Deep Reinforcement Learning in Macroeconomic Models

Matias Covarrubias

NYU

This paper

- Two questions:
  - 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?
- I apply Deep Reinforcement Learning (Deep RL) to macroeconomic problems.
- In the context of an investment under uncertainty model with heterogeneous 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.
  - equilibrium selection in models with multiple equilibrium.

,

An overview of Reinforcement Learning

- RL is a class of algorithms that adapt dynamic programming techniques to the problem of online learning, that is, to learn how to control a system by interacting and experimenting with it.
- 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 many transitions, if successful, they are able to estimate the consequences of their actions accurately and thus make optimal decisions.
- Deep RL adds neural nets as function approximators. It has shown state of the art performance in many tasks such as:
  - Games, including complex multi-agent games. (cite)
  - Robotics. (cite)
  - Autonomous driving. (cite)
  - Operations research. (cite)

Online Learning

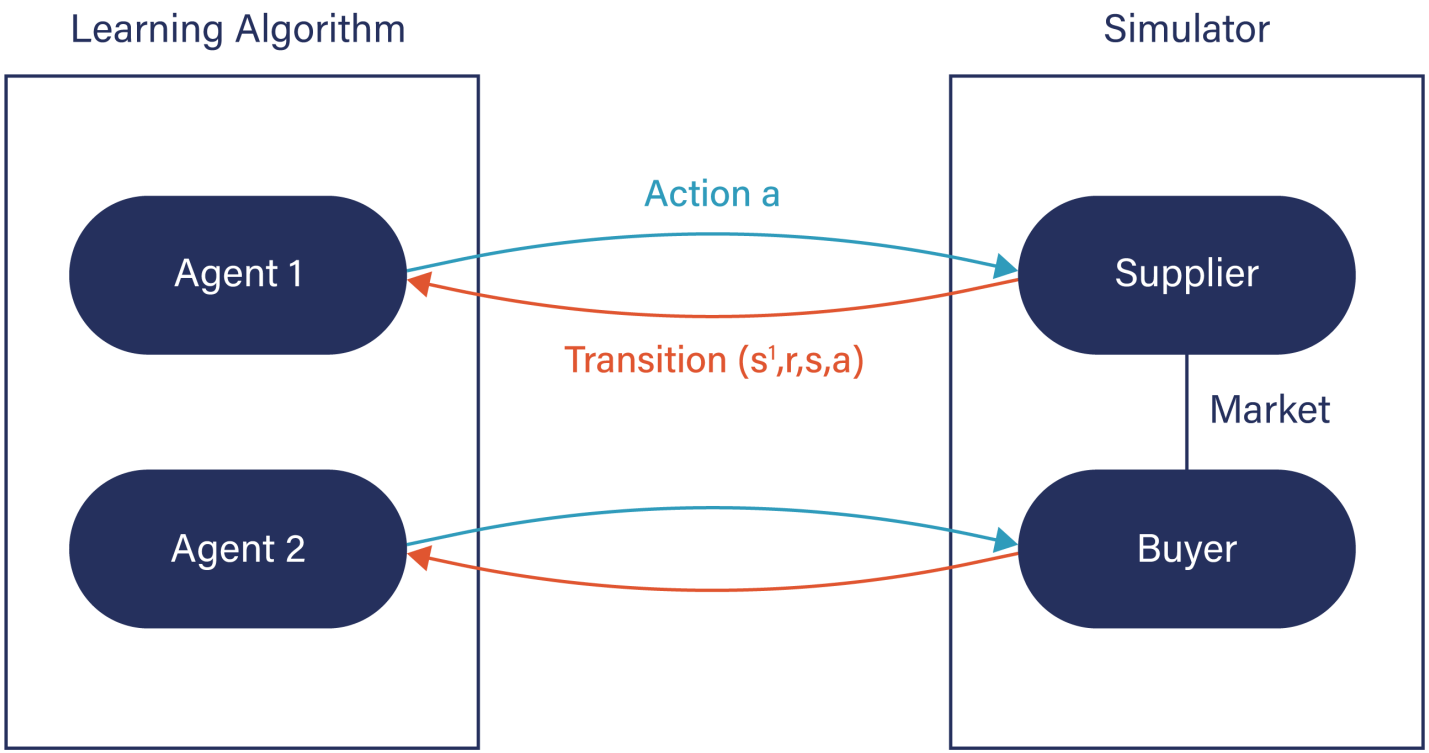


Figure: Online learning workflow

Deep RL in economics?

