Deep Reinforcement Learning in Macroeconomic Models

Matias Covarrubias

July 27, 2021

Abstract

Can artificial intelligent (AI) algorithms learn optimal intertemporal behavior without knowing any mathematical details of the economy beforehand? How should we specify economies and markets so as to make them easy to learn? In this paper, I apply Deep Reinforcement Learning (Deep RL), the current front-runner in the design of AI agents, to macroeconomic problems. In the context of an investment under uncertainty model with heterogenous agents, I show that Deep RL agents can learn the rational expectations solution and I suggest design choices that aid such learning. Then, by manipulating our baseline framework I highlight three topics in which this technology can be useful: high dimensional problems, incomplete information problems and equilibrium selection in models with multiple equilibrium.

1 Introduction

- Paragraph 1 (reframe) :Can artificial intelligent (AI) algorithms learn optimal intertemporal behavior without knowing any mathematical details of the economy beforehand? How should we specify economies and markets so as to make them easy to learn? In this paper, I apply Deep Reinforcement Learning (Deep RL), the current front-runner in the design of AI agents, to macroeconomic problems. In the context of an investment under uncertainty model, I show that Deep RL agents can learn the rational expectations solution and I suggest design choices that aid such learning. Then, by manipulating our baseline framework I highlight three topics in which this technology can be useful: high dimensional problems, incomplete information problems and equilibrium selection in models with multiple equilibrium.

- Paragraph 2: Explain model free RL.

  - RL is a class of algorithms that adapt dynamic programming techniques such as value function and policy function iteration to the problem of online learning, that is, to learn how to control a system by interacting and experimenting with it. Other historical names for this method are approximate dynamic programming, learning based control and automated optimization.

  - A key characteristic of RL algorithms is that they do not use any mathematical knowledge of the system they are controlling, other than the allowed actions and states. They feed actions to the system and observe the reward they get that period and how the state evolves. After millions of such transitions, if learning is successful, they are able to estimate the consequences of their actions accurately and thus make optimal decisions.

- Paragraph 3: Deep RL and the advent of this technology.

  - Deep RL adds neural nets as function approximator. Explain Mnih et al. [2013] paper.

  - Why it has been so effective? Scalability and computation power.

  - Success cases: games, robotics, self-driving vehicles, industrial optimization.

- Paragraph 3: Traditional challenges of Deep RL and why economics may be particularly well suited for this technology.

  - Rewards engineering is usually the main bottleneck. In many problems, such as games or robotics, you only get positive rewards after successfully achieving some final objective, so the researcher needs to design signals to indicate to the agents whether they are making progress or not. In economics, rewards are well specified in every period.

  - Also, RL algorithms usually require millions of transitions in order to learn, which may be computationally expensive. This problem is very serious in environments with high dimensional states, such as images, and with complex transitions logic, as in the case of games or robotics. In economics, sampling a transition is usually very fast, as we are only dealing with vectors whose transitions laws can be expressed with simple mathematical functions.

  - Finally, in many problems the algorithms get stuck in local optima. While this is also a potential problem in complex economic models, for many models we have nice regularity properties.

- Paragraph 5: But, economic problems present their own challenges.

  - Markets are a multi-agent problem. While multi agent Deep RL is a very active topic with some impressive success cases, it faces a crucial challenge: agents learn and explore while other agents learn and explore. This means that the environment is non-stationary and at the beginning it may be very chaotic. Whether order arises from such chaos is an empirical question with no theoretical guarantees.

  - In economics, rigorous insights often rely on exact solutions. That is, we are trying to solve very precise control problems, so usual assumptions such as discrete action spaces may not be satisfactory. Deep RL agents might get a very high percentage of the optimal discounted utility, but the solution may still lack some of the qualitative features of the exact solution. Example: growth model with fixed saving rate instead of decreasing saving rate.

- Paragraph 4: Explain why this could be useful besides the study of learning in macroeconomics:

  - With this technology, we only need to specify the economy's action space, observation space, initial state and transition laws. This makes it simple to setup an economy on which Deep RL agents can learn.

