GILLINGS SCHOOL OF GLOBAL PUBLIC HEALTH

PHAIR

# Introduction to Q-learning

Yang Ruan, Yuanyuan Yan, Yajie He

04/12/2024

# Content


**1: Basic setting in reinforcement learning**

**2:** *Intro to Q-learning*

**3:** *Case study: Q-learning in PM*

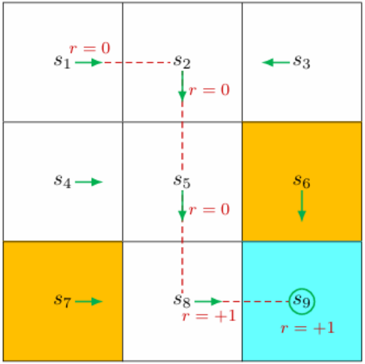
# Basic Concept

- **State**: Describes the agent's status with respect to the environment.
  **State Space**: The set of all the states, denote as $\mathcal{S} = \{s_1, \ldots, s_9\}$
- **Action**: For each state, the possible action the agent can take. e.g. moving upward, moving rightward.
  **Action Space**: The set of all actions, denoted as $\mathcal{A} = \{a_1, \ldots, a_5\}$
- **Policy**: A policy tells the agent which actions to take at every state, policies can be described by conditional probabilities.

$$\pi\left(a_1 \mid s_1\right) = 0$$
$$\pi\left(a_2 \mid s_1\right) = 1$$

# Reward and Trajectory

- **Reward** $R_t$: After executing an action at a state, the agent obtains a reward, denoted as $r$, as feedback from the environment.
- **Trajectory**: A state-action-reward chain.
  $\{S_1 = s_1, A_1 = a_2, R_1 = 0, S_2 = s_2, A_2 = a_2, R_2 = 0, S_3 = s_5, \ldots\}$



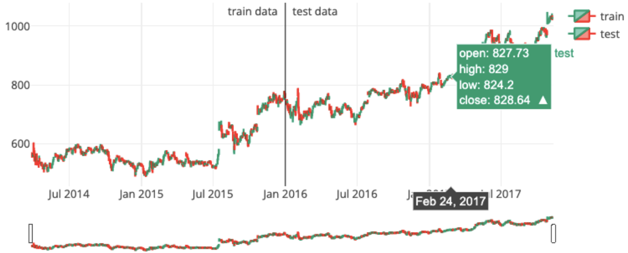

$$s_1 \xrightarrow[r=0]{a_2} s_2 \xrightarrow[r=0]{a_2} s_5 \xrightarrow[r=0]{a_3} s_8 \xrightarrow[r=1]{a_2} s_9.$$

# Return and discount rate

- **Return** $G_t$: The sum of all the rewards collected along the trajectory.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = r_{t+1} + \gamma r_{t+1}$$

- Here we introduce a discount rate $\gamma \in [0, 1)$:
  (1) If $\gamma$ is close to 0, the value of the discounted return is dominated by the rewards obtained in the near future.
  (2) If $\gamma$ is close to 1, the value of the discounted return is dominated by the rewards obtained in the far future.

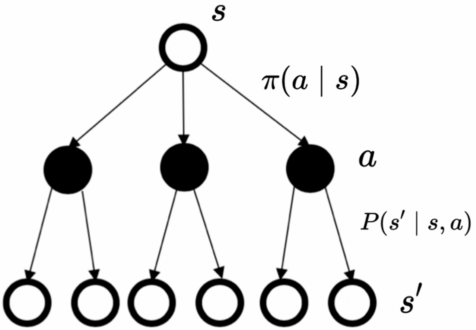


# Markov Decision Process (MDP)

- We define the state transition probability as:

$$p\left(s' \mid s, a\right) \doteq \Pr\left\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\right\}$$

- A Markov decision process is:



# State-Value function and Action-Value function

- **State-Value function**
  state value is defined as
  $$V(s) \doteq \mathbb{E}\left[G_t \mid S_t = s\right]$$
  which is the expected return one would get if starting from state $s$
- **Action-Value function**
  The action value of a state-action pair $(s, a)$ is defined as
  $$q(s, a) \doteq \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right]$$
  which is the expected return one would get if starting from state $s$ and take action $a$

# Bellman Optimality Equation (BOE)

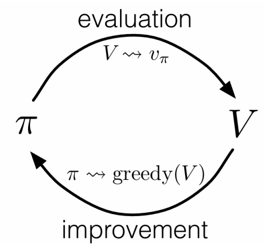
- The Bellman equation is obtained by conditional expectation:

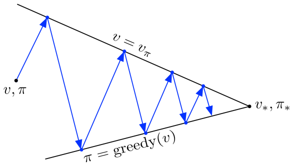
$$
\begin{aligned}
Q_\pi(s, a) &= \mathbb{E}\left[G_t \mid s_t = s, a_t = a\right] \\
&= \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s_t = s, a_t = a\right] \\
&= \mathbb{E}\left[r_{t+1} \mid s_t = s, a_t = a\right] + \gamma \mathbb{E}\left[r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + \ldots \mid s_t = s, a_t = a\right] \\
&= R(s, a) + \gamma \mathbb{E}\left[G_{t+1} \mid s_t = s, a_t = a\right] \\
&= R(s, a) + \gamma \mathbb{E}\left[V(s_{t+1}) \mid s_t = s, a_t = a\right] \\
&= R(s, a) + \gamma \sum_{s' \in S} p\left(s' \mid s, a\right) V_\pi\left(s'\right)
\end{aligned}
$$

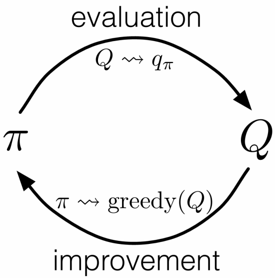
- Note that $V_\pi(s) = \mathbb{E}_\pi\left[G_t \mid s_t = s\right] = \sum_{a \in A} \pi(a \mid s) Q_\pi(s, a)$, we can have two Bellman optimality equations by plug-in either $Q$ or $V$:

$$
\begin{aligned}
V_\pi(s) &= \sum_{a \in A} \pi(a \mid s) \left( R(s, a) + \gamma \sum_{s' \in S} p\left(s' \mid s, a\right) V_\pi\left(s'\right) \right) \\
Q_\pi(s, a) &= R(s, a) + \gamma \sum_{s' \in S} p\left(s' \mid s, a\right) \sum_{a' \in A} \pi\left(a' \mid s'\right) Q_\pi\left(s', a'\right)
\end{aligned}
$$

# Pinciple of optimality




Based on the principle,
$V_\pi(s)$ and $Q_\pi(s, a)$ achieve the optimality at the same time.
$V_{\pi(s)}$ and $\pi_k$ achieve the optimality at the same time.

# Policy Iteration

- An important algorithm for optimizing policy would be **policy iteration**, which is closed related to Bellman optimality equation as well as Markov decision process.
- Policy evaluation step: this step evaluates a given policy by calculating the corresponding state value. That is to solve the following Bellman equation:

$$V_{\pi_k}(s) = \sum_{a \in A} \pi(a \mid s) \left( R(s, a) + \gamma \sum_{s' \in S} p(s' \mid s, a)\, V_{\pi_k}(s') \right)$$

  where $\pi_k$ is the policy obtained in the $k$-th iteration and $V_{\pi_k}$ is the state value to be calculated.
- Policy improvement step: this step aims to improve the policy. The updated policy $\pi_{k+1}$ based on policy evaluation step can be obtained by

$$a_k^*(s) = \arg\max_a q_{\pi_k}(s, a)$$

$$\pi_{k+1}(a \mid s) = \begin{cases} 1, & a = a_k^*(s) \\ 0, & a \neq a_k^*(s) \end{cases}$$

# Policy Iteration

- One can show that the state value sequence $\{V_{\pi_k}\}_{k=0}^{\infty}$ generated by the policy iteration algorithm converges to the optimal state value $V^*$. As a result, the policy sequence $\{\pi_k\}_{k=0}^{\infty}$ converges to an optimal policy $\pi^*$.

