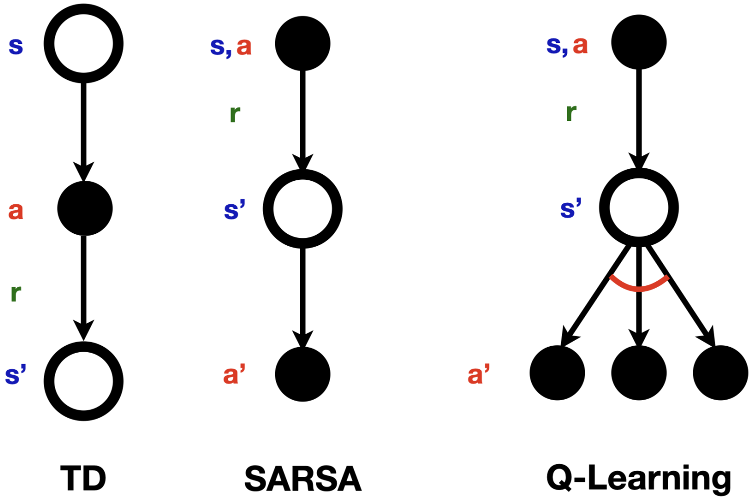
ST455: Reinforcement Learning

Lecture 7: TD Learning (Case Studies)

Chengchun Shi

Recap: Lecture 4, Introduction to TD



Recap: Lecture 5, TD with Function Approximation

- Introduction to Value Function Approximation
- Gradient Descent-based Methods
- Fitted Q-Iteration

Lecture Outline

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

Lecture Outline

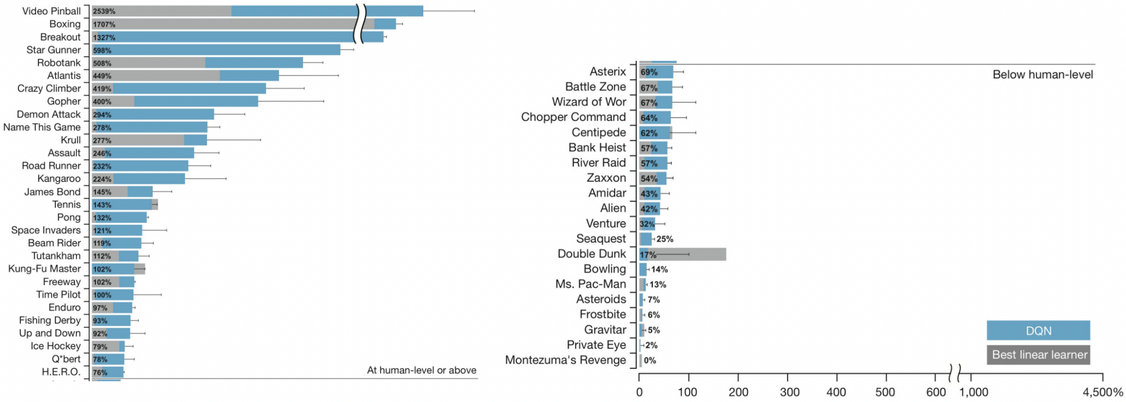
1. **Case Study I: Deep Q-Network (DQN) in Atari**

2. 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

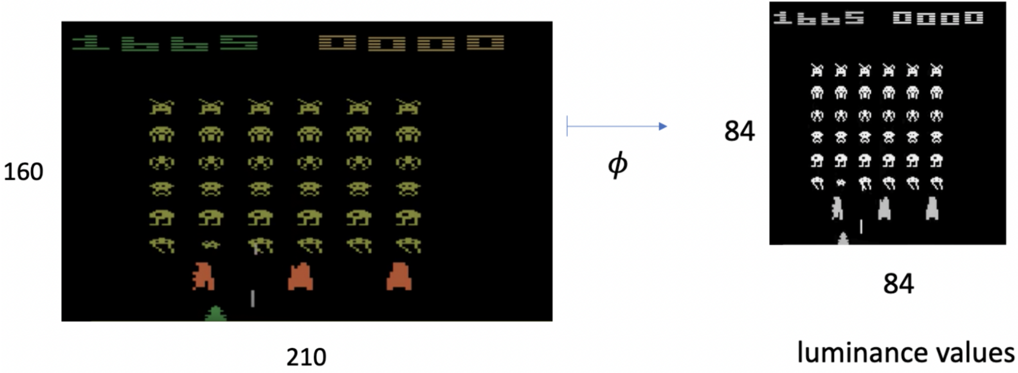


100% * (DQN score - random play score) / (human score - random play score)