- Traditional challenges of Deep RL and why economics may be particularly well suited for this technology.
  - Rewards engineering is usually the main bottleneck. In economics, rewards are well specified in every period.
  - RL algorithms usually require millions of transitions in order to learn. In economics, sampling a transition is usually very fast.
  - Algorithms may get stuck in local optima. In many problems, we have nice regularity properties.
- But, economic problems present their own challenges.
  - Markets are a multi-agent problem. Challenge: agents learn and explore while other agents learn and explore $\rightarrow$ environment is not stationary.
  - In economics, rigorous insights often rely on exact solutions. That is, we are trying to solve very precise control problems.

An economy in an RL framework

We formalize an economy as a a Partially Observed Markov Game. In each period:

- n agents indexed by i choose an action $a_i \in \mathcal{A}_i$, after observing a state $s_i \in \mathcal{S}_i$.
- Let $\mathcal{S}$ be set of states, that is, the collection of all possible states S_i.
- The initial states are determined by a distribution $\rho : \mathcal{S} \mapsto [0, 1]$.
- To choose actions, each agent i uses a stochastic policy $\pi_{\theta_i} : \mathcal{S}_i \times \mathcal{A}_i \mapsto [0, 1]$.
- The combined actions produces the next state according to the state transition function $p : \mathcal{S} \times \mathcal{A}_i \times ... \times \mathcal{A}_N \mapsto \mathcal{S}$.
- Each agent i observes its reward as a function of the state and agent's action $r_i : \mathcal{S} \times \mathcal{A}_i \times ... \times \mathcal{A}_N \mapsto \mathcal{R}$, and receives a private observation $s_i' : \mathcal{S} \mapsto \mathcal{S}_i$.

Overview of baseline framework

- 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.
  - Dynamics at the forefront: the key trade-offs of the model should be intertemporal.
  - Meaningful scalability: the problem needs to have a clear path to scalability. It cannot have aggregation properties.
- I present a version of the stochastic growth model in which we focus on the purchase market for capital goods and introduce heterogenous agents.
- N^h households produce the final good and choose how much consume vs invest.
- The final good is produced using bundle of N^c capital goods.
- The cost of producing new capital goods is quadratic in **aggregate** investment.
- We will consider both a market's formulation and a planner's formulation of the problem.

4 experiments

1. Experiment 1: Planner formulation of stochastic growth model with many households and aggregate convex cost of adjustment.
   - Single-agent $\rightarrow$ control problem is exactly the same as the one we solve in economics.
   - We learn:
     1. Deep RL agent learn optimal solution consistent with rational expectations.
     2. It can quickly solve high dimensional problems.
2. Experiment 2: Market formulation of the problem with a competitive supply.
   - Multi-agent problem: can be formalized as Partially Observed Markov Games $\rightarrow$ few theoretical guarantees.
   - Challenge: non-stationarity. Agents learn while others are learning (and experimenting) as well.
   - We learn:
     1. Order arises from chaos and agents find a solution.
     2. Deep RL agents behave strategically and they consider their effect on prices.

4 experiments (continued)

1. Experiment 3: Incomplete information.
   - Half of the households have full information, half observes only own state plus aggregate statistics of other's state.
   - The point of this experiment is to illustrate how easy is to modify the informational structure of the economy.
   - We can calculate the gain in expected utility of the additional information.
2. Experiment 4: Deep RL agents in both supply and demand
   - Challenging because we cannot use walrasian auctioneers to guarantee market clearing.
   - I explore alternative formulations of the market game and check whether increasing the number of sellers and buyers get the solution closer to competitive solution.

