INTERNATIONAL MONETARY FUND

AI and Macroeconomic Modeling: Deep Reinforcement Learning in an RBC Model

Tohid Atashbar and Rui (Aruhan) Shi

WP/23/40

2023
FEB

WORKING PAPER

 WP/23/40

IMF Working Paper[*]
Strategy, Policy and Review Department

AI and Macroeconomic Modeling: Deep Reinforcement Learning in an RBC Model

Prepared by Tohid Atashbar and Rui (Aruhan) Shi

Authorized for distribution by Stephan Danninger
February 2023

ABSTRACT: This study seeks to construct a basic reinforcement learning-based AI-macroeconomic simulator. We use a deep RL (DRL) approach (DDPG) in an RBC macroeconomic model. We set up two learning scenarios, one of which is deterministic without the technological shock and the other is stochastic. The objective of the deterministic environment is to compare the learning agent's behavior to a deterministic steady-state scenario. We demonstrate that in both deterministic and stochastic scenarios, the agent's choices are close to their optimal value. We also present cases of unstable learning behaviours. This AI-macro model may be enhanced in future research by adding additional variables or sectors to the model or by incorporating different DRL algorithms.

RECOMMENDED CITATION: Atashbar,T. and Shi, R.A. 2023. “AI and Macroeconomic Modeling: Deep Reinforcement Learning in an RBC model”, IMF Working Papers, WP/22/40.

JEL Classification Numbers:	C63, C54; D83; D87; E37
Keywords:	Reinforcement learning; Deep reinforcement learning; Artificial intelligence, RL; DRL; Learning algorithms; Macro modeling, RBC; Real business cycles; DDPG; Deep deterministic policy gradient; Actor-critic algorithms
Author’s E-Mail Address:	tatashbar@imf.org; ashi@imf.org

[*] The authors would like to thank Stephan Danninger for his helpful comments and suggestions. We appreciate the views and suggestions provided by Mico Mrkaic, Dmitry Plotnikov, Sergio Rodriguez and attendees at the IMF SPR Macro Policy Division Brownbag Seminar. Comments by Allan Dizioli are also gratefully acknowledged. All errors remain our own.

WORKING PAPERS

AI and Macroeconomic Modeling: Deep Reinforcement Learning in an RBC Model

Prepared by Tohid Atashbar and Rui (Aruhan) Shi

Contents

Glossary

AGI	Artificial General Intelligence
AI	Artificial Intelligence
ANN	Artificial Neural Networks
DDPG	Deep Deterministic Policy Gradient
DL	Deep learning
DNN	Deep Neural Network
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
RBC	Real Business Cycle
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SL	Supervised Learning
TD3	Twin Delayed DDPG
UL	Unsupervised Learning

Introduction

Macroeconomic modeling is the process of constructing a model that describes the behavior of a macroeconomic system. This process can be used to develop predictions about the future behavior of the system, to understand the relationships between different variables in the system, or to simulate behavior.

Artificial intelligence (AI) is a branch of computer science that deals with the design and development of intelligent computer systems. AI research deals with the question of how to create programs that are capable of intelligent behavior, i.e., the kind of behavior that is associated with human beings, such as reasoning, learning, problem-solving, and acting autonomously.

The two fields could be conceptually combined, as AI techniques could be used to develop more accurate macroeconomic models, or one could use macroeconomic models to help design artificial general intelligent systems that are better able to simulate economic (or more broadly social) behaviors, among many other tasks. AI can be used to automatically identify relationships between variables, or to develop new ways of representing economic systems. AI can also be used to develop methods for automatically learning from data, which can be used to improve the accuracy of predictions. AI also could be used to develop more sophisticated models that take into account a wider range of factors, including non-economic factors such as political instability or weather patterns.

An increasing body of work leverages machine learning for forecasting (Atashbar and Shi, 2022), besides some recent developments in optimization, market design, and algorithmic game theory, but AI's impact on economics, especially in the field of macroeconomic modeling, has been modest so far. This has been caused by a combination of factors including the relatively newness of the field, the difficulty of designing AI agents capable of realistically imitating human behavior in an economy, the lack of data available for training AI models, and the lack of computational resources needed to train and run large macroeconomic simulations.

But with the emergence of a new generation of AI models called reinforcement learning (RL), there's a growing belief that AI will have a transformative impact on macroeconomic modeling (Tilbury, 2022). This is primarily because RL models are much better suited than previous AI models for imitating human behavior. In addition, RL models require much less data to be trained (they generate their own data through interaction with their environment) and could be much more efficient in terms of computational resources in specific settings or algorithms.

The goal of this paper is to build a relatively simple and extendable macroeconomic model based on RL that can generate realistic macroeconomic dynamics that are comparable to models under the rational expectations assumption while not imposing unrealistic restrictions like perfect foresight on economic agents. The resulting model will be used as a prototype for future extensions in policy experiment or to customize it to better match the conditions, shocks or data of a particular or global economy.

To this end, we implement an advanced deep RL (DRL) algorithm (the deep deterministic policy gradient (DDPG)) in a real business cycle (RBC) macroeconomic model. We chose the DDPG algorithm for this basic model (with an eye on the possible extensions of the model in the future) for several reasons (Sutton and Barto (2018), Graesser and Keng (2019), Zai and Brown (2020) and Powell (2021)):

First, it is one of the modern RL algorithms that can be applied to continuous action space problems, which is crucial for modeling macroeconomic variables. Second, it is one of the RL algorithms that can handle high-dimensional state and action spaces, which are typical in macroeconomic models (e.g., the number of different economic sectors). Third, the separation of policy and value functions in the algorithm allows for analyzing each component independently during the learning process. Fourth, the DDPG algorithm is one of the few RL algorithms that can be applied to non-stationary problems, which are common in macroeconomic modeling. Fifth, it is one of the few RL algorithms that can be applied to problems with a very long-time horizon, which might be important for macroeconomic modeling. Sixth, the DDPG algorithm is one of the few RL algorithms that can be applied, in specific settings, to partially observable Markov decision process (POMDP) problems or,

in other words, to problems with a limited observation window or limited information settings. This could be important for some macroeconomic modeling works since the observation window is often limited by the frequency of the data. Finally, The DDPG algorithm has been shown to perform well in a variety of challenging problems in the RL literature. However, similar to other RL algorithms, the DDPG algorithm is also known to be unstable in some settings and can diverge if the learning process is not properly tuned.

We find that the RL augmented RBC model performs similar to the RBC model under the rational expectations assumption once the learning representative agent has learnt for many simulation periods. This is achieved from the stage when the representative agent does not understand the economic structure, its preference or how the economy transitions over time. However, the training takes a significant amount of simulation periods, in part due to the mechanism that the agent needs to generate its own experience to learn from it. To simulate realistic households' behaviors that match empirical learning periods, further work is needed to calibrate the parameters, or transfer past experience to the learning agent as a starting point of learning.

These encouraging results need to be put in perspective. In addition to the rudimentary (but extendible) character of our model structure, a disadvantage of our work is also the restricted scope of the RBC models. The business cycle variations are only propagated through an exogenous productivity shock. The empirically implied magnitude of true technology shock is likely to be smaller than what the RBC models predict. Unemployment is also explained in an overly simplified manner: intertemporal substitutions between labor and leisure explains employment variations. For workers to gain high utility, it is better to work more in productive periods, and less in unproductive periods. However, RBC models are the core component of the DSGE models that are largely applied in policy institutions and central banks. It is scalable and easily built on. It is well known and studied, and thus easy to compare learning results with existing theory.

We hope this work will encourage further research in the application of AI and deep RL for macroeconomic problems and will open up a new direction of research to combine deep RL with standard macroeconomic models. In particular, we expect it to be a base and extension for more advanced applications at the Fund that explore the use of deep RL for macroeconomic policy analysis.

The rest of the paper is organized as follows. Section I provides a brief literature review of AI and RL/deep RL applications in macroeconomic policy. Section II describes the RBC model. Section III introduces the DRL algorithm, the environment and the AI experiments we conduct, the results, and the issues during learning, and Section IV concludes.

I. An overview of the literature

Artificial intelligence (AI) is a growing field of computer science focused on creating intelligent computer systems, or machines, that can reason, learn, and act autonomously. AI systems are designed to mimic human cognitive abilities, such as learning, problem solving, and natural language processing.

The term "artificial intelligence" was first coined in 1956 by computer scientist John McCarthy (Andresen, 2002). AI research is highly interdisciplinary, involving disciplines such as computer science, psychology, neuroscience, linguistics, philosophy, and anthropology.

There are three broad categories of AI systems (Goertzel, 2007):

1. Narrow AI or weak AI systems are designed to perform a specific task, such as facial recognition or model financial markets.
2. General AI or strong AI systems are designed to perform a wide range of tasks, such as reasoning and planning.
3. Super AI or artificial general intelligence (AGI) are hypothetical AI systems that match or exceed human intelligence.

AI is already being heavily used across multiple fields and industries including health care, retail, finance, image processing, autonomous driving, and many more. The application of AI in economics is still in its early stages and has yet to be sufficiently developed in its application. Nonetheless, some theorize that sooner or later, AI-economist machines could catch up with the human economists in many areas (Atashbar, 2021a, 2021b). AI has been used in economics mostly for predictions and forecasts, market analysis and the impact analysis of alternative policies. Lu & Zhou (2021), Ruiz-Real et. al., (2021), Goldfarb et. al., (2019), Cao (2020), and Veloso et al., (2021) look at how AI is/could be used in economics and finance.

