BT-3102

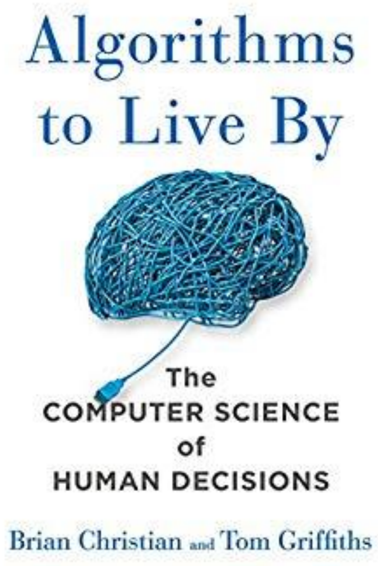
COMPUTATIONAL METHODS FOR BUSINESS ANALYTICS

L12 Markov Decision Process (MDP) & Reinforcement Learning (RL)

Aditya Karanam

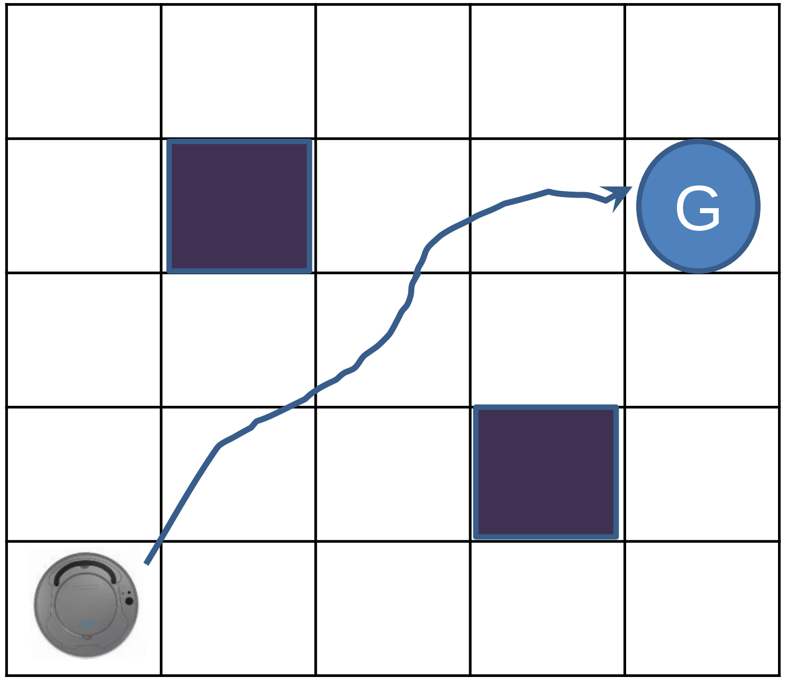
Optional Readings

- *N04-MDP-RL.pdf* (on Canvas)
- AIMA Chapters 17.1 to 17.3
  - *Sequential Decision Problems*; *Value Interation*; *Policy Iteration*
- *Reinforcement Learning: An Introduction*, Richard S. Sutton and Andrew G. Barto, 2018. MIT Press. (Selected chapters freely available online)
- *Algorithms to Live By: The Computer Science of Human Decisions,* Brian Christian and Tom Griffiths, 2017. Picador



Motivation

- Robot can move front, back, left, and right.
- It is now placed in a square space and our objective is to train it to reach a goal (or charging station) irrespective of its position
  - Assume that space is discrete, i.e., a grid
  - The state of the robot is its position in the grid
- The first step is to model the movement or transitions
  - How do we do that?



Transition Probability

- $T(s,a,s') = p(s'|a,s)$ gives the probability of going from s to s' upon taking action a
  - E.g., robot can select an action $a \in A = \{\text{front}, \text{back}, \text{left}, \text{right}\}$
- Assume that it follows first order Markov property
  - $p(s_t|a_{t-1}, s_{t-1}, a_{t-2}, s_{t-2}, \dots, a_0, s_0) = p(s_t|a_{t-1}, s_{t-1})$
- Action a is *non-deterministic*
  - E.g., when robot chooses to go front,