**Algorithm** Policy Iteration Algorithm

**Initialization:** System model, $p(r \mid s, a)$ and $p(s' \mid s, a)$ for all $(s, a)$, initial policy $\pi_0$
**Goal:** Search for the optimal state value and an optimal policy.
**while** $V_{\pi_k}$ has not converged, for the $k$-th iteration **do**
    **Policy Evaluation:**
    Initialization: an arbitrary $V_{\pi_k}^{(0)}$
    **while** $V_{\pi_k}^{(j)}$ has not converged, for the $j$-th iteration **do**
        **for** each state $s \in \mathcal{S}$ **do**
            $V_{\pi_k}^{(j+1)}(s) \leftarrow \sum_a \pi_k(a \mid s) \left[\sum_r p(r \mid s, a) r + \gamma \sum_{s'} p(s' \mid s, a) V_{\pi_k}^{(j)}(s')\right]$
            **Policy Improvement:**
            **for** each state $s \in \mathcal{S}$ **do**
                **for** each action $a \in \mathcal{A}$ **do**
                    $q_{\pi_k}(s, a) \leftarrow \sum_r p(r \mid s, a) r + \gamma \sum_{s'} p(s' \mid s, a) v_{\pi_k}(s')$
                    $a_k^*(s) \leftarrow \arg\max_a q_{\pi_k}(s, a)$
                    $\pi_{k+1}(a \mid s) \leftarrow 1$ if $a = a_k^*$ and $\pi_{k+1}(a \mid s) \leftarrow 0$ otherwise

# Content

1: *Basic setting in reinforcement learning*

2: **Intro to Q-learning**

3: *Case study: Q-learning in PM*

# Motivating Example

## From table to function

- As for discussion in previous slides, state and action values are represented by tables.

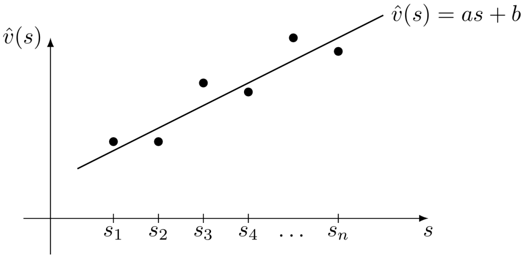
| State | $s_1$ | $s_2$ | $\cdots$ | $s_n$ |
|---|---|---|---|---|
| True value | $v_\pi(s_1)$ | $v_\pi(s_2)$ | $\cdots$ | $v_\pi(s_n)$ |
| Estimated value | $\hat{v}(s_1)$ | $\hat{v}(s_2)$ | $\cdots$ | $\hat{v}(s_n)$ |

| | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
|---|---|---|---|---|---|
| $s_1$ | $q_\pi(s_1, a_1)$ | $q_\pi(s_1, a_2)$ | $q_\pi(s_1, a_3)$ | $q_\pi(s_1, a_4)$ | $q_\pi(s_1, a_5)$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $s_9$ | $q_\pi(s_9, a_1)$ | $q_\pi(s_9, a_2)$ | $q_\pi(s_9, a_3)$ | $q_\pi(s_9, a_4)$ | $q_\pi(s_9, a_5)$ |

- Advantage: intuitive and easy to analyze
- Disadvantage: difficult to handle large or continuous state or action spaces.
  - storage
  - generalization ability

# Motivating Example

- For example, we can use a simple straight line to fit the dots. Suppose the equation of the straight line is