Machine learning (ML) is a branch of artificial intelligence that uses artificial neural networks (ANN) to learn from data, without being explicitly programmed. ANN is a data-driven approach to machine learning that is based on the idea of artificial neurons, or nodes, that are connected in layers. The input layer receives the input data, and the output layer produces the output. The hidden layers in between perform the learning by adjusting the weights of the connections between the nodes. Deep learning (DL) is a subset of machine learning that uses a deep neural network (DNN) to model complex patterns in data. A DNN is an ANN with a deep architecture. This means that the neural network contains not only an input layer and an output layer, but also one or more layers in between to add further non-linearities in order to recognize complex patterns in a dataset.

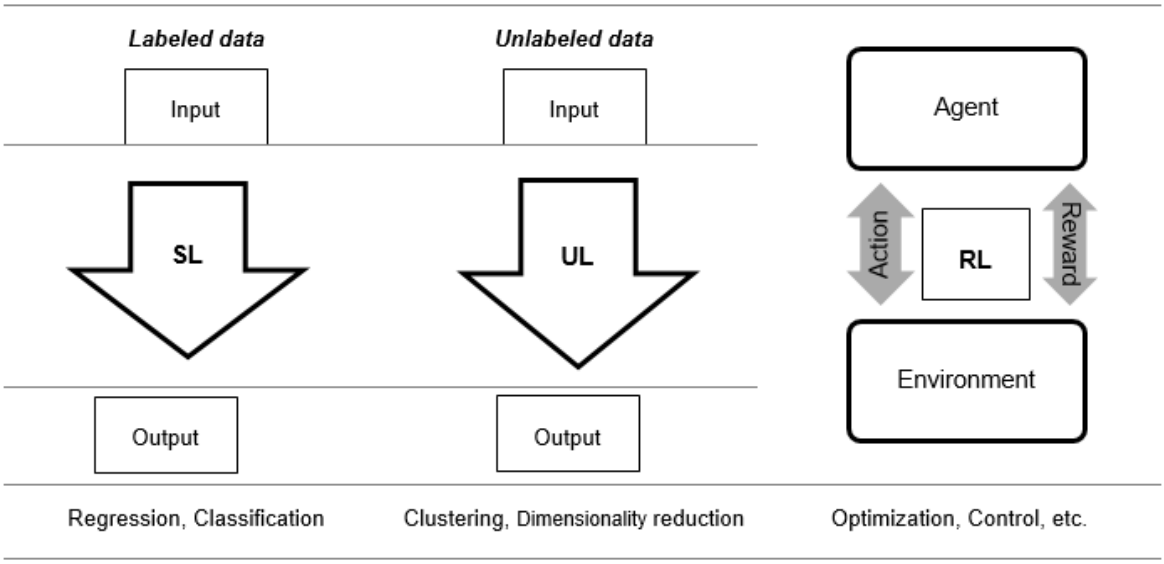
There are three general approaches to the learning processes in machine learning:

1. Supervised learning (SL): The machine is provided with a set of training data, which includes both the input data and the desired output. The data is labeled. The machine is then able to learn and generalize from this data in order to produce the desired output for new data. The main applications of supervised learning are classification, regression, and prediction.

2. Unsupervised learning (UL): The machine is provided with a set of input data, but not the desired output. The input is not labeled. The machine must then learn to find patterns and relationships in the data in order to produce the desired output. Semi-Supervised Learning combines supervised and unsupervised learning. This means that the training dataset contains both labelled data (i.e., every piece of input data is attached to a desired output) and unlabeled data (i.e., input data is not attached to a desired output). The main applications of unsupervised learning are clustering, dimensionality reduction (e.g., principal components), and association rule learning.

3. Reinforcement learning (RL): It is different from both supervised and unsupervised learning in that it is not given a set of training data. The machine is given a set of rules or objectives, and it must learn how to best achieve these objectives through repeated interactions with the environment. The main applications of reinforcement learning are control, robotics, optimization and gaming.

Figure 1. SL, UL and RL in ML



Source: authors' construction

Deep reinforcement learning (DRL) is a machine learning technique that combines reinforcement learning (RL) with deep learning (DL), meaning that it uses a DNN to represent the RL agent (Li, 2017). This approach is used to solve problems that are too difficult for simple RL algorithms alone. For an introduction of the theory and several algorithms in RL/RDL, see Atashbar and Shi (2022a).

Surveys by Athey (2018), Cameron (2019), Nosratabadi et al., (2020) and Hull (2021) provide a comprehensive review of the methods and use cases of ML and DL in economics. The application of RL and DRL in economics has a relatively short history and is in its early stages. The literature on deep reinforcement learning in economics mainly focuses on the application of deep reinforcement learning in microeconomic settings. Reinforcement learning has been applied to various economic problems, such as dynamic pricing in electricity markets, auction theory, portfolio management and asset pricing.

Feng et al. (2018) model the rules of an auction as a neural network and use deep learning for the automated design of optimal auctions. They discover new auctions with high revenue for multi-unit auctions with private budgets, including problems with unit-demand bidders. Zheng et al. (2020) employ reinforcement learning to examine and decide on the actions of agents and a social planner in a gather-and-build environment. They demonstrate that AI-driven tax policies enhance the trade-off between equality and productivity over baseline policies.

Dütting et al. (2021) model an auction as a multi-layer neural network, frame optimal auction design as a constrained learning problem, and show how it can be solved using standard machine learning pipelines. They demonstrate generalization limits and describe extensive experiments, recovering essentially all known analytical solutions for multi-item settings, and propose new mechanisms for settings in which the optimal mechanism is unknown.

While being limited, there is also a growing body of literature on the application of reinforcement learning to macroeconomic models. In macroeconomics literature, Deep RL algorithms have largely been used in a few domains. One of them is to use reinforcement algorithms to find the optimal possible policy or policy response function, as in Hinterlang and Tänzer (2022) and Covarrubias (2022). This field encompasses general equilibrium model solving as well, as demonstrated by Curry et al (2022).

The learnability of rational expectation solutions in a general equilibrium model with multiple equilibria is also a topic Chen et al (2021) study. By using a representative agent with numerous equilibria in a monetary model,

they demonstrate that the RL agent can locally converge to all of the stable states that the monetary model describes.

Modeling rationality and bounded rationality is another area of emphasis. Hill et al. (2021) demonstrate how to solve three rational expectations equilibrium models using discrete heterogeneous agents as opposed to a continuum of agents or a single representative agent. Shi (2021) investigates RL agents' consumption-saving behavior in a stochastic growth setting. She focuses on the differences in learning behaviors that occur when RL agents vary in terms of their exploration levels, and how this affects the convergence of optimum policy.

Similar to previous research, our work adds additional evidence testing a DRL algorithm in a macroeconomic model. However, we implement a representative DRL agent in an RBC model, which is served as a fundamental building block for the commonly used New Keynesian DSGE models.

II. A Real Business Cycle (RBC) Model

The baseline RBC model contains identical and infinitely lived households and firms. The business cycle fluctuations are generated by real shocks, i.e., a technology shock to productivity. In this specification, the households own the firms and rent out capital. The firms issue both debt (bonds) and equity (dividend).

A. Households

A household makes consumption-saving and work-leisure decisions. He maximizes expected utility:

$$E_0 \sum_{t=0}^{\infty} \beta^t u(c_t, 1 - h_t)$$

subject to the constraints:

$$x_t + c_t + b_{t+1} \leq w_t h_t + r_t k_t + R_t b_t + \Pi_t$$

$$k_{t+1} \leq (1 - \delta) k_t + x_t$$

$$k_t \geq 0$$

k_0 is given and the maximization also satisfies the transversality condition.

c_t denotes consumption, x_t denotes investment, b_{t+1} denotes bond holding, w_t hourly wage rate, h_t denotes hours worked, r_t denotes return on capital, k_t denotes capital, R_t denotes interest rate on bond holding, and Π_t denotes dividend payment.

$h_t \in [0,1]$ in period t, and the consumer receives utility from leisure.

The choices the consumer makes at time t are $(x_t \; or \; k_{t+1}, c_t, \; b_{t+1}, h_t)$, given time t information and the interest rate on bonds, R_{t+1}.

