the time (see Figure 6). For this reason, we chose to stick with REINFORCE as the reinforcement learning model for future simulations.

B. Investigating Economic Phenomena

V. Conclusion

We implemented the circular flow model using multi-agent reinforcement learning and demonstrated that the results were qualitatively correct and could be used to investigate more complex macroeconomic phenomena. In particular, we found reasonable results in a simple monopolization model with a single firm agent and five human agents. This model was tested with two different policy gradient

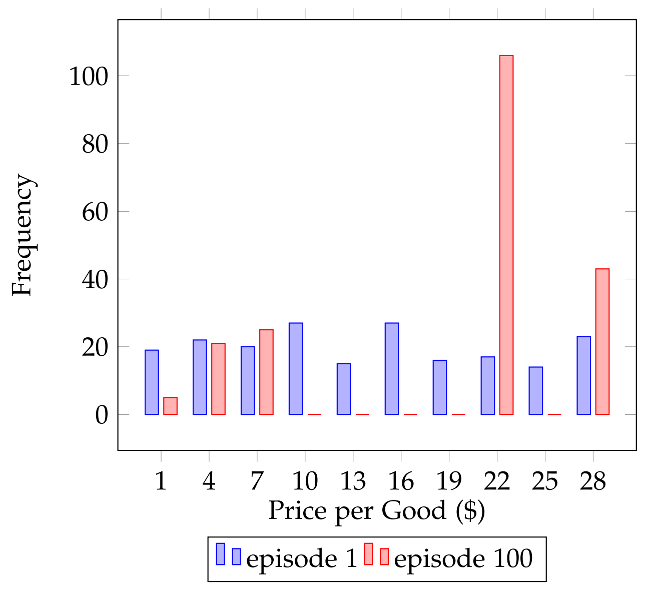

Figure 4: *With a single firm agent and five human agents, the firm learns to increase the price of goods to maximize its own profit. The bars shown correspond to the frequency at which the firm chooses a given price to offer for the first episode and the last episode. The possible prices are discretized to be in the set* $\{1, 4, 7, 10, 13, 16, 19, 22, 25, 28\}$. *This behavior is consistent with that of a monopoly.*

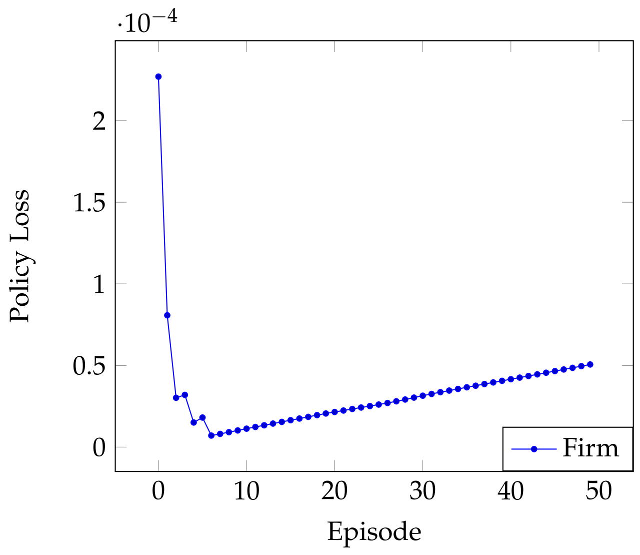

Figure 5: *Policy Network Loss in Q Actor Critic Method: with one actor-critic firm agent and five human agents, the firm's policy network loss converges rather quickly to 0 (note that the y-axis increments are rather small). Like in REINFORCE, when the policy becomes close to deterministic, the loss approaches zero.*

algorithms: REINFORCE and Q-actor critic. The latter was faster, but suffered from stability issues as compared to the former. Additionally, we simulated a more complex situation involving 5 firm agents and 25 human agents. The resulting dynamics had realistic features, including policies that corresponded to retirement behavior, slowly increasing wealth inequality, etc.

