Reinforcement Learning

Lecture 3: Q-Learning and Beyond

Chengchun Shi

Lecture Outline

1. Tabular Q-Learning
2. Fitted Q-Iteration
3. Case Study I: Deep Q-Network (DQN) in Atari
4. Case Study II: TD Learning in Ridesharing Platforms

Lecture Outline

1. **Tabular Q-Learning**
2. Fitted Q-Iteration
3. Case Study I: Deep Q-Network (DQN) in Atari
4. Case Study II: TD Learning in Ridesharing Platforms

Introduction to Q-Learning

- One of the most **popular** class of RL algorithms
- Variants include double Q-learning, fitted Q-iteration, deep Q-network (DQN), quantile DQN
- **Main idea**: learn the optimal Q-function $\boldsymbol{Q}^{\pi^{\mathbf{opt}}}$ based on the Bellman **optimality** equation and derive the optimal policy

$$\pi^{\mathbf{opt}}(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} \boldsymbol{Q}^{\pi^{\mathbf{opt}}}(\boldsymbol{s}, \boldsymbol{a})$$

- Focus on **tabular Q-learning** (finite MDP, discrete state and action)

$\pi^{\text{opt}}(s) = \arg\max_a Q^{\pi^{\text{opt}}}(s, a)$?

- $Q^{\pi^{\text{opt}}}(s, a)$ is the **value** of the policy that
  - Assigns a at the initial decision time;
  - Follow π^{opt} afterwards
- $Q^{\pi^{\text{opt}}}(s, \pi^{\text{opt}}(s)) = V^{\pi^{\text{opt}}}(s)$ is the value under the optimal policy π^{opt}
- π^{opt} is stationary and is no worse than any **history-dependent** policy (Lecture 1)

$$Q^{\pi^{\text{opt}}}(s, a) \leq V^{\pi^{\text{opt}}}(s) = Q^{\pi^{\text{opt}}}(s, \pi^{\text{opt}}(s)), \qquad \forall a.$$

- It follows that

$$\pi^{\text{opt}}(s) = \arg\max_a Q^{\pi^{\text{opt}}}(s, a)$$

Bellman Optimality Equation

- Bellman optimality equation for the optimal value function:

$$\boldsymbol{V}^{\pi^{\text{opt}}}(\boldsymbol{s}) = \max_{\boldsymbol{a}} \mathbb{E}[\boldsymbol{R}_t + \gamma \boldsymbol{V}^{\pi^{\text{opt}}}(\boldsymbol{S}_{t+1}) | \boldsymbol{A}_t = \boldsymbol{a}, \boldsymbol{S}_t = \boldsymbol{s}].$$

- Bellman optimality equation for the optimal Q-function:

$$\boldsymbol{Q}^{\pi^{\text{opt}}}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}\left[\boldsymbol{R}_t + \gamma \max_{\boldsymbol{a}'} \boldsymbol{Q}^{\pi^{\text{opt}}}(\boldsymbol{S}_{t+1}, \boldsymbol{a}') | \boldsymbol{A}_t = \boldsymbol{a}, \boldsymbol{S}_t = \boldsymbol{s}\right].$$

- **Proof**: according to Bellman equation,

$$\boldsymbol{Q}^{\pi^{\text{opt}}}(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}\left[\boldsymbol{R}_t + \gamma \boldsymbol{Q}^{\pi^{\text{opt}}}(\boldsymbol{S}_{t+1}, \pi^{\text{opt}}(\boldsymbol{S}_{t+1})) | \boldsymbol{A}_t = \boldsymbol{a}, \boldsymbol{S}_t = \boldsymbol{s}\right]$$

  This together with $\pi^{\text{opt}}(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} \boldsymbol{Q}^{\pi^{\text{opt}}}(\boldsymbol{s}, \boldsymbol{a})$ yields the result.

Q-Learning: an Off-Policy TD Control

- One-step SARSA update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

- One-step tabular Q-learning update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

- In Q-learning, the action in the target is independent of the **behavior policy**
- The behavior policy has an effect on which state-actions are visited

Tabular Q-Learning: Pseudocode

- **Initialization**: Q arbitrary
- **Repeat** for each episode:
  - **Initialize** state s
  - **Repeat** for each step of the episode:
    - $a \leftarrow$ action from s using policy derived from Q (e.g., ε-greedy)
    - Take action a, observe reward r and next state s'
    - $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
    - $s \leftarrow s'$
  - **until** s is a terminal state

On-Policy v.s Off-Policy

- Q-learning is **off-policy**:
  - Updates Q-values using Q-value of next state s' and **greedy** action a'
  - Q-value evaluates the value of **greedy policy** instead of **current policy**
- SARSA is **on-policy**:
  - Updates Q-values using Q-value of next state s' and **current policy's** action a'
  - Q-value evaluates the value of **current policy**

Recap: Convergence of SARSA

Theorem

SARSA converges to the optimal Q-function, $Q(s,a) \to Q^{\pi^{\text{opt}}}(s,a)$ *for any* s *and* a*, if*

- *All state-action pairs are explored infinitely many times,*

$$\sum_{t=0}^{\infty} \mathbb{I}(S_t = s, A_t = a) = \infty.$$

- *The policy converges to a greedy policy,*

$$\lim_{t \to \infty} \pi_t(a|s) = \mathbb{I}(a = \arg\max_{a'} Q_t(s,a'))$$

- *Robbins-Monro sequence of step-sizes [Robbins and Monro, 1951],*

$$\sum_{t=0}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Convergence of Tabular Q-Learning

Theorem (Melo [2001])

Q-learning converges to the optimal Q-function if

- *All state-action pairs are explored infinitely many times,*

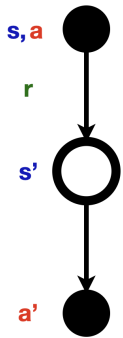
$$\sum_{t=0}^{\infty} \mathbb{I}(S_t = s, A_t = a) = \infty.$$

- *Robbins-Monro sequence of step-sizes [Robbins and Monro, 1951],*

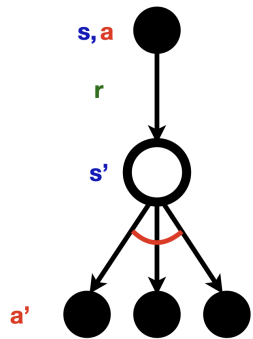
$$\sum_{t=0}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

- Only requires ε to be strictly positive
- **No** need to require ε to decay to zero
- Q-learning converges even if the behavior policy is far from the optimal

