Solving and Estimating Heterogeneous Agent Model Using Reinforcement and Machine Learning [*]

Yuanzhe Liu[†]

October 17, 2024

Abstract

In this paper, I introduce an innovative approach leveraging Deep Reinforcement Learning techniques to solve and estimate heterogeneous models. Contrasted with conventional solution methods, the deep learning approach offers a global solution while retaining the entirety of nonlinearity. Moreover, it exhibits remarkable scalability, making it suitable for handling models with hundreds or even thousands of state variables. I also explore the integration of this novel solution method with amortized likelihood-free Bayesian inference, opening up new possibilities for advanced probabilistic modeling and estimation.

Keywords: ML, NN, RL, Heterogeneous agent model

JEL Codes: C63, E21, C11

[*]I am grateful for continuous support and helpful comments from my advisor Peter Rupert, as well as from Kieran Walsh,and Yueyuan Ma. I am also grateful to all the participants of the UCSB Macro 290 group; special thanks to Xander Abajian and Yang Gao. All errors are my own.

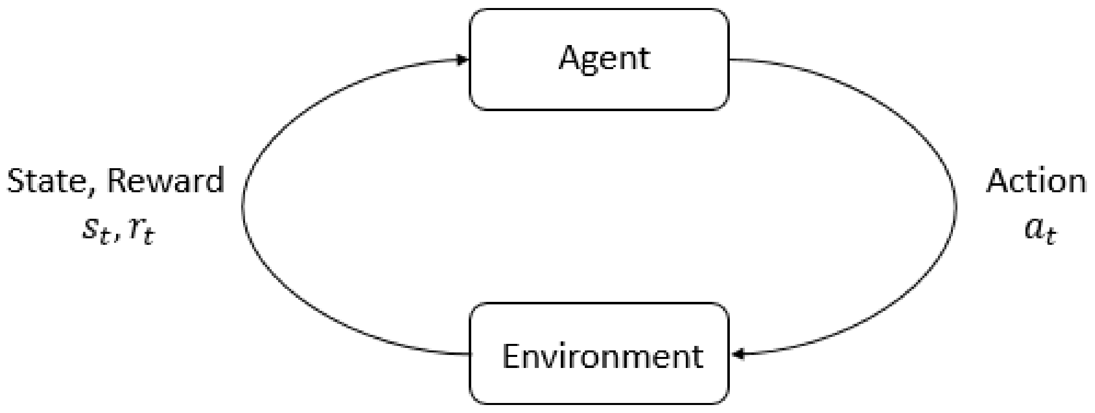
[†]yliu793@ucsb.edu

1 Introduction

Heterogeneous agent model has been the working horse in macroeconomics. It is widely used to study the distribution of wealth, income, and consumption. The traditional solution method for heterogeneous agent models is to use dynamic programming. However, this method has several limitations. First, it requires a lot of manual effort to solve the model. Second, it is difficult to scale up to models with hundreds or even thousands of state variables. Third, it is not guaranteed to find the global solution, typically, it only finds a local solution.

Reinforcement learning is a unsupervised machine learning technique that has been successfully applied to a wide range of problems, including game playing, robotics, and autonomous driving. It has been show that the performance of reinforcement learning is superior in many cases such as Go and multiplayer online battle arena (MOBA) games, where reinforcement learning has been able to beat the best human players.

In this paper I show how to adapt reinforcement learning to solve a heterogeneous agent model. I show that naively borrow reinforcement learning algorithms does not work well and I propose a method to adapt most commonly used RL algorithms to solving economic models. I then show that it can find the global solution to the model without any assumption, and it can be easily scaled up to models with hundreds or even thousands of state variables. Moreover, I show that we can easily obtain the impulse response function (IRF) of the model using the learned policy function. Finally, since we solve the model using reinforcement learning, if we would like to take the model to the data, we can easily do so using amortized likelihood free Bayesian inference. Since we use neural networks to approximate the policy function, we need to use some likelihood free method to estimate the model because the likelihood function is not available analytically.



1.1 Reinforcement Learning

In this subsection, I will give a brief introduction to reinforcement learning. Reinforcement learning is a type of machine learning that is used to teach an agent how to make decisions. The agent learns to make decisions by interacting with an environment and receiving rewards or punishments based on its actions. The goal of the agent is to learn a policy that maximizes its cumulative reward over time. RL agents learn by trial and error, exploring the environment and learning from the consequences of their actions. Most RL algorithms are based on the idea of estimating the value of different actions or states and using this information to make decisions.

There are two main types of RL algorithms: model-based and model-free. Model-based algorithms learn a model of the environment and use this model to make decisions. Model-free algorithms learn a policy directly from the data without explicitly modeling the environment. Model-free algorithms are more flexible and can be applied to a wider range of problems, but they can be less efficient than model-based algorithms.

For example, consider the game of snake where the goal is to eat as many apples as possible without running into the walls or the snake's own body. In this game, the agent is the snake, the environment is the game board, and the rewards are the number of apples it eats. The agent learns to move around the board and eat apples by exploring the environment and learning from the rewards it receives. The agent's policy is the set of rules it uses to decide how to move around the board. The agent learns this policy by trying different actions and observing the consequences of its actions.

