Imitation Learning in Macroeconomics: To Save or not to Save?

Sikata Sengupta (sikata) **, Rafael Esteves** (resteves)
CS 229 Final Project, Stanford University

1 Introduction

Within the field of Economics, Macroeconomists have long been interested in studying dynamic consumption savings problems [1]. In these models, agents optimize their discounted utility over some time horizon by choosing how much to consume or save at each period of time given some income streams and some known interest rates. This serves as the basis for whole classes of interesting real-world problems related to things like stimulus policy, income inequality policy, loaning markets etc. Moreover, many stochastic, continuous, and heterogeneous macroeconomic models are created as variants from this more baseline problem. However, in traditional Macroeconomic settings, when it is not possible to get an analytic solution, these types of problems are solved as a Markov Decision Problem (MDP) with dynamic programming. Below (in Subsection 2), we define one such Dynamic Consumption Savings Problem that Macroeconomics might traditionally solve through Value Function Iteration [6]. Such an agent not only knows his or her utility function, but also know the whole probabilistic model of the world. Because of the foresight the agent has over his or her entire lifetime, he or she is able to calculate future optimal consumption levels in advance. Such wisdom on behalf of the agent may at times seem unrealistic. It would seem more realistic to study the case where an agent has to learn more about the world as time goes on, and might not know his or her utility function and future horizon. It would be interesting to study how the agent's learning of his or her optimal policy is influenced by limited access to a few 'expert individuals' choices in certain states of the world. While Reinforcement Learning and Imitation Learning are certainly being employed within Economics, a lot is left to be done! We propose studying the following: Within the traditional deterministic consumption savings problem framework, could an agent learn his or her optimal policy function over time by combining his or her experience and imitation learning, with varying amounts of information provided by the expert? Aside from the dominant Dynamic Programming approach [5], the main use of modern machine learning we could find in this context was in using Gaussian Processes for interpolation along with Dynamic Programming [4]. Imitation Learning was discussed in the past as a model for agent behavior [2].

2 Environment

We will now introduce the toy Markov Decision Process (MDP) that we study for the remainder of the paper. In this problem, the state variable is given by the wealth value w_t, which we will subsequently also refer to as s_t and the action at any given state is given by the level of consumption c_t, which we will subsequently also refer to as a_t chosen at every period of time. Since we choose a deterministic problem, we do specify transition probabilities, but provide the transition rule below. We specify a discount factor β, finite horizon time period T, constant interest rate R, and constant income stream y. The felicity function is given by: $u(c_t) = \frac{(c_t/scale)^{1-1/\sigma}}{1-1/\sigma}$. Note that utility in this context refers to the discounted total sum of the felicity function. In our code we choose a scale of 100 based upon our choice of discretization of the state space ranging up until 2000 (so that we scaled between 0 and 20 in steps of $.01$). In this discrete version, our max wealth w_{max}, can hypothetically be thought of as laying the constraint that any money above that amount is taxed/given to the government. Our agent wants to solve the following optimization problem [7] to determine the optimal amount of

Preprint. Under review.

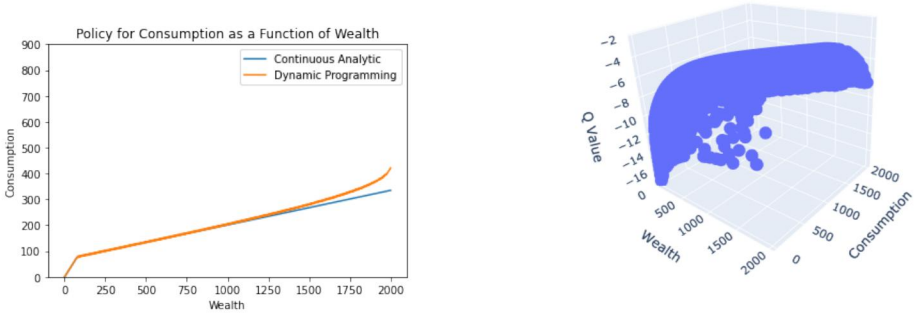

Figure 1: Here we see the policy function solved for through the traditional dynamic programming approach, matches up quite nicely with the analytic solution. We also see the curvature/shape of the Q function that the Q Value Iteration produces. The slight upturn in the the numerical value at the high wealth values is because saving there is often lost because of the w_{max} discretization.