Backup Diagram



SARSA



Q-Learning

Cliff Walking Example

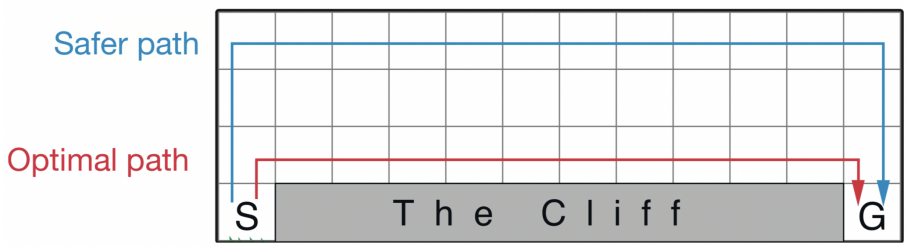


Figure: Illustrations of Cliff Walking

- Undiscounted, **episodic** task
- Actions: up, down, right and left
- Reward of -100 if stepping into **cliff**
- Reward of -1 on other transitions

Cliff Walking Example (Cont'd)

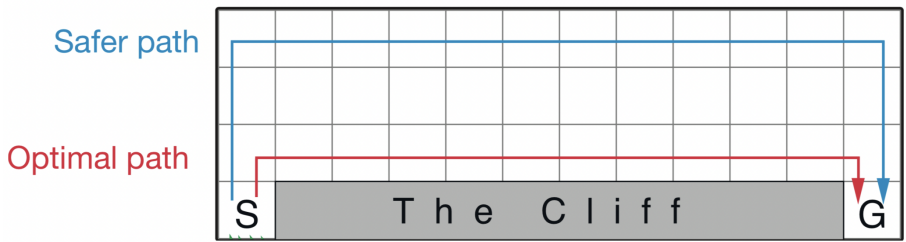


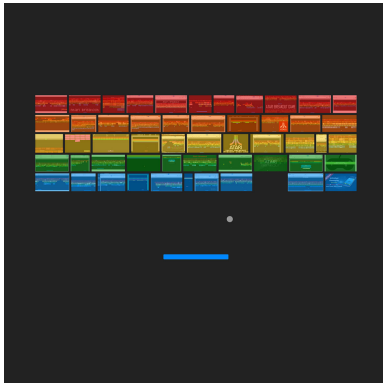
Figure: Illustrations of Cliff Walking

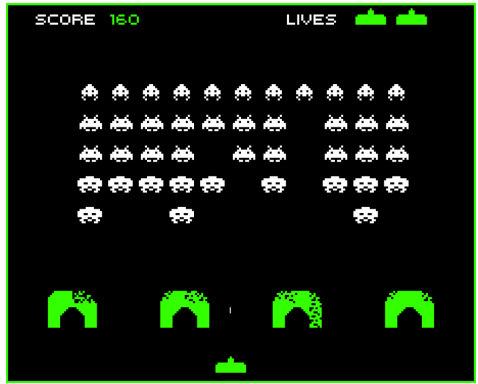
- Q-learning identifies the **optimal** path
- SARSA identifies a **safer** path (the optimal path is not optimal here due to that the ϵ-greedy policy, which might force the agent to fall into the cliff when walking along the optimal path, yielding a low value)

Limitations of Tabular Methods

- So far, we studied reinforcement learning methods using a **tabular representation**
  - Focus on **finite** MDPs
  - Value function represented by a **table**
  - Each state s has an entry for value $V(s)$
  - Each state-action pair (a, s) has an entry for value $Q(s, a)$
- **Limitations** of tabular methods
  - Cannot handle **large-scale** RL problems or **continuous** state space
  - **Scalability**: storage needed to maintain estimates
  - **Slow learning**: learning the value of each state individually

Large Scale RL Problems (Examples)





- Image-valued states (e.g., 210$\times$ 160 pixel image fames, 129 colours)

Large Scale RL Problems (Examples)

- $19 \times 19 = 361$ Go board, each location (empty, black or white) $\rightarrow 3^{361} \approx 10^{170}$ states

Lecture Outline

1. Tabular Q-Learning
2. **Fitted Q-Iteration**
3. Case Study I: Deep Q-Network (DQN) in Atari
4. Case Study II: TD Learning in Ridesharing Platforms

Efficient Control with Function Approximation

- **Batch** (offline) setting with pre-collected data $\{S_t, A_t, R_t, S_{t+1}\}_t$
- Bellman optimality equation

$$Q^{\pi^{\text{opt}}}(S_t, A_t) = \mathbb{E}\left[R_t + \gamma \max_a Q^{\pi^{\text{opt}}}(S_{t+1}, a) \,\middle|\, S_t, A_t\right]$$

- Supervised learning is sample efficient in **batch** settings
- Use supervised learning to learn $Q^{\pi^{\text{opt}}}$ by solving Bellman optimality equation

Challenge

- Bellman optimality equation

$$Q^{\pi^{\text{opt}}}(S_t, A_t) = \mathbb{E}\left[R_t + \gamma \max_a Q^{\pi^{\text{opt}}}(S_{t+1}, a) \,\middle|\, S_t, A_t \right]$$

- Both LHS and RHS involve $Q^{\pi^{\text{opt}}}$
- A naive approach: minimize the **mean squared Bellman error**

$$\sum_t \left[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]^2$$

This would yield a **biased** estimator!

The Bellman Error is Not Learnable

- For a given random variable Z, $\mathbb{E}Z^2 = (\mathbb{E}Z)^2 + \mathrm{Var}(Z)$
- The mean squared Bellman error can be decomposed into **squared bias** + **variance**

$$\begin{aligned} & \mathbb{E}\left[\left\{R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right\}^2 \middle| A_t, S_t\right] \\ = \; & \left[\mathbb{E}\left\{R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\middle| A_t, S_t\right\}\right]^2 \\ + \; & \mathrm{Var}\left[R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\middle| A_t, S_t\right] \end{aligned}$$

- The second line is **zero** when $Q = Q^{\pi^{\text{opt}}}$
- The third line is **nonzero** for any Q and is a function of Q as well
- There is no guarantee $Q^{\pi^{\text{opt}}}$ is the minimizer

Fitted Q-Iteration [Riedmiller, 2005]

- Bellman optimality equation

$$Q^{\pi^{\text{opt}}}(S_t, A_t) = \mathbb{E}\left[R_t + \gamma \max_a Q^{\pi^{\text{opt}}}(S_{t+1}, a) \,\middle|\, S_t, A_t \right]$$