A.1 Optimization under Rational Expectation

This section as well as Section B.1 derive the optimization conditions under the rational expectations assumption. The aim is to compare learning results of a DRL RBC model with the rational expectations solution. In implementing a DRL algorithm, the first order conditions including the Euler equation are not required. The representative household's Lagrangian is,

$$L = E_0 \sum_{t=0}^{\infty} \beta^t \{u(c_t, h_t) + \lambda_t(w_t h_t + r_t k_t + R_t b_t + \Pi_t - c_t - k_{t+1} + (1-\delta)k_t - b_{t+1})\}$$

The first order conditions are:

$$\frac{\partial L}{\partial c_t} = 0 \leftrightarrow u^c(c_t, h_t) = \lambda_t$$

$$\frac{\partial L}{\partial h_t} = 0 \leftrightarrow u^h(c_t, h_t) = \lambda_t w_t$$

$$\frac{\partial L}{\partial k_{t+1}} = 0 \leftrightarrow \lambda_t = \beta E_t \lambda_{t+1}\{r_{t+1} + (1-\delta)\}$$

$$\frac{\partial L}{\partial b_{t+1}} = 0 \leftrightarrow \lambda_t = \beta E_t \lambda_{t+1}(R_{t+1})$$

$\frac{\partial L}{\partial c_t}$ and $\frac{\partial L}{\partial k_{t+1}}$ yield:

$$u^c(c_t, h_t) = \beta E_t u^c(c_{t+1}, h_{t+1})(r_{t+1} + 1 - \delta)$$

$\frac{\partial L}{\partial c_t}$ and $\frac{\partial L}{\partial b_{t+1}}$ yield:

$$u^c(c_t, h_t) = \beta E_t u^c(c_{t+1}, h_{t+1}) R_{t+1}$$

$\frac{\partial L}{\partial c_t}$ and $\frac{\partial L}{\partial h_t}$ yield:

$$u^h(c_t, h_t) = w_t u^h(c_t, h_t)$$

B. Firms

A profit maximizing firm's problem is:

$$\max_{K_t, H_t} e^{z_t} F(K_t, H_t) - w_t H_t - r_t K_t$$

where K_t is the capital input, H_t is the labour input, F is a neoclassical production function, such as the Cobb-Douglas production function, z_t follows an AR(1) process as follows.

$$z_t = \rho z_{t-1} + \epsilon_t$$

ϵ_t is sampled from a white noise process.

B.1 Optimization under Rational Expectations

The firms first order conditions give wage rate and capital rental rate equations:

$$w_t = e^{z_t} F^K(K_t, H_t)$$

$$r_t = e^{z_t} F^H(K_t, H_t)$$

The debt the firm issues is indeterminate in this setup.

C. Functional forms and parameters

Table 1 presents baseline parameters follow Cooley and Prescott (1995) for the US data.

Table 1. Baseline parameters for RBC model

Description	Parameter value	Relevant equations
Utility function parameters	χ = 1 (logarithmic utility) α = 0.64	$u(c_t, h_t) = \frac{(c_t^{1-\alpha}\,(1-h_t)^{\alpha})^{1-\chi}}{1-\chi}$ $u(c_t, h_t) = (1-\alpha)\ln c_t + \alpha \ln (1-h_t)$
Production function	$\theta = 0.4$	$F(K,H) = K^{\theta}H^{1-\theta}$
Discount rate – β	0.99	$E_0 \sum_{t=0}^{\infty} \beta^t u(c_t, 1-h_t)$
Autoregressive parameter - ρ	0.95	$z_t = \rho z_{t-1} + \epsilon_t$
Standard deviation of ϵ_t, σ_{ϵ}	0.007	$z_t = \rho z_{t-1} + \epsilon_t$
Capital depreciation – δ	0.012	$k_{t+1} \leq (1-\delta)k_t + x_t$

D. A deterministic steady state

Assume for the parameter values and functional forms presented in section C. At a deterministic steady state, $z_t = 0$, $k_{t+1} = k_t = k^*$, $c_{t+1} = c_t = c^*$. The first order conditions in section A.1 and B.1 become the following steady state conditions:

$$\frac{1}{\beta} - 1 + \delta = \theta\left(\frac{k^*}{h^*}\right)^{\theta-1} \leftrightarrow \frac{k^*}{h^*} = \left(\frac{\frac{1}{\beta} - 1 + \delta}{\theta}\right)^{\frac{1}{\theta-1}} (= 124.7)$$

$$y^* = \left(\frac{k^*}{h^*}\right)^{\theta} h^* \leftrightarrow \frac{y^*}{h^*} = \left(\frac{k^*}{h^*}\right)^{\theta} (= 6.89)$$

$$i^* = \delta k^* = \delta\left(\frac{k^*}{h^*}\right) h^* \leftrightarrow \frac{i^*}{h^*} = \delta\left(\frac{k^*}{h^*}\right) (= 1.5)$$

The accounting identity gives the value of consumption[1]:

$$c^* = y^* - i^* \leftrightarrow \frac{c^*}{h^*} = \frac{y^*}{h^*} - \frac{i^*}{h^*} (= 5.39)$$

The values in parenthesis are steady state values calculated based on the parameters presented in Table 1. The steady state values can be calculated for all real variables per unit of labor input, i.e., $\frac{k^*}{h^*}$, $\frac{y^*}{h^*}$, $\frac{c^*}{h^*}$, $\frac{i^*}{h^*}$. Wage rate and capital rental rate are as follows.

$$w^* = (1-\theta)\left(\frac{k^*}{h^*}\right)^{\theta}$$

$$r^* = \theta\left(\frac{k^*}{h^*}\right)^{\theta-1} - \delta$$

III. AI Experiments

The following simulations demonstrate the learning behaviors of a representative RL agent and the economic dynamics. We first compare the agent's decisions (e.g., choice of labor hour) at the beginning of a learning process with the same agent's decisions after many simulation periods of learning. This is to show that the agent's progress of learning in an unknown environment following the framework of learning from past its own experience. We then compare the learning agent's decisions with what a rational expectations agent would make in the same environment. We also plot series of macroeconomic variables to show that the RBC model with a RL agent makes similar qualitative predictions to a conventional RBC model.

We setup two environments, one is a dynamic[3] and deterministic environment without any shocks, and the other is stochastic with technology shocks. This is to first offer a clear comparison of RL agent's behaviors with a rational expectations agent in a deterministic environment. As most macro insights are derived from stochastic models, we then highlight that the RL agent behaves and learns well in a stochastic environment as well.

Implementation

We implement DDPG algorithm in this paper. It is first introduced by Lillicrap et al (2015)[4] in the paper "Continuous Control with Deep Reinforcement Learning". The algorithm was designed to improve the issue of applying RL methods to continuous action spaces. The main idea behind DDPG is to use a DNN to approximate the action-value function. DDPG was an extension of DPG (Deterministic Policy Gradient) to continuous action spaces, using DQN (Deep Q-Network) to estimate the Q-function. Q-function refers to an action-value function. It reflects expected cumulative rewards. It is a mapping from a state-action pair to the expected value. For more information on deep RL algorithms, please refer to Atashbar and Shi (2022a).

DDPG algorithm has been the harbinger of modern Reinforcement Learning and has been the launchpad for the development of many other interesting RL algorithms. One offshoot of DDPG is called TD3 (Twin Delayed DDPG) which uses a clipped double-Q function for learning policies. Another offshoot is MADDPG (Multi-Agent Deep Deterministic Policy Gradient), which is an extension of DDPG to the so-called "centralized training with

[1] Ratio $\frac{i^*}{y^*} = 1 - \left(\frac{\frac{c^*}{h^*}}{\frac{y^*}{h^*}}\right) = 0.28$

[3] The state variables depend on past actions of the AI agent, as illustrated in the transition equations cell in Table 3. The first environment is both deterministic (absence of exogenous shocks) and dynamic.

[4] Full algorithm is attached in the annex. More advanced algorithm such as soft actor critic is also developed that can achieve a more stable learning.

decentralized execution" or learning to play multi-agent environments that are partially observable to each agent. DDPG also inspired a more recent algorithm called SAC (Soft Actor-Critic), which is a variation of TD3 that uses entropy to encourage exploration. As DDPG was developed by Google DeepMind, it is sometimes called DeepMind DDPG.

DDPG has been used in a variety of tasks and has been shown to be successful in learning a variety of control tasks. In particular, DDPG has been used to control robotic arms, flying drones, and walking robots.

DDPG concurrently learns a Q-function and a policy. It uses the Bellman equation to learn the Q-function and uses the Q-function to learn the policy. The DDPG algorithm is an Actor-Critic algorithm. The Actor represents the policy, and the Critic represents the Q-function. The Actor Network is trained using the Policy Gradient, while the Critic Network is trained using Temporal Difference Learning.

The algorithm uses replay memory in order to break the correlation between samples. The replay memory stores samples and then randomly selects a batch of samples to train on. This prevents the algorithm from getting stuck in a local minimum.

The algorithm also uses a target network for both the Actor and the Critic. The target network is a copy of the original network which is updated using a Polyak averaging. This prevents the algorithm from diverging.

One advantage of DDPG is that it can learn from actions that are suboptimal or even wrong according to the current policy. The off-policy nature of DDPG is useful in real-world situations where the agent is allowed to explore to find the optimal policy. It's also useful in many tasks where the agent may not always have complete information about the environment, so it may have to learn from suboptimal actions. For example, in a 3D game, the agent may not always know where enemies are, so it may have to learn from its mistakes in order to find them. In another example, the agent has to learn to open a door and occasionally smashes it with its head. DDPG can learn from this mistake and still learn to open the door correctly most of the time.

In an economic example, a central bank may want to keep inflation low, but sometimes it will have to allow some inflation in order to achieve other goals, such as full employment. The central bank can learn from its mistakes during fine-tuning and still keep inflation low most of the time.

A disadvantage of DDPG is that it can take a long time to converge to the optimal policy. This is because the agent is constantly exploring and trying new actions, even if they are suboptimal. This can be a problem in tasks where the agent needs to find the optimal policy quickly, such as in a real-time game.

