# 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 introduce Deep Reinforcement Learning (Deep RL) as a "learning from experience" technology, in the spirit of least square learning (e.g., Macet and Sargent (1989)) but with less information about the economy.
- In a simple heterogeneous agents 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 potential applications:
  - high dimensional and incomplete information problems.
  - equilibrium selection in models with multiple equilibrium.
  - Strategic problems in dynamic settings.

,

# Deep 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.
- Deep RL has three main characteristics:
  1. **Model-free**: The algorithms do not use any mathematical knowledge of the system they are controlling, other than other than the allowed actions and states.
  2. **Simulation-Based**: They feed actions to the system and observe the reward they get that period and how the state evolves. With that information, they update their estimations of value function and policy function.
  3. **Deep**: they use Deep Neural Nets as function approximators for the value function and/or the policy function.
- It has shown state of the art performance in many tasks such as:
  - Games, including complex multi-agent games. (e.g., Go and Dota 2)
  - Robotics and autonomous driving. (e.g., Tesla autopilot and Covariant robotic arms).
  - Operations research and logistics. (e.g., Deepmind on Google Data Center Cooling).

# Related Literature

- Reinforcement Learning algorithms in economics:
  - Calvano et al (2020) and Asker, Fershtman and Pakes (2021) uses first generation RL algorithms to study algorithmic collusion in repeated Bertrand games.
  - Marimon, Sargent and McGrattan (1989) uses a "pre-RL" AI algorithm to study equilibrium selection in a Kiyotaki and Morre (2002) monetary model.
- Use of neural nets in computation macroeconomics
  - Maliar, Maliar and Winant (2020) solve Krusell and Smith (1998) without using bounded rationality.
  - Kahou, Fernandez-Villaverde, Perla and Sood (2021) exploits symmetry in policy functions suing neural nets.
  - Azinovic, Gaegauf and Scheidegger (2020) also uses a simulation based approach with neural net approximators to solve the equilibrium differential equations.
- This paper is the first to use Deep RL algorithms on dynamic stochastic problems.