While this simulation is a successful proof-of-concept for using multi-agent reinforcement learning for macroeconomic simulations, there are many extensions that can make the results more realistic and interpretable. We outline several of these extensions here:

- **Government.** In practice, the GDP tended to decrease in our simulations. One way to combat this would be to add a government agent that can control the interest rates and taxes for firms and humans. The reward for the government agent would be related to increasing GDP, decreasing unemployment, and keeping the Gini coefficient in a desired range.
- **Multiple Goods.** We only simulated one specific market on one type of good. By expanding to n goods, each person could specialize in the good they were the best at (the skill of a person would become a n-vector of skills). Additionally, firms could either produce multiple goods or a single good.
- **Birth, Death, Bankruptcy.** In practice, people can die and wealth can be passed on within a family. Similarly, firms can go bankrupt and new firms can emerge. Adding a system by which firms and people can enter and leave markets would allow for more generalizable dynamics.
- **Unions and Trusts.** Allowing cooperation amongst firms and humans would make the simulation even more realistic. This can be approximated by allowing one neural network to choose the actions for multiple people or multiple firms at once.
- **Equity and the Stock Market.** One feature missing from the circular flow model is the existence of financial markets. One interesting addition would be allowing human agents to bet on the stock market or to own shares of a firm. However, this may require increasing the number of agents significantly in order to prevent a single person's actions from influencing the remainder of the market.

As computational power increases and the field of reinforcement learning progresses, the ability to create accurate economic simulations will continue to improve. In particular, economics is a field in which controlled experiments are extremely difficult to setup. We hope that the techniques presented in this paper may allow future economists to investigate niche phenomena and situations more easily.

VI. Acknowledgments

The authors feel that they all contributed equally to the work presented. In particular, Andrew Chen implemented the actor-critic reinforcement learning algorithm, Nicholas Hirning implemented the circular flow model architecture,

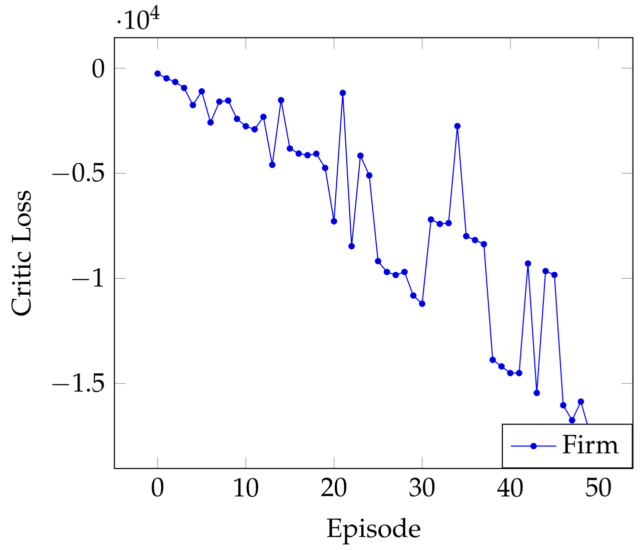

Figure 6: *Critic Network Loss in Q Actor Critic Method: with one actor-critic firm agent and five human agents, the firm's critic loss does not converge, even with changing the learning rate. This was also true when we used A2C, which is supposed to address variance issues with a baseline. Note that the scale on the y-axis is quite large.*

and Rory Lipkis performed hyperparameter optimization and data analysis. All three authors collaborated on implementing and debugging the REINFORCE policy gradient algorithm and other areas of the code.

The authors would like to thank the CS 238 staff for a great time throughout this course.

References

[1] Fernando Lozano, Jaime Lozano, and Mario García. An artificial economy based on reinforcement learning and agent based modeling. Documentos de Trabajo 003907, Universidad del Rosario, April 2007.

[2] Gerald Tesauro and Jeffrey Kephart. Pricing in agent economies using multi-agent q-learning. *Autonomous Agent and Multi-Agent Systems*, 5, 10 1999.

[3] Wouter van den Bos, Arjun Talwar, and Samuel M. McClure. Neural correlates of reinforcement learning and social preferences in competitive bidding. *Journal of Neuroscience*, 33(5):2137–2146, 2013.

[4] Yoella Bereby-Meyer and Alvin E. Roth. The speed of learning in noisy games: Partial reinforcement and the sustainability of cooperation. *American Economic Review*, 96(4):1029–1042, September 2006.

[5] Ville Könönen. Dynamic pricing based on asymmetric multiagent reinforcement learning. *International Journal of Intelligent Systems*, 21(1):73–98, 2006.

[6] Lilian Weng. Policy gradient algorithms, April 2018. `https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html`.

[7] Emma Brunskill. CS 234 Lecture 8: Policy Gradient I, February 2018.