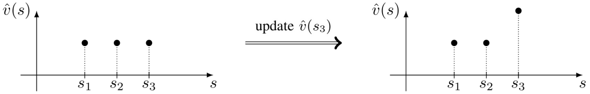



$$\hat{v}(s, w) = as + b = \underbrace{[s, 1]}_{\phi^T(s)} \underbrace{\begin{bmatrix} a \\ b \end{bmatrix}}_{w} = \phi^T(s) w$$

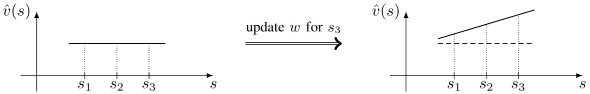
where $w$ is the parameter vector; $\phi(s)$ the feature vector of $s$; $\hat{v}(s, w)$ is linear in $w$.
**Benefit** storage: we do not need to store $|\mathcal{S}|$ state values. We only need to store a lower-dimensional $w$.

# Motivating Example

- How do we update the state-value function?




(a) Tabular method: when $\hat{v}(s_3)$ is updated, the other values remain the same.




(b) Function approximation method: when we update $\hat{v}(s_3)$ by changing $w$, the values of the neighboring states are also changed.

$$\hat{v}(s, w) = \phi^T(s) w$$

**Benefit** generalization ability: When we update $\hat{v}(s)$ by changing $w$, the values of the neighboring states are also changed.

# Stochastic gradient for objective function

- The objective function is

$$J(\boldsymbol{w}) = \mathbb{E}\left[\left(v_\pi(S) - \hat{v}(S, \boldsymbol{w})\right)^2\right].$$

- Our goal is to find the best $\boldsymbol{w}$ that can minimize $J(\boldsymbol{w})$.
- To minimize the objective function $J(\boldsymbol{w})$, we can use the gradient-descent algorithm:

$$\begin{aligned} \boldsymbol{w}_{k+1} &= \boldsymbol{w}_k - \alpha_k \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_k) \\ \longrightarrow \boldsymbol{w}_{k+1} &= \boldsymbol{w}_k + 2\alpha_k \mathbb{E}_S\left[\left(v_\pi(S) - \hat{v}(S, \boldsymbol{w}_k)\right) \nabla_{\boldsymbol{w}} \hat{v}(S, \boldsymbol{w}_k)\right] \end{aligned}$$

- However, the true gradient above involves the calculation of an expectation. We can use the stochastic gradient to replace the true gradient:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \left(v_\pi(s_t) - \hat{v}(s_t, \boldsymbol{w}_t)\right) \nabla_{\boldsymbol{w}} \hat{v}(s_t, \boldsymbol{w}_t) \tag{1}$$

- Notably, $(2)$ is not implementable because it requires the true state value v , which is unknown and must be estimated.

# Temporal-difference with function approximation

- Note that $v(s_t) = r_{t+1} + \gamma v(s_{t+1})$, we replace $v(s_{t+1})$ with its estimate and the algorithm becomes

$$w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{v}\left(s_{t+1}, w_t\right) - \hat{v}\left(s_t, w_t\right)\right] \nabla_w \hat{v}\left(s_t, w_t\right)$$

- This is the algorithm of TD learning with function approximation. An algorithm can be summarized as:

---

**Algorithm** TD learning of state values with function approximation

---

1: **Initialization:** A function $\hat{v}(s, w)$ that is differentiable in $w$. Initial parameter $w_0$.
2: **Goal:** Learn the true state values of a given policy $\pi$.
3: **for** each trajectory $\{(s_t, r_{t+1}, s_{t+1})\}_t$ generated by $\pi$ **do**
4:     **for** each sample $(s_t, r_{t+1}, s_{t+1})$ **do**
5:         In the general case, $w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \hat{v}\left(s_{t+1}, w_t\right) - \hat{v}\left(s_t, w_t\right)\right] \nabla_w \hat{v}\left(s_t, w_t\right)$
6:         In the linear case, $w_{t+1} = w_t + \alpha_t \left[r_{t+1} + \gamma \phi^T\left(s_{t+1}\right) w_t - \phi^T\left(s_t\right) w_t\right] \phi\left(s_t\right)$
7:     **end for**
8: **end for**=0