    it moves front with probability 0.8, and

    it moves left with probability 0.1, and

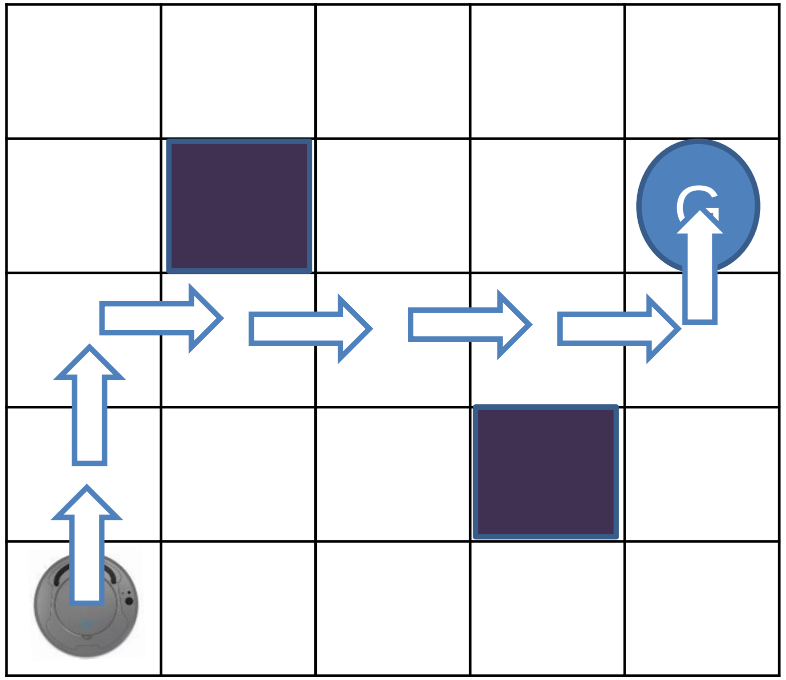
    it moves right with probability 0.1

What else is required?

- Robot can reach its goal by taking:

  {Front, Front, Left, Left, Left, Left, Front}

- Consider the probability for each of these actions is 0.9
- Probability it reaches the goal: $0.9^7 = 0.48$



- Apart from the transition probability,
  - Also, need to communicate or incentivize the robot to achieve what we want

Reward and Utility

- *Reward* is a way of communicating to the robot (or agent) what you want it to achieve and ask it to maximize the *cumulative reward in the long run*.
  - $R(s, a, s')$ is a reward function that gives the reward at each state s
    - *Assumption*: Reward is obtained after taking action a and moving out of state s

- Examples:
  - To make a robot learn to walk, the reward for each time step can be proportional to the robot's forward motion
  - In making a robot learn to escape from a maze, the reward is often -1 for every time step it spends inside the maze
  - How would you design the reward to a robot in Kentridge Cafe so that it learns to deliver the food to the table?

Utility Function and Discounting Factor

- The *utility function* over a path $s_0, s_1, s_2 \dots$ gives the sum of discounted rewards,

- $U[s_0, s_1, s_2, \dots] = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots = \sum_{t=0}^{\infty} \gamma^t R(s_t)$
  - γ: discounting factor with value in (0, 1)
  - Rewards obtained later are worth much less today

- Utility of a path is bounded:
  - $U[s_0, s_1, s_2, \dots] \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \dfrac{R_{max}}{1-\gamma}$

Markov Decision Process (MDP)

- An MDP is defined by
  - A set of states S
  - A set of actions A
  - A transition probability function $T(s, a, s') = p(s'|a, s)$
  - A reward function $R(s, a, s')$
  - Discount: γ (*excluded sometimes*)

MDP: Policy Function

- Goal is to learn an *optimal* policy $\pi^*: S \rightarrow A$

  - Policy: A function π that specifies an action for each state

  - *Optimal* policy ($\pi^*(s)$): a function that specifies the action to take in state s that results in the highest *expected* utility

- Why *expected*?

Notations

- $V^*(s)$: value of state s
  - i.e., the expected utility of starting in state s and acting optimally thereafter

* : represents the optimal value

- $Q^*(s, a)$: Q-value of state s and action a
  - i.e., the expected utility of starting in state s, taking action a and acting optimally thereafter

- $\pi^*(s)$: The optimal policy
  - i.e., it specifies the action that should be taken in state s in order to obtain the highest expected utility
  - Assumes that you act optimally at every step.

Relationship between V^*, Q^*, π^*

1. $V^*(s) = \quad Q^*(s,a)$

2. $Q^*(s,a) =$

3. $\max_a \sum_{s'} T(s,a,s')[\, R(s,a,s') + \gamma V^*(s')\,]$

$= \sum_{s'} T(s,\pi^*(s),s')[\, R(s,\pi^*(s),s') + \gamma V^*(s')\,]$

How to express optimal policy $\pi^*(s)$?