Most problems in reinforcement learning can be formulated as a Markov decision process (MDP). An MDP is a mathematical framework that describes the interaction between an agent and an environment. An MDP consists of a set of states, a set of actions, a transition function that describes how the environment changes in response to the agent's actions, and a reward function that assigns a reward to each state-action pair. The agent's goal is to learn a policy that maximizes its cumulative reward over time.

This is very similar to how we build most economic models. In an economic model, the agent is household or firm. They make decisions based on the state of the economy and receive rewards or punishments based on their actions. The goal of the agent is to maximize their expected discounted utility over time. Most economic models can be formulated as a MDP and therefore can be solved using reinforcement learning.

What is the difference between reinforcement learning and dynamic programming? In dynamic programming, we solve the model by backward induction or value function iteration. We start from the terminal period and solve for the value function at each period by maximizing the expected value of the next period's value function. In reinforcement learning, we solve the model by trial and error. We start with a random policy and update the policy based on the rewards we receive. The key difference is that in dynamic programming, we know the transition function and the reward function, while in reinforcement learning, we have to learn the transition function and the reward function from the simulated data.

Then what are the disadvantages of traditional method? For example, in the case of a heterogeneous agent model, the state space is very large. It is difficult to solve the model using dynamic programming because the state space is too large. In this case, reinforcement learning can be a good alternative because it can handle large state spaces. Moreover, reinforcement learning can find the global solution to the model, while traditional methods can only find a local solution. The most popular method in solving heterogeneous agent model is perturbation method. However, this method is not guaranteed to find the global solution and we have to make some assumptions to make the model tractable. In contrast, reinforcement learning does not require any assumptions and can find the global solution to the model.

There is no free lunch in machine learning. Reinforcement learning is not a panacea. It has its own limitations. For example, reinforcement learning requires a lot of data to learn the policy. If the model is too complex, it may take a long time to learn the policy. Moreover, reinforcement learning is not guaranteed to find the optimal policy. It may get stuck in a local minimum and fail to find the global solution. Therefore, it is important to carefully design the model and the algorithm to ensure that reinforcement learning can find the global solution.

1.2 Solution method

In this subsection, I will discuss the traditional solution method for heterogeneous agent model and what kind of improvement reinforcement learning can bring to the table. The most widely used method to solve heterogeneous agent model with aggregate shocks is proposed byKrusell and Smith (1998). Followed by many extensions, such as more cite here. All existing approaches have in common that they try to represent the cross section distribution of agents using a small number of statistics to reduce the dimensionality of the state space.

Reiter (2009) proposed a projection and perturbation method to solve the heterogeneous agent model. However, we have to linearize around the stationary steady state and if the model has more than one aggregate shocks or multidimensional cross-sectional distribution, the method becomes less efficient and accurate. Auclert et al. (2021) follows Reiter (2009) by perturbing the model to first order in aggregates. However, instead of represents the equilibrium as a system of linear equations in the state space, Auclert et al. (2021) write it as a system of linear equations in the sequence space. However, their method still requires linearization and therefore, it is not guaranteed to find the global solution.

Han et al. (2021) proposed a new method to solve the heterogeneous agent model using deep learning. They use a neural network to approximate both the value function and the policy function and then use fictitious play to update the policy and value function. They show that their method can find the global solution to the model and can be easily scaled up to models with hundreds or even thousands of state variables. Azinovic et al. (2022) also proposed a new method to solve the heterogeneous agent model using deep learning. They use a neural network to approximate the value function and then use euler equations as constraints to update the value function.

The rest of the paper is organized as follows. In Section 2, I briefly describe the heterogeneous agent model I used as a benchmark. In Section 3, I describe the Reinforcement Learning method to solve the model. In Section 4, I present the results of the RL method and show the implus response function (IRF). In Section 7, I conclude and discuess future work.

2 Model

Here is a little review of Krusell and Smith (1998). I tried to rephrase it use the language of reinforcement learning. We assume there is a continuum(infinite) of household in this economy. The household's Bellman equation of the model is:

$$v(k, s; \lambda, z) = \max_{c,k'} \left\{u(c) + \beta E\left[v\left(k', s'; \lambda', z'\right) \mid (s, z, \lambda)\right]\right\} \tag{1}$$

where the maximization is subject to

$$c + k' = \tilde{r}(K, N, z)k + w(K, N, z)s + (1 - \delta)k \tag{2}$$

$$\tilde{r} = \tilde{r}(K, N, z) = z\alpha \left(\frac{K}{N}\right)^{\alpha - 1} \tag{3}$$

$$w = w(K, N, z) = z(1 - \alpha) \left(\frac{K}{N}\right)^{\alpha} \tag{4}$$

$$\lambda' = H(\lambda, z) \tag{5}$$

where (K, N) is a stochastic processes determined from

$$K = \int k\lambda(k, s)dkds \tag{6}$$

$$N = \int s\lambda(k, s)dkds. \tag{7}$$

Here $\lambda(k, s)$ is the joint distribution of k, s across households and $H()$ in equation 5 is the transition kernel. The distribution is itself a random function disturbed by the aggregate shock (common noise) z. Typically, we can assume it follows an $AR(1)$ process $z' = (1 - \rho_z)\mu_z + \rho_z z + \epsilon_z$ where ϵ_z is from a Gaussian distribution. Similarly, we can assume the labor s also follow an $AR(1)$ process.