  Both LHS and RHS involve $Q^{\pi^{\text{opt}}}$

- **Main idea**: Fix $Q^{\pi^{\text{opt}}}$ on the RHS
- **Repeat** the following
  1. Compute $\widehat{Q}$ as the argmin of

$$\arg\min_Q \sum_t \left[R_t + \gamma \max_a \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t) \right]^2$$

  2. Set $\widetilde{Q} = \widehat{Q}$

Fitted Q-Iteration (Cont'd)

- During each iteration, consider the objective function

$$\begin{aligned} & \mathbb{E}\left[R_t + \gamma \max_a \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t) \middle| A_t, S_t \right]^2 \\ = & \left[\mathbb{E}\left\{ R_t + \gamma \max_a \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t) \middle| A_t, S_t \right\} \right]^2 \\ + & \operatorname{Var}\left[R_t + \gamma \max_a \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t) \middle| A_t, S_t \right] \end{aligned}$$

- When $\widetilde{Q}$ is close to $Q^{\pi^{\text{opt}}}$, the second line is **small** when $Q = Q^{\pi^{\text{opt}}}$
- The third line is the same for any Q, since $\widetilde{Q}$ is fixed

Fitted Q-Iteration: Algorithm

- **Initialization**: $\widehat{Q}, \widetilde{Q}$ arbitrary, $k = 0$
- While $(k < K)$ **Repeat**

  Generated data $\{(S_t, A_t, R_t, S_{t+1})\}$ using policy derived from $\widehat{Q}$ (e.g., ε-greedy)

  Compute $\widehat{Q}$ as the argmin of

$$\arg\min_{Q} \sum_{t} \left[R_t + \gamma \max_{a} \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t) \right]^2$$

  Set $\widetilde{Q} = \widehat{Q}$

Advantages of Fitted Q-Iteration

- **Flexibility**: any supervised learning method (e.g., deep learning, boosting, random forest) is applicable to learn the Q-function during each iteration.
- **Efficiency**: borrows the strength of supervised learning for sample-efficient estimation. Allows high-dimensional state information.

Theoretical Analysis of Fitted Q-Iteration

- Let $\widehat{Q}_k$ denote the Q-estimator during the kth iteration
- Error decomposition: **bias due to initialization** + **stochastic estimation error**
- The initialization bias $\to \mathbf{0}$ as $k \to \infty$
- The estimation error $\to \mathbf{0}$ when supervised learning provides a **consistent** estimator at each iteration

Theoretical Analysis of Fitted Q-Iteration (Cont'd)

- At the kth iteration,

$$\widehat{Q}_k = \arg\min_{Q} \sum_{t} \left[R_t + \gamma \max_{a} \widehat{Q}_{k-1}(S_{t+1}, a) - Q(S_t, A_t) \right]^2$$

- Supervised learning target:

$$Q_k(s, a) = \mathbb{E}\left[R_t + \gamma \max_{a} \widehat{Q}_{k-1}(S_{t+1}, a) \,\middle|\, S_t = s, A_t = a \right]$$

Theoretical Analysis of Fitted Q-Iteration (Cont'd)

- A key inequality

$$\sup_{s,a} |\widehat{Q}_k(s,a) - Q^{\pi^{\text{opt}}}(s,a)| \leq \sup_{s,a} |\widehat{Q}_k(s,a) - Q_k(s,a)| + \gamma \sup_{s,a} |\widehat{Q}_{k-1}(s,a) - Q^{\pi^{\text{opt}}}(s,a)|$$

- Iteratively applying the inequality

$$\sup_{s,a} |\widehat{Q}_k(s,a) - Q^{\pi^{\text{opt}}}(s,a)| \leq \gamma^k \sup_{s,a} \underbrace{|\widehat{Q}_0(s,a) - Q^{\pi^{\text{opt}}}(s,a)|}_{\text{Initialization Bias}} + \frac{1}{1-\gamma} \sup_{s,a} \max_{j=\{1,\cdots,k\}} \underbrace{|\widehat{Q}_j(s,a) - Q_j(s,a)|}_{\text{Estimation Error}}$$

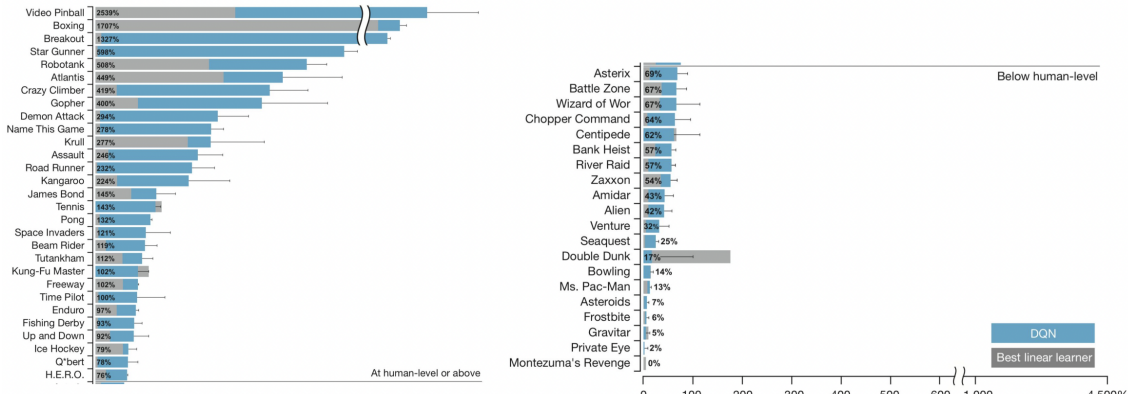
Lecture Outline

1. Tabular Q-Learning
2. Fitted Q-Iteration
3. **Case Study I: Deep Q-Network (DQN) in Atari**
4. Case Study II: TD Learning in Ridesharing Platforms

Deep Q-Network [Mnih et al., 2015]

- **Q-learning type method** that uses a neural network Q-function approximator and several tricks to mitigate instability
- Showed superior performance to previously known methods for playing **Atari 2600 games**
- Q-function approximated by a **convolutional neural network**
- Additional tricks: **experience replay**, **target network**