consumption at each period of time:

$$\begin{aligned} \max_{c} \sum_{t=0}^{T} \beta^t u(c_t) \quad & \text{s.t.} \\ w_{t+1} = R(w_t - c_t) + y \quad & \forall t : t < T \\ w_t \geq y \quad & \forall t \\ w_t \geq c_t \geq 0 \quad & \forall t \end{aligned}$$

where w_t is the wealth variable corresponding to the agent's wealth at time period t and c_t is the amount of consumption that the agent chooses at time t. Note here that our state transitions are deterministic, and hence we use this MDP as a toy problem, which can serve the basis for future exploration on more complicated MDPs. Though we do not derive this solution here for the sake of space, an analytic benchmark solution for the policy function $c^*(w)$ that we can use at higher wealth levels (especially to compare slopes of policies) is given by:

$$c^*(w) = \min \left\{ w, (1 - \beta^\sigma R^{\sigma-1}) \left(w + \frac{y}{R-1} \right) \right\}.$$

3 Model and Discussion

We will go back and forth between discussing the results of various modeling experiments we try and discussing how they performed. We will write out the loss functions we used for our different experiments, where applicable, but for this exploration, our main method for evaluating the success of our model will be through comparing the resulting policy function that we can solve for with an analytic solution and the solution that would traditionally be solved for in a Macroeconomics context through Dynamic Programming, or essentially what we will call Q Value Iteration [9]. See Figure 1. Note, that in many of these methods we will first solve for a Q function, though ultimately we will use that Q function to solve for the resulting policy function.

3.1 Q Temporal-Difference (Q-TD) Learning

We implement a tabular 1 step Temporal Difference [8] learning model with the agent being ϵ-greedy for estimating the Q-Value. This is a useful point of comparison for our more advanced models because ultimately it represents for us the case where the agent receives no information from the expert, but still samples enough of the state-action space to potentially learn something about the Q function. The update rule for Q-TD that we use with epsilon greediness while following the trajectory given by the MDP is:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\big(r_t + \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big)$$

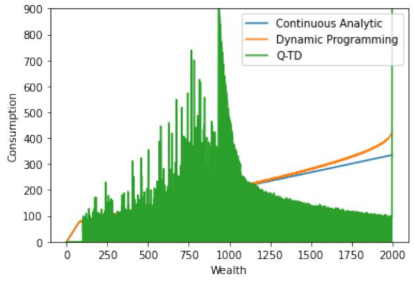

Figure 2: Q-TD Resuling Policy Solution is Compared with Analytic and Macro Dynamic Programming Solution (through Q Value Iteration)

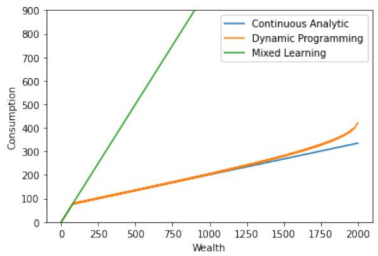

Figure 3: Mixed Imitation-Q Learning Solution

Where $r_t = r(a_t)$ is the reward obtained from following action a_t at state s_t and s_{t+1} is the state that results from following action a_t at state s_t. The loss we use here is a lowerbound on the ℓ_∞ norm loss because we simply compute $|Q_{old}(s_t, a_t) - Q_{new}(s_t, a_t)|$. See Figure 2. Though we can see the agent learns something through the Q-TD procedure, it is clear that the agent is not successful in learning the ultimate policy function. The agent in the tabular method does not know that it has to learn a smooth function. We try to address this by using a neural network in the following section.

3.2 Mixed Neural Network Model

In this model, we combine concepts from both imitation learning [3] and Q-Learning. As opposed to our tabular approach before, our Q function (with inputs s_t, a_t) is now modeled by a neural network with 1 layer with 100 hidden units each and with ReLU activation functions. The loss is a combination of a term corresponding to the agent's experience as it follows a certain MDP trajectory along with a expert supervised loss. More specifically we compute the experience loss for $interval$ iterations which we follow by computing the expert loss for $gamma$ iterations after which we optimize. Both of these losses are described below when given a state and action pair s_t and a_t:

$$l_{experience}(Q_{pred}) = ||R_t + \beta \max_{a_{t+1}} Q_{pred}(s_{t+1}, a_{t+1}) - Q_{pred}(s_t, a_t)||_2^2$$

$$l_{expert}(Q_{pred}) = \max_a (Q_{pred}(s_t, a)) + (a - a_E)^2 - Q_{pred}(s_t, a_E)$$

where a_E stands for the action of the expert. Note that $l_{expert}(Q_{pred})$ is always positive by nature of how the maximization is computed. Now, we define total loss to be:

$$l_{total}(Q_{pred}) = l_{experience}(Q_{pred}) + \lambda l_{expert}(Q_{pred}).$$

The quantities $gamma/interval$ and λ decide the importance of the expert. In Figure 3, we can see that the solution learned by the RL agent through this mixed procedure was quite poor. This motivated us to investigate what would happen in the purely supervised setting, in order to determine whether our issue was in the ability of the neueral network to approximate the function or was in the procedures.