  - The algorithm's structure is mostly the same regarding the problem at hand. The algorithm only needs to adapt the dimensions of its function approximators (e.g., neural nets) to the state space and action space of the environment, which can be easily automated. Thus, the same algorithm can learn to play chess or to calculate optimal saving behavior. As a corollary, the information available to each agent can be manipulated easily, without making any changes to the algorithm. Also, different versions of the economy can be constructed without need to recalculate the set of equations that describe the optimal solution

1.1 Preview of leading framework and exercises

- My criteria for choosing a baseline framework are the following:

  - Simple economics, challenging solution: the problem should be easy to explain and with clear economics. Nevertheless, the control problem should be challenging.
  - Dynamics at the forefront: the key trade-offs of the model should be intertemporal, since we are interested in the application of deep RL to dynamic problems.
  - Meaningful scalability: the problem needs to have a clear path to scalability in which the dimensionality truly affects the economics of the problem. As such, it cannot have aggregation properties, so each variable in the state space should affect optimal behavior.

- Consistent with this criteria, I present a model of investment under uncertainty in which we focus on the purchase market for capital goods. There are a discrete number of buyers (household who produce, consume and invest) and sellers (each producing a different type of capital good). More precisely:
- N^h households indexed by choose how much to consume and how much to invest in order to maximize the discounted intertemporal utility..
- The final good is produced using N^c capital goods. All capital goods are bundled with a Cobb-Douglas aggregation and the production function have decreasing returns to scale over that bundle. The total factor productivity of each households follows a markov chain.
- The agent allocates current production between consumption and investment in new capital goods. The cost in term of the final good of producing new capital goods is quadratic in investment.
- We will consider both a market's formulation and a planner's formulation of the problem. In the market formulation, for each capital good there is a firm who pays the quadratic cost and sells the capital good. In the planner formulation, the planner decides how much of the final good to allocate in each capital good for each household, and investment is determined by how many units can be produced given the total allocation of resources by all households on that capital good.
- Within this framework we perform the following exercises:
  - Exercise 1: The planner problem. For single agent problems, we can ask the Deep RL agents to solve the exact same problem we formulate in macroeconomic models. Thus, we can check how close does the Deep RL agent gets to the solution obtained by classical dynamic programming methods. If convergence is achieved, we confirm that the expectations implicit in the behavior of the Deep RL agent are rational. Also, we can scale the problem to many state variables and evaluate the potential of this learning methodology to deal with high dimensional dynamic control problems.
  - Exercise 2: Many households. In order to evaluate the ability of deep RL agents to learn in multi-agent environments, we consider the problem of many households who need to purchase

capital goods from the capital good firm. The supply side stays competitive, that is, the price paid by households correspond to the marginal cost evaluated at the aggregate demand level. In this formulation, the deep RL solution differs from the competitive market solution because agents do not assume that the price they pay is independent of the actions. In the tradition competitive equilibrium formulation, we impose such price-taking behavior by taking the price as given when we calculate the first order conditions. In the RL formulation, we do not use first order conditions. Instead, we formulate the economy as a markov game, in which the consequences of actions are well specified for any behavior, optimal or not. Within this formulation, we increase the number of households to evaluate whether a larger number of buyers increase competition on the demand side and thus increase quantity purchased. In the limit, the solution should approximate the competitive solution.

  - Exercise 3: Incomplete Information. In order to showcase the flexibility with which we can manipulate the information structure of the economies under study, we simulate an economy with many households in which half of the households observes the stock and idiosyncratic shock of all households, and the other half only observes his own state and aggregate statistics of other households state variables. With this exercise, we can calculate the increase in expected utility from having the extra information, and compare the investment behavior.

  - Exercise 4: AI agents in both supply and demand. Finally, I simulate an economy in which both sellers and buyers are Deep RL agents. This is challenging because we cannot make use of a "walrasian auctioneer" that take magnets first order conditions and determine prices by solving a system of equations. Also ,we can study the strategic behavior of agents by varying both the number og sellers and buyers.

- Results

1.2 Related Literature

- paragraph 1: Learning in Macroeconomics.

- paragraph 2: Reinforcement Learning in economics.

  - Calvano et al. [2020] study algorithmic collusion by evaluating the behavior of q-learning algorithms, the most classical RL algorithm, when they compete in a repeated static game consisting on a differentiated bertrand with constant marginal costs. They show that algorithms learn to collude, in the sense that they charge a markup that is above the static nash solution, even though it is not as high as the monopolistic solution. Their problem, though repeated, is static in the sense that there is no state variable nor inter-temporal choice.
  - Asker et al. [2021] study a similar problem as Calvano et al. [2020] but their focus is on how the specific RL algorithm affect the collusive behavior. They also use first generation reinforcement learning algorithms.