DQN: Empirical Results

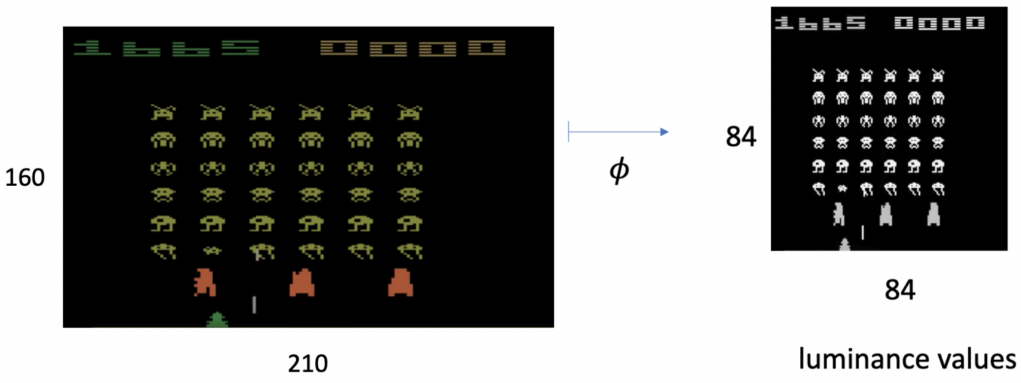


100% * (DQN score - random play score) / (human score - random play score)

Atari 2600 Observation

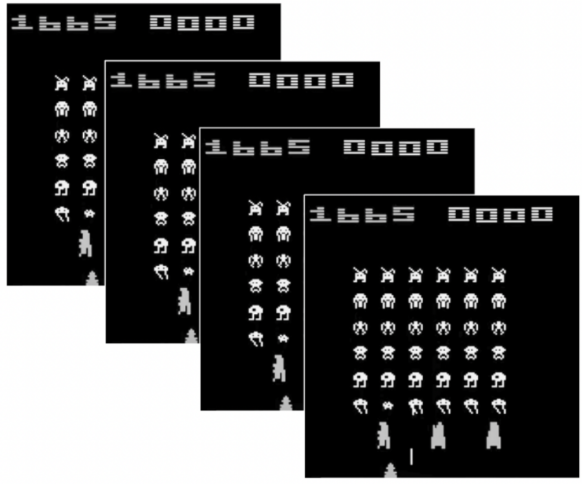
- 210×160 pixel image frames
- 129 colours
- 60Hz frame rate
- Non-Markovian

Input Preprocessing

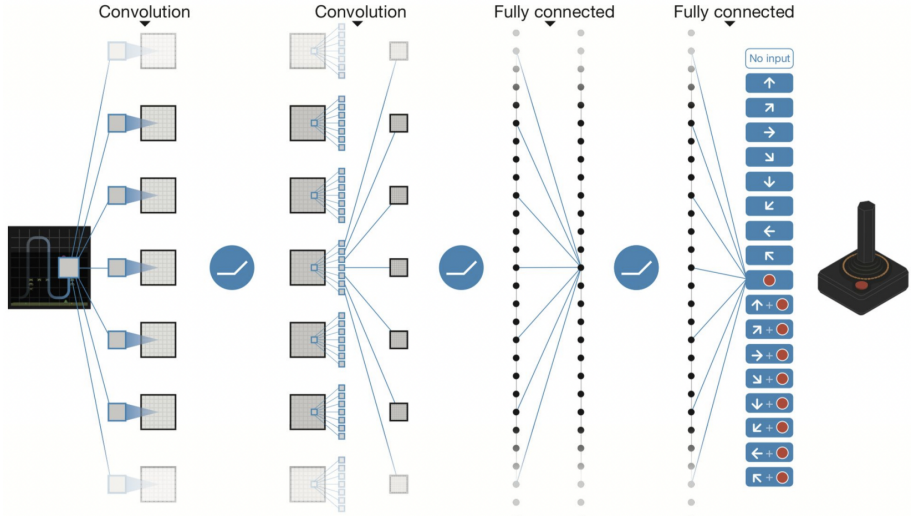


Mitigating Non-Markovianity by Stacking Frames

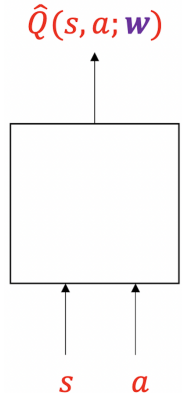
Input is a stack of 4 most recent frames

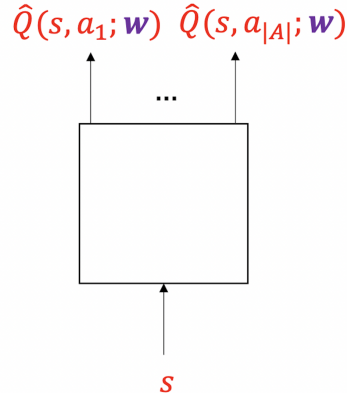


Action Value Approximator



Action Value Approximator (Cont'd)





Two Tricks Used in DQN: Experience Replay

- **Experience Replay**
  - Store transitions (S_t, A_t, R_t, S_{t+1}) in **replay memory** $\mathcal{M}$ at time t
  - Sample **minibatch** of transitions $\{(s_i, a_i, r_i, s_{i+1}) : i \in [n]\}$ from n and update parameters based on this sub-dataset
  - Differs from tabular Q-learning update

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \max_a [R_t + \gamma Q(S_{t+1}, a) - Q(S_t, A_t)],$$

    where only one tuple is used to update the Q-function.
  - Use historical data more **efficiently** to mitigate instability
  - For sufficiently large $\mathcal{M}$, the sampled transitions become asymptotically **independent** (correlation decays with time), yielding more accurate estimate

Recap: Fitted Q-Iteration

- Bellman optimality equation

$$Q^{\pi^{\text{opt}}}(S_t, A_t) = \mathbb{E}\left[R_t + \gamma \max_a Q^{\pi^{\text{opt}}}(S_{t+1}, a) \,\middle|\, S_t, A_t \right]$$

  Both LHS and RHS involve $Q^{\pi^{\text{opt}}}$

- **Main idea**: Fix $Q^{\pi^{\text{opt}}}$ on the RHS
- **Repeat** the following
  1. Compute $\widehat{Q}$ as the argmin of

$$\arg\min_Q \sum_t \left[R_t + \gamma \max_a \widetilde{Q}(S_{t+1}, a) - Q(S_t, A_t) \right]^2$$

  2. Set $\widetilde{Q} = \widehat{Q}$

Two Tricks Used in DQN: Target Network

- According to Bellman optimality equation

$$\underbrace{Q^{\pi^{\text{opt}}}(S_t, A_t; \theta)}_{Q\text{-network}} = \mathbb{E}\left[R_t + \gamma \max_a \underbrace{Q^{\pi^{\text{opt}}}(S_{t+1}, a; \theta^*)}_{\text{target-network}} \middle| S_t, A_t\right]$$