Household choose their consumption c for this period and saving k' for next period subject to the budget constrain equation 2. $\tilde{r}$ is interest rate and w is wage. On the right hand side of equation 2 we have capital income, labor income, and undepreciated capital, δ is the depreciation rate of capital.

The equilibrium is defined as the following:

1. Given the price functions, 3 and 4, and H, the value function solves the Bellman equation 1 and the optimal decision rule is $f(k, s, \lambda, z)$. ($f(k, s, \lambda, z)$ returns c and k')

2. The decision rule $f(k, s, \lambda, z)$ and the processes for s and z imply that today's distri-

bution $\lambda(k, s)$ is mapped into tomorrow's $\lambda'(k, s)$ by H (I think is part is similar to the forward backward phrase in the mean field game literature.)

In the original Krusell Smith paper, what they did is that instead of keep tracking λ which is infinite dimension, they only track of the first moment of λ and solve it using value/policy function iteration. It turns out that this strategy works very well in this model. However, only keep track of the first moment may not work in other models where the distribution itself matters a lot.

The intuition of why this method works is that when people make consumption saving decisions, they forecast the next period interest rate $\tilde{r}'$. To forecast $\tilde{r}'$, it is enough to know the first moment of λ. Some people try to add higher moments to the method described above, however, it only improves the result marginally.

Using reinforcement learning (RL) to solve this model is unnecessary because the old method is already very good. However, it might be a good benchmark model to test if RL can be applied to this classes of model. For more complicated models, only tracking the first moment may not work and the dynamic of K is not even close to linear.

what exactly are we expecting from a good solution method? A good solution method should be

- **Efficiency**: This is necessary in order to use the method for calibration, estimation, and further quantitative analysis.

- **Reliability**: In particular, it should be applicable beyond the local perturbation regime if nonlinearity matters, e.g. ZLB.

- **Generality**: The method should be applicable to a wide variety of different HA models. Multiple shocks and multidimensional cross-sectional distribution should be handled without much modification.

3 Method

Reinforcement Learning has four essential elements:Agents, Environment, Actions, Rewards. In the context of the Krusell and Smith (1998) model, the agents are the households, the environment is the transition function, the actions are the consumption and saving decisions, and the rewards are the utility from consumption. The goal of the agent is to learn a policy that maximizes its cumulative reward over time.

3.1 Policy Gradient Methods

Policy gradient methods are a class of methods that update the parameters of the function approximator (NN in our case) in the direction of increasing the value function. The policy gradient theorem states that the gradient of the expected return with respect to the policy parameters is proportional to the sum of the rewards weighted by the gradient of the log probability of the action. The policy gradient theorem can be written as:

$\nabla_\theta J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a \mid s, \boldsymbol{\theta})$

Here, $q_\pi(s,a)$ represents the action-value function, which denotes the expected cumulative future rewards an agent can obtain by starting from state s, taking action a, and then following a specific policy π_θ in this context. More details and the proof of the policy gradient theorem can be found in Sutton and Barto (2018).

The distribution μ here is the on-policy distribution under π. All parts of $\nabla_\theta J(\boldsymbol{\theta})$ can be estimated from simulated data. Let $J(\boldsymbol{\theta}) = v_{\pi_{\boldsymbol{\theta}}}(s)$ be the performance measure, where v_{π_θ} represents the true value function for the policy function π_θ. Here, θ denotes the parameter of the policy function. For instance, when approximating the policy function using a neural network (NN), θ corresponds to the weights and biases of the network.The task is to update θ to maximize the value function.

There are many different policy gradient methods, such as REINFORCE, PPO, DDPG, and TRPO. In this paper, I use the Deep Deterministic Policy Gradient (DDPG) 1 algorithm to solve the heterogeneous agent model. DDPG is an off-policy actor-critic algorithm that uses a deterministic policy to select actions. It is well-suited for continuous action spaces and has been shown to be effective in a wide range of problems. Off-policy means that the policy used to generate the data is different from the policy used to update the value function. This can be useful in practice because it allows us to reuse data collected from previous policies. Actor-critic means that the algorithm learns both a policy (actor) and a value function (critic) simultaneously. The policy is updated using the gradient of the value function, which provides a more stable estimate of the policy gradient.