# Online Learning

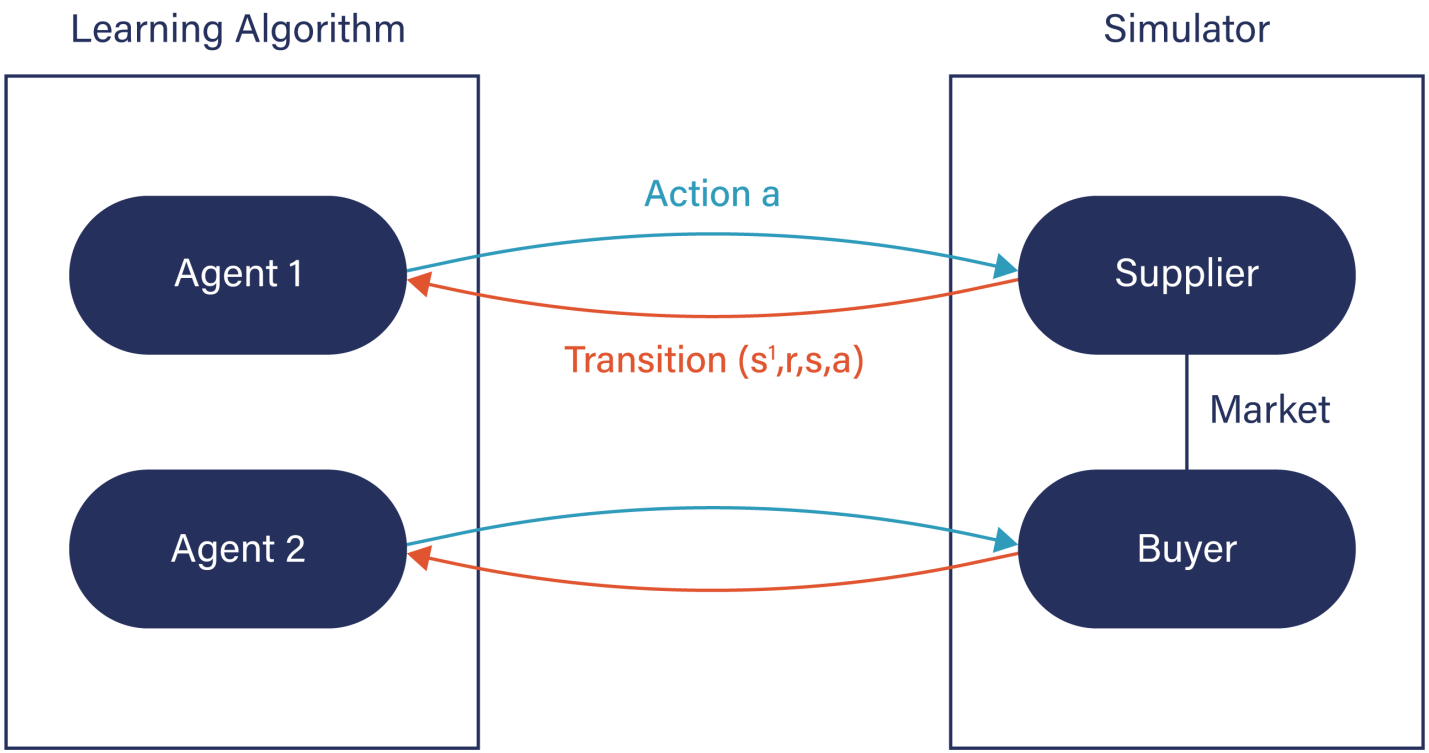



Figure: Online learning workflow

# 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 (e.g., in chess). 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

- I present a version of the stochastic growth model with heterogeneous agents in which we focus on the purchase market for capital goods.
- $N^h$ 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 total investment on that good.
- We will consider both a market's formulation and a planner's formulation of the problem.
- As in Krusell and Smith (1998) model, we have heterogenous agents that accumulate capital and face idiosyncratic risk and aggregate risk.
  - But, different source of idiosyncratic risk and many separate production functions.
  - Also, here we model a finite set of households, instead of a continuum.
  - Cao (2020) solves the Krusell and Smith (1998) framework with finite agents using the GDSGE toolbox (Cao, Luo, and Nie (2020)).

# 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.
     3. Can be implemented as a multi-agent cooperative game with "centralized learning, decentralized execution".
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.

# Market formulation, $N^h$ 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 $I_{i,t}^h = s_{i,t} Y_{i,t} / p_t^k$.

- There are $N^h$ households that operate a production fraction and decide how much capital goods to purchase. 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 pays a cuadratic adjustment cost $C(I_t) = \frac{\phi}{2} I_t^2$.
  - It maximizes profits
$$\pi_t = p_t^k I_t - \frac{\phi}{2} I_t^2$$
.
  - Price is then determined by
$$p_t^k = \phi I_t$$
- A key part of the problem with many households is that the covex adjustment cost are over the total amount of produced capital.
  - Then, given decreasing returns to scale, agents need information on the entire distribution of capitals stock to predict prices.

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

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

$$V_i\left(\{K_{i,j,t}\}_{i,j}\{Z^{ind}_{i,t}\}_i, Z^{agg}_t\right) = \max_{\{s_{i,j,t}, K_{i,j,t+1}\}_j} U(C_{i,t}) + \beta \mathbb{E}_t V_i(\{K_{i,j,t+1}\}_{i,j}\{Z^{ind}_{i,t+1}\}_i, Z^{agg}_{t+1})$$

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

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

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

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

# Planner, $N^h$ HHs, 1 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 $\mathcal{P}^{ind}$.

# 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 $\rho$ 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.

# Way Neural Nets?

- They are an universal approximator.
- It is easy to impose constraint by manipulating the input or output layer.
- They are efficient for high dimensional problems (Grohs, Jentzen and Salimova (2019)).
- Since they are not aligned to a basis, they satisfy a symmetry property called "isotropy".
- Neural networks are "easy" to compute. There's good software for them and are particularly suited for GPU-acceleration and other computational tools.
- There are proofs that in many scenarios for neural networks the local minima are the global minima (Watt, Lacotte and Pilanchi 2020)

# 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 $\alpha$. 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 $\alpha$-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).