- Fix θ^* in the target network when updating θ in the Q-network
- Perform minibatch SGD T_{target} steps to update θ and set $\theta^* \leftarrow \theta$
- In Mnih et al. [2015], $T_{target} \leftarrow \mathbf{10000}$
- For sufficiently large T_{target}, performing minibatch SGD T_{target} steps is equivalent to

$$\theta \leftarrow \arg\min_{\theta'} \sum_t \left[R_t + \gamma \max_a Q^{\pi^{\text{opt}}}(S_{t+1}, a; \theta^*) - Q^{\pi^{\text{opt}}}(S_t, A_t; \theta')\right]^2$$

- Share similar spirits as **fitted Q-iteration**

The Complete Algorithm

Input: MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, replay memory $\mathcal{M}$, number of iterations T, minibatch size n, exploration probability $\epsilon \in (0,1)$, a family of deep Q-networks $Q_\theta \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, an integer T_{target} for updating the target network, and a sequence of stepsizes $\{\alpha_t\}_{t \geq 0}$.
Initialize the replay memory $\mathcal{M}$ to be empty.
Initialize the Q-network with random weights θ.
Initialize the weights of the target network with $\theta^\star = \theta$.
Initialize the initial state S_0.
for $t = 0, 1, \ldots, T$ **do**
With probability ϵ, choose A_t uniformly at random from $\mathcal{A}$, and with probability $1 - \epsilon$, choose A_t such that $Q_\theta(S_t, A_t) = \max_{a \in \mathcal{A}} Q_\theta(S_t, a)$.
Execute A_t and observe reward R_t and the next state S_{t+1}.
Store transition (S_t, A_t, R_t, S_{t+1}) in $\mathcal{M}$.
Experience replay: Sample random minibatch of transitions $\{(s_i, a_i, r_i, s_i')\}_{i \in [n]}$ from $\mathcal{M}$.
For each $i \in [n]$, compute the target $Y_i = r_i + \gamma \cdot \max_{a \in \mathcal{A}} Q_{\theta^\star}(s_i', a)$.
Update the Q-network: Perform a gradient descent step

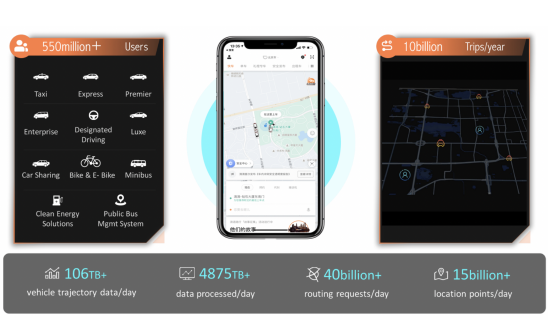
$$\theta \leftarrow \theta - \alpha_t \cdot \frac{1}{n} \sum_{i \in [n]} \big[Y_i - Q_\theta(s_i, a_i) \big] \cdot \nabla_\theta Q_\theta(s_i, a_i).$$

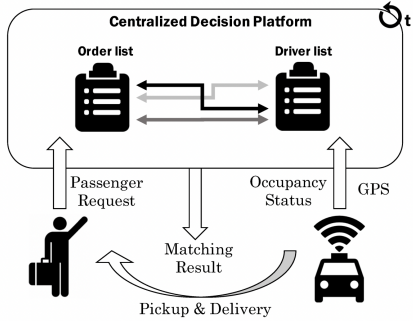
Update the target network: Update $\theta^\star \leftarrow \theta$ every T_{target} steps.
end for
Define policy $\overline{\pi}$ as the greedy policy with respect to Q_θ.
Output: Action-value function Q_θ and policy $\overline{\pi}$.

Lecture Outline

1. Tabular Q-Learning
2. Fitted Q-Iteration
3. Case Study I: Deep Q-Network (DQN) in Atari
4. **Case Study II: TD Learning in Ridesharing Platforms**

Ridesharing: Order-Dispatching





Objective: learn an optimal policy to maximize

- answer rate (proportions of call orders being answered)
- completion rate (proportions of call orders being completed)
- drivers' income

Order Dispatch Policies

- Closest Driver Policy
- MDP Order Dispatch Policy [Xu et al., 2018]
  - **Simple**: no neural networks, no deep learning, use tabular methods
  - **Useful**: performance improvement consistent in all cities, gains in completion rate ranging from 0.5% to 5%, successfully deployed for more that 20 cities
- Some Follow-up Works [Tang et al., 2019, Wan et al., 2021]

Closest Driver Policy

Assign the call order to the closest available driver

$$\arg\min_{a_{i,j}} \sum_{i=1}^{m}\sum_{j=1}^{n} d(i,j)a_{i,j} \qquad \text{Minimize driver-passenger total distance}$$

$$s.t. \sum_{i=1}^{m} a_{i,j} \leq 1,\ j = 1,\cdots,n \qquad \text{Order assigned to at most one driver}$$

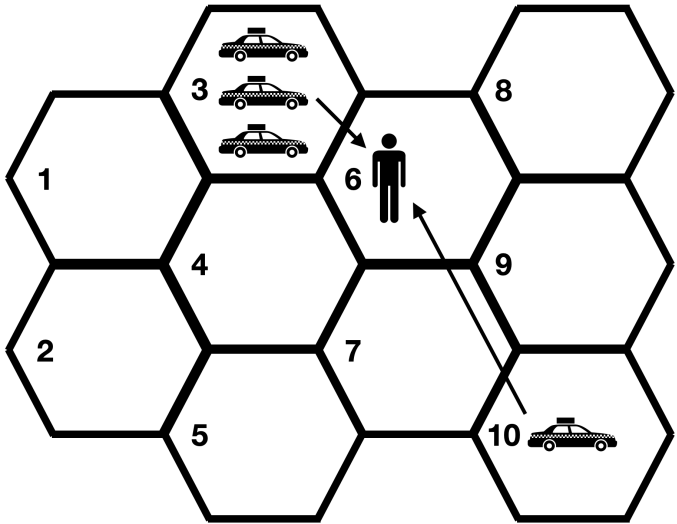
$$\sum_{j=1}^{n} a_{i,j} \leq 1,\ i = 1,\cdots,m \qquad \text{Driver assigned to at most one order}$$

- i indexes the ith driver
- $d(i,j)$ = distance between i and j
- One of the two equalities shall hold
- j indexes the jth order
- $a_{i,j} = 1 \Leftrightarrow$ order j is assigned to i