4. $\pi^*(s) =$

$=$

Value Iteration Algorithm

- $V_i^*(s)$ is the estimate of $V^*(s)$ in iteration i

1. Start with $V_0^*(s) = 0 \quad \forall s \in S$
2. Compute V_{i+1}^* given V_i^*:

   $V_{i+1}^*(s) \leftarrow \max\limits_{a} \sum_{s'} T(s,a,s')[\,R(s,a,s') \ + \gamma V_i^*(s')\,]$ *(Bellman equation)*

3. Repeat until convergence
   - i.e., $V_{i+1}^*(s)$ is very close to $V_i^*(s)$ $\forall s$

Convergence Proof of Value Iteration: Basic Idea

- Consider a simple MDP with a single state and a single action
  - $V_{i+1}^* = R(s) + \gamma V_i^*$
- For true optimal V^*, $V^* = R(s) + \gamma V^*$
- Subtracting the above two equations:

$$V_{i+1}^* - V^* = \gamma(V_i^* - V^*) \; (\gamma < 1)$$

- Distance between the estimated and optimal would decrease by a factor γ for each iteration

Practice - Value Iteration

0	1	2	3

- $S = \{0, 1, 2, 3\}$; $A = \{M, B\}$ (M: mini step; B: big step)
- Take mini step, move forward by 1 with prob 1.0
- Take big step, move forward by 2 with prob 0.3 and
  stay in same place with prob 0.7
- Can never exceed state 3. If you take a step and exceed state 3, then you will stay in state 3. If you take any action in state 3, you will always remain in state 3
- $T(k, M, k+1) =$ $\forall k \in \{0, 1, 2\}$
  - $T(3, M, 3) =$
  - $T(k, B, k+2) =$ $T(k, B, k) =$ $\forall k \in \{0, 1\}$
  - $T(2, B, 3) =$ $T(2, B, 2) =$ $T(3, B, 3) =$
- $R(s, a, s') = |s - s'|$
- What are the values of $V_1^*(s)\ \forall s$? (Let $\gamma = 0.5$)

Practice: Value of $V_1^*(0)$

- $V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[\,R(s,a,s') \;+\gamma V_i^*(s')\,]$

- $V_1^*(0) \leftarrow$
  $$\max \begin{Bmatrix} T(0,M,1)[R(0,M,1)+\gamma\, V_0^*(1)], \\ T(0,B,2)[R(0,B,2)+\gamma\, V_0^*(2)] + T(0,B,0)[R(0,B,0)+\gamma\, V_0^*(0)] \end{Bmatrix}$$

- $V_1^*(0) \leftarrow \max \begin{Bmatrix} 1.0\cdot[1+0.5\cdot 0], \\ 0.3\cdot[2+0.5\cdot 0] + 0.7\cdot[0+0.5\cdot 0] \end{Bmatrix}$

- $V_1^*(0) \leftarrow 1.0$

Practice: Value of $V_1^*(1)$

- $V_{i+1}^*(s) \leftarrow \max\limits_a \sum_{s'} T(s,a,s')[\,R(s,a,s') \;+\gamma V_i^*(s')\,]$

- $V_1^*(1) \leftarrow$
$$\max \begin{array}{l} \{\, T(1,M,2)[R(1,M,2)+\gamma\, V_0^*(2)], \\ \;\; T(1,B,3)[R(1,B,3)+\gamma\, V_0^*(3)] + T(1,B,1)[R(1,B,1)+\gamma\, V_0^*(1)]\} \end{array}$$

- $V_1^*(1) \leftarrow \max \begin{array}{l} \{\, 1.0 \cdot [1 + 0.5 \cdot 0], \\ \;\; 0.3 \cdot [2 + 0.5 \cdot 0] + 0.7 \cdot [0 + 0.5 \cdot 0]\} \end{array}$

- $V_1^*(1) \leftarrow 1.0$

Practice: Value of $V_1^*(2)$

- $V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[\ R(s,a,s')\ + \gamma V_i^*(s')\ ]$

- $V_1^*(2) \leftarrow$
$$\max \begin{cases} T(2,M,3)[R(2,M,3) + \gamma\ V_0^*(3)], \\ T(2,B,3)\ [R(2,B,3) + \gamma\ V_0^*(3)] + T(2,B,2)[R(2,B,2) + \gamma\ V_0^*(2)] \end{cases}$$

