Solving a McCall Search Model via Reinforcement Learning

Trevor Gallen
Econ 64200

Fall 2022

Deliverables

- You should have a word/LaTeXdocument that has three sections:
  1. Discusses the model and answers the questions I pose throughout.
  2. Contains the tables and figures you will produce.
  3. Contains a discussion of your programming choices if you had to make any.
- You should have a Matlab file or set of files (zipped) that contain **all** your programs and raw data. There should be a file called "Main.M" that produces everything I need in one click.

1 Model

In this homework, we're going to try to solve a simple McCall search model with Reinforcement Learning.

Each period, agents wage up with a wage draw $w \sim \log \mathcal{N}(1,3)$, they have a choice whether or not to accept the offer or not. If they accept, they receive that $w^{accepted}$ draw every period for the rest of their life. If they reject, then (if they don't die) receive zero and get a new wage draw in the next period, discounted by a factor of β. The probability of death is $\alpha = 0.067$, discount factor is $\beta = 0.95$. Their problem is can therefore be written:

$$V(w) = \max_{accept,reject} \left\{ \frac{w}{1-(1-\alpha)\beta}, (1-\alpha)\beta V(w') \right\}$$

2 Problem

Set this problem up and solve it with a reinforcement learning agent in Matlab. Note that while the state is continuous, the decision is discrete. However, you may make a discrete decision continuous (round a sigmoid function, for instance), and approximate a continuous state with a discrete set of N levels. To solve this, you must therefore:

1. Set up an action space (accept/reject) and observation space (wage draw)
2. Set up neural networks (if actor critic, then a critic network that takes in one action and one state and spits out a scalar, and an action network that takes in a state and spits out an action)
3. Set up a reset function that starts your actor
4. Set up a step function that advances them one period, or terminates the problem if they accept (or die)

This can be as simple as creating the observation info (rlNumericSpec(dimensions)), the act info (rlFiniteSetSpec(discrete actions), set up the environment (rlFunctionEnv(obsInfo,actInfo,stepfcn,resetfcn)) and then setting up a default DQN agent (rlDQNAgent(obsInfo,actInfo)), telling it the discount factor (agent.AgentOptions.DiscountFactor = xxx), and training it train(agent,env,opt). **I leave it to you** how you solve it, this is merely a suggestion.
Once you've solved the problem, you should graph out:

- The value function of each action, contingent on draw (getValue(getCritic(agent,wage draw)))
- The action at each wage draw getAction(agent,wage draw)

Suggested Solution

The simplest way to do this is to take Matlab's default Neural Networks. I choose to use a DQN agent, a "deep Q-network" agent. This agent is a little different than our actor-critic agents, it focuses on the "critic" part, and has a deep neural network to approximate it. The basic idea is the same, however–if tries to learn the Q-function (value of state+action) rather than the value-function (value of state, assuming action maximizes). It does so by starting out with an NN that takes in a state action pair, selects either a random action (exploration) or what it thinks is best (greedy), and stores the experience. After doing so multiple times, it has a dataset of state-action-rewards/experiences that it can use to update its network. Maximizing

is easy in this case, because actions are discrete.

When I do this, I have three functions:

- Main.m, which:
  - Sets up the observation info
  - Sets up the action info
  - Defines the environment (observation info, action info, reset function, and step function)
  - Creates the agent (passes observation & action info to rlDQNagent)
  - Trains the agent
  - What is spit out is a little different than a critic we saw, as evaluating it gives two values (value of accept & value of reject).
  - It retrieves & plots the values, and the actions.
- myresetfunction, which:
  - Draws one random draw from a lognormal distribution as the state (that's it!)
- mystepfunction, which:
  - Pulls in the action & the state
  - Determines if you die this period (draws a draw)
  - If you don't die and accept, then it gives you the NPV reward, tells simulator it's done
  - If you don't die and reject, then gives you a new draw, tells simulator it isn't done
  - If you die, then you get no reward, tells simulator it's done