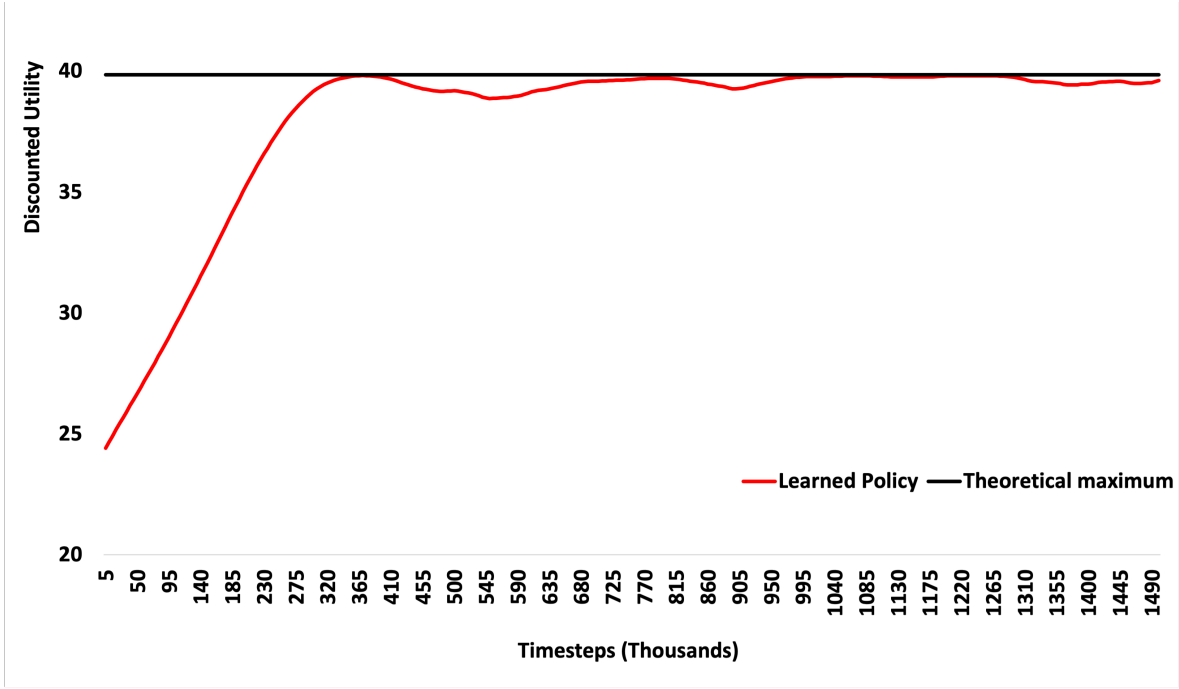
# Results: Experiment 1

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

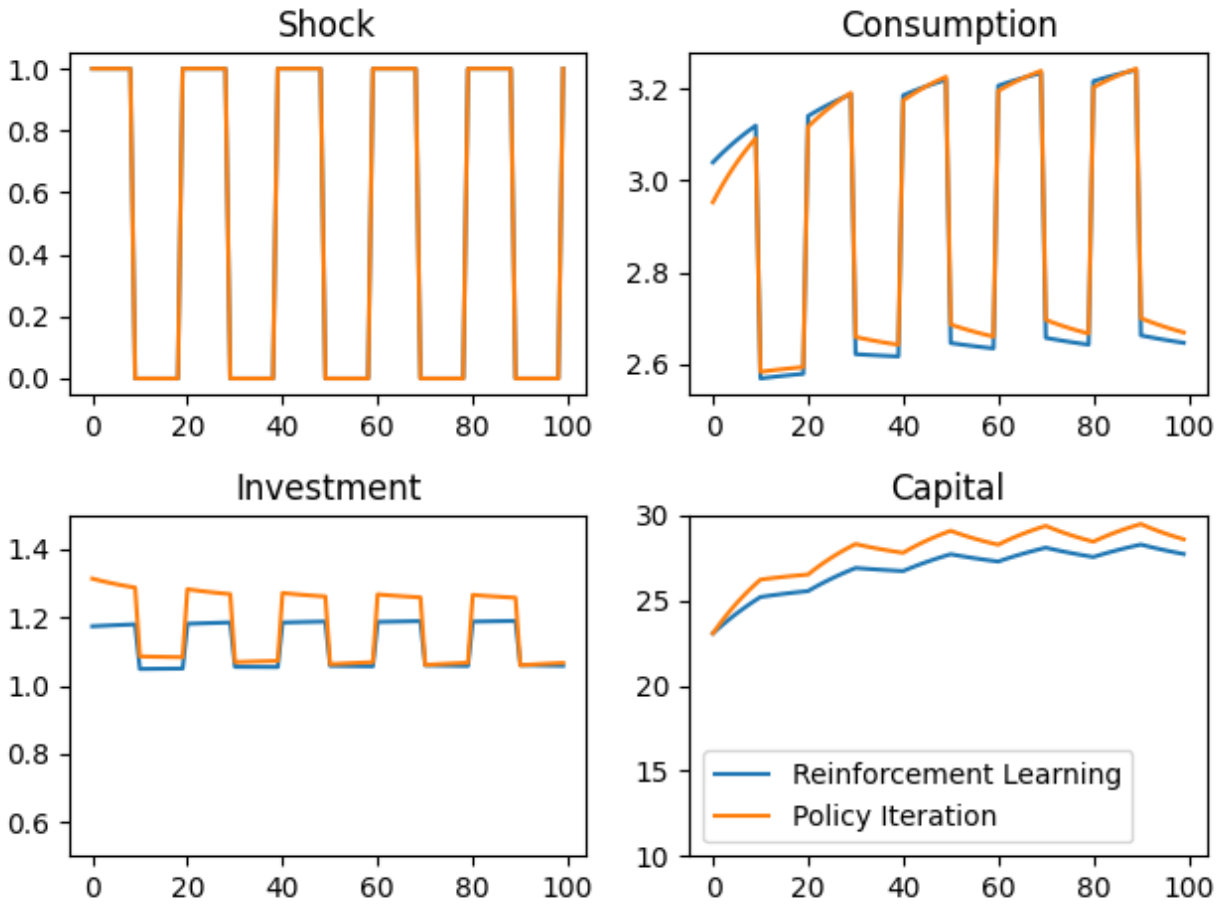
- We can compare the discounted utility on a simulated time window of a 1000 periods.
- W calculate the theoretical maximum by solving the model with global