  - Graf et al. [2021] also study algorithmic collusion but with a deep reinforcement learning algorithm that can be used on environments with continuous action and observation space.
  -

- paragraph 3: Literaure on machine learning learning for high-dimensional problems.

2 A Primer on Reinforcement Learning

- Define Markov Decision Problems.
- Example 1: Monopolist problem.
- Algorithm: Q-learning
  - We start from simplest single agent algorithm. The problem is:

$$\max_{a \in A} \sum_{t=s}^{\infty} \gamma^{s-t} E_t[r_{t+s}]$$

  - Two value functions:
    1. state-value function: $V(s) = \max_{a \in A}\{E[r|s,a] + \gamma E[V(s')|s,a]\}$
    2. action-value function: $Q(s,a) = E(r|s,a) + \gamma E[\max_{a' \in A} Q(s',a')|s,a]$
  - Q-Learning:
    * Initialize $Q_0(s,a)$. Loop until converge:
      · agent chooses $a = \arg\max Q(s,a)$ and observes $\{r, s', s, a\}$.
      · Old Q: $Q(s,a)$.
      · Target Q: $r_t + \gamma \max_{a' \in A} Q(s',a')$.
      · New Q:

$$Q'(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a')]$$

- Both in theory and in practice, one of the most important ingredients for successful learning is exploration. Agents do not always act according to what they currently believe to be optimal but they sometimes choose actions at random in order to learn. How much exploration to perform is an open problem, called the exploration-exploitation dilemma.

- Proof of convergence and theory.

- Tsitsiklis [1994] perform an extensive convergence analysis of q-learning algorithms. His proof of convergence is based on contraction mapping theorem.
- Bertsekas and Yu [2012] introduce a mix of policy iteration and q-learning that improves efficiency even for approximate methods. It seems great check it.
- Ma et al. [2020] show formally that the recursive specification of the q values can be considered as a transformation of the bellman operator. The authors use this q-transform to solve problems with unbounded rewards.
- Ma and Stachurski [2021] use computational complexity theory and numerical experiments to conclude that the q-transform leads to gains in computational efficiency. NOTES: WHY?

2.1 Deep Reinforcement Learning

- Deep Q Network (DQN)
  - DQN consists in approximating the Q function with a neural net: $\hat{Q}(s,a) \equiv \mathcal{N}_\rho \sim Q(s,a)$.
  - The parameters ρ are estimated in order to minimize

$$\mathcal{L}(\rho) = \mathbb{E}\left[\left(\hat{Q}(s,a) - (r_t + \gamma \max_{a' \in A} \hat{Q}(s',a'))\right)^2\right]$$

    in observed transitions $\{r, s', s, a\}$.
  - The most widely used NN is a composite function made of nested linear functions:

$$\mathcal{N}_\rho(x) = \sigma_K(W_K ... \sigma_2(W_2 \sigma_1(W_1 x + b_1) + b_2) ... + b_K)$$

    where x is the input (in our case the transitions $\{r, s', s, a\}$), $W_i \in \mathbb{R}^{m_{i+1} \times m_i}$ are the weights, $b_i \in \mathbb{R}^{m_i+1}$ are the biases, and $\sigma()$ are activation functions.
- Theory: explain proofs by Maei et al. [2009].

2.2 State of the Art Algorithms

- Before, we gobtaingin a policy by choosing the action that maximizes a parametrized action-value functions. For policy gradient methods, we parametrized the policy directly:

$$\pi(a,s) = P(a|s,\theta)$$

- Three reason to consider Policy based methods:

  - There may situation in which storing the value function may be hard but the policy may be simple.
  - Another reson why it may be better to use policy gradient is that they have better convergence properties. Value function methods sometimes go wild.
  - So there are cases in which the policy may be more compact. Another big one is that these methods work better at continuous or large state spaces, since they circumvent the need to calculate the max over Q. That may be very intensive. This last one may be the bigger one.
  - We can estimate stochastic policies.

- Disadvantages:
  - BUT, they are less sample efficient. Maximizing the Q-value is very aggressive, you are trying to summarilze all your current info.
  - They typically converge to local rather than global maximum.

- Proximal Policy Optimization Schulman et al. [2017] is a policy gradient algorithms that has been used in many of the recent success cases (cite all). The first step is to comput an estimator of the policy gradient:

$$\hat{g} = \hat{\mathbb{E}}_t \left[\Delta_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t \right]$$

  where $\hat{\mathbb{E}}_t$ is the empirical average over a finite batch of transitions.