An example of on-policy method is the REINFORCE algorithm. In the REINFORCE algorithm, the policy is updated using the gradient of the expected return with respect to the policy parameters. The expected return is estimated by running a simulation for many periods and computing the sum of the rewards. The policy is then updated using the gradient of the log probability of the action multiplied by the return. However, for off-policy methods, we simulate the environment to obtain the data and then we sample from the data to estimate the gradient.

```
initial policy and action-value function parameters $\theta$ and $\phi$, and a reply buffer $\mathcal{D}$;
while Not Converge do
    Observe state $s$ and select action $a = \operatorname{clip}\left(\pi_\theta(s) + \epsilon, a_{\text{Low}}, a_{\text{High}}\right)$, where $\epsilon \sim \mathcal{N}$;
    Execute $a$ ;
    Observe next state $s'$, utility $r$ ;
    Store $(s, a, r, s')$ in reply buffer $\mathcal{D}$ ;
    if it is time to update then
        Randomly sample a batch $B = \{(s, a, r, s')\}$ from $\mathcal{D}$;
        $y(s') = r + \beta Q_{\phi_{\text{targ}}}\left(s', \pi_{\theta_{\text{targ}}}(s')\right)$;
        $\phi = \phi - \nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s'\}\in B} \left(Q_\phi(s, a) - y(s')\right)^2$;
        $\theta = \theta + \nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi\left(s, \pi_\theta(s)\right)$;
        $\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$
        $\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1-\rho)\theta$
    end
end
```

Algorithm 1: Deep Deterministic Policy Gradient

The key idea of DDPG is to use a deterministic policy to select actions and to learn a value function that estimates the expected return of the policy. The policy is updated using the gradient of the value function, which provides a more stable estimate of the policy gradient. The value function is updated using the Bellman equation, which is the same as in Q-learning.

DDPG trains a deterministic policy in an off-policy fashion, therefore, in the beginning it may not try a wide enough variety of actions to find useful improvement and get stuck at a local solution. To address this issue, we add noise to the action selection process. This is called exploration noise. The noise is added to the output of the policy function to encourage exploration. The noise is annealed over time to reduce the amount of exploration as the policy becomes more confident. More details can be found in the original paper Silver et al. (2014) and Lillicrap et al. (2015).

In short summary, DDPG algorithm works as follows. We simulate the model for many periods and store the data in the reply buffer. We then sample from the data to estimate the gradient of the value function and the policy function. We update the value function using the Bellman equation and the policy function using the gradient of the value function. We repeat this process until the policy converges.

3.2 Implementation details

For each agent in the KS economy, each period they observe the economy such as r, w and the cross section distribution, and then make consumption and saving decision, and then transition to the next period.

We simulate 100 KS agents. For $i \in \{1, 2, 3...100\}$

$s_{t,i} = \{k_{t,1}, k_{t,2}..., k_{t,100}; l_{t,i}, z_t\}$

$a_{t,i} = \pi_\theta(s_{t,i})$

$utility_{t,i} = log(a_{t,i})$

$k_{t+1,i} = (1 + r_t) * k_{t,i} + w_t * l_{t,i} - a_{t,i}$

Once every agent has made their decision, we draw a new TFP shock, z_{t+1}, and transition to the next period.

$s_{t+1,i} = \{k_{t+1,1}, k_{t+1,2}..., k_{t+1,100}; l_{t+1,i}, z_{t+1}\}$

Store $\{s_{t,i}, a_{t,i}, utility_{t,i}, s_{t+1,i}\}$ in the memory reply buffer and update policy and action-value function.

Table 1: Model Parameter Values

Parameter	**Description**	**Value**
β	Discount Factor	0.98
μ_z	Mean of TFP	1
ρ_z	Persistence of TFP process	0.95
ϵ_z	st of TFP process	0.01
μ_s	Mean of labor	0.117
ρ_s	Persistence of labor process	0.95
ϵ_s	st of idio labor shock	0.01
α	Weight of capital in production function	0.33
δ	Depreciation rate of Capital	0.025
σ	utility	1

The following table shows the parameters of the learning algorithm.

Table 2: DDPG Parameter Values

Parameter	**Description**	**Value**
lr_{actor}	Learning rate of policy function	$5e^{-4}$
lr_{critic}	Learning rate of action-value function	$1e^{-3}$
B	Batch size	512
S_θ	Structure of NN_{policy}	$[256, 128, 32]$
S_ϕ	Structure of NN_{value}	$[256, 128, 32]$

Table 2 – Continued

Parameter	**Description**	**Value**
τ	soft update	$1e^{-3}$
ϵ	Exploration noise	0.001

3.3 Extra: need to be edited later

Most RL algorithms work well with deterministic environment, and in theory they should also work with stochastic environment. As shown in Lillicrap et al. (2015); Silver et al. (2014), the gradient policy theorem is proved in stochastic environment setting. However, in practice, it is very difficult to make the algorithm work in a stochastic environment. In the RL literature, theorems are proven in very general setting which is typically stochastic, but the environments used in the experiments are typically deterministic. For example, Lillicrap et al. (2015) used the most popular RL environments, Gymnasium, an API standard for reinforcement learning with a diverse collection of reference environments, more details about Gymnasium can be found in Towers et al. (2023).

Most economics models are stochastic, for example, the heterogeneous agent literature following Krusell and Smith (1998); Aiyagari (1994) assume ex-ante identical agents face idiosyncratic shocks. If we simply borrow the RL algorithm from the computer science literature and apply it to the economics models, it will not work well, mostly because the algorithm is not designed to handle stochastic environment. Not only it won't converge, but it will also be very unstable.