methods on a very fine grid using the GDSGE package.




- 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

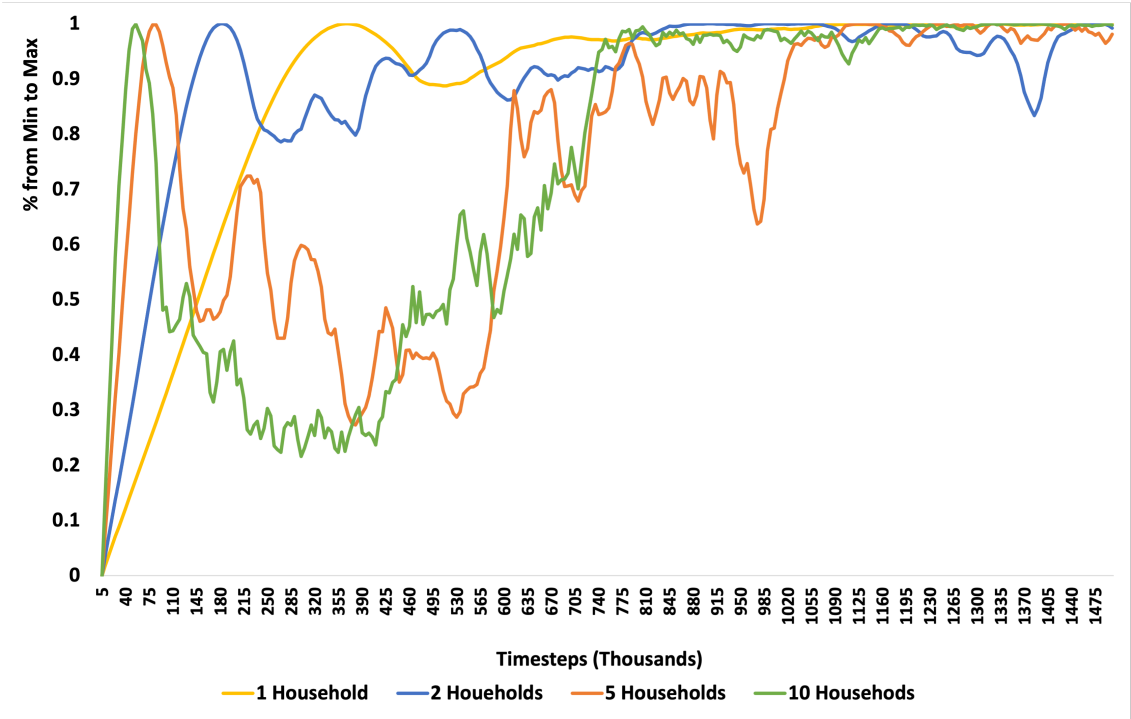
- We simulate a series of shocks for 100 periods and compare the optimal policy calculated through policy iteration vs the policy learned trough Deep RL.