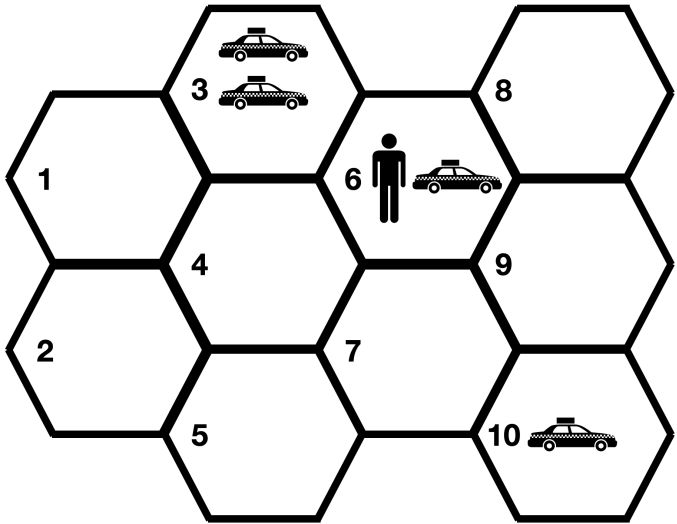
Closest Driver Policy: Limitations

- The company implements the policy every 2 seconds
- **Myopic** policy (e.g., maximize immediate rewards)
- No guarantee it will maximize long-term rewards
- Example given in the next slide

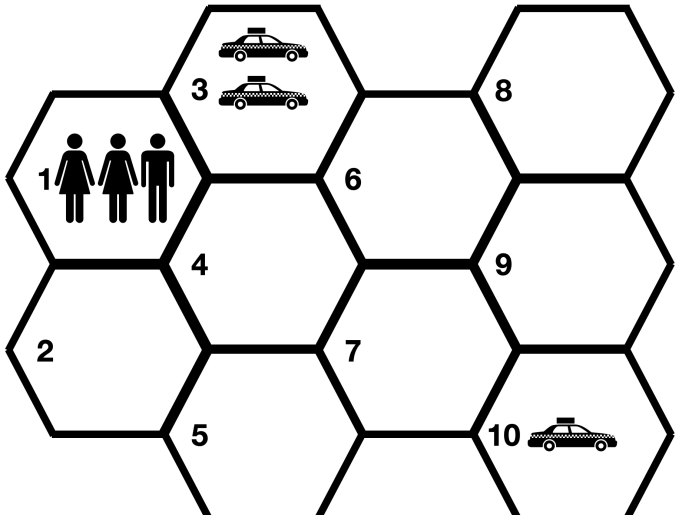
Illustration of Limitations of Closest Driver Policy