---

# Sarsa with function approximation

- Note that by principle of optimality, we can intuitively replace the estimate of $v$ with $q$. The Sarsa algorithm with value function approximation is

$$w_{t+1} = w_t + \alpha_t \left[ r_{t+1} + \gamma \hat{q}\left(s_{t+1}, a_{t+1}, w_t\right) - \hat{q}\left(s_t, a_t, w_t\right) \right] \nabla_w \hat{q}\left(s_t, a_t, w_t\right)$$

---

**Algorithm** Sarsa with function approximation

---

1: **Aim:** Search a policy that can lead the agent to the target from an initial state-action pair $(s_0, a_0)$.
2: **for** each trajectory **do**
3: **if** the current $s_t$ is not the target state **then**
4: Take action $a_t$ following $\pi_t(s_t)$, generate $r_{t+1}, s_{t+1}$, and then take action $a_{t+1}$ following $\pi_t(s_{t+1})$
5: **Value update (parameter update):**

$$w_{t+1} = w_t + \alpha_t \left[ r_{t+1} + \gamma \hat{q}\left(s_{t+1}, a_{t+1}, w_t\right) - \hat{q}\left(s_t, a_t, w_t\right) \right] \nabla_w \hat{q}\left(s_t, a_t, w_t\right)$$

6: **Policy update:**

$$\pi_{t+1}\left(a \mid s_t\right) = \begin{cases} 1 - \frac{\varepsilon}{|\mathcal{A}(s)|}\left(|\mathcal{A}(s)| - 1\right) & \text{if } a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}\left(s_t, a, w_{t+1}\right) \\ \frac{\varepsilon}{|\mathcal{A}(s)|} & \text{otherwise} \end{cases}$$

7: **end if**
8: **end for**

---

# Q-learning with function approximation

- The Q-learning algorithm with value function approximation is

$$w_{t+1} = w_t + \alpha_t \left[ r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

---

**Algorithm** Q-learning with function approximation (**on-policy version**)

---

1: **Initialization:** Initial parameter vector $w_0$. Initial policy $\pi_0$. Small $\varepsilon > 0$.

2: **Aim:** Search a good policy that can lead the agent to the target from an initial state-action pair $(s_0, a_0)$.

3: **for** each trajectory **do**

4:     **if** the current $s_t$ is not the target state **then**

5:         Take action $a_t$ following $\pi_t(s_t)$, and generate $r_{t+1}, s_{t+1}$

6:         **Value update (parameter update):**

$$w_{t+1} = w_t + \alpha_t \left[ r_{t+1} + \gamma \max_{a \in \mathcal{A}(s_{t+1})} \hat{q}(s_{t+1}, a, w_t) - \hat{q}(s_t, a_t, w_t) \right] \nabla_w \hat{q}(s_t, a_t, w_t)$$

7:         **Policy update:**

$$\pi_{t+1}(a \mid s_t) = \begin{cases} 1 - \frac{\varepsilon}{|\mathcal{A}(s)|}(|\mathcal{A}(s)| - 1) & \text{if } a = \arg\max_{a \in \mathcal{A}(s_t)} \hat{q}(s_t, a, w_{t+1}) \\ \frac{\varepsilon}{|\mathcal{A}(s)|} & \text{otherwise} \end{cases}$$

# Motivation for off-policy learning

- In X-ray Computed Tomography (CT), projections from many angles are acquired and used for 3D reconstruction.[a] To make CT suitable for in-line quality control, reducing the number of angles while maintaining reconstruction quality is necessary.
- However, it is expensive and even unethical to employ different policies on patients for the purpose of training an optimized policy.
- **off-policy learning**: learning from experiences generated by a different policy than the one it's currently following.