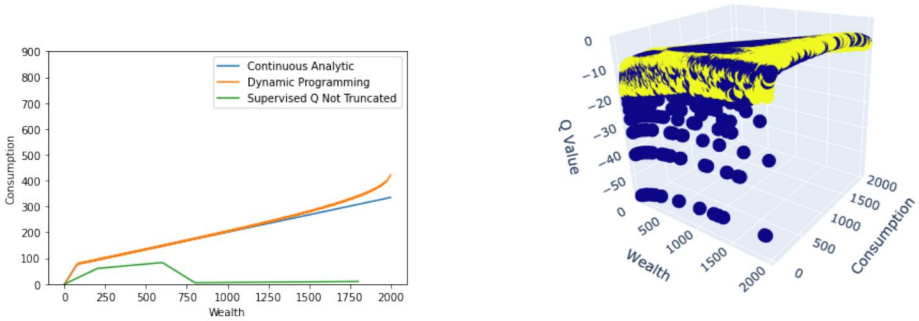

Figure 4: These figures show that the numerical Q value and the full state, action space supervised Q value do not line up near very low consumption values, unlike for high consumption values. The yellow dots are the supervised solution and the blue dots are the numerical solution. We can also see here that the resulting policy solution is not similar to the true solution.

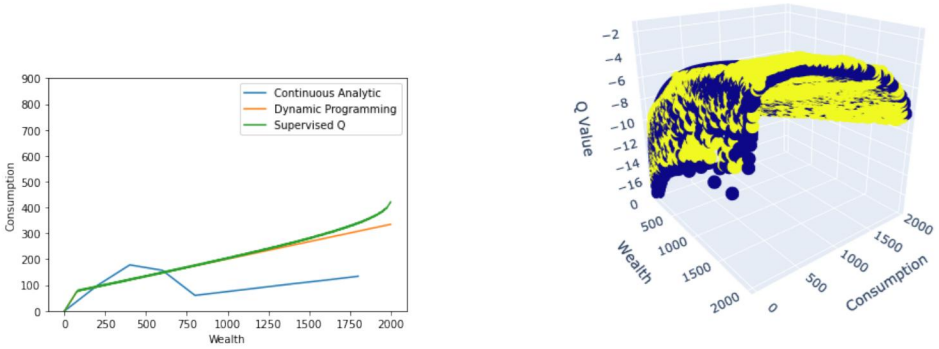

Figure 5: These figures show that the numerical Q value and the truncated state space supervised Q value are nearly identical. The yellow dots are the supervised solution and the blue dots are the numerical solution. We can see here that even though the Q functions learned are quite similar, the resulting policy solutions are not.

3.3 Exploration of Issue through Supervised Problem

Initially, we tried studying the supervised case on the entire state space of wealth, as will be discussed first.

3.3.1 Full State Action Space

Figure 4 showed us that at extremely low consumption values, due to the sharp drop off in the Q value function (because of the concavity of the utility function we define), even with full expert information, it is difficult to learn both the optimal Q function and therefore the optimal policy function. This motivated us to try studying the supervised problem on a truncated state space which did not focus on low consumption values.

3.3.2 Truncated State Action Space

Even in this circumstance, see Figure 5, what we saw is that even when the agent is able to successfully learn the Q value function, it does not do great in solving for the resulting policy function (which makes sense given that if there are tiny variations in the Q values that are not visible on this scale, the argmax will return the incorrect optimal action). Thus, since even in the best case scenario for the supervised procedure, the agent would still ultimately face great difficulty in learning the optimal policy function, we decided that it was better to try to directly learn the policy function itself, rather than going through the Q value.
From this, we also realized that our choice of MSE loss was not a good metric for capturing our