To start: Planner formulation, 1 Household, 1 capital good

- A representative household chooses how much to consume and how much to invest in order to maximize $\sum_{t=0}^{\infty} \beta^t u(c_t)$.
- The final good is produced using capital K according to the production function $Y = ZK^{\alpha}$. The agent needs to decide how much of that product to consume and how much to use for production of new capital goods.
- **Adjustment costs:** In order to produce I_t new capital goods the representative agent needs to sacrifice $\frac{\phi}{2} I_t^2$ final goods.
- We express the problem in terms of choosing the saving rate s such that consumption is $C_t = (1 - s_t)Y_t$ and investment is defined by the equation $\frac{2}{\phi} I_t^2 = s_t Y_t$. Thus:

$$I_t = \sqrt{\frac{2}{\phi} s_t Y_t}$$

Planner, 1HH, 1 capital: recursive formulation

- The recursive formulation of the problem is:

$$V_t\left(K_t, Z_t\right) = \max_{s_t, K_{t+1}} U(C_t) + \beta \mathbb{E}_t V_{t+1}(K_{t+1}, Z_{t+1}) \qquad \text{s.t.}$$

$$C_t = Y_t(1 - s_t)$$

$$Y_t = Z_t K_t^{\alpha}$$

$$K_{t+1} \leq (1-\delta)K_t + \sqrt{\frac{2}{\phi} s_t Y_t}$$

where Z_t follows a markov chain with transition probability $\mathcal{P}$.

Planner, N^h HHs, 1 capital

- A key part of the problem with many households is that the covex adjustment cost are over the total amount of produced capital.
- For the planner formulation, we assume that all the resources that are committed by households are employed in production, and then total production is distributed among households according to their contribution. Total production is defined implicitly by:

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

where i indexes a household. Solving for I_t we get:

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

We can then distribute it among households according to

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

Planner, N^h HHs, 1 capital: recursive formulation

- The recursive formulation of the problem is:

$$V\left(\{K_{i,t}, Z^{id}_{i,t}\}_i, Z^{agg}_t\right) = \max_{\{s_{i,t}, K_{i,t+1}\}_i} \sum_{i=1}^{N^h} U(C_{i,t}) + \beta \mathbb{E}_t V((\{K_{i,t+1}, Z^{id}_{i,t+1}\}_i, Z^{agg}_{t+1}) \qquad \text{s.t.}$$

$$C_{i,t} = (1 - s_{i,t}) Y_{i,t} \quad \text{for } i \in [1, ..., N^h]$$

$$Y_{i,t} = Z^{agg}_t Z^{id}_{i,t} K^{\alpha}_{i,t} \quad \text{for } i \in [1, ..., N^h]$$

$$K_{i,t+1} \leq (1 - \delta) K_{i,t} + \frac{s_{i,t} Y_{i,t}}{\sqrt{\frac{\phi}{2} \sum_{i=1}^{N^h} s_{i,t} Y_{i,t}}} \quad \text{for } i \in [1, ..., N^h]$$

where Z^{agg}_t and Z^{id}_{i,t_i} follow markov chains with transition probability $\mathcal{P}^{agg}$ and $\mathcal{P}^{ind}$.

Market, N^h HHs, 1 capital: household 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_t^k. Thus, from the point of view of the household, investment is $I_{i,t}^h = s_{i,t}/p_t^k$.
- The recursive formulation of the problem for household i is:

$$V_i\left(\{K_{i,t}, Z_{i,t}^{id}\}_i, Z_t^{agg}\right) = \max_{\{s_{i,t}, K_{i,t+1}\}_j} U(C_{i,t}) + \beta \mathbb{E}_t V_i((\{K_{i,t+1}, Z_{i,t+1}^{id}\}_i, Z_{t+1}^{agg}) \qquad \text{s.t.}$$

$$C_{i,t} = (1 - s_{i,t}) Y_{i,t}$$

$$Y_{i,t} = Z_t^{agg} Z_{i,t}^{id} K_{i,t}^{\alpha}$$

$$K_{i,t+1} \leq (1 - \delta) K_{i,t} + \frac{s_{i,t} Y_{i,t}}{p_t^k}$$

Market, N^h HHs, 1 capital: capital good firm

- At first, we will assume that the capital producer is price taker:
  - It maximizes
  $$\pi_t^c = p_t^k I_t^c - \frac{\phi}{2} (I_t^c)^2$$
  .
  - Price is then determined by
  $$p_t^k = \phi I_t^c$$
- Then, in order to study strategic behavior in environments where both buyers and sellers are Deep RL agents, we will allow a deep RL agent to fix the mark-up over price.