Atari 2600 Observations

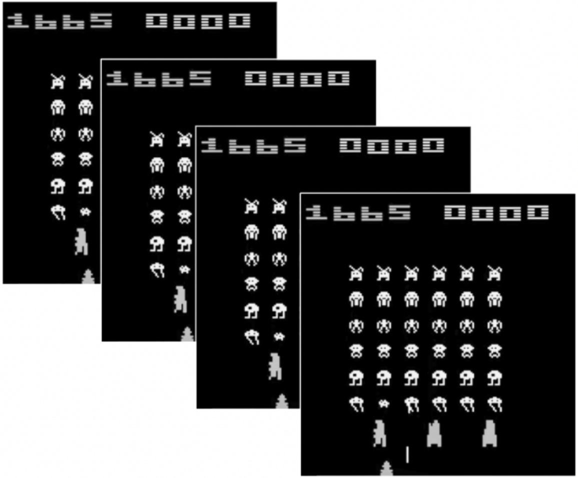
- 210×160 pixel image frames
- 129 colours
- 60Hz frame rate
- Non-Markovian

Input Preprocessing



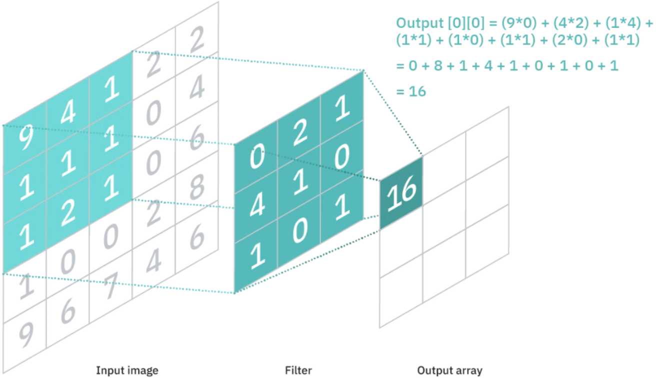
Mitigating Non-Markovianity by Stacking Frames

Input is a stack of 4 most recent frames



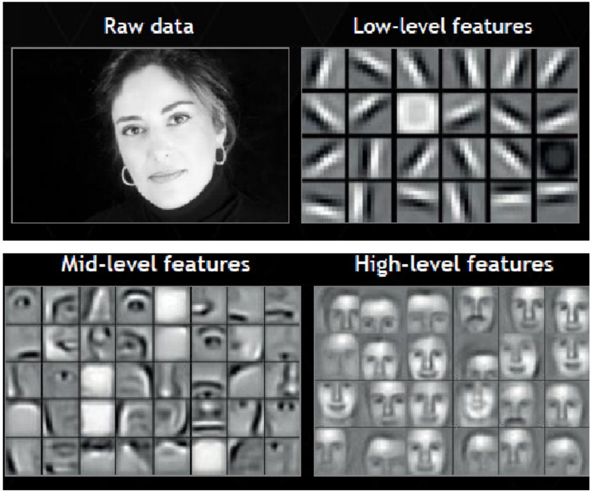
Convolutional Neural Networks

- **Filter**, typically a 3×3 matrix, determines the size of output array
- **Parameter sharing**, weights fixed as filter moves across the image