- $V_1^*(2) \leftarrow \max \begin{cases} 1.0 \cdot [1 + 0.5 \cdot 0], \\ 0.3 \cdot [1 + 0.5 \cdot 0] + 0.7 \cdot [0 + 0.5 \cdot 0] \end{cases}$

- $V_1^*(2) \leftarrow 1.0$

Practice: Value of $V_1^*(3)$

- $V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[\,R(s,a,s') \;+ \gamma V_i^*(s')\,]$
- $V_1^*(3) \leftarrow \max \begin{Bmatrix} T(3,M,3)\,[R(3,M,3) + \gamma\, V_0^*(3)], \\ T(3,B,3)\,[R(3,B,3) + \gamma\, V_0^*(3)] \end{Bmatrix}$
- $V_1^*(3) \leftarrow \max \begin{Bmatrix} 1.0 \cdot [0 + 0.5 \cdot 0], \\ 1.0 \cdot [0 + 0.5 \cdot 0] \end{Bmatrix}$
- $V_1^*(3) \leftarrow 0.0$

- $V_1^*(0) = 1.0,\;\; V_1^*(1) = 1.0,\;\; V_1^*(2) = 1.0,\;\; V_1^*(3) = 0.0$

Quiz: Value of $V_2^*(1)$

- $V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[\,R(s,a,s') \;+ \gamma\, V_i^*(s')\,]$

- $V_2^*(1) = ?$

Practice: Value of $V_2^*(2)$

- $V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[\, R(s,a,s') \;+\gamma V_i^*(s') \,]$

- $V_2^*(2) \leftarrow$
  $$\max \begin{Bmatrix} T(2,M,3)[R(2,M,3) + \gamma\, V_1^*(3)], \\ T(2,B,3)[R(2,B,3) + \gamma\, V_1^*(3)] + T(2,B,2)[R(2,B,2) + \gamma\, V_1^*(2)] \end{Bmatrix}$$

- $V_2^*(2) \leftarrow \max \begin{Bmatrix} 1.0 \cdot [1 + 0.5 \cdot 0], \\ 0.3 \cdot [1 + 0.5 \cdot 0] + 0.7 \cdot [0 + 0.5 \cdot 1] \end{Bmatrix}$

- $V_2^*(2) \leftarrow 1.0$

Practice: Value of $V_2^*(3)$

- $V_{i+1}^*(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[\,R(s,a,s') \;+\gamma V_i^*(s')\,]$

- $V_2^*(3) \leftarrow \max \begin{matrix}\{\,T(3,M,3)[R(3,M,3)+\gamma\, V_1^*(3)], \\ T(3,B,3)[R(3,B,3)+\gamma\, V_1^*(3)]\,\}\end{matrix}$
- $V_2^*(3) \leftarrow \max \begin{matrix}\{1.0\cdot[0+0.5\cdot 0], \\ 1.0\cdot[0+0.5\cdot 0]\}\end{matrix}$
- $V_2^*(3) \leftarrow 0.0$

$V_2^*(0) = 1.5, \qquad V_2^*(1) = 1.5, \qquad V_2^*(2) = 1.0, \qquad V_2^*(3) = 0.0$

Practice: $\pi^*(s)$?

- Assume we obtained convergence: $V^* = V_2^*$
- $\pi^*(s) = \text{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
- $\pi^*(0) = \text{argmax}_a \sum_{s'} T(0,a,s')[R(0,a,s') + \gamma V^*(s')]$
  - $a = M$: $T(0,M,1)[R(0,M,1) + \gamma\, V^*(1)] = 1.0\,[1 + 0.5 \cdot 1.5] = 1.75$
  - $a = B$: $T(0,B,2)[R(0,B,2) + \gamma\, V^*(2)] + T(0,B,0)[R(0,B,0) + \gamma\, V^*(0)]$
    $= 0.3\,[2 + 0.5 \cdot 1.0] + 0.7\,[0 + 0.5 \cdot 1.5] = 1.275$
- $\pi^*(1) = \text{argmax}_a \sum_{s'} T(1,a,s')[R(1,a,s') + \gamma V^*(s')]$
  - $a = M$: $T(1,M,2)[R(1,M,2) + \gamma\, V^*(2)] = 1.0\,[1 + 0.5 \cdot 1.0] = 1.5$
  - $a = B$: $T(1,B,3)[R(1,B,3) + \gamma\, V^*(3)] + T(1,B,1)[R(1,B,1) + \gamma\, V^*(1)]$
    $= 0.3\,[2 + 0.5 \cdot 0] + 0.7\,[0 + 0.5 \cdot 1.5] = 1.125$

How to find $Q^*(s,a)$?