- The consumption paths are very close, but we still observe differences in time series.

# Planner, Learning charts by number of households

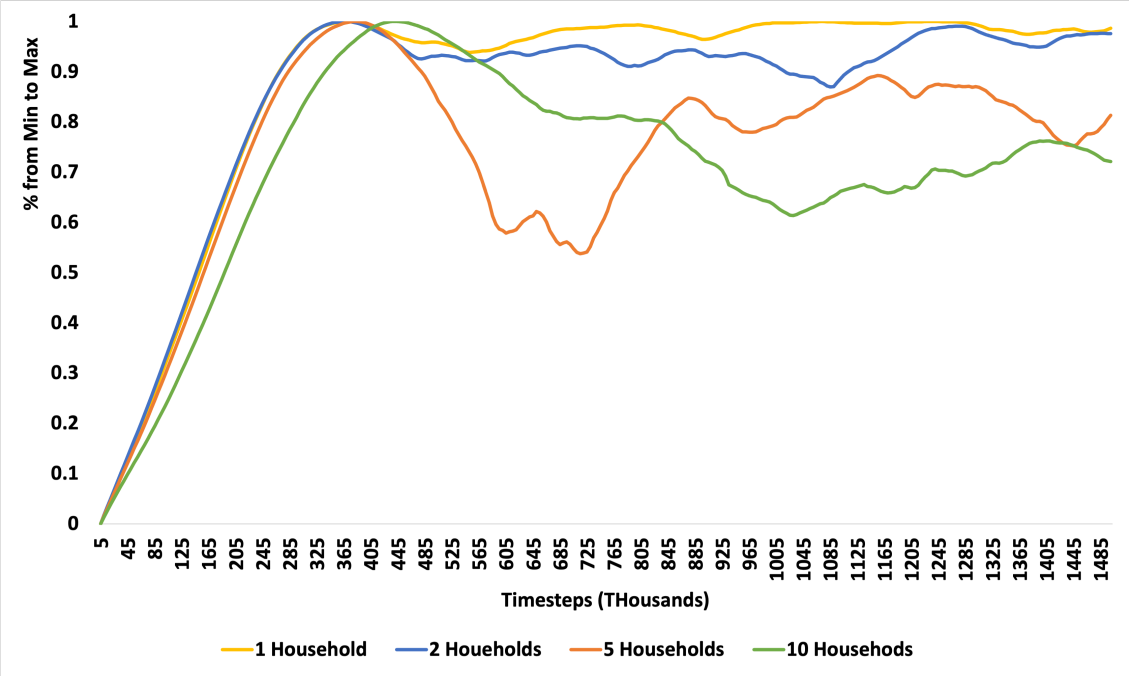
- We compare the number of transitions it takes to reach peak performance according to number of households.




- 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,no centralized learning.

- We now compare scalability in the case when we solve map from the global state to $N^h$ policies (no centralized learning).




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

# Comments on Experiment 1

- 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.