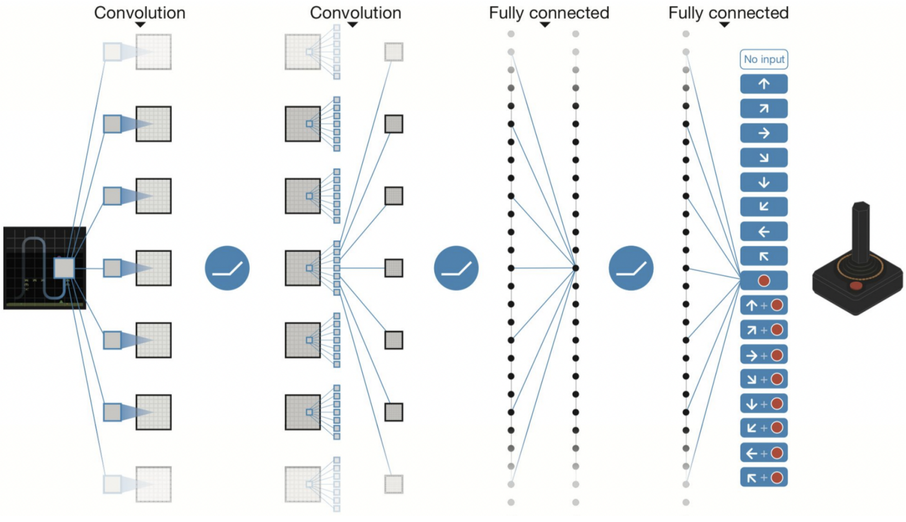


Taken from `https://www.ibm.com/cloud/learn/convolutional-neural-networks`

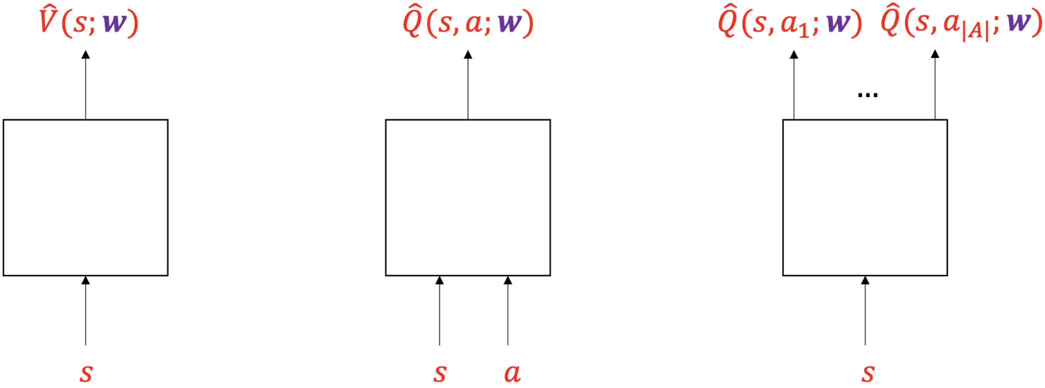
Feature Extraction



Action Value Approximator



Action Value Approximator (Cont'd)



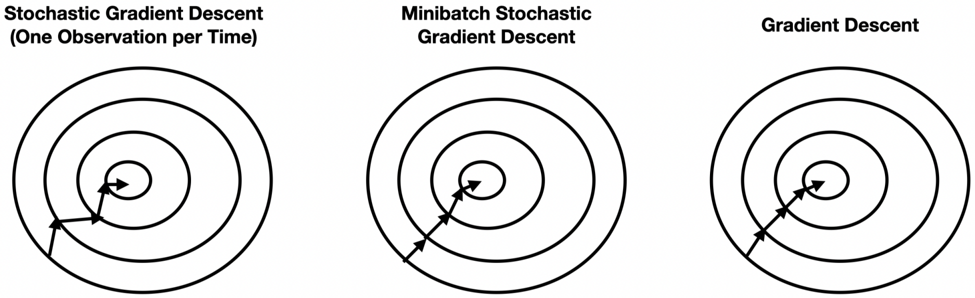
Two Tricks Used in DQN: Experience Replay

- **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 classical 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

Minibatch Stochastic Gradient Descent



- Each point represents a parameter
- Circle represents parameters with the same loss function

Recap: Fitted Q-Iteration

- Bellman optimality equation

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

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

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

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

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

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

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

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.

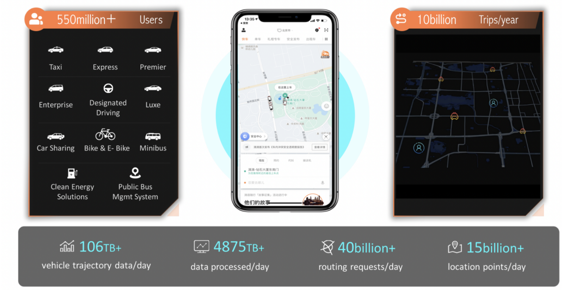
Output: Action-value function Q_θ and policy $\overline{\pi}$.