Market, N^h HHs, N^c capital goods.

- Now we have N^c capital goods indexed by j.
- The recursive formulation of the problem for household i is:

$$V_i\left(\{K_{i,j,t}\}_{i,j}\right) = \max_{\{s_{i,j,t}, K_{i,j,t+1}\}_j} U(C_{i,t}) + \beta \mathbb{E}_t V_i(\{K_{i,j,t+1}\}_{i,j}) \qquad \text{s.t.}$$

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

$$Y_{i,t} = Z^h_{i,t} \Pi_{j=1}^{N^c} K^{\alpha/N^c}_{i,j,t}$$

$$K_{i,j,t+1} \leq (1-\delta) K_{i,j,t} + \frac{s_{i,j,t} Y_{i,t}}{p^k_{j,t}} \quad \text{for } j \in [1, ..., N^c]$$

- The supply is give by the equation $p^k_{j,t} = \phi I^c_{j,t}$ where

Reinforcement Learning Algorithm

- 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:

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

Getting deeper: Deep RL

- Deep Q-learning 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 $\{r, s', s, a\}$.
- A common example of a 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.

Convergence results in RL

- Tsitsiklis (1994) If state space and action space are discrete, Q-learning converges globally if:
  - All state-action pairs are visited infinitely often.
  - Conditions on learning rate α. The sum $\sum_{t=0}^{\infty} \alpha_t^2$ is finite and $\sum_{t=0}^{\infty} \alpha_t$ is infinite.
- Tsitsikli and Van Roy (1997) prove convergences in the case of linear function approximators and Maei et al (2009) prove convergence with smooth non-linear approximators (e.g., neural nets).
- For multi-agent, convergence is hard to prove. Muller et al (2019) show convergence in a large class of games to a solution concept they called α-rank.

Taxonomy of State of the Art Algorithms

There are three objects that the agents can learn:

1. q function $Q(s,a)$
   - **Q-based methods** use NNs to represent $Q(s,a)$.
   - off-policy method: the NN can learn from steps taken by any policy.
   - Model-free method.
2. policy function $\pi(s,a)$
   - **Policy-gradient methods** use NN to represent $\pi(s,a)$.
   - on-policy method. the NN only learns from steps taken from optimal policy.
   - Model-free. More stable than Q but less sample efficient.
3. Model $p(r,s'|s,a)$
   - **Model-based methods** uses different methods to represent $p(r,s'|s,a)$.
   - In recent stage of development compared to other methods.

In practice: How to create environments and agents

An environment consists of a **reset** and a **step** function.

An agent consist of a **choose_action** and a **learn** function. Loop:

- `initial_state=env.reset()`
  - The reset function gives you the initial state.
- `action = agent.choose_action(state)`
  - takes the state as input and choose the optimal action.
- `next_state, rewards = env.step(actions)`
  - Takes as input the actions of all the players and outputs the rewards for all the agents and the next state.
- `agent.learn(r,s',s,a)`
  - Takes as input the transition info $\{r, s', s, a\}$ and updates the internal estimates (e.g., policy functions).