Another disadvantage of DDPG is that it can be unstable. This is because the agent is constantly updating its policy and value function, which can lead to oscillations in the values. This can make it difficult for the agent to converge to the optimal policy. In the DDPG algorithm, the agent must collect experience by directly interacting with the environment. The experience for each period contains state, action, reward and next state. The collected experience is used to update the agent’s policy and value functions. Both policy and value functions are approximated by their respective ANNs. The policy network updates with the goal of maximizing the prediction made by the value function. The value function parameters update with the goal of minimizing the temporal difference errors.

In a typical DDPG algorithm, there are generally four types of parameters that need to be tuned:

1. Hyperparameters (exploration parameters): These parameters are used to control the learning rate, exploration rate, and related factors.

2. Neural network parameters: These parameters are used to control the weights and biases of the neural networks.

3. Replay parameters (Memory parameters): The size of the replay buffer and the batch size for training need to be tuned.

4. Algorithm parameters: The parameters of the DDPG algorithm itself need to be tuned. Actor and Critic learning rates, soft update parameters, and other parameters may need to be tuned. These parameters are also used to control the algorithm's behavior, such as the number of episodes or the number of steps per episode.

Table 2 shows parameters related to the AI algorithm and the ANN architecture. Parameters such as number of episode and steps per episode indicate the length of simulation periods and can be adjusted depends on the underlying question. Feedforward ANN with two hidden layers are used to approximate the policy and the value functions. Each hidden layer has 16 nodes, which is a simple ANN architecture comparing to those used for the game of Go[5], or speech recognition.

Table 2 Algorithm related parameters

Parameters	Baseline	Descriptions
Number of episodes	200	A measure of simulation periods. The number of episodes can affect the performance of the agent in reinforcement learning. If the agent experiences too few episodes, it may not have enough data to learn from. If the agent experiences too many episodes, it may overfit to the data and not generalize well to new situations.
Steps per episode	2000	The number of periods within an episode. It is a measure of how difficult the task is. A task that is easy to learn will require fewer timesteps to complete an episode, while a task that is difficult to learn will require more timesteps. The number of timesteps can also be used as a measure of how efficient the learning algorithm is. A learning algorithm that is very efficient will require fewer timesteps to complete an episode, while a learning algorithm that is less efficient will require more timesteps.
Updates per step	5	Number of times the ANNs are updated within a simulation period. In reinforcement learning, updates per step can be thought of as an analogy for how many times an agent has experienced a particular environment. In general, the more updates per step, the faster the learning process. However, too many updates per step can lead to instability and divergence.
Batch size	128	Sample size refers to the number of episodes or experiences sampled in each training iteration. A larger batch size generally results in a model that converges faster, but may be more prone to overfitting, instability and poorer performance. A smaller batch size allows for more fine-grained updates but may take longer to converge.
Hidden layer size	16	Number of neurons in a hidden layer. Size of hidden layers is important because it determines the capacity of the network to learn complex patterns. If the hidden layer size is too small, the network will not be able to learn complex patterns. If the hidden layer size is too large, the network will overfit the data.
Number of hidden layers	2	Number of hidden layers. The number of hidden layers in a deep neural network is often referred to as the

[5] Due to its enormous search space and the difficulty of evaluating board positions and moves, Go has long been regarded as the most difficult of the traditional board games for artificial intelligence. (Silver et al., 2016).

		model's depth. The more hidden layers in a network, the more complex the model. In general, as the depth of a network increases, the network becomes more difficult to train. This is because each additional layer introduces more parameters that need to be optimized, and the optimization problem becomes more non-linear.
Learning rate	Actor: 1e-4 Critic: 1e-3	Governs how quickly the ANN updates based on new information. It controls to what extent newly acquired information overrides old information.
Activation functions	Actor: ReLU[6] for each linear layer and Tanh for output layer Critic: ReLU for all	Activation functions are used in artificial neural networks to map the output of the network to a value that is usable by the next layer. In DDPG, a common activation function is ReLU, or Rectified Linear Unit. ReLU is a common activation function because it is simple to compute and has a range of 0 to 1. ReLU is also a common activation function because it is differentiable. This means that the gradient of the ReLU function can be computed, which is important for training the neural network. Tanh is another common activation function and maps the output of a node to a value between -1 and 1. Tanh is also differentiable, which means that the gradient of the Tanh function can be computed. There are a variety of other activation functions that can be used, such as sigmoid, softmax, and linear. Each of these activation functions has different properties that make it suitable for different types of neural networks.
Exploration	From 1.0 to 0.3, diminish by episode	It is used to control the amount of randomness in the exploration of the state space. A higher exploration parameter results in more randomness, meaning that the agent is more likely to explore new states that it has not visited before. A lower exploration parameter results in less randomness, meaning that the agent is more likely to exploit the states that it has already visited and that it knows are rewarding.

Table 3 shows different components of RL that are represented in the RBC model. For the learning representative household, only its utility function and budget constraint are needed. Its first order conditions, such as the Euler equation, is not used when implementing the RL algorithm. This means that the only information needed for setting up a RL agent is the state variables, the choices of the agents and how state transitions without needing the agent's optimality conditions.

At each period, the agent observes the state variables, and makes an action. The action includes labor hour, h_t, proportion of investment, λ_t^i, and changes in bond position, λ_t^b. The reward this agent receives is dependent on the consumption level and labor hour, and the reward function is logarithmic. As we consider an RBC model in this paper, the state space and action space are relatively small. DRL algorithms are designed to function well in a situation that involves large state and action spaces, such as learning the game of Go. Depends on the economic question and the models adopted, the algorithms can be implemented with large state and action spaces.[7]

[6] ReLU is short for rectified linear unit. Is it the default activation function for most recent applications of feedforward ANNs. Tanh is short for hyperbolic tangent activation function, and it is often used when the desired output is within range (-1, 1).

[7] This may require a larger ANN, i.e., an ANN with more depth or more nodes per layer.

The state of the environment transitions according to the transition equations presented in Table 3. Only objective functions, budget constraints, as well as aggregate identities are used for the DRL learning implementation. When modelling through a DRL algorithm, the agent does not need to know the first order conditions (including the Euler equation). It gathers experiences through taking actions and exploring the environment. This means that at each period the RL agent observes realized state variables, takes an action, and receives the reward given its action. The experiences are then used as past data to update the policy and value functions (two ANNs).

Table 3 RL set up of the RBC model

Component	Setup
State	Period t state variables include: $k_t, z_t, \frac{k_{t-1}}{h_{t-1}}, \lambda^b_{t-1}, R_{t-1}$
Action and action space	Period t action variables include: $h_t \in (0,1)$ $\lambda^i_t \in (0,1)$ $\lambda^b_t \in [1,2]$
Reward function	Logarithmic utility: $u(c_t, h_t) = (1-\alpha)\ln c_t + \alpha \ln(1-h_t)$
Transition equations	$K_t = k_t$ $H_t = h_t$ $y_t = e^{z_t} K_t^{\theta} H_t^{(1-\theta)}$ $i_t = \lambda^i_t y_t$ $r_t = \theta e^{z_t} K_t^{\theta-1} H^{1-\theta}$ $R_t = r_t + 1 - \delta$ $\Delta^b_{t+1} = \frac{b_{t+1}}{b_t} - R_t = \lambda^b_t - R_t$[8] $c_t = (y_t - i_t) - a\Delta^b_{t+1}$ $a > 0$ is a constant and assumed to be $a = 10$

A. Experiment I: deterministic environment

In reinforcement learning, an environment is said to be deterministic if the next state of the environment is completely determined by the current state and the agent's current action. For instance, in chess, the next state of the board is completely determined by the current state of the board and the move that the agent makes. In contrast, an environment is stochastic if the next state is only partially determined by the current state and the agent's current action. For instance, in backgammon, the next state of the board is determined by the current state of the board and the roll of the dice, which is random.

The type of environment can have a significant impact on the performance of a reinforcement learning algorithm. In general, algorithms tend to perform better in deterministic environments than in stochastic environments. This is because in a deterministic environment, the agent can always be sure of what the next

[8] Does not run into issues with negative Δ^b because the production is Cobb-Douglas and its first derivative with respect to capital is greater than 0.

state will be, and so it can plan its actions accordingly. In a stochastic environment, on the other hand, the agent can never be sure of what the next state will be, and so it has to take into account the possibility of different outcomes. However, they can also be difficult to learn in since the agent may become stuck in a local optimum. In contrast, stochastic environments may be more difficult to learn initially, but can often lead to better long-term performance, since the agent can explore different states and find new optimums.

We first assume for a deterministic environment, and no productivity shock, i.e., $e^{z_t} = 1$ and $z_t = 0$. This is for an easy comparison between learning behaviors and the optimal behaviors under rational expectation scenario.

The representative household is assumed to be the learning agent. Firms are "rational", meaning it maximizes profit, sets competitive wage and assume for a rational household. Rate of return on capital and return on bonds satisfy the equilibrium condition: $R_t = r_t + 1 - \delta$.

Results

This section shows simulation results under the baseline parameters in Table 2. Most figures show simulated series during both training and testing periods. When an agent engages in exploratory behavior to interact with the environment, it is referred to as training. The agent's policy and value functions are then updated as a result of the interactive experience. Testing entails demonstrating how the agent behaves if the learning process is halted, i.e., when the policy and value functions are not updated and when the agent's actions are not accompanied by noise from exploration.