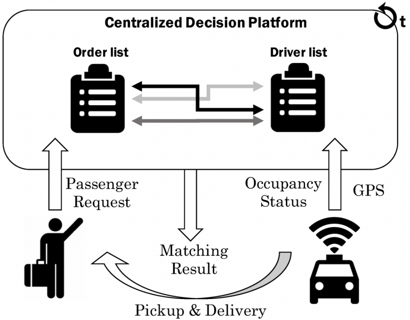
Lecture Outline

1. Case Study I: Deep Q-Network (DQN) in Atari

2. 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

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

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

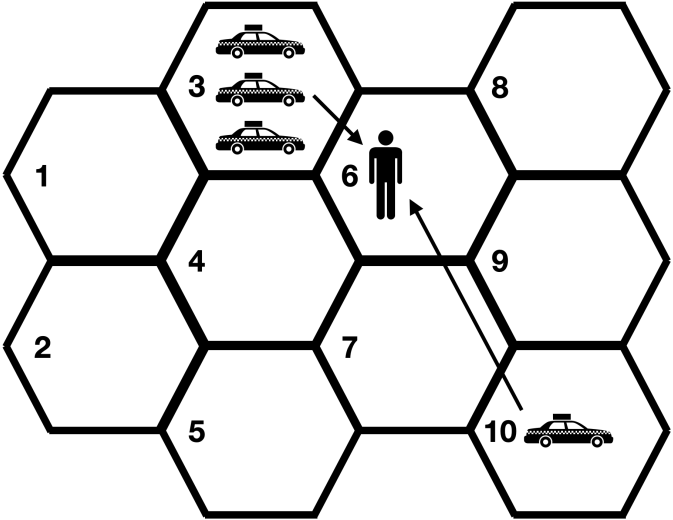
$$\sum_{j=1}^{n} a_{i,j} \leq 1,\ i = 1,\cdots,m \qquad \text{Driver assigned to at most one order}$$

- 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

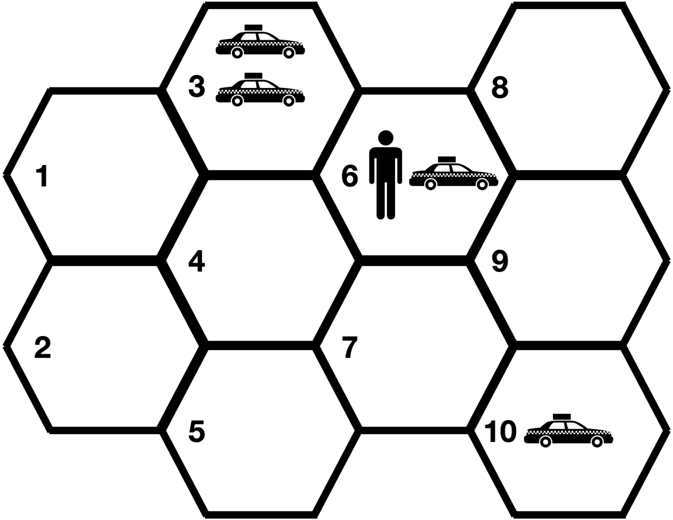
Closest Driver Policy: Limitations

- 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

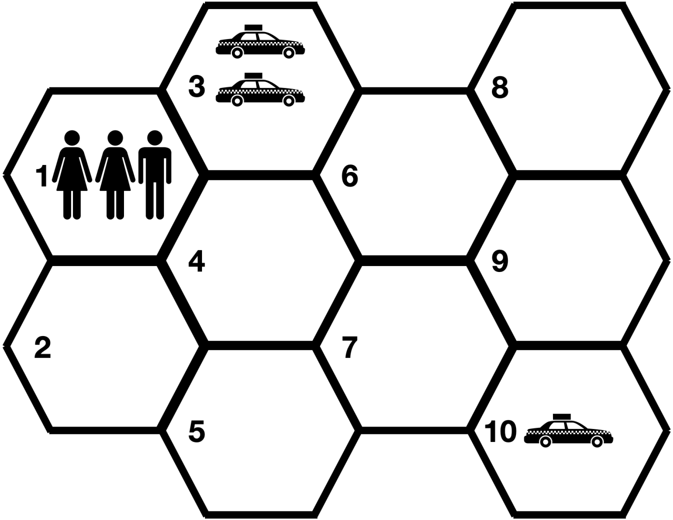
Illustration of Limitations of Closest Driver Policy