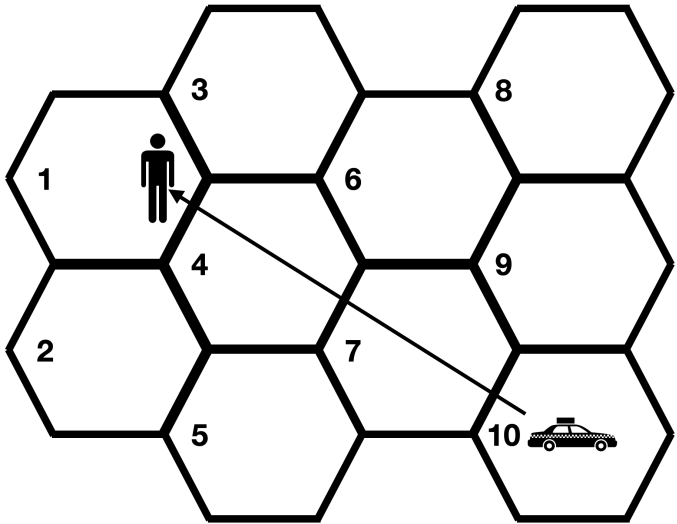
Adopting the Closest Driver Policy



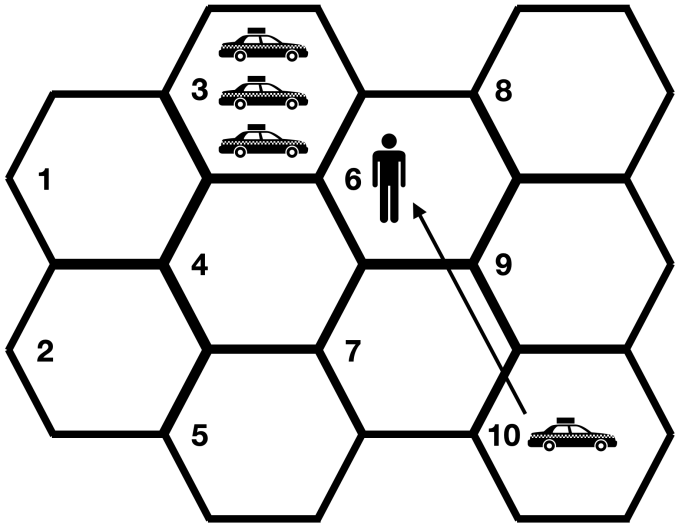
Some Time Later · · ·



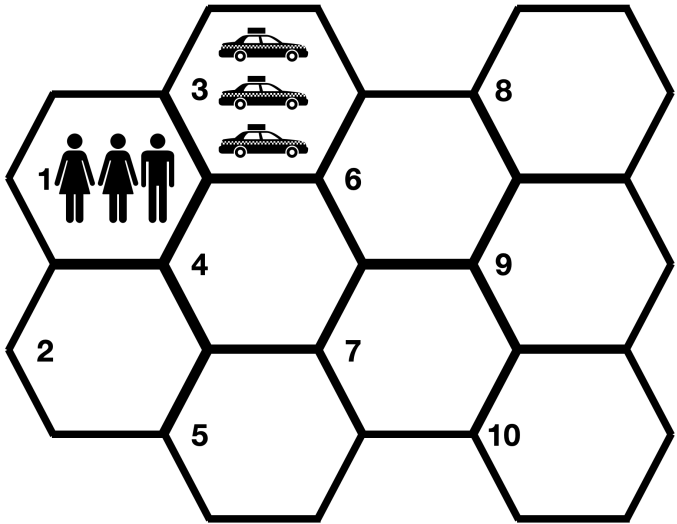
Miss One Order



Consider a Different Action



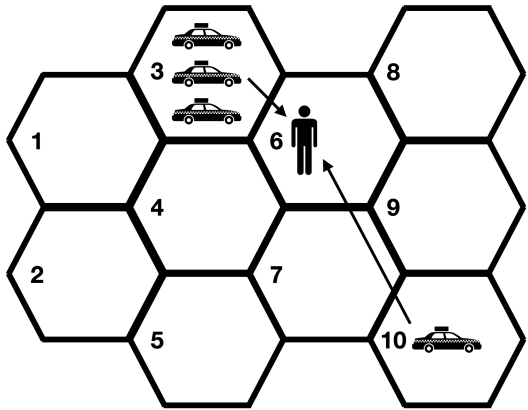
Able to Match All Orders



MDP Order Dispatch Policy

- Adopts a **reinforcement learning** framework to optimize long-term rewards
- Delivers **remarkable improvement** on the platform's efficiency
- **Challenges**:
  - Huge state space (e.g., origin/destination of call orders, location of available drivers)
  - Huge action space: number of matchings grows exponentially with number of orders/drivers. With $\boldsymbol{n}$ orders and $\boldsymbol{n}$ drivers, number of potential matchings $= \boldsymbol{n!}$

Main Idea



- Closest driver is myopic because its objective function (e.g., total distance) only considers immediate rewards
- Use an objective function that involves long-term rewards (e.g., value)

Main Idea (Cont'd)

- A learning and planning approach
- **Learning**: policy evaluation based on historical data
- **Planning**: order dispatch by maximizing total value

An MDP Framework

- Model each driver as an agent
- **State**: 2-dim vector (time, location)
- **Action**: two types of actions
  1. Serving action: assign the driver to server an order
  2. Idle action: allows drivers to stay in the same location, to serve an order in the next time
- **Reward**:
  1. an order is completed or not (0/1) (completion rate)
  2. driver's revenue from an order (driver's income)

An MDP Framework (Cont'd)

- **Discounted Factor**: $\gamma = \mathbf{0.9}$. An order that lasts for time $\boldsymbol{T}$ with reward $\boldsymbol{R}$

$$\boldsymbol{r} = \sum_{\boldsymbol{t}=\mathbf{0}}^{\boldsymbol{T}-\mathbf{1}} \gamma^{\boldsymbol{t}} \frac{\boldsymbol{R}}{\boldsymbol{T}}$$

- **Example**:
  - An driver in area **A** receives an order from **B** to **C** at time 00:00
  - The driver arrives **C** at 30min and earns 30£
  - 10min as one time unit, $\gamma = 0.9$
  - **State transition**: $(\mathbf{0}, \boldsymbol{A}) \rightarrow (\mathbf{3}, \boldsymbol{C})$
  - **Reward**: $\mathbf{10} + \mathbf{0.9} \times \mathbf{10} + \mathbf{0.9^2} \times \mathbf{10} = \mathbf{27.1}$

Learning: Policy Evaluation

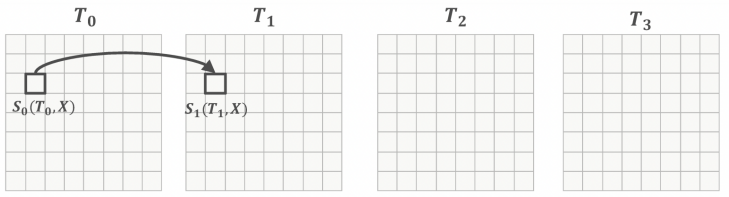
- Break down historical data into a set of transitions pairs $(s, a, r, s', \Delta t)$, where Δt denotes the time of pickup, waiting and deliver process
- TD update rule for the **idle** action

$$V(s) \leftarrow V(s) + \alpha[0 + \gamma V(s') - V(s)]$$

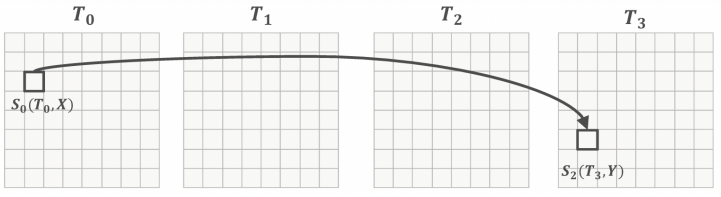
- TD update rule for the **serving** action

$$V(s) \leftarrow V(s) + \alpha[r + \gamma^{\Delta t} V(s') - V(s)]$$

Policy Evaluation: Example



Idle action: $V(S_0) \leftarrow V(S_0) + \alpha(0 + \gamma V(S_1) - V(S_0))$



Serving action: $V(S_0) \leftarrow V(S_0) + \alpha(R_\gamma + \gamma^3 V(S_2) - V(S_0))$

Policy Evaluation: Pseudocode

- **Input**: Collect historical state transitions $\boldsymbol{D} = \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'}, \Delta \boldsymbol{t})\}$ where each state is composed of a time and space index
- **Initialize** $\boldsymbol{V}(\boldsymbol{s})$ and $\boldsymbol{N}(\boldsymbol{s})$ to zero for any $\boldsymbol{s}$
- **For** $\boldsymbol{t} = \boldsymbol{T} - \boldsymbol{1}$ to $\boldsymbol{0}$ **do**
  Find a data subset $\boldsymbol{D_t}$ where the time index of the state is $\boldsymbol{t}$
  For each sample $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'}, \Delta \boldsymbol{t})$ in $\boldsymbol{D_t}$ **do**
  $\boldsymbol{N}(\boldsymbol{s}) \leftarrow \boldsymbol{N}(\boldsymbol{s}) + \boldsymbol{1}$
  $\boldsymbol{V}(\boldsymbol{s}) \leftarrow \boldsymbol{V}(\boldsymbol{s}) + \boldsymbol{N}^{-\boldsymbol{1}}(\boldsymbol{s})[\boldsymbol{r} + \gamma^{\Delta \boldsymbol{t}} \boldsymbol{V}(\boldsymbol{s'}) - \boldsymbol{V}(\boldsymbol{s})]$
  End For
  End For
- **Return** $\boldsymbol{V}$

Planning: Order Dispatch

Recall the closest driver policy

$$\arg\min_{a_{i,j}} \sum_{i=0}^{m}\sum_{j=1}^{n} d(i,j)a_{i,j} \qquad \text{Minimize driver-passenger total distance}$$

$$s.t. \sum_{i=1}^{m} a_{i,j} \leq 1,\ j = 1,\cdots,n \qquad \text{Order assigned to at most one driver}$$

$$\sum_{j=0}^{n} a_{i,j} \leq 1,\ i = 1,\cdots,m \qquad \text{Driver assigned to at most one order}$$