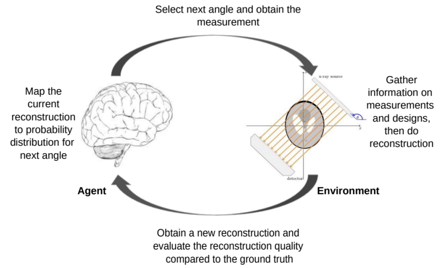



Fig. 1. The interaction between the environment and the agent during policy training

[a] Tianyuan Wang, Felix Lucka, and Tristan van Leeuwen. "Sequential experimental design for X-ray CT using deep reinforcement learning". In: *arXiv preprint arXiv:2307.06343*

# Deep Q-learning

- Deep Q-learning aims to optimize the following objective function:

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}\left(S', a, w\right) - \hat{q}(S, A, w)\right)^2\right]$$

- where $(S, A, R, S')$ are random variables that denote a state, an action, the immediate reward, and the next state, respectively.
- Note that the parameter $w$ appears in $R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}\left(S', a, w\right)$, this bring some computational challenge as it is nontrivial to calculate the gradient.
- One potential method is to fix $w$ when computing the gradient and update $w$ every $C$ iterations.
- We introduce two networks: one is a main network representing $\hat{q}(s, a, w)$ and the other is a target network $\hat{q}\left(s, a, w_T\right)$. The objective function in this case becomes

$$J = \mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}\left(S', a, w_T\right) - \hat{q}(S, A, w)\right)^2\right],$$

where $w_T$ is the target network's parameter.

# Deep Q-learning

- Now we can easily compute the gradient. When $w_T$ is fixed, the gradient of $J$ is

$$\nabla_w J = -\mathbb{E}\left[\left(R + \gamma \max_{a \in \mathcal{A}(S')} \hat{q}\left(S', a, w_T\right) - \hat{q}(S, A, w)\right) \nabla_w \hat{q}(S, A, w)\right]$$

---

**Algorithm** Deep Q-learning (**off-policy version**)

---

1: **Initialization:** A main network and a target network with the same initial parameter.
2: **Goal**: Learn an optimal target network to approximate the optimal action values from the experience samples generated by a given behavior policy $\pi_b$.
3: Store the experience samples generated by $\pi_b$ in $\mathcal{B} = \{(s, a, r, s')\}$
4: **for** each iteration **do**
5: Uniformly draw a mini-batch of samples from $\mathcal{B}$
6: For each sample $(s, a, r, s')$, Calculate the target value as $y_T = r + \gamma \max_{a \in \mathcal{A}(s')} \hat{q}(s', a, w_T)$, where $w_T$ is the parameter of the target network
7: Update the main network to minimize $(y_T - \hat{q}(s, a, w))^2$ using the mini-batch of samples
8: Set $w_T = w$ every $C$ iterations
9: **end for**

---

# Content

1: *Basic setting in reinforcement learning*

2: *Intro to Q-learning*

3: Case study: Q-learning in PM

# Q Learning in PM

## Multi-decision Setting[1]

- Recall that the observed data is of the form $\{(X_{t,i}, A_{t,i}, Y_{t,i})\}_{i=1}^{n}$. At each decision point $t = 1, \ldots, T$, assume that there is a finite set of all possible treatment options $\mathcal{A}_t$ with elements $A_t \in \mathcal{A}_t$.
- Let $Y_T$ be the proximal outcome measured after the treatment at stage $T$.
- Denotes $H_t$ as the set of available patient history at time $t$ such that
  - $H_1 = X_1$
  - $H_t = (H_{t-1}, A_{t-1}, Y_{t-1}, X_t)$
- A dynamic Treatment Regime is a sequence of functions $\boldsymbol{d} = (d_1, \ldots, d_T)$ such that $d_t : \mathcal{H}_t \to \mathcal{A}_t$ for $t = 1, \ldots, T$.
- An optimal treatment regime maximizes the expectation of some (prespecified) cumulative outcome measure $Y = y(Y_1, \ldots, Y_T)$, e.g., $y(v_1, \ldots, v_T) = \sum_{t=1}^{T} v_t$, or $y(v_1, \ldots, v_T) = \max_t v_t$, or $y(v_1, \ldots, v_T) = v_T$.