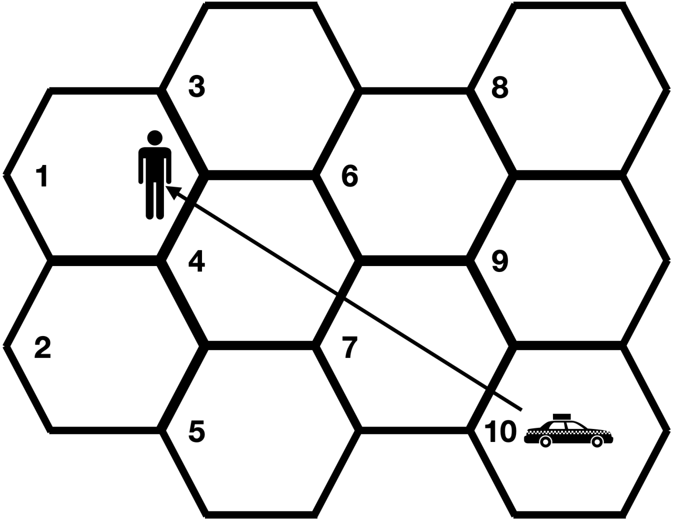
Adopting the Closest Driver Policy



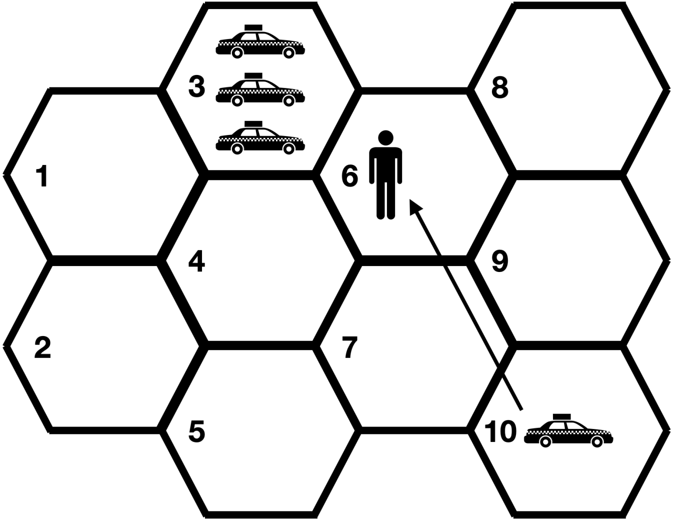
Some Time Later · · ·



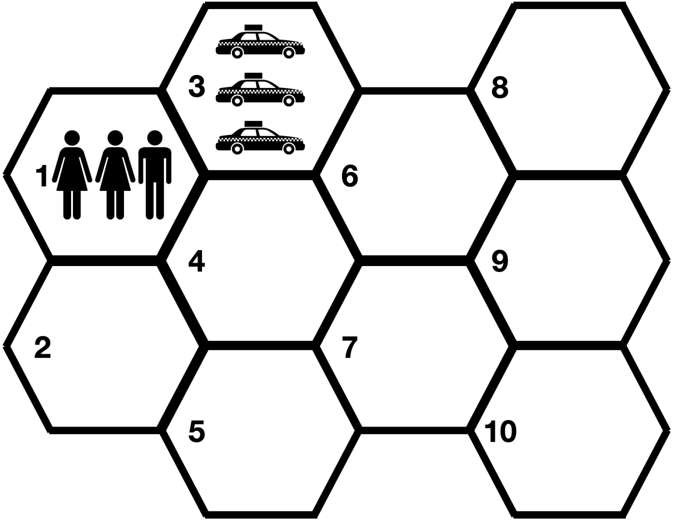
Miss One Order



Consider a Different Action



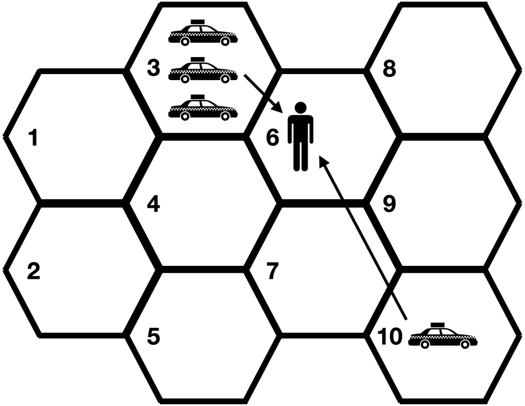
Able to Match All Orders



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 $\boldsymbol{R}$

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

- **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**: $\mathbf{10 + 0.9 \times 10 + 0.9^2 \times 10 = 27.1}$