- We transform this estimate in a loss function so we can apply grandient ascent. The loss function is :

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \hat{A}_t \right]$$

2.3 Multi-Agent Reinforcement Learning

- Define Partially Observed Markov Games.
- Example 1: Algorithmic Collusion.
- Algorithm: Independent learning.
- Challenges of multi-agent.
- Soutions: QMix and other algorithms.
- Prominent success cases:

- Berner et al. [2019] achieve super-human performance at the game Dota 2. The game is challenging for multiple reasons. It is multi-agent, stochastic, it has a large action and observation space and it requires long term strategy. They authors used a single neural net to approximate both the value function and the policiy function. In particular, they used a single-layer 4096-unit LSTM.
- Baker et al. [2019] show that agents self-playing hide and seek in a complex environment exhibit emergent curricular learning, in which they progressively discover strategies and then device counter-strategies. They use PPO with GAE. The distributed computation framework is called rapid.
- Vinyals et al. [2019] achieve grand-master level at the game StarCraft 2. "Observations of player and opponent units are processed using a self-attention mechanism. To integrate spatial and non-spatial information, we introduce scatter connections. To deal with partial observability, the temporal sequence of observations is processed by a deep long short-term memory (LSTM) system. To manage the structured, combinatorial action space, the agent uses an auto-regressive policy and recurrent pointer network.". "For every training agent in the league, we run 16,000 concurrent StarCraft II matches and 16 actor tasks (each using a TPU v3 device with eight TPU cores23) to perform inference. The game instances progress asynchronously on preemptible CPUs (roughly equivalent to 150 processors with 28 physical cores each), but requests for agent steps are batched together dynamically to make efficient use of the TPU. Using TPUs for batched inference provides large efficiency gains over previous work14,29. Actors send sequences of observations, actions, and rewards over the network to a central 128-core TPU learner worker, which updates the parameters of the training agent. The received data are buffered in memory and replayed twice. The learner worker performs large-batch synchronous updates. Each TPU core processes a mini-batch of four sequences, for a total batch size of 512. The learner processes about 50,000 agent steps per second. The actors update their copy of the parameters from the learner every 10 s. We instantiate 12 separate copies of this actor–learner setup: one main agent, one main exploiter and two league exploiter agents for each StarCraft race. One central coordinator maintains an estimate of the payoff matrix, samples new matches on request, and resets main and league exploiters. Additional evaluator workers (running on the CPU) are used to supplement the payoff estimates. See Extended Data Fig. 6 for an overview of the training setup."

3 Baseline Framework

- N^h households indexed by i choose how much to consume and how much to invest in order to maximize $\sum_{t=0}^{\infty} \beta^t U(C_{i,t})$.

- The final good is produced using N^c capital goods indexed by j according to technology

$$Y_{i,t} = Z^h_{i,t}\left(\Pi^{N^c}_{j=1} K^{1/N^c}_{i,j,t}\right)^{\alpha} = Z^h_{i,t}\Pi^{N^c}_{j=1} K^{\alpha/N^c}_{i,j,t}$$

  where total factor productivity $Z^h_{i,t}$ follows the stochastic process $Z^h_{i,t+1} \sim \mathcal{P}(Z^h_{i,t})$. We assume that $\alpha < 1$ in order to avoid having aggregation, that is, given decreasing returns to scale the distribution of capital stocks matters.

- The agent allocates $Y_{i,t}$ between consumption and investment in new capital goods. The evolution of the stock of capital is

$$K_{i,j,t+1} = (1-\delta)K_{i,j,t} + I^h_{i,j,t}$$

  where $I^h_{i,j,t}$ represents investment in capital good j by household i in period t[1].

- The cost in term of the final good of producing $I^h_{j,t}$ new units of capital good j is $\frac{\phi}{2}\left(I^h_{j,t}\right)^2$.

- In order to frame the problem, we define the saving rate $s_{j,t}$ as the expenditure on new capital goods j in terms of the final good. Thus, consumption is

$$C_{i,t} = (1 - \sum_{j=1}^{N^c} s_{i,j,t})Y_{i,t}$$

- We will consider both a competitive market's formulation and a planner's formulation of the problem:

  - First, we assume that there is a market for investment goods and we introduce capital good firms that pay the adjustment cost and sell the investment good at price $p^k_{j,t}$. Thus, from the point of view of the household, investment is $I^h_{i,j,t} = s_{i,j,t}/p^k_{j,t}$.