[1] Phillip J Schulte et al. "Q-and A-learning methods for estimating optimal dynamic treatment regimes". In: *Statistical science: a review journal of the Institute of Mathematical Statistics* 29.4 (2014), p. 640.

# Q Learning in PM

## Optimal Treatment Regime[3]

- Under these assumptions, we can express the optimal regimes in terms of the observed data. We now define the following:

$$Q_T\left(h_T, a_T\right) = E\left(Y_T \mid H_T = h_T, A_T = a_T\right)$$
$$V_T\left(h_T, a_T\right) = \max_{a_T} Q_T\left(h_T, a_T\right)$$

- and for $t = T-1, \ldots, 1$,

$$Q_t\left(h_t, a_t\right) = E\left(V_{t+1}(h_{t+1}, a_t) \mid H_t = h_t, A_t = a_t\right)$$
$$V_t\left(h_t, a_t\right) = \max_{a_t} Q_t\left(h_t, a_t\right)$$

- The optimal DTRs is:

$$d_t^{\text{opt}}\left(h_t\right) = \arg\max_{a_t} Q_t\left(h_t, a_t\right), \qquad \text{for} \quad t = 1, \ldots, T \tag{2}$$

- Q-learning is an approximate dynamic programming[2] algorithm based on (2). This immediately suggests a regression-based estimator $\widehat{Q}_{t,n}(h_t, a_t, \xi_t)$ of $Q_{t,n}(h_t, a_t)$ by regressing $Y$ on $H_t$ and $A_t$, where $\xi_t$ is the parameters for estimating $\widehat{Q}_{t,n}$.

[2] Richard Bellman. "Dynamic programming". In: *Science* 153.3731 (1966), pp. 34–37.
[3] Schulte et al., "Q-and A-learning methods for estimating optimal dynamic treatment regimes".

# Q Learning in PM

## Q-learning for Two Stages[4]

- Considering DTR with only two stages. We may fit linear models for
$$Q_1\left(h_1, a_1; \xi_1\right) = \mathcal{H}_1^T \beta_1 + a_1\left(\mathcal{H}_1^T \psi_1\right)$$
$$Q_2\left(h_2, a_2; \xi_2\right) = \mathcal{H}_2^T \beta_2 + a_2\left(\mathcal{H}_2^T \psi_2\right)$$
- where
$$\mathcal{H}_1 = (1, x_1^T)^T \quad \mathcal{H}_2 = (1, x_1^T, a_1, x_2^T)^T$$
$$\xi_t = (\beta_t^T, \psi_t^T)^T \quad t = 1, 2$$
- Here $Q_2\left(h_2, a_2; \xi_2\right)$ is a model for $E(Y \mid H_2 = h_2, A_2 = a_2)$, a standard regression problem involving observable data, whereas $Q_1\left(s_1, a_1; \xi_1\right)$ is a model for $E(V_2(h_2, a_1) \mid H_1 = h_1, A_1 = a_1)$
- The corresponding V-functions are
$$V_2\left(h_2, a_2; \xi_2\right) = \max_{a_2 \in \{-1,1\}} Q_2\left(h_2, a_2; \xi_2\right)$$
$$= \mathcal{H}_2^T \beta_2 + \left(\mathcal{H}_2^T \psi_2\right) \times \operatorname{sign}\left(\mathcal{H}_2^T \psi_2\right),$$ and
$$V_1\left(s_1; \xi_1\right) = \max_{a \in \{-1,1\}} Q_1\left(s_1, a_1; \xi_1\right)$$
$$= \mathcal{H}_1^T \beta_1 + \left(\mathcal{H}_1^T \psi_1\right) \operatorname{sign}\left(\mathcal{H}_1^T \psi_1\right)$$
- We can see that
$$d_1^{\text{opt}}\left(h_1; \xi_1\right) = \operatorname{sign}\left(\mathcal{H}_1^T \psi_1\right)$$
$$d_2^{\text{opt}}\left(h_2, a_1; \xi_2\right) = \operatorname{sign}\left(\mathcal{H}_2^T \psi_2\right)$$
- We can see that we only need to estimate the regression coefficients $\psi_1$ and $\psi_2$, which can be done via OLS and WLS, etc.

