Modeling an Economy with Policy Gradients

Nicholas Hirning
Stanford University
nhirning@stanford.edu

Rory Lipkis
Stanford University
rlipkis@stanford.edu

Andrew Chen
Stanford University
asjchen@stanford.edu

Abstract

TODO

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.

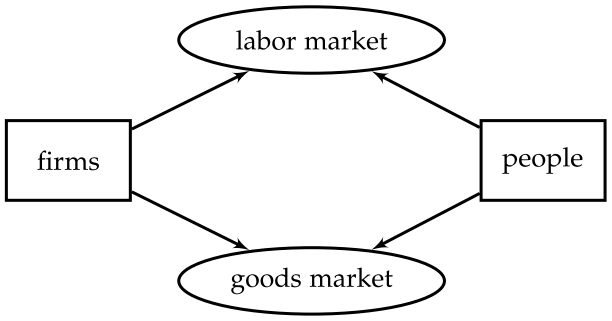

Figure 1: *Basic schematic of the circular flow model, a simple economic model in which society is composed of two types of agents (firms and people) that interact in two distinct markets.*

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 can be directly investigated 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. Thus, such a model can lead to investigation of how GDP changes over time and as a function of the number of agents of each type.

Another important economic focus is on income and wealth inequality, both of which are manifested in the circular flow model. A common approach to measuring inequality is to use the Gini coefficient. The Gini coefficient of a distribution p with mean μ 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 just becomes the average absolute difference of all pairs of items (e.g., income or absolute wealth) divided by twice the mean. Given its relevance to modern society, understanding how income inequality develops and progresses over time and as a function of initial wealth and skill distribution is quite important.

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 both GDP and income inequality. As this is a multi-agent problem, agents choose actions and learn simultaneously, making the simulation dynamics quite complex. Thus, we will 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. In this case, 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. In this case, 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 limited 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. In this case, 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. In our model, we attempt to combine and extend many of the successful techniques presented in past studies to create an accurate representation of the circular flow model.

III. Methods

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).

Each firm receives a reward at each step equal to the profit they generate. Thus, firms aim to maximize their future profit. Human agents receive a reward equal to the number of goods they consume. Furthermore, human agents are each seeded with a constant skill level which governs how many goods a human agent can produce in an hour of work.

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 *i*th entry of the demand curve corresponds to the price the firm will pay to produce its *i*th good. In the goods market, the *i*th entry of the demand curve corresponds to the price a human will pay for its *i*th good.

In practice, this action space is vast. In order to make the problem tractable, the offered price and offered units are discretized into one of 20 or so values. The demand curve is parameterized by TODO.
[multiagent -> individual RL agents] [describe state, s_s] [describe actions, s_a, as discretized to more easily track the] [describe rewards] [parameterizing the demand curve] [full transparency of staters]

i. RL Algorithms******

Following the literature we reviewed, we worked within the realm of policy gradient methods; we also choose this family of algorithms because they have stronger convergence guarantees than value-based RL methods (albeit potentially to local optimum that are not global optimum).

As framed, our problem contains large — in fact, high dimensional — state spaces. Specifically, the dimension of a given state is $s_s = 2 \cdot (\text{number of firms}) + (\text{number of people})$. Furthermore, while the values in each dimension were integers, they could take on a wide range of values. Therefore, in every policy, we use neural network architectures to convert a given state to a vector, containing log probabilities of taking 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, we generally use a learning rate of $\alpha = 0.01$.

i.1 REINFORCE Policy Gradient

Perhaps the simplest policy gradient method is the REINFORCE algorithm. This algorithm uses a Monte Carlo-type approach to gathering training data. We manually specify the length of the episode, generally using $T = 100$. This algorithm runs reasonably quickly

Algorithm 1 REINFORCE Policy Gradient

```
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 $v_t = \sum_{i=t}^{n} \gamma^{i-t} r_i$.
7:             Update $\theta \leftarrow \theta + \alpha \nabla_\theta \pi_\theta(s_t, a_t) v_t$
8:     return $\theta$
```

per episode. However, we also need to run more episodes because:

- 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.
- In our multi-agent problem, the environment dynamics constantly change because every agent responds to each timestep in the markets.

i.2 Q 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 functions $Q(s,a)$. We consider this set of algorithms because while they carry the convergence advantages of general policy gradient methods, they also approximate the Q-values with better sample efficiency. In our context, we again use neural networks with one hidden layer each for both the actor and critic. In the current version we

Algorithm 2 Q Actor Critic

```
1: procedure ACTOR_CRITIC
2:     Initialize policy and Q-network parameters $\theta, \omega$
3:     for $i = 1, 2, \dots, M$ do
4:         Initialize state and action $s_1, a_1 \sim \pi_\theta(a|s_1)$.
5:         for $t = 1, 2, \dots, T-1$ do
6:             Record reward $r_t$ and next state $s_{t+1}$
7:             Sample the next action $a_{t+1} \sim \pi_\theta(a|s_{t+1})$
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 $\theta, \omega$
```

have implemented, the
[************* Q-network doesn't converge, maybe because we have an unstable target, requiring a target network]

IV. Results and Data Analysis

V. Conclusion

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, 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 really 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.