  - Second, in order to formulate the planner's problem, we assume that all the resources that are committed by households for investment on capital good j are actually employed in production, and then total production of capital good j is distributed among housholds according to their contribution. In this case, total production is defined implicitly by:

$$\sum_{i=1}^{N^h} s_{i,j,t}Y_{i,t} = \frac{\phi}{2}I^2_{j,t} \quad \text{for } j \in [1, ..., N^h]$$

    Solving for $I_{j,t}$ we get:

[1]The superscript h is added because we will consider both planner and market solutions to the problem, and in the latter it is useful to keep track of the agent that is choosing the variable I.

$$I_{j,t} = \sqrt{\frac{2}{\phi}\sum_{i=1}^{N^h} s_{i,j,t} Y_{i,t}}$$

We can then distribute it among households according to

$$I_{i,j,t} = \frac{s_{i,j,t}Y_{i,t}}{\sum_{i=1}^{N^h} s_{i,j,t}Y_{i,t}} I_{j,t} = \frac{s_{i,j,t}Y_{i,t}}{\sqrt{\frac{\phi}{2}\sum_{i=1}^{N^h} s_{i,j,t}Y_{i,t}}}$$

3.1 Competitive Market's Formulation

- Now we have two types of agents. The household purchases capital goods and consume the final good. The capital good firm, which produces the capital good subject to a quadratic cost.

3.1.1 Household

- The recursive formulation of the problem for household i is:

$$\begin{aligned} V_i\left(\{K_{i,j,t}\}_{i,j}\right) = \max_{\{s_{i,j,t}, K_{i,j,t+1}\}_j} & U(C_{i,t}) + \beta \mathbb{E}_t V_i(\{K_{i,j,t+1}\}_{i,j}) \qquad \text{s.t.} \\ & C_{i,t} = (1 - \sum_{j=1}^{N^c} s_{i,j,t}) Y_{i,t} \\ & Y_{i,t} = Z_{i,t}^h \Pi_{j=1}^{N^c} K_{i,j,t}^{\alpha/N^c} \\ & K_{i,j,t+1} \leq (1-\delta) K_{i,j,t} + \frac{s_{i,j,t} Y_{i,t}}{p_{j,t}^k} \quad \text{for } j \in [1, ..., N^c] \end{aligned}$$

- The lagrangian of the problem is:

$$\begin{aligned} \mathcal{L}_t = \mathbb{E}_t \sum_{r=0}^{\infty} \beta^r \Bigg(& U\left((1 - \sum_{j=1}^{N^c} s_{i,j,t+r}) Z_{i,t+r}^h \Pi_{j=1}^{N^c} K_{i,j,t+r}^{\alpha/N^c} \right) + \\ & \sum_{j=1}^{N^c} Q_{i,j,t+r} \left[(1-\delta) K_{i,j,t+r} + \frac{s_{i,j,t+r} Z_{i,t+r}^h \Pi_{j=1}^{N^c} K_{i,j,t+r}^{\alpha/N^c}}{p_{j,t+r}^k} - K_{i,j,t+r+1} \right] \Bigg) \end{aligned}$$

- The first order conditions are:

$$
\begin{aligned}
[s_{i,j,t}] \qquad & Q_{i,j,t} = p^k_{j,t} U'(C_{i,t}) \\
[K_{i,j,t+1}] \qquad & Q_{i,j,t} = \beta \mathbb{E}_t \left(U'(C_{i,t+1})(1 - \sum_{j=1}^{N^c} s_{i,j,t+1}) \frac{\alpha}{N^c} \frac{Y_{i,t+1}}{K_{i,j,t+1}} \right. \\
& \qquad \left. + Q_{i,j,t+1} \left[(1-\delta) + \frac{s_{i,j,t+1}}{p^K_{j,t+1}} \frac{\alpha}{N^c} \frac{Y_{i,t+1}}{K_{i,j,t+1}} \right] \right)
\end{aligned}
$$