The key point here is that the RL algorithm cannot learn the value-action function very well while the environment is stochastic. As shown in algorithm 1 the way it learns the value-action function is to sample from the memory reply buffer and then use the sample to update the value-action function by minimizing the temporal difference error. The problem here is that the sample is not informative enough to learn the value-action function in a stochastic environment. When an economic agent making decisions, they are trading off between the current period utility and the future period expected utility. In typically economic model settings, part of the transition is deterministic, for example, the capital stock, however, some will be stochastic, such as the TFP or labor productivity. Most RL algorithms are designed to handle the deterministic case and often they perform very well because they do not need that much of sample to learn the value-action function. However, in the stochastic environment, the sample is not informative enough to learn the value-action function and simply increasing the sample size will not help.

4 RL Result

The learning rate for the policy function $5e^{-4}$, the learning rate for the action-value function $1e^{-3}$, the batch size $|B|= 512$, the target update rate $\tau = 1e^{-3}$, and the exploration noise $\epsilon = 0.01$. I use a neural network with three hidden layers of 128 units each and Tanh activation functions to approximate the policy function and the action-value function.

The following figures show the results of the RL method. The first figure 1 shows the distribution of wealth in the economy after half million iterations. The second figure 2 shows the consumption function. The third figure 5 shows the IRF of a transfer policy.

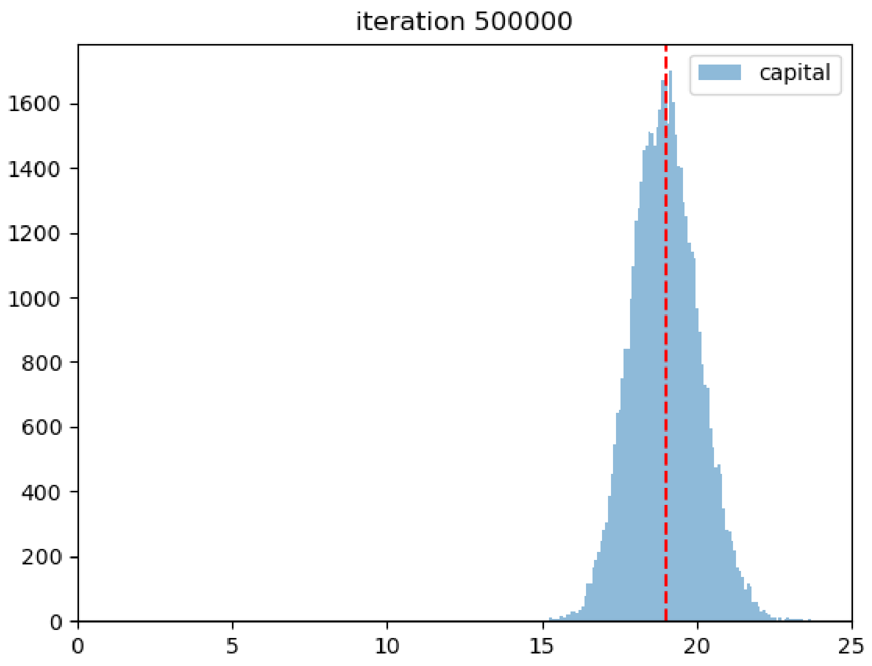


Figure 1: Wealth distribution after 500,000 iterations.

The following figures show the consumption and saving policy function of the RL algorithm. Figure 2 shows the consumption function and figure 3 shows the saving function. Both figure looks very similar to the policy function obtained from the traditional method. As mentioned in the original paper, the policy function is very close to linear becuase the marginal propensity to consume is almost constant across the wealth distribution.

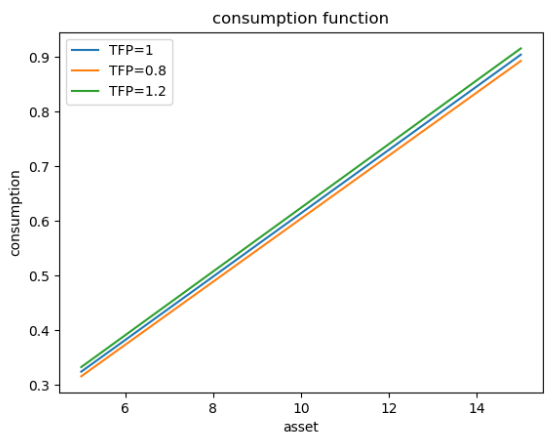


Figure 2: consumption function.