- i indexes the ith driver
- $d(i,j)$ = distance between i and j
- j indexes the jth order
- $a_{i,j} = 1 \Leftrightarrow$ order j is assigned to i

Planning: Order Dispatch (Cont'd)

The MDP order dispatch policy

$$\arg\max_{a_{i,j}} \sum_{i=0}^{m}\sum_{j=1}^{n} A(i,j)a_{i,j} \qquad \text{Maximize total advantage function}$$

$$s.t. \sum_{i=1}^{m} a_{i,j} \leq 1,\ j = 1,\cdots,n \qquad \text{Order assigned to at most one driver}$$

$$\sum_{j=0}^{n} a_{i,j} \leq 1,\ i = 1,\cdots,m \qquad \text{Driver assigned to at most one order}$$

- i indexes the ith driver
- $A(i,j)$ = advantage function
- j indexes the jth order
- $a_{i,j} = 1 \Leftrightarrow$ order j is assigned to i

Advantage Function Trick

- **What** is advantage function?
  - Difference between Q-function and value function.
- **Why** use advantage function trick?
  - Optimize long-term rewards
  - Send drivers in areas with lower values ("cold regions") to areas with higher values ("hot regions")

What is Advantage Function

- $A(i,j) = r_{i,j} + \gamma^{\Delta t_{i,j}} V(s'_{i,j}) - V(s_i)$
- i indexes ith driver, j indexes jth order
- $r_{i,j}$: expected gain for ith driver to serve jth order
- s_i: initial state of ith driver
- $s'_{i,j}$: state of ith driver after serving jth order
  - **Time**: $s'_{i,j}(t) = s_i(t) + \Delta t_{i,j}$
  - **Location**: $s'_{i,j}(\ell)$, the destination of jth order
- The first two term corresponds to the state-action value function (Q-function) of assigning ith driver to jth order

Why Use Advantage Function Trick

- $A(i,j) = r_{i,j} + \gamma^{\Delta t_{i,j}} V(s'_{i,j}) - V(s_i)$
- **Order Price**: an order with a **high utility** leads to a high advantage
- **Driver's Location**:
  - Value of a driver's current location has a **negative** impact on the advantage
  - When # drivers $>$ # orders (**oversupplied**), drivers in areas with lower values ("cold regions") are more likely to be selected
- **Order's Destination**:
  - Value of an order's destination has a **positive** impact on the advantage
  - When # drivers $<$ # orders (**undersupplied**), orders whose destinations have higher values ("hot regions") are more likely to be selected
- **Pickup Distance**:
  - Contributes to the advantage **implicitly**
  - A larger pickup distance $\Longrightarrow$ a larger $\nabla t_{i,j} \Longrightarrow$ a lower advantage
  - Considers **immediate reward** as well

Simulations: Toy Example

- A simple map of $\mathbf{9} \times \mathbf{9}$ spatial grids with $\mathbf{20}$ time steps
- Orders can only be dispatched to drivers in distance that are no greater than $\mathbf{2}$
- Simulate realistic traffic patterns with a morning-peak and a night-peak, centralized on different locations of residential areas and working areas
- Competing methods
  - Distance-based
  - Myopic ($\gamma = \mathbf{0}$)

Toy Example (Cont'd)

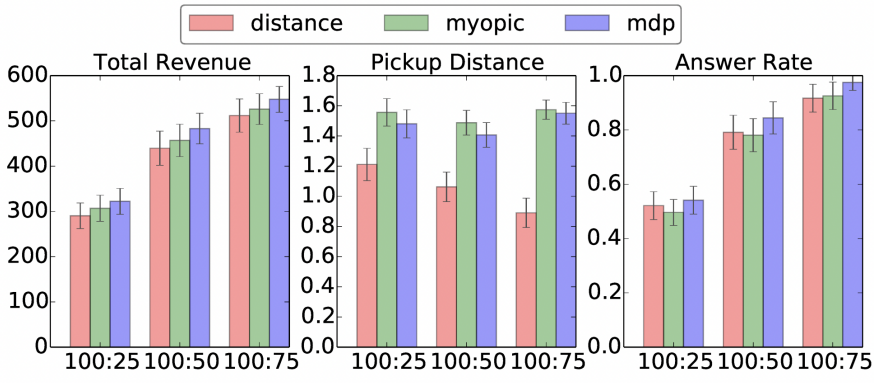
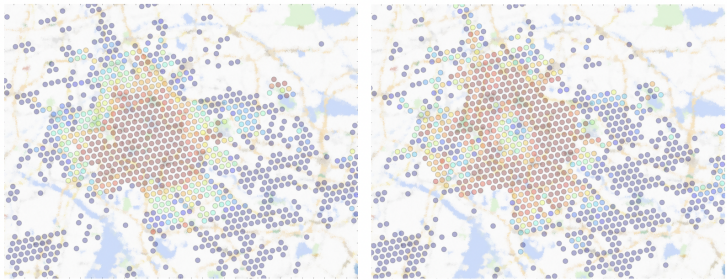


Figure 6: Comparison of distance-based method, myopic method and the proposed MDP method in three metrics on the toy example environment. X-axis stands for the order-driver ratios. Better viewed in color.

Real-World Experiment

- Performance improvement brought by the MDP method is consistent in all cities
- Gains in global GMV and completion rate ranging from 0.5% to 5%
- Successfully deployed for more than **20** cities
- Serving millions of trips in a daily basis

Real-World Experiment (Cont'd)



(a) 18:00-18:10, evening peak (b) 09:00-09:10, after morning peak

Figure 8: Sampled value function for the same city at different times. Red indicates higher values, blue for lower ones. Better viewed in color.

Summary

- Tabular Q-Learning
- Fitted Q-Iteration
- Deep Q-Network
- Experience Replay
- Target Network
- Convolutional Neural Networks
- Closest Driver Policy
- MDP Order Dispatch
- Advantage Function Trick

References I

Francisco S Melo. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pages 1–4, 2001.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, pages 317–328. Springer, 2005.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

References II

Xiaocheng Tang, Zhiwei Qin, Fan Zhang, Zhaodong Wang, Zhe Xu, Yintai Ma, Hongtu Zhu, and Jieping Ye. A deep value-network based approach for multi-driver order dispatching. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1780–1790, 2019.

Runzhe Wan, Sheng Zhang, Chengchun Shi, Shikai Luo, and Rui Song. Pattern transfer learning for reinforcement learning in order dispatching. *arXiv preprint arXiv:2105.13218*, 2021.

Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 905–913, 2018.

Questions