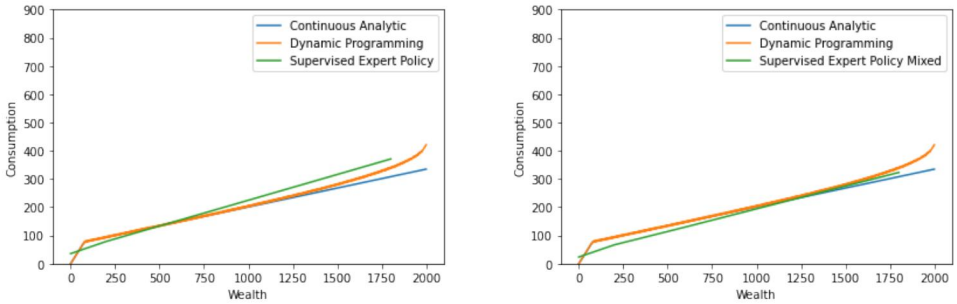

Figure 6: Here we see the policy function solved for when directly trying to learn the policy function with the left side corresponding to the one in which the agent is purely given information from the expert and the right one corresponding to the case where the agent has more individual experiences than expert datapoints, but weighs the loss function of the expert difference much higher.

broader target of ultimately producing good policies. Specifically because a policy is derived by finding the maximum possible value, what really matters is the ordering of the top few values. Our model was performing well on the Q values which were not particularly large (magnitude-wise), and had a bigger error on those Q values relevant to the policy. We considered altering our loss term in order to reflect this importance, but we decided against it. Since this supervised case investigation was to consider the 'best-case scenario' of our mixed imitation Q learning setting, we felt as if making large changes would be a fundamental reframing of the problem. Because even in the best case scenario for the supervised procedure, the agent would still ultimately face great difficulty in learning the optimal policy function, we decided that it was better to try to directly learn the policy function itself, rather than going through the Q value.

3.4 Expert Supervised Neural Network–Explicitly Learning the Policy Function

Here, in Figure 6, we see that the agent is able to learn the optimal policy function quite well when studying the supervised setting for directly learning the policy function. We implement a procedure where for $interval$ number of iterations, the agent follows the MDP based upon the policy it is learning, and for $gamma$ iterations, it follows the optimal policy the expert suggests. The figure we provide on the right side, shows that if we provide 300 experience data points to 200 expert data points, with a loss function that weighs the MSE action difference of the expert datapoints 100 times more than those of experience, the agent can still learn a decent policy function, though of course it is worse than the fully supervised case. This suggests that we ultimately have to conduct a diluted supervised method to make the agent learn. Of course, to optimally study this procedure in the future, we should pursue alternative policy gradient methods like REINFORCE [8] and see if we could combine them with some form of imitation learning, like behavioral cloning.

4 Conclusion

In this deterministic dynamic consumption savings problem, we found that both traditional Q Learning and hybrid Q-Imitation Learning methods perform quite poorly in terms of learning the optimal policy function, despite their potential to do well in learning the optimal Q function for high values of consumption. Ultimately, though, we found that when our agent tries to learn the optimal policy directly, with enough input and weight on the expert, it is able to perform quite well. This suggests the need for studying other policy gradient methods to explore this toy problem in an RL context. Clearly, how an agent can operate in a world without full knowledge of a probabilistic model remains to be understood. The same issues plague more realistic macroeconomic models.

References

[1] Fabio C Bagliano and Giuseppe Bertola. *Models for dynamic macroeconomics*. Oxford University Press on Demand, 2004.

[2] Erdem Başçı. "Learning by imitation". In: *Journal of Economic Dynamics and Control* 23.9-10 (1999), pp. 1569–1585.

[3] Takayuki Osa et al. "An algorithmic perspective on imitation learning". In: *arXiv preprint arXiv:1811.06711* (2018).

[4] Philipp Renner and Simon Scheidegger. "Machine learning for dynamic incentive problems". In: *Available at SSRN 3282487* (2018).

[5] John Rust. "Has dynamic programming improved decision making?" In: *Annual Review of Economics* 11 (2019), pp. 833–858.

[6] John Rust. "Numerical dynamic programming in economics". In: *Handbook of computational economics* 1 (1996), pp. 619–729.

[7] Nancy L Stokey, RE Lucas, and E Prescott. "Recursive methods in dynamic economics". In: *Cambridge, MA: Harvard University* (1989).

[8] Richard S Sutton and Andrew G Barto. "Reinforcement learning: an introduction MIT Press". In: *Cambridge, MA* 22447 (1998).

[9] Jordi Torres. *Value Iteration for Q-function*. 2020. URL: `https://towardsdatascience.com/value-iteration-for-q-function-ac9e508d85bd` (visited on 06/15/2020).