- Combining both F.O.C.s we ge the inverse demand:

$$
p^k_{j,t} = \mathbb{E}_t \left(\beta \frac{U'(C_{i,t+1})}{U'(C_{i,t})} \left[(1 - \sum_{j=1}^{N^c} s_{i,j,t+1} + s_{i,j,t+1}) \frac{\alpha}{N^c} \frac{Y_{i,t+1}}{K_{i,j,t+1}} + p^k_{j,t+1}(1-\delta) \right] \right) \tag{1}
$$

3.1.2 Capital good firms (superscript c)

- Capital good firms maximize the discounted sum of profits. Profits in each period are

$$
\pi^c_{j,t} = p^k_{j,t} I^c_{j,t} - \frac{\phi}{2} \left(I^c_{j,t} \right)^2
$$

- The lagrangian of the problem for firm j is:

$$
\mathcal{L}_{j,t} = \mathbb{E}_t \sum_{r=0}^{\infty} \beta^r_f \left(p^k_{j,t+r} I^c_{j,t+r} - \frac{\phi}{2} \left(I^c_{j,t+r} \right)^2 \right)
$$

- The first order conditions is:

$$
[I^c_{j,t}] \qquad\qquad p^k_{j,t} = \phi I^c_{j,t} \tag{2}
$$

3.1.3 Market clearing and equilibrium definition

- Given that $I^h_{i,j,s} = s_{i,j,t} Y_{i,t} / p^k_{j,t}$, we can write the market clearing condition as:

$$
\sum_{i=1}^{N^h} \frac{s_{i,j,t} Y_{i,t}}{p^k_{j,t}} = I^c_{j,t} \quad \text{for } j \in [1, ..., N^c]
$$

- **Equilibrium:** A competitive equilibrium is a sequence of prices $\{p^k_{j,t}\}_{j,t}$ and allocations $\{s_{i,j,t}, I^c_{j,t}\}_{i,j,t}$ such that:

  - Given prices, household maximize the intertemporal utility of consumption and capital good firms maximize intertemporal profits.

– Market for capital goods $j \in [1, ..., N^c]$ clear.

3.1.4 System of Equations

- The system of equations is:

$$p^k_{j,t} = \mathbb{E}_t \left(\beta \frac{U'(C_{i,t+1})}{U'(C_{i,t})} \left[(1 - \sum_{j=1}^{N^c} s_{i,j,t+1} + s_{i,j,t+1}) \frac{\alpha}{N^c} \frac{Y_{i,t+1}}{K_{i,j,t+1}} + p^k_{j,t+1}(1-\delta) \right] \right) \quad \text{for } i, j$$

$$p^k_{j,t} = \phi I^c_{j,t} \quad \text{for } j$$

$$I^c_{j,t} = \sum_{i=1}^{N^h} \frac{s_{i,j,t} Y_{i,t}}{p^k_{j,t}} \quad \text{for } j$$

$$K_{i,j,t+1} = (1-\delta) K_{i,j,t} + \frac{s_{i,j,t} Y_{i,t}}{p^k_{j,t}} \quad \text{for } i, j$$

- We want to reduce this system to $N^h * N^c$ equations that only depend on $\{s_{i,j,t}, s_{i,j,t+1}\}_{i,j}$. We start by replacing the market clearing condition in the supply curve:

$$p^k_{j,t} = \sqrt{\phi \sum_{i=1}^{N^h} s_{i,j,t} Y_{i,t}}$$

- We now replace this expression for $p^k_{j,t}$ in the transition law for capital $K_{i,j,t+1}$:

$$K_{i,j,t+1} = (1-\delta) K_{i,j,t} + \frac{s_{i,j,t} Y_{i,t}}{\sqrt{\phi \sum_{i=1}^{N^h} s_{i,j,t} Y_{i,t}}} \tag{3}$$

$$\tag{4}$$

- Finally, we can replace these expressions for $p^k_{j,t}$ and $K_{i,j,t+1}$ in the inverse demand function. Notice that $Y_{i,t+1}$ depends on the set of $\{K_{i,j,t+1}\}_j$ and $C_{i,t+1}$ is a known function of the set $\{s_{i,j,t+1}, K_{i,j,t+1}\}_j$, so we get $N^c * N^h$ differential equations whose only unknowns are $\{s_{i,j,t}, s_{i,j,t+1}\}_{i,j}$.