- With Q-value iteration algorithm

1. Start with $Q_0^*(s,a) = 0,\ \forall\, s \in S, \forall\, a \in A$
2. Compute $Q_{i+1}^*(s,a)$ given $Q_i^*(s,a)$:

$$Q_{i+1}^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[\,R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')\,]$$

3. Repeat until convergence
   - i.e., $Q_{i+1}^*(s,a)$ is very close to $Q_i^*(s,a)\quad \forall s \in S$

MDP Application: Health Care

- Optimal timing of living-donor liver transplant
  - States: patient health status
  - Actions: Transplant, Wait
  - Transition function: Obtained by the progression of the disease
  - Reward: total discounted life expectancy
  - Discount factor: 0.99
- https://pubsonline.inorms.org/doi/10.1287/mnsc.1040.0287

MDP Application: Finance

- Consumption-Investment Problem
  - States: Wealth, i.e., amount of money owned
  - Actions: Amount of money invested in stocks, bonds (remainder is consumed)
  - Reward function: Amount of money consumed (or utility function based on money owned)
  - Transition probability: estimated historical performance of stocks, bonds
  - https://www.minet.uni-jena.de/Marie-Curie-ITN/SMIF/talks/Baeuerle.pdf

Reinforcement Learning

- Given states S and actions A
- But don't know transition probs $T(s, a, s')$
- When agent (e.g., robot) in state s, it can experience a reward of getting to that state
- How to find $Q^*(s, a)$?

Naïve Sampling Approach to Estimate $Q^*(s,a)$

- Recall: $Q^*_{i+1}(s,a) = \sum_{s'} T(s,a,s')\,[\,R(s,a,s') + \gamma \max_{a'} Q^*_i(s',a')\,]$
- Estimate $Q^*_{i+1}(s,a)$ using $Q^*_i(s,a)$ and samples obtained by:
  - Repeatedly taking one action at a time, and observing the reward and next state
  - Pretend can restart at s after action a

- Sample 1: $R(s,a,s'_1) + \gamma \max_{a'} Q^*_i(s'_1,a')$
- Sample 2: $R(s,a,s'_2) + \gamma \max_{a'} Q^*_i(s'_2,a')$

  …
- Sample k: $R(s,a,s'_k) + \gamma \max_{a'} Q^*_i(s'_k,a')$
- $Q^*_{i+1}(s,a) = \frac{1}{k} \sum_{j=1}^{k} R\left(s,a,s'_j\right) + \gamma \max_{a'} Q^*_i\left(s'_j,a'\right)$

Naïve Sampling: Analysis

- Problem with naïve sampling:
  - Extremely slow
    - Q-value is updated after every *k* samples or after *k* moves

- In practice, we collect the samples when we actually move.
  - Update Q-value after every move (s, a)
  - After every move, the agent (e.g., robot) observes the next state s' and experiences reward, $R(s, a, s')$

Q-Learning Algorithm

Q-Learning Algorithm:

1. For $i = 1,2,3 \ldots$ (till convergence)
2. Collect a sample: s, a, s' and $R(s, a, s')$
3. Update running average of Q-values

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[\,R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

- α: learning rate
  - Usually, start with $\alpha = 1$, and slowly shrink it to 0 as the number of iterations increases
- How to choose actions or collect samples?

How to choose action a in Q-learning?

- With probability ϵ, pick a randomly (exploration)
- With probability $1 - \epsilon$, pick a that maximizes current estimate of $Q(s, a)$, i.e., $\text{argmax}_\text{a} Q(s, a)$ (exploitation)
- Decrease ϵ with time as we gather more samples and obtain a better estimate of $Q(s, a)$

Applications

- Alpha Go: Deep Mind
  - Documentary: https://www.youtube.com/watch?v=WXuK6gekU1Y