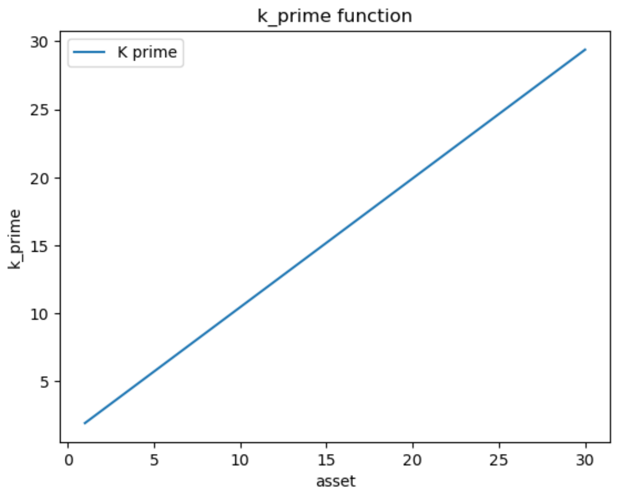


Figure 3: k'

Figure 5 shows the impulse response function of a transfer policy. The transfer policy is a policy that transfers a fixed amount of wealth from the rich to the poor. The social planner take one unit of wealth from agents whose wealth is above the median and give it to agents whose wealth is below the median. The figure 5 shows that the transfer policy has a positive effect on aggregate consumption. The original method proposed by Krusell and Smith (1998) states that only the mean of the distribution matters and the MPC is almost the same across the wealth distribution, therefore, the transfer policy should not have a big effect on the economy through the channel of inter-temporal substitution.

It is very easy to obtain the IRF of the model using the policy function learned by the RL algorithm. We can simply simulate this transfer for many time and then compute the average of the outcome. If we were to use the traditional method, we would have to solve the model for each period and then compute the outcome. The RL method is much faster and more efficient than the traditional method when it comes to compute the IRF.

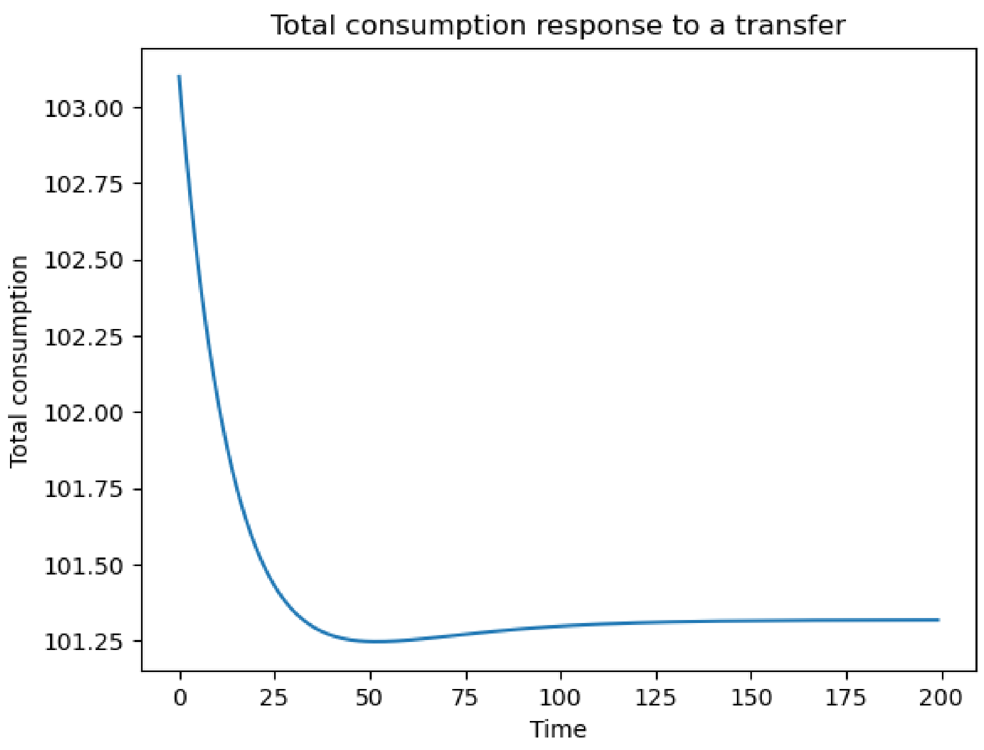


Figure 4: Impulse response function of a transfer policy.

5 Simulation Based inference

After solving the model we might want to take the model to data. However, since we use neural networks to approximate the policy function, we need to use some likelihood free method to estimate the model because the likelihood function is not available analytically.

Traditionally, if we solve the model by linearizing around its steady state then we can follow Herbst and Schorfheide (2016) to write down the likelihood function and use sampling method such as Metropolis-Hasting to obtain the posterior distribution of the parameters. However, in this case, we cannot use this method because the likelihood function is not analytically available.

Liu and Plagborg-Møller (2023) proposed a new method to incorporate the cross section distribution of variables such as consumption, saving, and wealth into the Bayesian estimation. It requires solve for the conditional probability of the cross section distribution given the state variables.

Kase et al. (2022) also proposed a new method to use the information from cross section distribution to estimate the model. Their method is to use machine learning method to approximate the particle filter.

Another advantage of using the likelihood free method is that combining with RL solution method we can have a end to end approach to solve and estimate our model.

5.1 SBI method

As an analytical expression for the likelihood function given observations is not available if we solve the model using RL or any other methods that use neural network to approximate the policy and value function, conventional Bayesian inference method is not applicable. An alternative class of methods, for likelihood free estimation has been developed in the literature. Approximate Bayesian computation (ABC) is a popular method in this class. ABC is a simulation-based method that approximates the posterior distribution of the parameters by comparing the simulated data with the observed data. The basic idea of ABC is for a given set of parameters drawn from the prior we simulate the model for many periods and then compare the simulated data with the observed data. The distance between the simulated data and the observed data is used as a measure of the likelihood of the parameters. If the distance is greater than a certain threshold, the parameters are rejected, otherwise, the parameters are accepted. The accepted parameters are then used to form an approximate posterior distribution of the parameters.