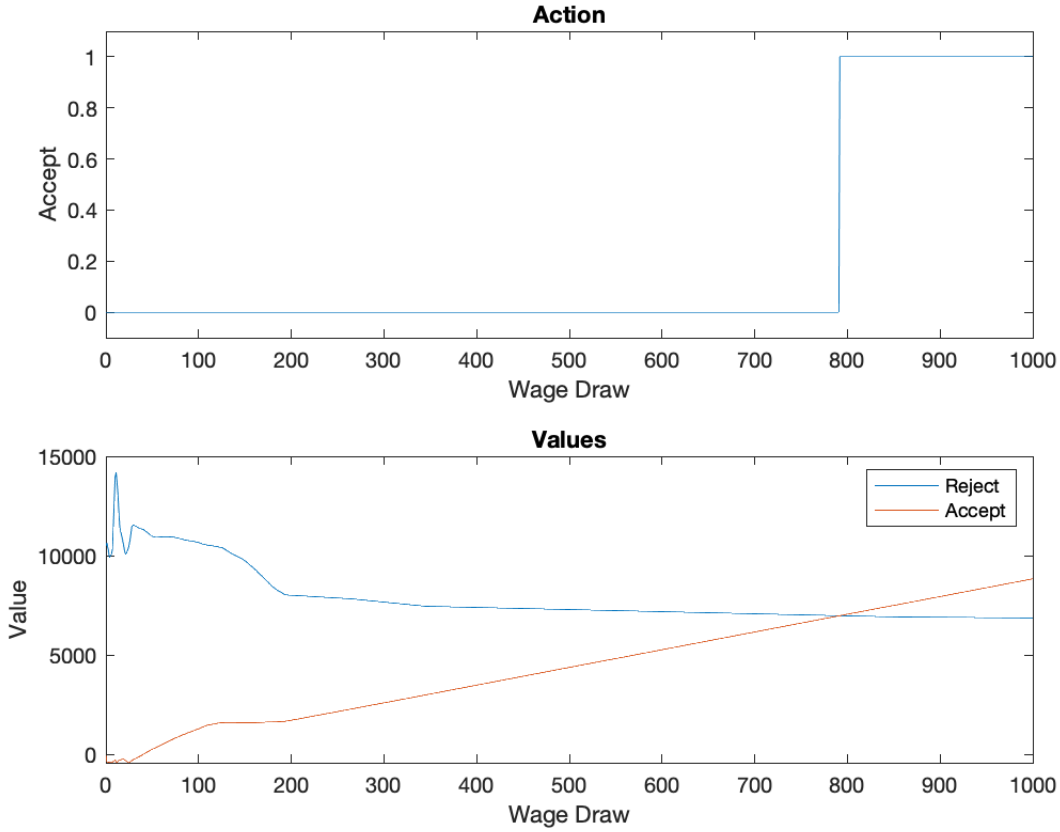
That's it! Shockingly easy to read, as opposed to VFI, which can be less easy to read (simulating a single draw is simpler than integration, no need to write a maximization step, no explicit looping over values). A few interesting lessons:

- We aren't integrating–the program is doing that for us, via monte carlo
- However, it's hard to learn about "rare" events, by their very nature, and lognormal has a lot of rare events!
- Consequently it takes time to converge

- I let it run for 16,000 trials, and plotted the results below, which aren't insane, but you can see some of the issues:
  - the reject value should be flat (it is, approximately, but not perfect)
  - the accept value should be linear (it is, approximately, but not perfect)
  - If you solve for the reservation wage in closed form, with w^* the reservation wage, β the discount factor*chance of living, and $F(w)$ the cumulative distribution function for the wage lognormal with mean 1 and variance 3, you get:

$$w^* = \frac{\beta}{1-\beta}\int_{w^*}^{\infty}(w - w^*)dF(w) \Rightarrow w^* = 1367$$

  - Which suggests we should have let it run a little longer!



3 Research

Change the model in some interesting way and report your results.