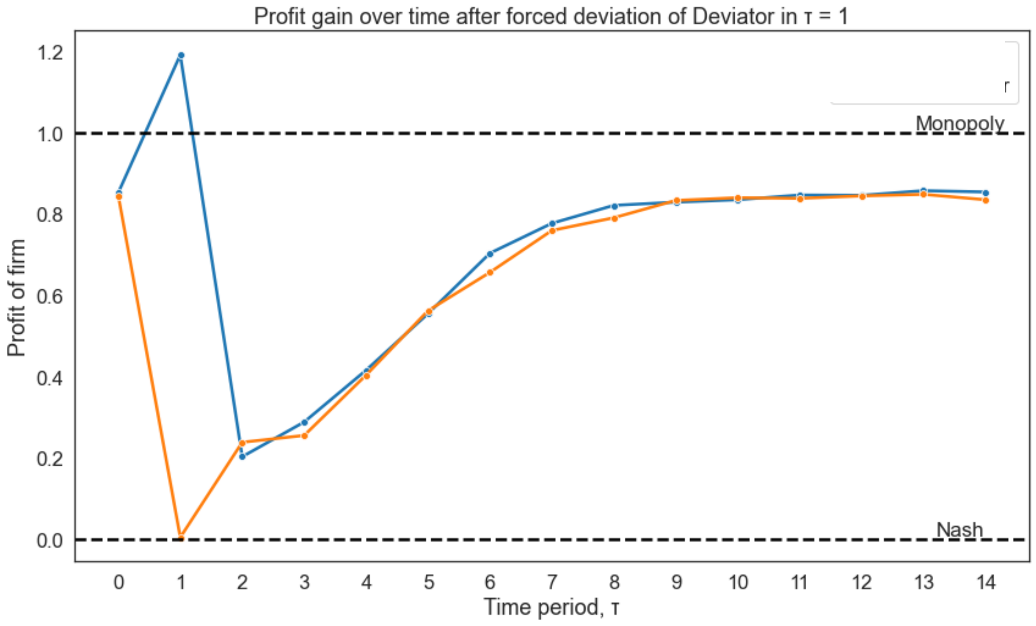
- ChatGPT updates its responses
  - https://huggingface.co/blog/rlhf

- Trading, Advertising, Healthcare …

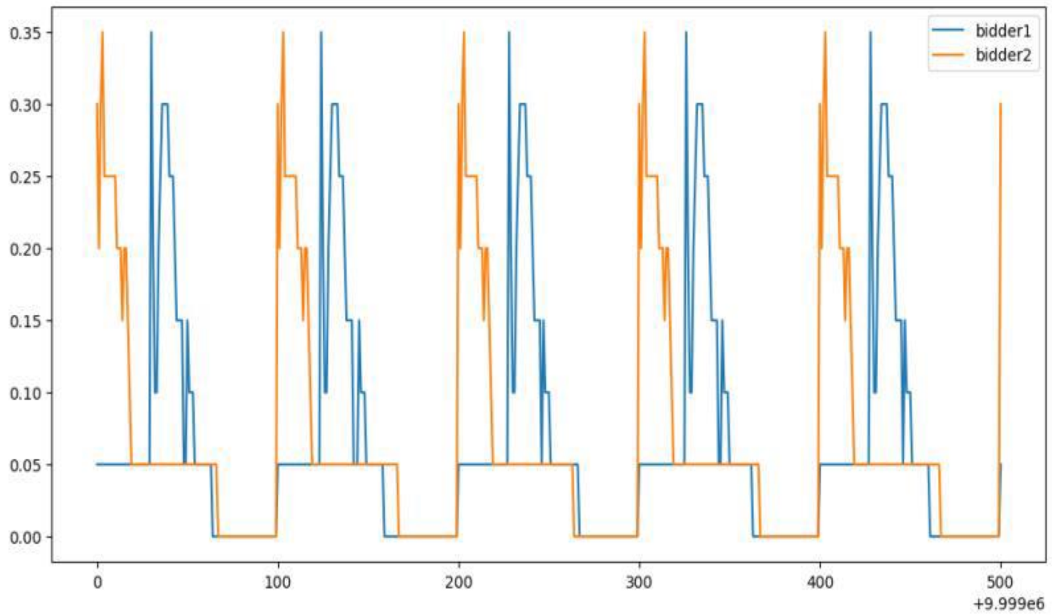
Applications: RL algorithms for pricing

- Solve these questions by examining the behavior of Q-learning and Deep Q-Learning algorithms in repeated repeated economic games (pricing and auctions)

Preliminary results (work in progress)



E-commerce



Auctions
(First and Second Price Auctions)

Thank You