Learning: Policy Evaluation

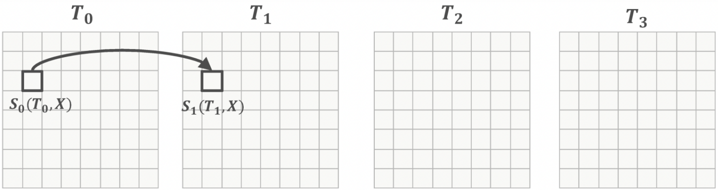
- Break down historical data into a set of transitions pairs $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'}, \Delta \boldsymbol{t})$, where $\Delta \boldsymbol{t}$ denotes the time of pickup, waiting and deliver process
- TD update rule for the **idle** action

$$\boldsymbol{V}(\boldsymbol{s}) \leftarrow \boldsymbol{V}(\boldsymbol{s}) + \alpha[\mathbf{0} + \gamma \boldsymbol{V}(\boldsymbol{s'}) - \boldsymbol{V}(\boldsymbol{s})]$$

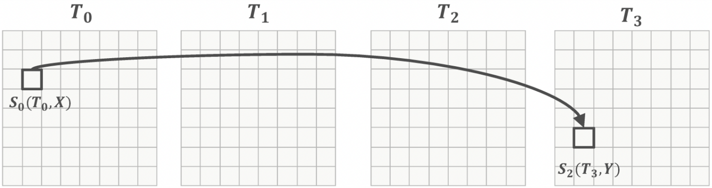
- TD update rule for the **serving** action

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

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 $D = \{(s, a, r, s', \Delta t)\}$ where each state is composed of a time and space index
- **Initialize** $V(s)$ and $N(s)$ to zero for any s
- **For** $t = T - 1$ to 0 **do**

  Find a data subset D_t where the time index of the state is t

  For each sample $(s, a, r, s', \Delta t)$ in D_t **do**

  $N(s) \leftarrow N(s) + 1$

  $V(s) \leftarrow V(s) + N^{-1}(s)[r + \gamma^{\Delta t} V(s') - V(s)]$

  End For

  End For
- **Return** V

Planning: Order Dispatch

Recall the closest driver policy

$$\begin{aligned}
&\arg\min_{a_{i,j}} \sum_{i=0}^{m}\sum_{j=1}^{n} d(i,j)a_{i,j} && \text{Minimize driver-passenger total distance}\\
&s.t. \sum_{i=1}^{m} a_{i,j} \leq 1,\ j = 1,\cdots,n && \text{Order assigned to at most one driver}\\
&\sum_{j=0}^{n} a_{i,j} \leq 1,\ i = 1,\cdots,m && \text{Driver assigned to at most one order}
\end{aligned}$$

- 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

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

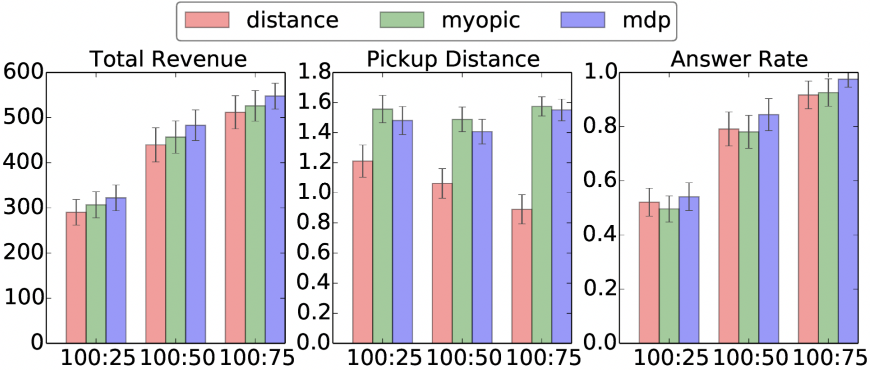
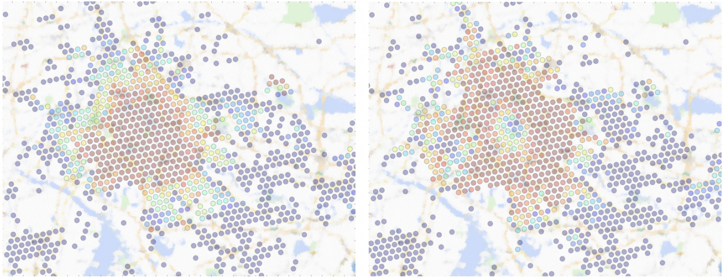


Figure 6: Comparison of distance-based method, myopic method and the proposed MDP method in three metrics on the toy example environment. X-axis stands for the order-driver ratios. Better viewed in color.