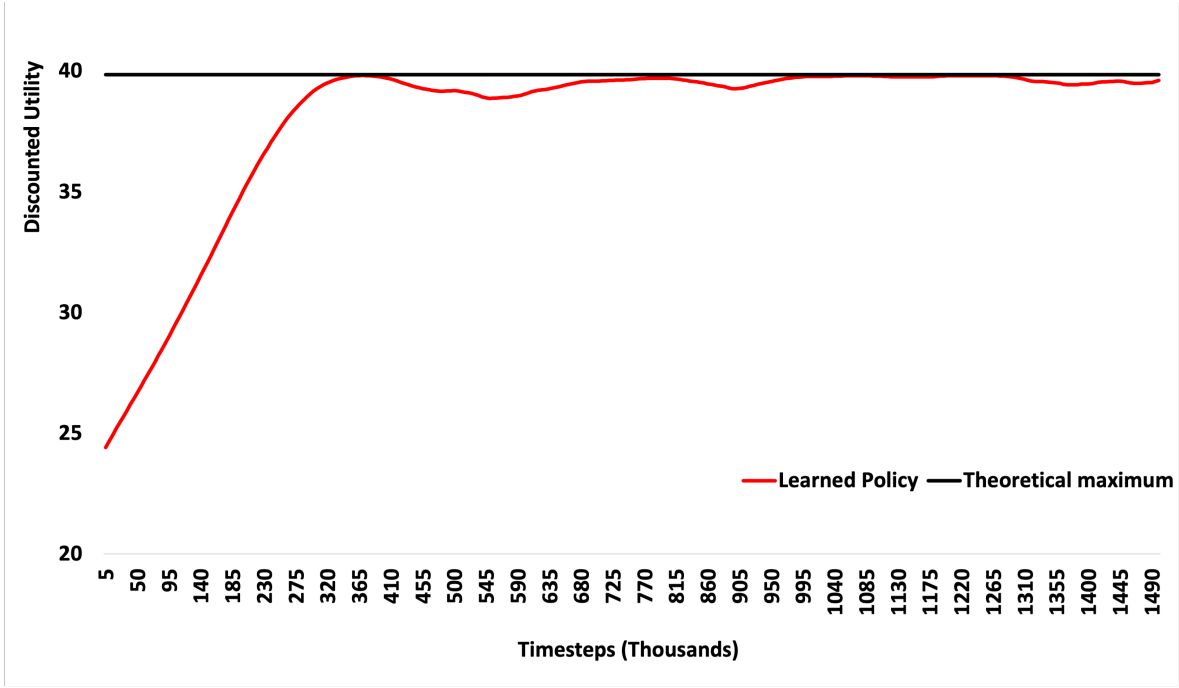
Experiment 1: Results

We start with the plannner's problem. Exercises:

- We vary the number of households ($N^h \in \{1, 2, 5, 10, 100\}$) to evaluate scalability.
- Two ways to formulate planner solutions
  1. Single-agent maps global state S to N^h actions.
     - Pro: Environment need only to expose the global state.
     - Con: Does not impose simmetry, so it needs to calculate probabilities of taking each of N^h actions $\rightarrow$ slower for normal sized states.
  2. Centralized learner learns policy that maps individual states S_i to one action. All households receive same aggregate reward.
     - Pro: imposes symmetry so it learns fast. Also, implementation is a couple of lines of code of difference with decentralized solution.
     - Con: The environment exposes a dictionary giving a reorder version of the global state for each houshold $\rightarrow$ slow transitions due to information overhead.