Following our analytical strategy, Figure 2 compares a RL agent's choices of labor hour at the beginning of a learning process with its decisions after it has learnt for 190 episodes. It plots the frequencies of labor hour choices within the action set, i.e., $(0, 1)$. The green bins denote choices made at the first 10,000 steps, whereas the blue bins plot the last 10,000 steps of training. The green bins show that at the beginning of a learning process, the agent makes more random actions. As the agent acquires more knowledge about the environment and learns more about its own preference, its decision of labor hour is more centered and concentrate towards the deterministic steady state value of 0.33.

Figure 3 plots the simulated series of labor hours for 500 periods (or steps). The dash-dot line represents the deterministic steady state value of labor hour. The solid line is the series during testing periods. The dotted line is the series during training period. We demonstrate that the agent learns to choose the labor hour that is close to the optimal choice as measured by the deterministic steady state value.

Figure 2 Labor hours during training (200 episodes)

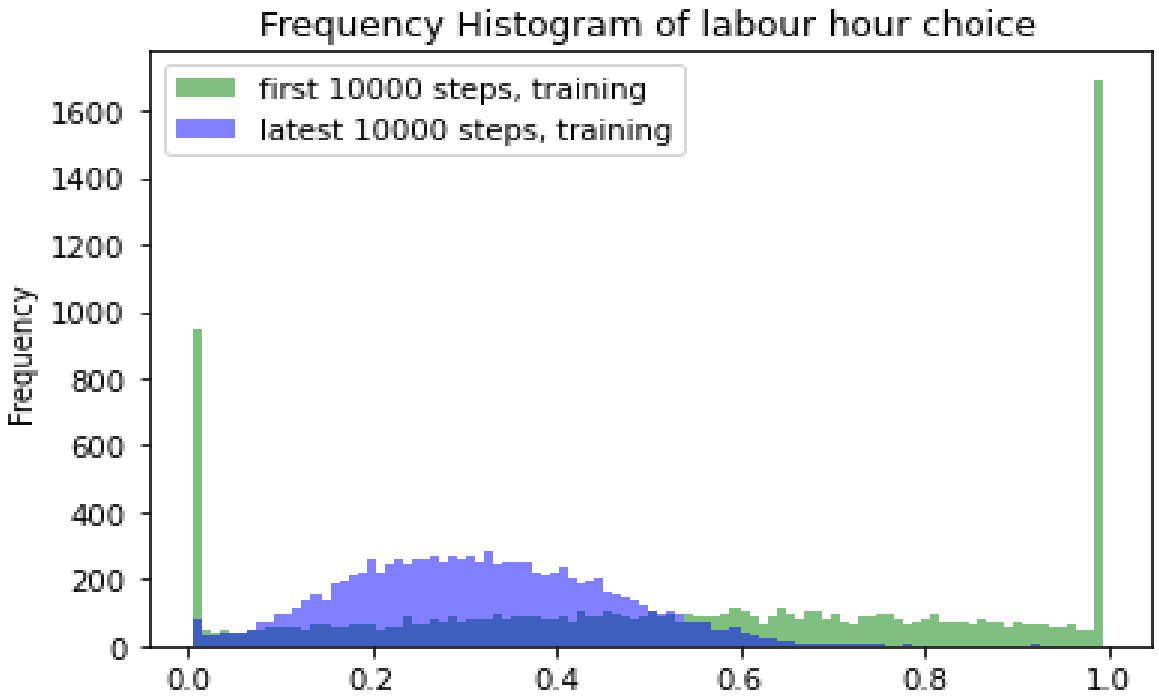
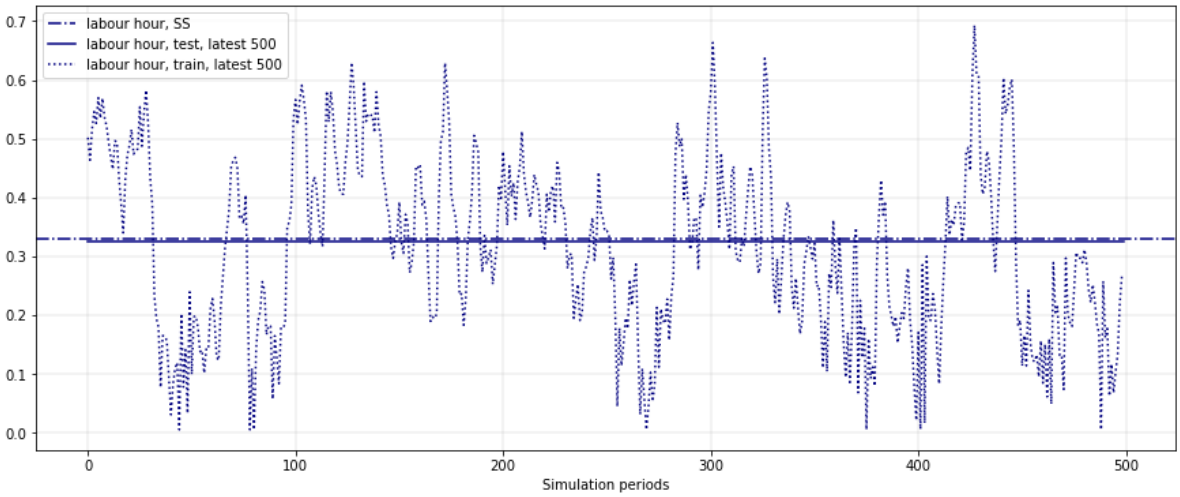


Figure 3 Labor hour series during training and testing



To compare and illustrate the learning process, we calculate the mean squared distance between the RL agent's choice and the deterministic steady state value, similar to Shi (2021) and Chen et al (2021). In particular, the distance is calculated as

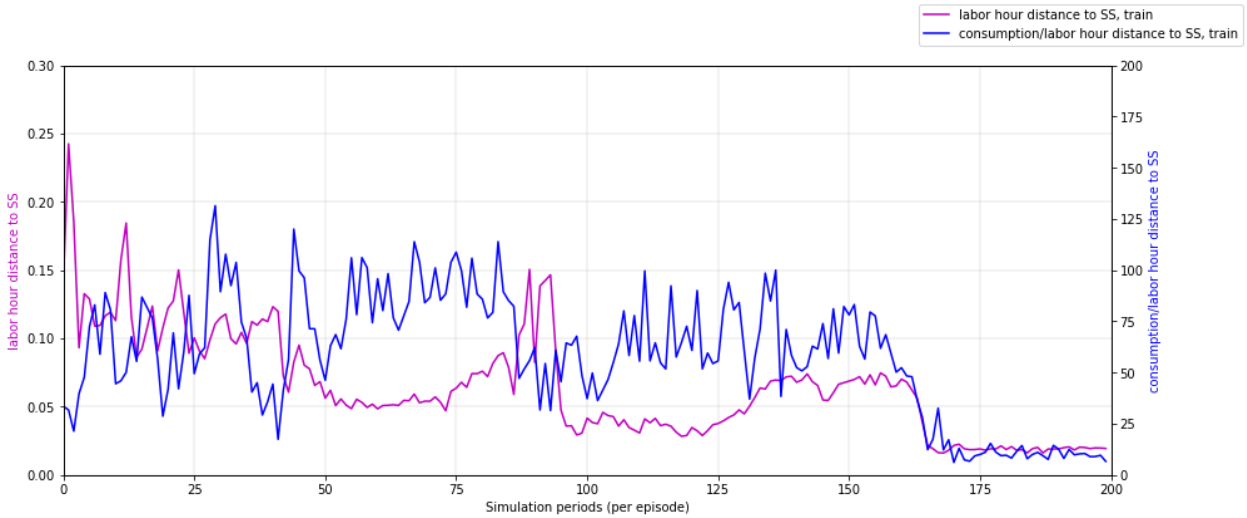
$$d_t = (\bar{x}_t - x^{ss})^2$$

where t denotes episode, x denotes a variable of interest, e.g., labor hour or consumption. $\bar{x}_t$ represents the mean of variable of interest for learning episode t. x^{ss} represents the steady state value.

A smaller distance, i.e., d_t, means that the RL agent's decision is close to a rational agent's decision.

We calculate the distance between the RL agent's choice and the steady state value for each learning episode. Figure 4 plots the distance per learning episode for labor hour and consumption per labor hour. The left y-axis shows the distance series for labor hour over learning episodes, and the righthand y-axis shows the distance series for consumption. Both lines show gradual decline as the agent learns for longer. However, the decline is not monotonic to learning episode. This could be due to a number of reasons and requires further investigation. For example, a change in learning parameters, e.g., exploration level and the neural networks learning rate, may have an impact on how smooth the series can be over time. In addition, aggregate many simulations might average out the volatility observed in the plot.

Figure 4 Distance the steady state (SS) values for labor hour and consumption



Using varying parameter values and ANN architectures, the agent could acquire this belief prior to 200 episodes of learning. Due to the fact that the agent must generate its own experience, learning requires a substantial number of iterations.