[4] Schulte et al., "Q-and A-learning methods for estimating optimal dynamic treatment regimes".

# Q-learning for Multi-stages

- Here we further explore to multi-stage scenario. Consider the model $Q_t\left(h_t, a_t ; \xi_t\right)$, say, for $t= T, T-1, \ldots, 1$, each depending on a finite-dimensional parameter $\xi_t$
- The models may be linear or nonlinear in $\xi_t$ and include main effects and interactions in the elements of $h_t$ and $a_t$ .
- Estimators $\widehat{\xi}_t$ can be obtained in a backward iterative fashion for $t= T, T-1, .., 1$ by solving suitable estimating equation (e.g., ordinary (OLS) or weighted (WLS) least squares)

$$\sum_{i=1}^n \frac{\partial Q_t\left(H_{ti}, A_{ti} ; \xi_t\right)}{\partial \xi_t} \Sigma_t^{-1}\left(H_{ti}, A_{ti}\right)\left\{\widehat{V}_{(t+1) i}-Q_t\left(H_{ti}, A_{ti} ; \xi_t\right)\right\}=0$$

- where $\Sigma_t\left(h_t, a_t\right)$ is a variance covariance matrix and $\widehat{V}_{ti}$ is an estimation of $V_{ti}$ by incorporating the $\widehat{\xi}_t$ across all $i=1, \ldots, n$. For $t= T$, letting $\widehat{V}_{(T+1) i}= Y_i$.

# Q-learning for Multi-stages

- We can then optimize $Q_t$ by substituting $\widehat{\xi}_t$ for $\xi_t$ and maximizing the equation

$$d_t^{\text{opt}}\left(h_t, a_{t-1}; \widehat{\xi}_T\right) = \arg\max_{a_t \in \Psi_t(h_t)} Q_t\left(h_t, a_{t-1}, \widehat{\xi}_t\right)$$

  where $\Psi_t\left(h_t\right)$ represents the set of allowable treatments for patients at the $t$-th treatment stage given available patient history at time t.
- For each $i$, update

$$\widehat{V}_{ti} = \max_{a_t \in \Psi_t(H_{ti})} Q_t\left(H_{ti}\ , A_{(t-1)i}, a_t; \widehat{\xi}_t\right)$$

- Summarize the estimated optimal regime as $\widehat{d}_Q^{\text{opt}} = \left(\widehat{d}_{Q,1}^{\text{opt}}, \ldots, \widehat{d}_{Q,T}^{\text{opt}}\right)$

$$\widehat{d}_{Q,1}^{\text{opt}}\left(h_1\right) = d_1^{\text{opt}}\left(h_1; \widehat{\xi}_1\right)$$
$$\widehat{d}_{Q,t}^{\text{opt}}\left(h_t, a_{t-1}\right) = d_t^{\text{opt}}\left(h_t, a_{t-1}; \widehat{\xi}_t\right)$$

# Thank you!

You can contact us at:

yangruan@unc.edu    yuanyyan@unc.edu    yajie@unc.edu