Planner, 1 household. Exact solution vs Learned policy

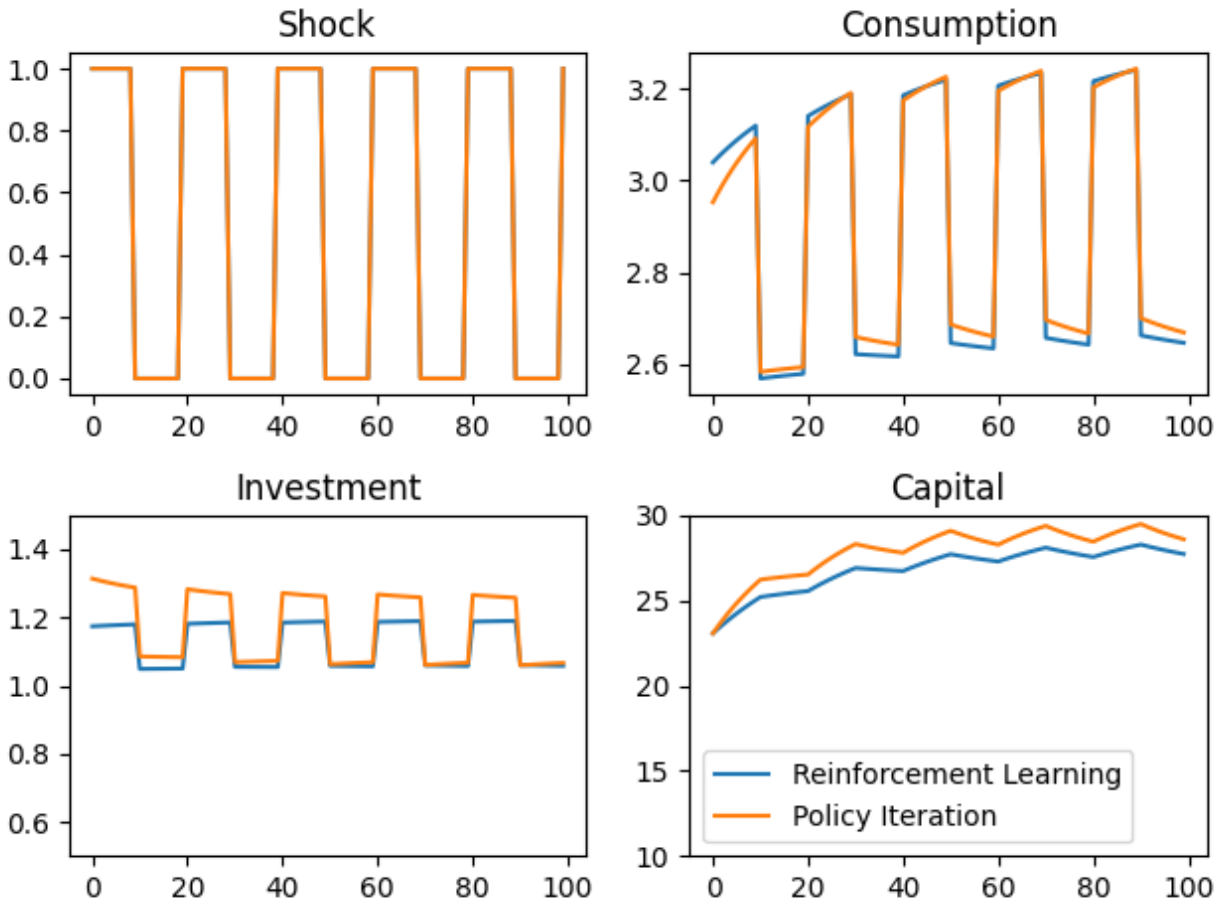
- We can compare the discounted utility on a simulated time window of a 1000 periods.



- The RL policy has a discounted utility that is 99.991% of the discounted utility under the optimal policy.

Planner, 1 household. Simulating the exact solution vs Learned policy

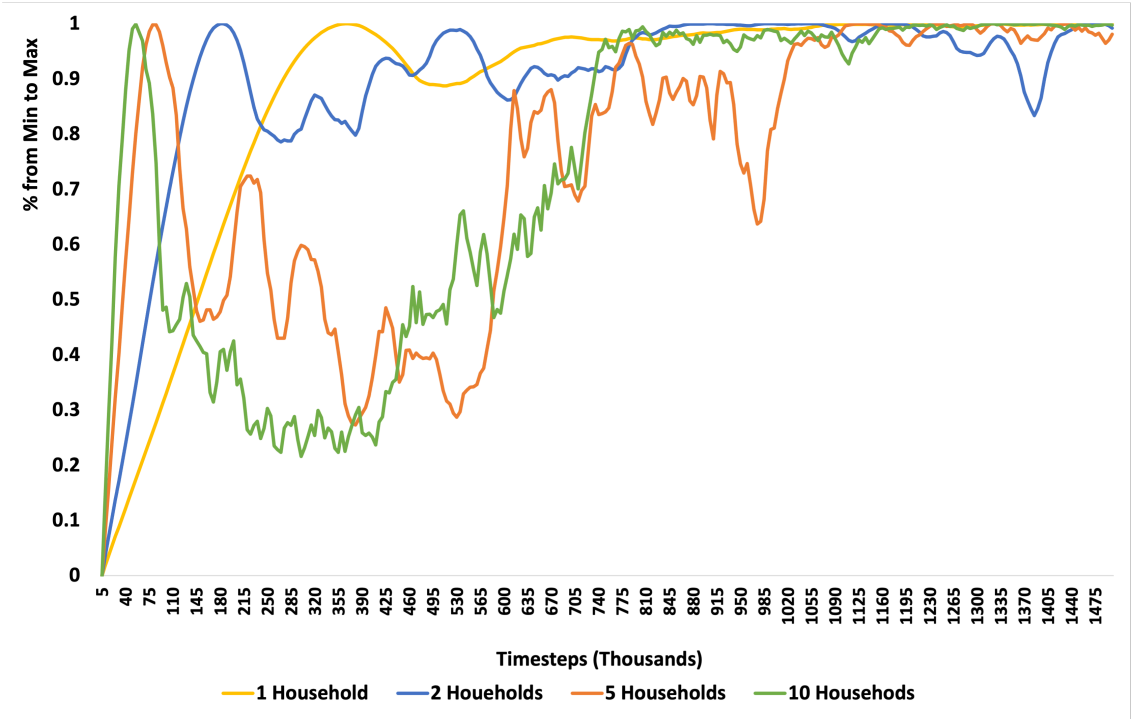
- We simulate a series of shocks and compare the optimal poly calculated through policy iteration vs the policy learned trough Deep RL.