In a deterministic environment, we show that an RL agent at first make random decisions. This is part of the process to collect experiment through an agent-environment interactive process. Once the agent starts to update its beliefs, its decisions gradually converge to what a rational agent would do in the RBC model. This is consistent to the findings in Hinterlang and Tänzer (2022), Chen et al (2021), and Shi (2021). Moreover, the distance between an RL agent's decision and a rational expectations agent reduces over time. However, the convergence, as measured by the mean squared distance, is not monotonic in this experiment. This could be due to the exploration parameter that leads to volatility in the RL agent's decisions. The extend of this requires further simulation evidence. Furthermore, in this experiment, parameters of the learning framework is uncalibrated. This corresponds to a slow learning process.

B. Experiment II: stochastic environment

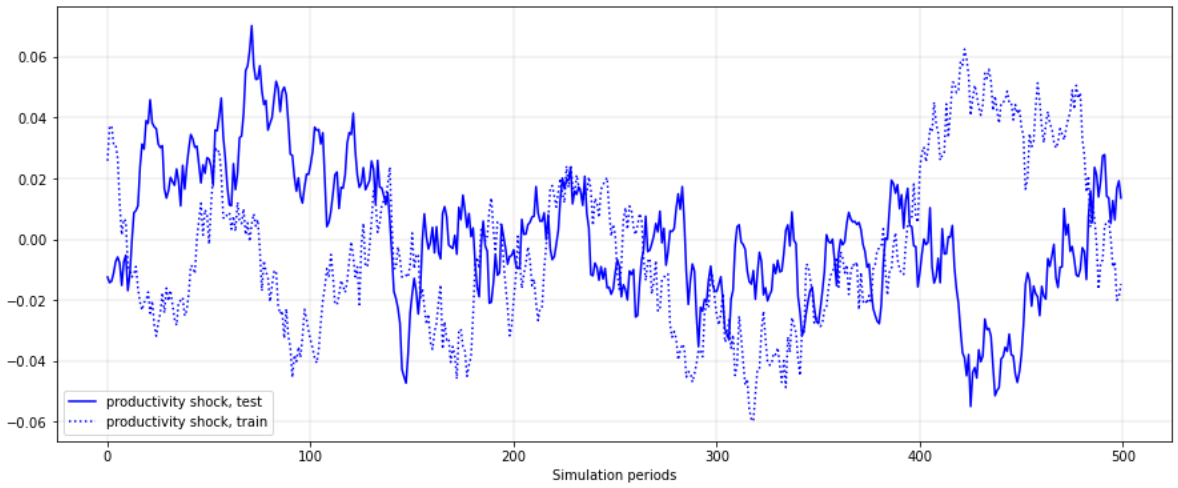
In the stochastic environment, productivity shock is present, and it is as follows.

$$z_t = \rho z_{t-1} + \epsilon_t$$

The household is assumed to be the learning agent, and the firm is assumed to be rational.

Figure 5 plots the stochastic factor in this experiment, i.e., the z_t series for 500 simulation periods.

Figure 5 Productivity shock series z_t



Following five episodes of learning, the agent's consumption, output, and investment per unit of labor are shown in Figure 6 over the course of 100 testing periods. For the same simulation period, it also plots this agent's labor hour preference. The blue and green lines (consumption per unit of labor and output per unit of

labor, respectively) move in the same direction, as predicted by many macro theories and empirical evidence[9]. A high investment in one period causes a high output in the following period, as the yellow line shows.

Figure 6 Simulated series during 100 testing periods

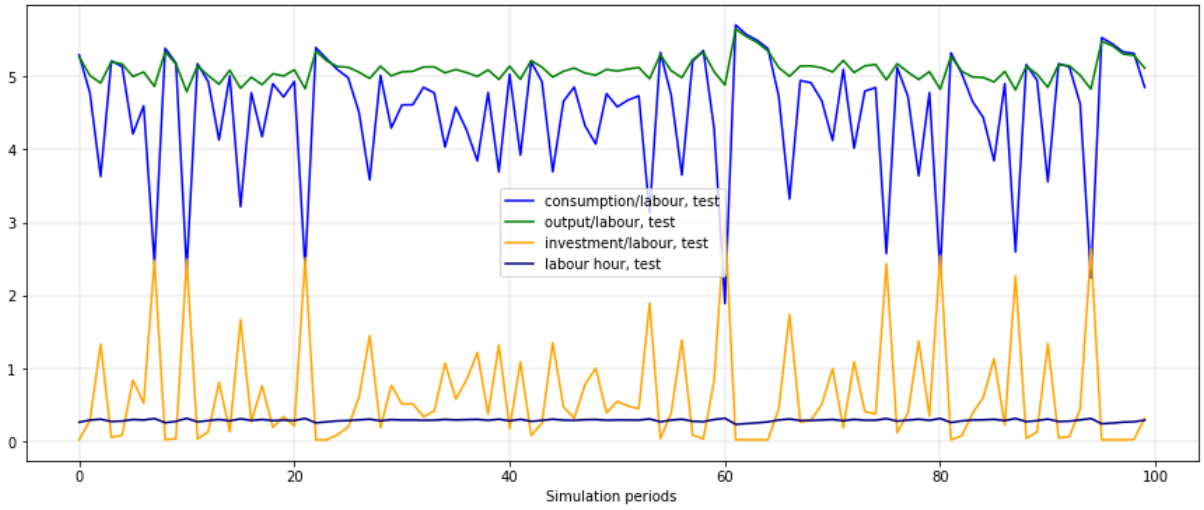


Figure 7 plots the frequency histogram of AI agent's choice of labor hour at the first 10,000 steps of learning and at the last 10,000 steps of learning. The green bins are the choices the agent made at the beginning of a learning process. As the agent learns in this environment, its labor hour choices concentrate towards the optimal choice of 0.3.

[9] In such models, as workers become more productive, they also raise their consumption. This is consistent with the models in which workers are rewarded with wages and consumption growth (more productive workers are typically given higher wages, and as a result, generally consume more). This is also consistent with the neoclassical growth theory, which suggests that technological progress is labor augmenting, i.e., it raises both output per worker and consumption per worker. When technological progress is labor augmenting, the correlation between consumption per worker and output per worker tends to be positive.

Figure 7 Labor hour choice before and after learning (200 episode)

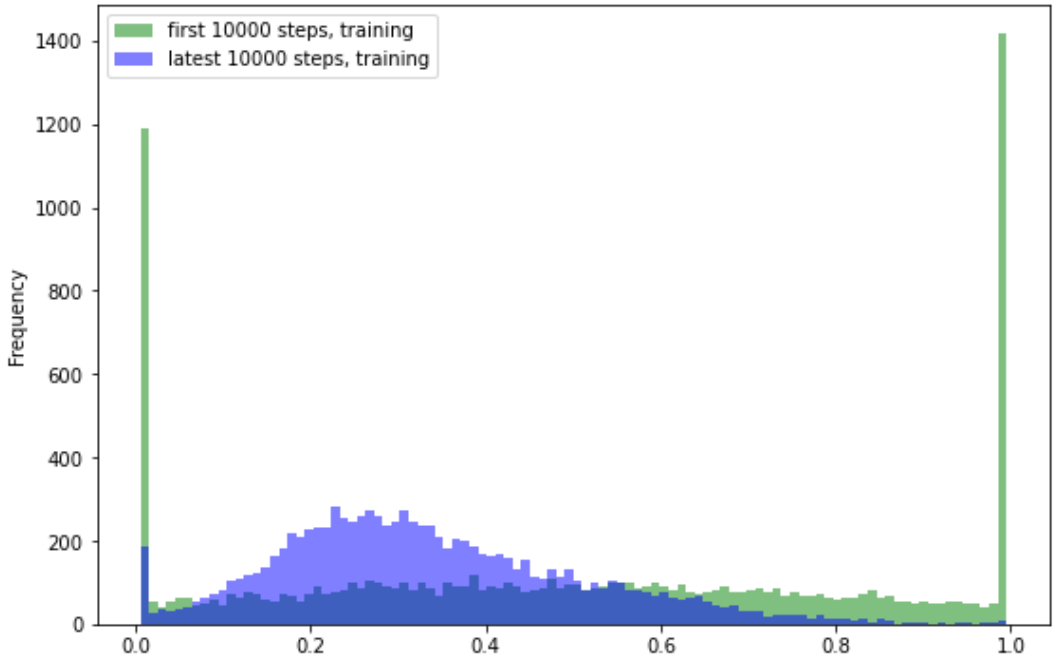


Figure 8 and 9 plots the distance to deterministic steady states for variables including consumption, output, investment per unit of labor for 200 simulation periods. The distance at each episode is calculated according to the equation presented in Experiment I, i.e.,

$$d_t = (\bar{x}_t - x^{ss})^2$$

All distance series in figures 8 and 9 show a gradual decline, which means the RL agent behaves similar to a rational expectations agent as it learns in the environment for longer. Similar to the results in the deterministic environment, the decline is not monotonic to learning episodes.

Due to the stochastic nature of this environment, the distance series can still be volatile once the RL agent acquires enough knowledge and information to act similar to a rational expectations agent. For example, the series after episode 125 in figure 8 appear to be more volatile than the series plotted in figure 4.