3.2 Deterministic Steady state

- In the deterministic steady state, we assume that $Z^h_i = 1, \forall i \in [1, ..., N^h]$. Then, all households have a symmetric problem and all capital goods are equivalent. Consequently, we can drop subscripts i and j on all variables. The steady state equations become:

$$
\begin{aligned}
Y &= K^{\alpha} \\
p^k &= \sqrt{\phi N^h s Y} \\
\delta K &= \frac{sY}{\sqrt{\phi N^h s Y}} \\
p^k &= \frac{\beta}{1-(1-\delta)\beta}\left[(1-(N^c-1)s)\frac{\alpha}{N^c}K^{\alpha-1}\right]
\end{aligned}
$$

- The solution is:

$$
K = \left[\phi\delta N^h N^c\left(\frac{1-\beta*(1-\delta)}{\alpha\beta}+\frac{\delta(N^c-1)}{N^c}\right)\right]^{\frac{1}{\alpha-2}}
$$

- For $N^f = 1$, $N^c = 1$, The steady state is:

$$
K = \left(\frac{\beta_f}{1-(1-\delta)\beta_f}\frac{\alpha^f}{\phi\delta}\right)^{\frac{1}{2-\alpha^f}}
$$

3.3 Planner's Formulation

- The recursive formulation of the problem is:

$$
\begin{aligned}
V\left(\{K_{i,j,t}\}_{i,j}\right) = \max_{\{s_{i,j,t},K_{i,j,t+1}\}_{i,j}} & \sum_{i=1}^{N^h} U(C_{i,t}) + \beta\mathbb{E}_t V(\{K_{i,j,t+1}\}_{i,j}) \qquad \text{s.t.} \\
C_{i,t} &= (1-\sum_{j=1}^{N^c} s_{i,j,t})Y_{i,t} \quad \text{for } i \in [1,...,N^h] \\
Y_{i,t} &= Z^h_{i,t}\Pi_{j=1}^{N^c} K_{i,j,t}^{\alpha/N^c} \quad \text{for } i \in [1,...,N^h] \\
K_{i,j,t+1} &\leq (1-\delta)K_{i,j,t} + \sqrt{\frac{2}{\phi}s_{i,j,t}Y_{i,t}} \quad \text{for } i \in [1,...,N^h] \text{ and } j \in [1,...,N^c]
\end{aligned}
$$

- The lagrangian of the problem is:

$$\mathcal{L}_t = \mathbb{E}_t \sum_{r=0}^{\infty} \beta^r \left(\sum_{i=1}^{N^h} U \left((1 - \sum_{j=1}^{N^c} s_{i,j,t+r}) Z^h_{i,t+r} \Pi_{j=1}^{N^c} K^{\alpha/N^c}_{i,j,t+r} \right) + \right.$$
$$\left. \sum_{i=1}^{N^h} \sum_{j=1}^{N^c} Q_{i,j,t+r} \left[(1-\delta) K_{i,j,t+r} + \sqrt{\frac{2}{\phi} s_{i,j,t+r} Z^h_{i,t+r} \Pi_{j=1}^{N^c} K^{\alpha/N^c}_{i,j,t+r}} - K_{i,j,t+r+1} \right] \right)$$

- The first order conditions are:

$$[s_{i,j,t}] \qquad Q_{i,j,t} = \phi \sqrt{\frac{2}{\phi} s_{i,j,t} Y_{i,t}} \frac{\partial U(C_{i,t})}{\partial C_{i,t}} = \phi I_t U'(C_{i,t})$$

$$[K_{i,j,t+1}] \qquad Q_{i,j,t} = \beta \mathbb{E}_t \left[\frac{\partial U(C_{i,t})}{\partial C_{i,t}} (1 - \sum_{j=1}^{N^c} s_{i,j,t+1}) \frac{\alpha}{N^c} \frac{Y_{i,t+1}}{K_{i,j,t+1}} \right.$$
$$\left. + Q_{i,j,t+1} \left((1-\delta) + \left(\frac{2}{\phi} s_{i,j,t+1} Y_{i,t+1} \right)^{-1/2} \frac{s_{i,j,t+1}}{\phi} \frac{\alpha}{N^c} \frac{Y_{i,t+1}}{K_{i,j,t+1}} \right) \right]$$

  To do:

  - Change proble to overal cost. b
  - Simplify FOC for K. you can use x/sqrt(x) o nthe right side and qrite in terms of investment.
  - Calculate steady state system of equations.
  - Order the rest of the doc.
  - Why do we need the planner for? Just as an analog for the program?