- We still observe differences differences in time series.

Planner, Learning charts by number of households

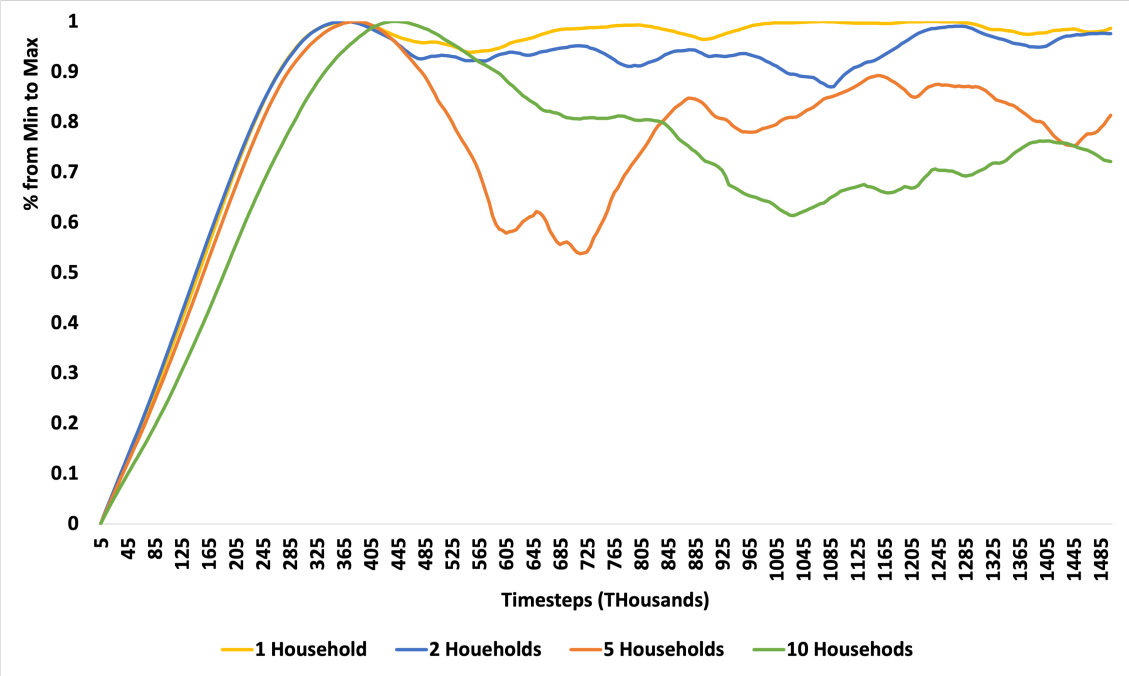
- We compare the number of transitions it takes to reach peak performance according to number of households.



- The number of transitions needed to reach the peak decreases with amount of households.
- Even though the problem is more complex, each transitions convey more information given that we are using a centralized learner with decentralized execution.

