Deep Reinforcement Learning in Macroeconomic Models

Matias Covarrubias

July 14, 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, 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 the abstract in relation to Learning in Macroeconomics. Look at Learning in Macro book for queues. Motivate this particular learning technology.

- Paragraph 2: Explain model free RL. look at Calvano for queues.

  - RL is a class of algorithms that adapt dynamic programming techniques such as value function and policy iteration to the problem of online learning, that is, to learn how to control a system by interacting 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. 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.
  - 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.

- Paragraph 3: 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 a learning environment for the Deep RL agents.
  - 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.

- Paragraph 4: 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 is 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.

1.1 Preview of leading framework and exercises

- Paragraph 6: Explain the leading framework.
- Paragraph 7: Explain learning results.
- Paragraph 8: Explain high dimensional, incomplete information, and multiple equilibrium exercises.
- Paragraph 9: Explain design lessons.

1.2 Related Literature

- paragraph 1: Learning in Macroeconomics.
- paragraph 2: Reinforcement Learning in economics.

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:

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

- 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 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.
- Explain SAC or PPO.

2.2 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

3.1 Planing Formulation

3.2 Market Formulation

3.3 Steady State

3.4 Global Solution

4 Applying Reinforcement Learning

4.1 Learning by a Planner

- Exercise 1: 1 capital good, many households (1 vs 4 vs 100)
- Exercise 2: many capital goods, (1v1, 2v2, 10v10)

4.2 Multi-agent learning

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

bla bla

References

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.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

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.

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.