However, an interesting observation is that the RL agent appears to learn quicker in a stochastic environment than the deterministic one by comparing figure 8 with figure 4. In figure 4, a small distance is only observed after around episode 165, whereas in figure 8, the RL agent acts similar to a rational expectations agent after episode 125. One explanation is that in a stochastic environment, the agent gathers experience in a wide range of states in early learning periods due to the stochasticity in the environment. Hence, the agent learns to act quicker.

Figure 8 Distance to deterministic steady states (SS) for labor hour and consumption

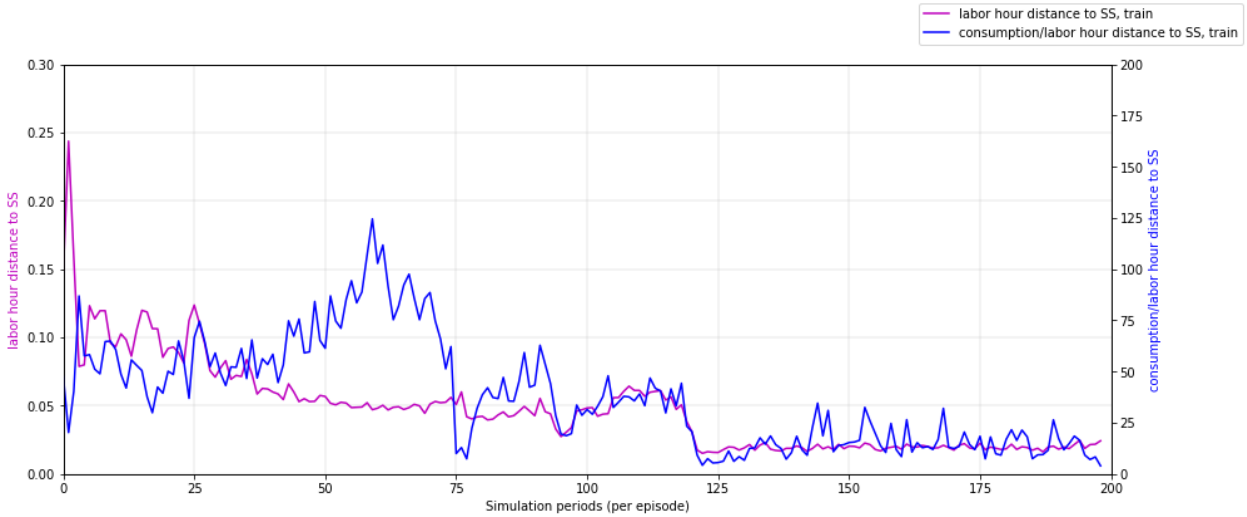
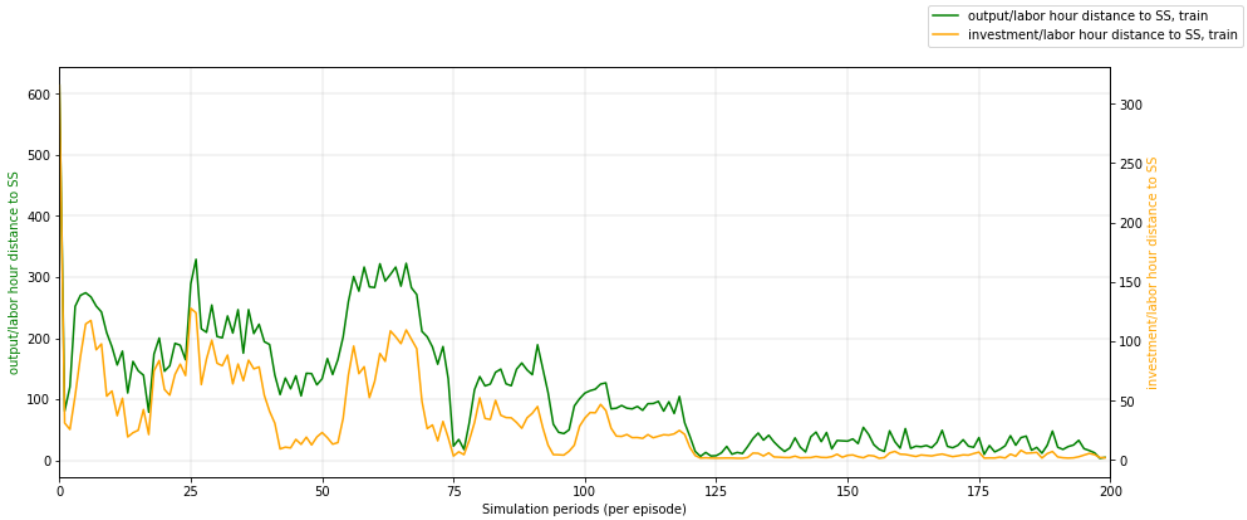


Figure 9 Distance to deterministic steady states (SS) for output and investment



In this experiment with the stochastic environment, we show that the RBC with a representative RL agent makes similar predictions to the model with a rational expectations agent. Moreover, we show that the RL agent learns to make optimal decisions over time through a distance metric. Although, the agent's behaviors can be more volatile in the stochastic environment, it learns quicker than the agent in a deterministic environment. We propose that this is because the agent experiences more states in a stochastic environment than in a deterministic one, Therefore, it learns to make optimal choices quicker than living in a deterministic environment.

C. Issues during learning

Instability of learning

Many DRL algorithms are not proven to converge to optimal solutions of a problem. This might be brought on by the exploration noise that was added to the action. Additionally, it's possible that the learning rate is not set correctly, the gradient estimation is too noisy, or even the network is not deep enough. These could result in convergence to suboptimal solutions. Matheron et al. (2019) argue that DDPG algorithm can be unstable in trivial tasks, which is still not well understood. There are a number of ways to try to improve the stability of DDPG. One approach is to use a different noise process for exploration, such as adding Ornstein-Uhlenbeck noise to the actions instead of Gaussian noise. Another approach is to use a different function approximation, such as a long short-term memory (LSTM) network instead of a fully connected network. Finally, it is also possible to try to modify the way in which the DDPG algorithm learns, such as using a different learning rate or using a different optimization algorithm.

In Figure 10 and 11, we plot output and investment per unit of labor after an AI agent learns for 200 episodes in a deterministic environment. This is an example of the instability of learning. The agent is trapped in a state where it does not make an optimal choice but refuses to move towards the optimal direction. Even after further learning and training, the agent remains in this state. Kuriksha (2021) also documented learning traps using ANNs.

The frequency of such events occurring remains unclear and it is also case-dependent. This requires researchers' subjective judgement when analyzing results.

Given that the environment is set up correctly, in some cases, this learning trap can be resolved simply by starting the learning from the beginning. In other cases, it might imply issues with hyperparameters or ANN architectures. Different parameter values might need to be explored.

Figure 10 Output per unit of labor

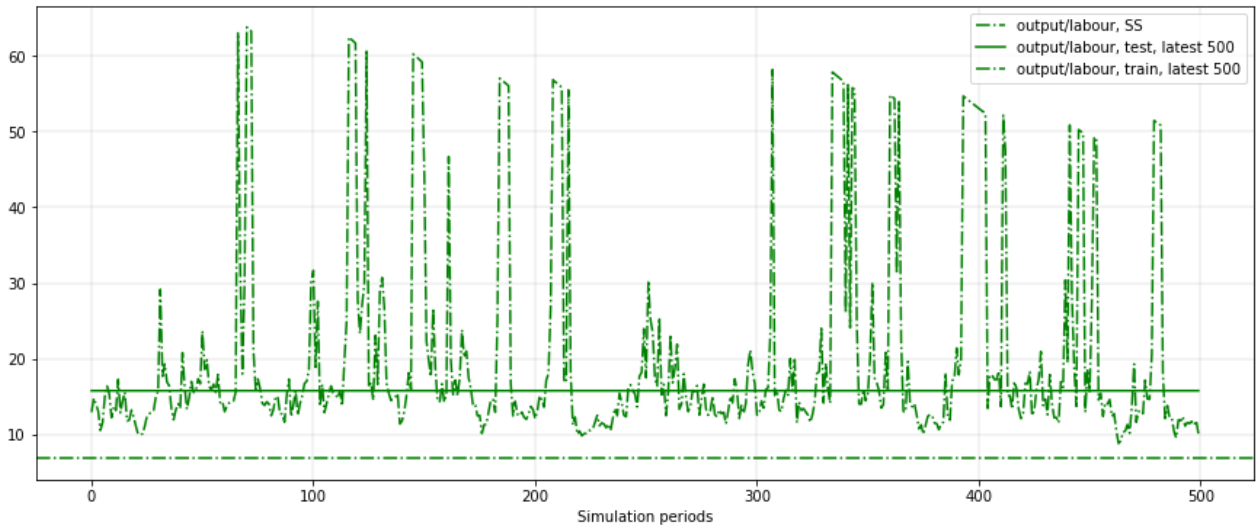
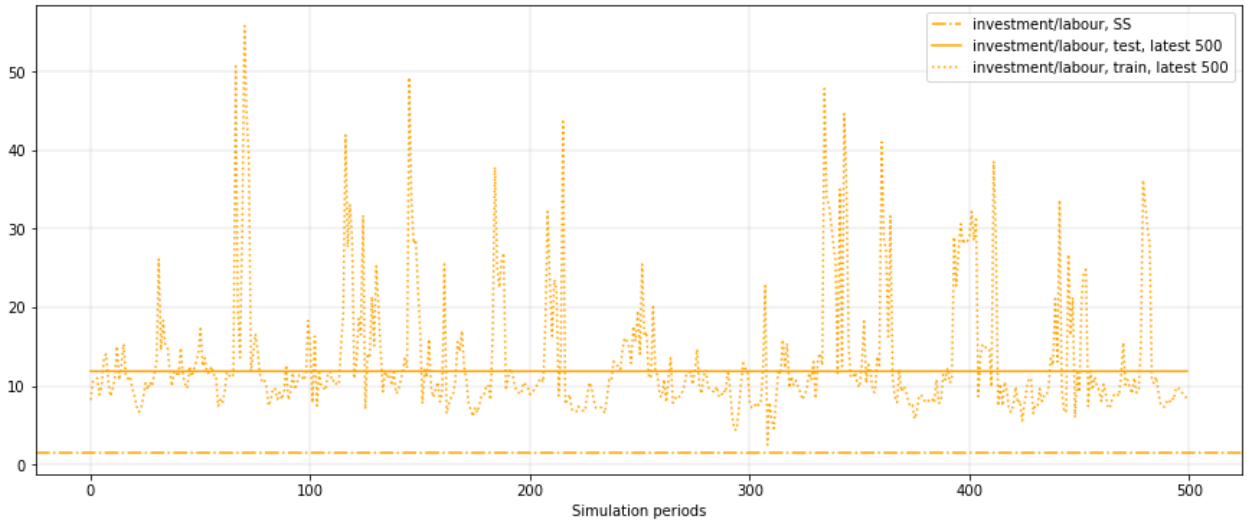