Another way of seeing this is to think of this process as of obtaining the joint distribution. $p(X, \theta) = p(\theta) * p(X|\theta)$. The posterior distribution of the parameters is then given by $p(\theta|X) = \frac{p(X,\theta)}{p(X)}$. The likelihood function $p(X|\theta)$ is not available analytically so does the joint distribution. If we draw many sets of parameters from the prior and simulate the model for many periods, we can obtain the joint distribution. The posterior distribution of the parameters is then given by the joint distribution divided by the marginal distribution of the data $p(X)$.

5.2 Normalizing Flow

Papamakarios and Murray (2018) proposed a new method to estimate the posterior distribution of the parameters using normalizing flow. Normalizing flow is a generative model that learns the distribution of the data by transforming a simple distribution into a complex distribution. The key idea of normalizing flow is to learn a series of invertible transformations that map a simple distribution to a complex distribution. The parameters of the transformations are learned using a neural network. The normalizing flow model can then be used to sample from the posterior distribution of the parameters. The advantage of normalizing flow is that it can capture complex dependencies in the data and can be trained efficiently using stochastic gradient descent. Normalizing flow has been shown to be effective in a wide range of problems, including density estimation, generative modeling, and variational inference.

Radev et al. (2023) developed a more advanced method that amortized the estimation process. That is for any given observations, without the need to re-train the model, we can obtain the posterior distribution of the parameters simply by evaluating the trained NN.

5.3 Estimation Result

More results will be added here.

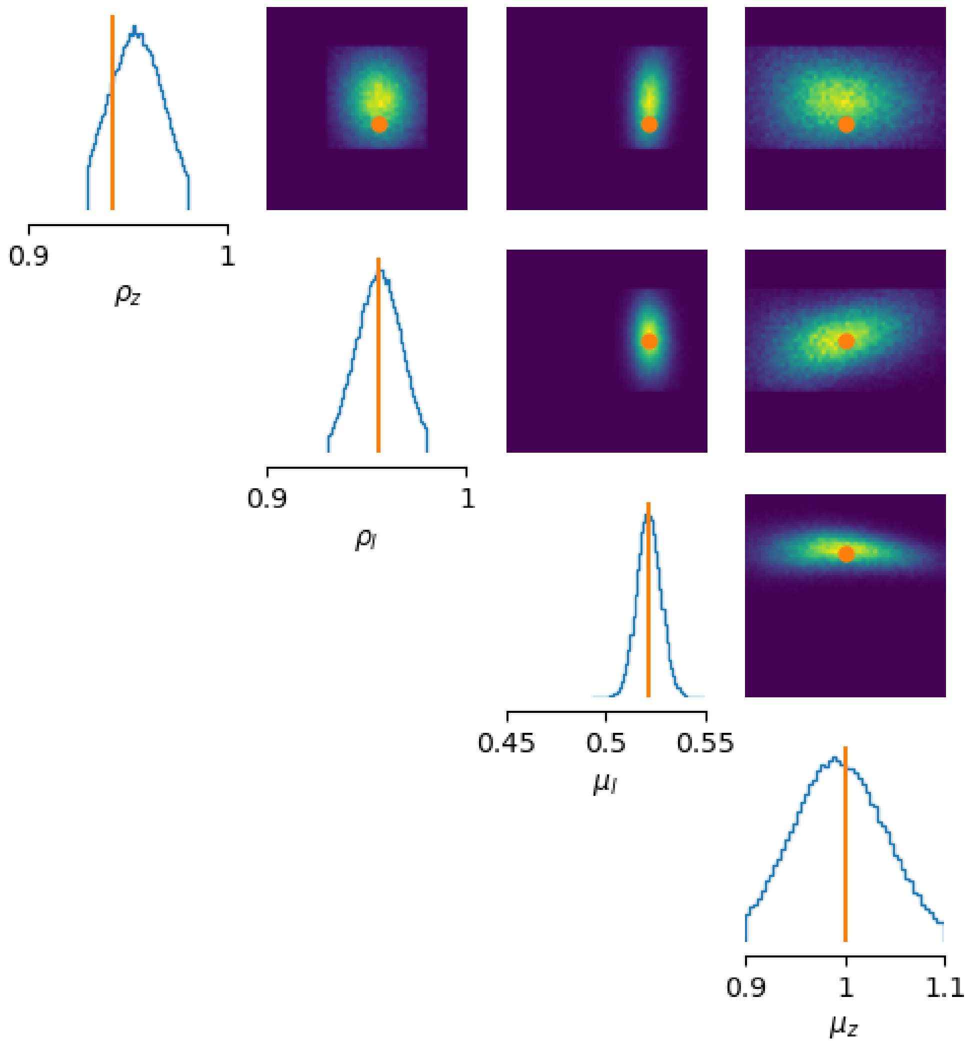


Figure 5: Posterior distribution of the parameters.