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

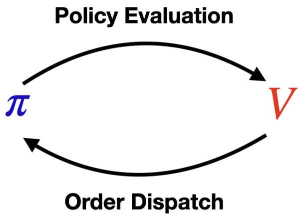
Real-World Experiment (Cont'd)



(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.

Extension: Policy Iteration



$$\pi_0 \longrightarrow V^{\pi_0} \longrightarrow \pi_1 \longrightarrow V^{\pi_1} \longrightarrow \cdots$$

- **Policy Evaluation**: evaluate the value under a given policy π
- **Order Dispatch**: implement the order dispatch policy based on $\boldsymbol{V}$ for data collection

Extension: Function Approximation

- Bellman equation:

$$V(S_i) = \mathbb{E}\Big[\, R_i(\gamma) + \gamma^{\Delta t_i} V_{k-1,t}(S_i') \Big|\, S_i \Big]$$

- Use **fitted value iteration** (similar to fitted Q-iteration) to allow function approximation
  - Use previous estimate to construct the target
  - Update the value using supervised learning
- Repeat for $k = 1, 2, \cdots$

$$V_k = \arg\min_{V} \sum_{i \in \mathcal{D}} \Big[R_i(\gamma) + \gamma^{\Delta t_i} V_{k-1}(S_i') - V(S_i) \Big]^2$$

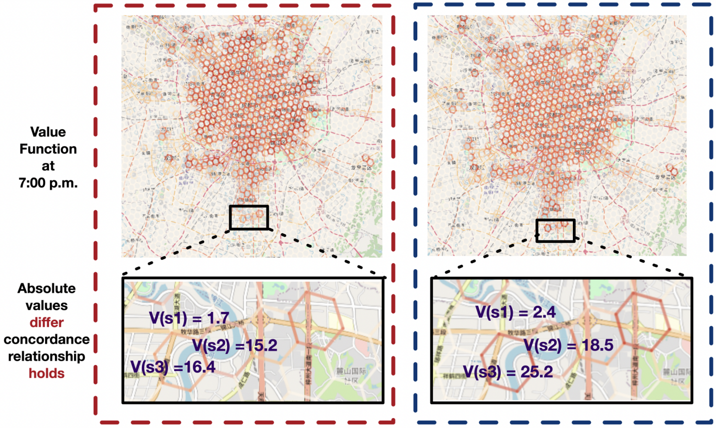
- VNet [Tang et al., 2019]: Combine fitted value iteration with deep value-network

Extension: Pattern Transfer Learning

- **Motivation**: violation of **time-homogeneity** assumption in data collected from ridesharing platforms, leading to TMDPs
  - The system dynamics is likely to vary over time
- **Naive solution**:
  - Use more **recent** data for policy evaluation (learning)
  - Use advantage function trick for order dispatching (planning)
  - **Disadvantage**: discard a lot of data
- **Research question**: how should we efficiently utilize historical dataset to improve the efficiency of value function estimation

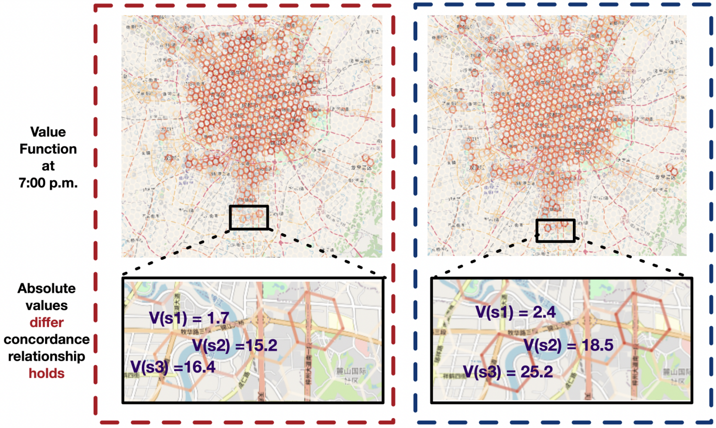
Nonstationarity

- Value function estimated based on data from [KDD CUP 2020]
- 30-day's data collected from Didi Chuxing
- Left plot: value based on first 15-day's data
- Right plot: value based on last 15-day's data
- Absolute values differ



Main Idea [Wan et al., 2021]