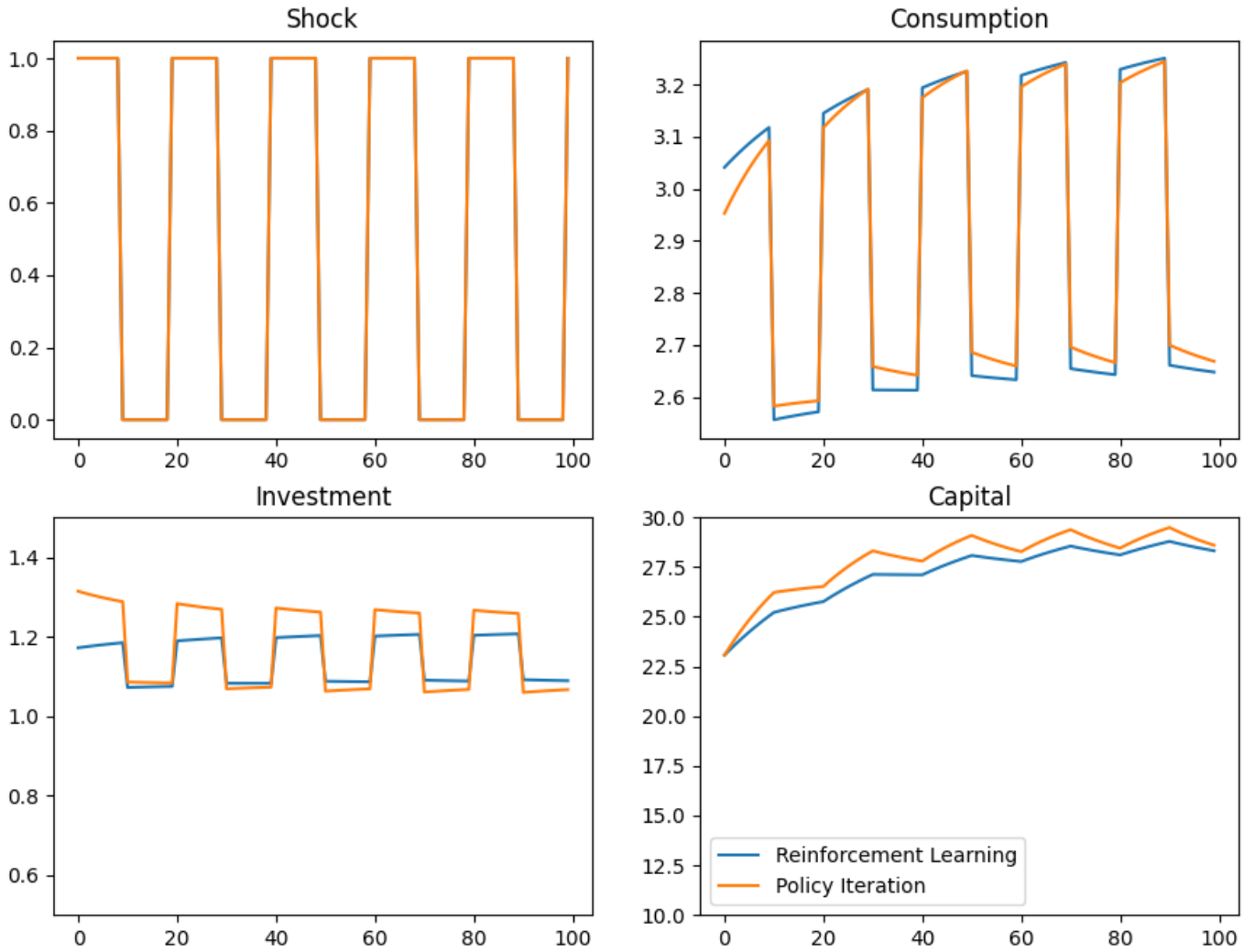
Planner, solving for N^h actions instead of only one map.

- We now compare scalability in the case when we solve map from the global state to N^h policies.



- We still observe scalability, but now the timesteps required to converge increase monotonically.

Planner, 1 household (3 state variables)



Comments on learning

- Learning is fast even in high dimensional problems. Time to reach the peak using only three physical cores:
  - 1 HH (3 state variables): 3 minutes 21 second.
  - 2 HHs (5 state variables): 1 minute 54 seconds.
  - 5 HHs (11 state variables): 1 min 10 seconds.
  - 10 HH (21 state variables): 1 min 13 seconds.
- For very large problems (e.g., 100 households), the centralized learning approach faces a bottleneck in that the environment needs to pass the whole state space to each household. That could be fixed.
- In those cases, modeling the problem as finding a mapping from the global state to many different policies (which is slower for lower dimensions), works, but it requires parallelizing over more cores as it takes hours instead of minutes.