The orange vertical line represents the true value that I used to generate the observation.

6 Benchmark

In this section, we use a simple RBC model to demonstrate that the performance of the RL approach. The RBC model is a standard model in macroeconomics that is used to study the business cycle. The model consists of a representative agent who maximizes utility subject to a budget constraint. The agent chooses consumption and saving to maximize utility subject to the budget constraint. The agent's utility function is given by $U(c) = \log(c)$, where c

is consumption. The agent's budget constraint is given by $c + k' = (1-\delta)k + y$, where k is capital, k' is next period's capital, δ is the depreciation rate of capital. The agent's production function is given by $y = zk^{\alpha}$, where y is output, z is total factor productivity which follows an $AR(1)$ process, and α is the capital share of output. The agent's Euler equation is given by $\frac{1}{c} = \beta E\left[\frac{1}{c'}(1-\delta+\alpha z k^{\alpha-1})\right]$, where β is the discount factor, c' is next period's consumption, and E is the expectation operator. The agent's capital accumulation decision is given by $k' = (1-\delta)k - c$.

The golden rule of solve this class of simple model is using value function iteration. The value function iteration is a dynamic programming method that solves for the value function of the agent by iterating on the Bellman equation. The Bellman equation is given by $V(k,z) = \max_{c,k'} \log(c) + \beta E[V(k',z')|z]$. We first discrete the sate variable k around its steady state ,and discrete z using method proposed by Rouwenhorst (1995). Then we iterate on the Bellman equation until convergence. The policy function is then obtained by taking the argmax of the Bellman equation. The policy function is the optimal decision rule of the agent. Value function iteration is a standard method in macroeconomics that is used to solve dynamic stochastic general equilibrium (DSGE) models. It is guaranteed to find the global solution to the model. However, it is computationally expensive and can be slow for large models.

We discrete k into 3000 grid points and z into 31 grid points. We measure the performance of the solution method using the mean squared error of the euler equation. The mean squared error of the euler equation is given by $\frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{c_i} - \beta E\left[\frac{1}{c'}(1-\delta+\alpha z k^{\alpha-1})\right]\right)^2$, where N is the number of grid points, c_i is the consumption decision of the agent at grid point i, and c' is the next period's consumption decision. The mean squared error of the euler equation is a measure of how well the policy function satisfies the euler equation. The lower the mean squared error, the better the policy function. The MSE of the value function iteration is around 10^{-7}. The RL method I proposed in this paper has a MSE of around 10^{-}, which is in the same ballpark to the value function iteration method. Value function iteration is guaranteed to find the global solution to the model, therefore, in terms of accuracy the RL method is as good as the value function iteration method. However, the RL method is much faster and more efficient (less memory usage) than the value function iteration method.

7 Conclusion

References

Aiyagari, S. R. (1994). Uninsured idiosyncratic risk and aggregate saving. *The Quarterly Journal of Economics*, 109(3):659–684.

Auclert, A., Bardóczy, B., Rognlie, M., and Straub, L. (2021). Using the sequence-space jacobian to solve and estimate heterogeneous-agent models. *Econometrica*, 89(5):2375–2408.

Azinovic, M., Gaegauf, L., and Scheidegger, S. (2022). Deep equilibrium nets. *International Economic Review*, 63(4):1471–1525.

Han, J., Yang, Y., and E, W. (2021). Deepham: A global solution method for heterogeneous agent models with aggregate shocks.

Herbst, E. P. and Schorfheide, F. (2016). *Bayesian estimation of DSGE models.* Princeton University Press.

Kase, H., Melosi, L., and Rottner, M. (2022). *Estimating nonlinear heterogeneous agents models with neural networks.* Centre for Economic Policy Research.

Krusell, P. and Smith, Jr., A. (1998). Income and wealth heterogeneity in the macroeconomy. *Journal of Political Economy*, 106(5):867–896.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971.*

Liu, L. and Plagborg-Møller, M. (2023). Full-information estimation of heterogeneous agent models using macro and micro data. *Quantitative Economics*, 14:1–35.

Papamakarios, G. and Murray, I. (2018). Fast ϵ-free inference of simulation models with bayesian conditional density estimation.

Radev, S. T., Schmitt, M., Pratz, V., Picchini, U., Köthe, U., and Bürkner, P.-C. (2023). Jana: Jointly amortized neural approximation of complex Bayesian models. In Evans, R. J. and Shpitser, I., editors, *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence*, volume 216 of *Proceedings of Machine Learning Research*, pages 1695–1706. PMLR.

Reiter, M. (2009). Solving heterogeneous-agent models by projection and perturbation. *Journal of Economic Dynamics and Control*, 33(3):649–665.

Rouwenhorst, K. G. (1995). Asset pricing implications of equilibrium business cycle models. *Frontiers of business cycle research*, 1:294–330.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China. PMLR.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., and Younis, O. G. (2023). Gymnasium.

Appendix A. Optimal Conditions

Appendix B. Benchmark RBC model