- Magnitude of value is nonstationary
- **Concordance** relationship of value remains stationary
- Values of **hot** zones (e.g., centers) are consistently larger than those of **cold** zones (e.g., suburbs)
- Overall, concordance relationship holds on more than 80% state pairs



Concordance

- Widely used in the statistics and economics literature
  - Maximum rank correlation estimator for regression [Han, 1987]
  - Concordance-assisted estimator for learning optimal dynamic treatment regimes [Fan et al., 2017, Shi et al., 2021]

- For two states s_1 and s_2 and two value functions V_1 and V_2
  - Concordance is 1 if $\{V_1(s_1) - V_1(s_2)\}\{V_2(s_1) - V_2(s_2)\} \geq 0$ and 0 otherwise

- Concordance penalty:

$$c(V_1, V_2) = \frac{1}{n(n-1)} \sum_{i<j} \#[\{V_1(S_i) - V_1(S_j)\}\{V_2(S_i) - V_2(S_j)\} < 0]$$

Algorithm

- Use past data to learn $\boldsymbol{V}^{old}$
- Use more recent data to learn $\boldsymbol{V_0}$ as an initial estimator
- Use **fitted value iteration** to update value
- Solve a **constrained optimisation** to incorporate concordance penalty
- Repeat for $\boldsymbol{k} = \mathbf{1}, \mathbf{2}, \cdots$

  Repeat for $\boldsymbol{t} = \mathbf{0}, \mathbf{1}, \cdots$

$$\boldsymbol{V_{k,t}} = \arg\min_{\boldsymbol{V_t}} \sum_{\boldsymbol{i} \in \mathcal{D}(\boldsymbol{t})} \left[\boldsymbol{R_i}(\gamma) + \gamma^{\boldsymbol{\Delta t_i}} \boldsymbol{V_{k-1,t}}(\boldsymbol{S_i'}) - \boldsymbol{V_t}(\boldsymbol{S_i}) \right]^2$$

$$s.t. \quad \boldsymbol{c}(\boldsymbol{V_t^{old}}, \boldsymbol{V_{k,t}}) \leq \epsilon$$

for some $\mathbf{0} < \epsilon < \mathbf{1}$.

Simulation

- Build dispatch simulator using the KDD dataset

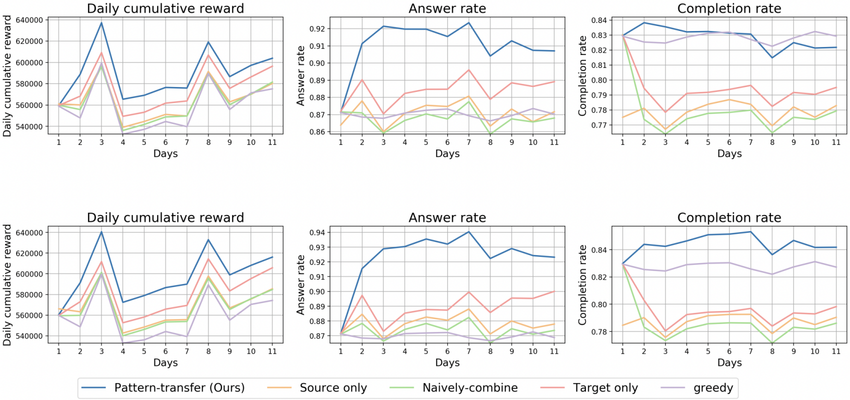


Figure 2: Performance of different methods when $\gamma = 0.9$ (upper) and $\gamma = 0.95$ (lower). The x-axis represents consecutive weekdays in the target environment. Our method outperforms the baseline methods under different metrics.

Summary

- Deep Q-Network
- Experience Replay
- Target Network
- Convolutional Neural Networks
- Minibatch Stochastic Gradient Descent
- Closest Driver Policy
- MDP Order Dispatch
- Advantage Function Trick
- Fitted Value Iteration
- Pattern Transfer Learning

References I

Caiyun Fan, Wenbin Lu, Rui Song, and Yong Zhou. Concordance-assisted learning for estimating optimal individualized treatment regimes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 79(5):1565–1582, 2017.

Aaron K Han. Non-parametric analysis of a generalized regression model: the maximum rank correlation estimator. *Journal of Econometrics*, 35(2-3):303–316, 1987.

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.

Chengchun Shi, Rui Song, and Wenbin Lu. Concordance and value information criteria for optimal treatment decision. *The Annals of Statistics*, 49(1):49–75, 2021.

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