Figure 11 Investment per unit of labor



Parameter selection

Algorithm-related parameter selection and the choice of ANN architecture can be a challenge. There is no general guidance on how to set these parameters, and it might be difficult to find values that work well for a specific problem. But a rule of thumb might be to start with simple ANN architectures and then increase the complexity if needed. Some techniques for alleviating the issue are to use a grid search or an evolutionary algorithm.

Difficulty in reproducing results

It can be difficult to reproduce results with DRL algorithms, since there is a lot of randomness involved in the learning process. This is especially true for algorithms that use stochasticity in the learning process. To try and reproduce results, it is important to use the same random seed for the algorithm. Additionally, it is important to keep track of all the hyperparameters that were used, as well as the specific details of the problem that was being solved.

IV. Conclusion

This work aimed to develop a reasonably basic and extensible AI-macroeconomic simulator based on reinforcement learning. To do that, we deployed a modern deep RL (DRL) method (DDPG) in a real business cycle (RBC) macroeconomic model.

In this AI experiment, we have one learning agent, which is the representative household. It learns to make consumption-saving, and labor-leisure decisions to maximize long-term rewards. Reward function is assumed to be the same as a logarithmic utility function. The learning agent's policy and value functions are approximated by randomly initialized ANNs. The agent observes current period realized technology shock, and its previous choices (consumption, investment, bonds, labor).

We setup two learning environments, one is deterministic without the technology shock, and the other is a stochastic environment. The purpose of a deterministic environment is to compare the learning agent's behaviors with a deterministic steady state scenario.

Parameters underlying the RBC model follow Cooley and Prescott (1995). Parameters related to the DDPG algorithm, and the ANN architecture largely follow existing AI literature with adaptations to the current problems. For example, most existing DDPG implementation uses a large ANN architecture with at least hundreds of nodes for each layer. Given the current problem is relatively simple, we reduce the number of ANN nodes to be 16 per hidden layer.

In the simulation experiments, we plot simulated series of consumption, output, investment, and labor hour. We show that the deep RL agent RBC model provides predictions that are consistent to conventional RBC models. As a benchmark, we also plot the deterministic steady state values of the RBC model under the rational expectations assumption. We demonstrate that in both deterministic and stochastic contexts, the agent's decisions after learning are near to the optimal value. Due to the stochastic character of the environment, however, the series in stochastic setup are more volatile than in deterministic setup, and learning tends to be less stable.

This basic AI-macro structure might be expanded by adding more variables or sectors to the model or using other DRL algorithms in the model in the next studies.

Annex I. DDPG algorithm

The DDPG algorithm follows from Lillicrap et al (2019, page5)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

References

Andresen, S. L. (2002). John McCarthy: father of AI*. IEEE Intelligent Systems*, 17(5), 84-85.

Atashbar, T. (2021) The Future of Economics in an AI-Biased World. *World Economic Forum*.

Atashbar, T. (2021) R4: A transformer machine fine-tuned with the IMF public knowledge, V4.4.2

Atashbar, T., and Shi, R. A., (2022a) Deep Reinforcement Learning: emerging trends in macroeconomics and future prospects, *IMF Working Papers.*

Athey, S. (2018). The impact of machine learning on economics. In *The economics of artificial intelligence: An agenda* (pp. 507-547). University of Chicago Press.

Cameron, A. C. (2019). Machine Learning Methods in Economics. *Machine Learning*, 1, 67.

Cao, L. (2020). AI in finance: A review. *Available at SSRN 3647625.*

Chen, M., Joseph, A., Kumhof, M., Pan, X., Shi, R. and Zhou, X., (2021) Deep reinforcement learning in a monetary model. arXiv preprint *arXiv:2104.09368.*

Cooley, T.F. and Prescott, E.C., 1995. *Frontiers of business cycle research* (Vol. 3). Princeton, NJ: Princeton University Press.

Covarrubias, M. (2022), Dynamic Oligopoly and Monetary Policy: A Deep Reinforcement Learning Approach.

Curry, M., Trott, A., Phade, S., Bai, Y., & Zheng, S. (2022). Finding General Equilibria in Many-Agent Economic Simulations Using Deep Reinforcement Learning. *arXiv preprint arXiv:2201.01163.*

Dangeti, P. (2017). *Statistics for machine learning*. Packt Publishing Ltd.

Dütting, P., Feng, Z., Narasimhan, H., Parkes, D. C., & Ravindranath, S. S. (2021). Optimal auctions through deep learning. *Communications of the ACM,* 64(8), 109-116.

Feng, Z., Narasimhan, H., & Parkes, D. C. (2018, July). Deep learning for revenue-optimal auctions with budgets. In Proceedings of the *17th International Conference on Autonomous Agents and Multiagent Systems* (pp. 354-362).

Goertzel, B. (2007). *Artificial general intelligence* (Vol. 2). C. Pennachin (Ed.). New York: Springer.

Goldfarb, A., Gans, J., & Agrawal, A. (2019). *The Economics of Artificial Intelligence: An Agenda*. University of Chicago Press.

Graesser, L., & Keng, W. L. (2019). *Foundations of deep reinforcement learning: theory and practice in Python*. Addison-Wesley Professional.

Hill, E., Bardoscia, M. and Turrell, A., (2021) Solving heterogeneous general equilibrium economic models with deep reinforcement learning. *arXiv preprint arXiv:2103.16977.*

Hinterlang, N., & Tänzer, A. (2021). Optimal monetary policy using reinforcement learning.

Hull, I. (2021). Machine Learning and Economics. In *Machine Learning for Economics and Finance in TensorFlow 2* (pp. 61-86). Apress, Berkeley, CA.

Kuriksha, A., 2021. An Economy of Neural Networks: Learning from Heterogeneous Experiences. arXiv preprint arXiv:2110.11582.

Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

Lu, Y., & Zhou, Y. (2021). A review on the economics of artificial intelligence. *Journal of Economic Surveys,* 35(4), 1045-1072.

Matheron, G., Perrin, N. and Sigaud, O., 2019. The problem with DDPG: understanding failures in deterministic environments with sparse rewards. *arXiv preprint* arXiv:1911.11679.

Nosratabadi, S., Mosavi, A., Duan, P., Ghamisi, P., Filip, F., Band, S. S., ... & Gandomi, A. H. (2020). Data science in economics: comprehensive review of advanced machine learning and deep learning methods. *Mathematics*, 8(10), 1799.

Powell, W. B. (2021). From reinforcement learning to optimal control: A unified framework for sequential decisions. *In Handbook of Reinforcement Learning and Control* (pp. 29-74). Springer, Cham.

Ruiz-Real, J. L., Uribe-Toril, J., Torres, J. A., & De Pablo, J. (2021). Artificial intelligence in business and economics research: Trends and future. *Journal of Business Economics and Management*, 22(1), 98-117.

Shi, R. A., (2021b) Can an AI agent hit a moving target? *arXiv preprint arXiv:2110.02474.*

Shi, R.A., (2021a) Learning from Zero: How to Make Consumption-Saving Decisions in a Stochastic Environment with an AI Algorithm. *arXiv preprint arXiv:2105.10099.*

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587), 484-489.

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

Tilbury, C. (2022). Reinforcement Learning in Macroeconomic Policy Design: A New Frontier? arXiv preprint arXiv:2206.08781.

Veloso, M., Balch, T., Borrajo, D., Reddy, P., & Shah, S. (2021). Artificial intelligence research in finance: discussion and examples. *Oxford Review of Economic Policy*, 37(3), 564-584.

Zai, A., & Brown, B. (2020). *Deep reinforcement learning in action*. Manning Publications.

PUBLICATIONS