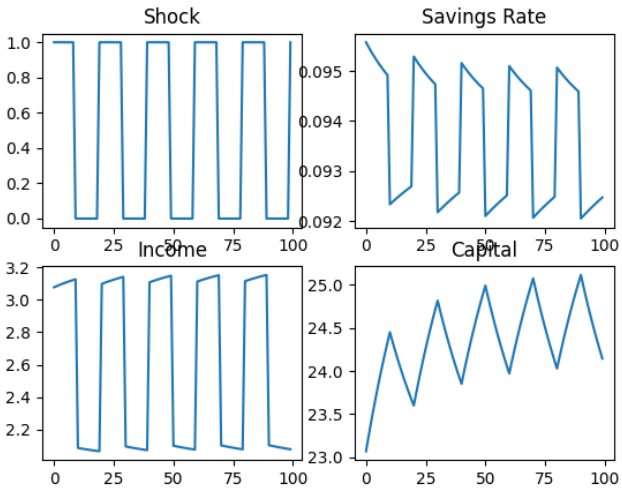
4 Applying Reinforcement Learning

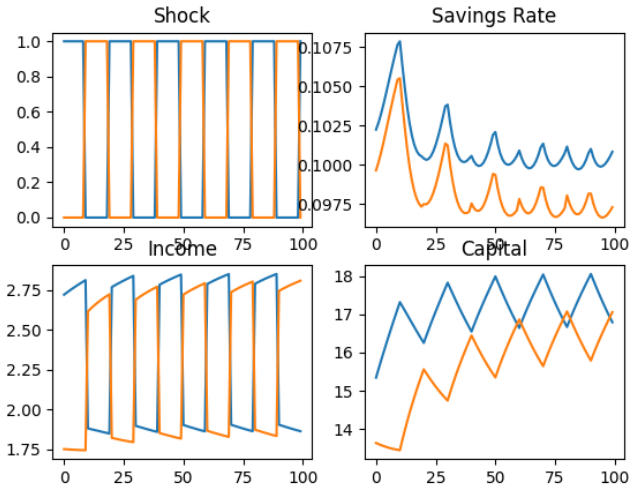
4.1 Learning by a Planner

- Exercise 1: 1 houehold, 1 capital (2 state variables)
  - 1 household
  - 2 houeholds, 1 capital (4 state variables)
- Exercise 2: many capital goods, (1v1, 2v2, 10v10)

4.2 Multi-agent learning

- Explain centralized critic algorithm.
- Explain parallelized computation framework.


Shock
Savings Rate
Income
Capital


Shock
Savings Rate
Income
Capital

- Exercise 1: 1 vs 5 vs 100 households.
- Exercise 2: Capital Good firms as AI agents. 1 vs 1, 2 vs 2, 10 vs 10. Does it converge to competitive equilibrium?
- Exercise 3: Half the households have full info and half the households only observe own and aggregate stock.

5 Design Lessons

- Exercise 1: Choosing spending vs choosing quantities. General normalization issues.
- Exercise 2: How to specify markets.
- Exercise 3: How to impose constrains.
- Exercise 3: Learning rates and algorithmic issues?

6 Conclusion

to be done

References

John Asker, Chaim Fershtman, and Ariel Pakes. Artificial intelligence and pricing: The impact of algorithm design. Technical report, National Bureau of Economic Research, 2021.

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Dimitri P Bertsekas and Huizhen Yu. Q-learning and enhanced policy iteration in discounted dynamic programming. *Mathematics of Operations Research*, 37(1):66–94, 2012.

Emilio Calvano, Giacomo Calzolari, Vincenzo Denicolo, and Sergio Pastorello. Artificial intelligence, algorithmic pricing, and collusion. *American Economic Review*, 110(10):3267–97, 2020.

Christoph Graf, Viktor Zobernig, Johannes Schmidt, and Claude Klöckl. Computational performance of deep reinforcement learning to find nash equilibria. *arXiv preprint arXiv:2104.12895*, 2021.

Qingyin Ma and John Stachurski. Dynamic programming deconstructed: Transformations of the bellman equation and computational efficiency. *Operations Research*, 2021.

Qingyin Ma, John Stachurski, and Alexis Akira Toda. Unbounded dynamic programming via the q-learning transform. *arXiv preprint arXiv:2012.00219*, 2020.

Hamid Reza Maei, Csaba Szepesvari, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *NIPS*, pages 1204–1212, 2009.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202, 1994.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.