Scalable Ride-hailing using Reinforcement Learning (Team 2)

Wangwei Wu, Yucheng Chen

May 17, 2021

ORIE 6590 Final Presentation

Performance difference equality for the MDP

Lemma 1. *We consider two policies π_θ and π_ξ, $\theta, \xi \in \Theta$. Their value functions satisfy*

$$V_\theta(s_{1,1}) - V_\xi(s_{1,1}) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=1}^{H}\sum_{i=1}^{I_t} A_\xi(s_{t,i}, a_{t,i})\right].$$

$$A_\pi(s_{t,i}, a_{t,i}) := \begin{cases} c(s_{t,i}, a_{t,i}) + V_\pi(s_{t,i+1}) - V_\pi(s_{t,i}), & \text{if } i \neq I_t \\ c(s_{t,i}, a_{t,i}) + \sum_{y \in S} \mathcal{P}(s_{t,i}, a_{t,i}, y) V_\pi(y) - V_\pi(s_{t,i}), & \text{if } i = I_t, \end{cases}$$

PPO framework

Collect Trajectories on old policy with Monte Carlo simulation

---

Algorithm 1: The PPO algorithm

Result: policy π_{θ_J}

Initialize policy function π_{θ_0} and value function approximator $V_{\psi_{-1}}$;

for *policy iteration* $j = 1, 2, \ldots, J$ **do**

- Run policy $\pi_{\theta_{j-1}}$ for K episodes and collect dataset (3.3).
- Construct Monte-Carlo estimates of the value function $V_{\theta_{j-1}}$ following (3.5).
- Update function approximator V_ψ minimizing (3.6).
- Estimate advantage functions $\hat{A}(s_{t,i,k}, a_{t,i,k})$ by (3.7).
- Maximize surrogate objective function (3.4) w.r.t. θ. Update $\theta_j \leftarrow \theta$

end

$$D_\xi^{(K)} := \left\{ \left(\left(s_{t,1,k}, a_{t,1,k}, \hat{A}(s_{t,1,k}, a_{t,1,k}) \right), \cdots, \left(s_{t,I_{t,k},k}, a_{t,I_{t,k},k}, \hat{A}(s_{t,I_{t,k},k}, a_{t,I_{t,k},k}) \right) \right)_{t=1}^{H} \right\}_{k=1}^{K}, \qquad (3.3)$$

PPO framework

Estimate Value function

Algorithm 1: The PPO algorithm

Result: policy π_{θ_J}

Initialize policy function π_{θ_0} and value function approximator $V_{\psi_{-1}}$;
for *policy iteration* $j = 1, 2, \ldots, J$ **do**
  Run policy $\pi_{\theta_{j-1}}$ for K episodes and collect dataset (3.3).
  Construct Monte-Carlo estimates of the value function $V_{\theta_{j-1}}$ following (3.5).
  Update function approximator V_ψ minimizing (3.6).
  Estimate advantage functions $\hat{A}(s_{t,i,k}, a_{t,i,k})$ by (3.7).
  Maximize surrogate objective function (3.4) w.r.t. θ. Update $\theta_j \leftarrow \theta$
end

$$\hat{V}_{t,i,k} := \sum_{j=i}^{I_{t,k}} c(s_{t,j,k}, a_{t,j,k}) + \sum_{\ell=t+1}^{H} \sum_{j=1}^{I_{\ell,k}} c(s_{\ell,j,k}, a_{\ell,j,k}),$$

PPO framework

Update function approximator

Algorithm 1: The PPO algorithm

Result: policy π_{θ_J}

Initialize policy function π_{θ_0} and value function approximator $V_{\psi_{-1}}$;

for *policy iteration* $j = 1, 2, \ldots, J$ **do**

  Run policy $\pi_{\theta_{j-1}}$ for K episodes and collect dataset (3.3).

  Construct Monte-Carlo estimates of the value function $V_{\theta_{j-1}}$ following (3.5).

  Update function approximator V_ψ minimizing (3.6).

  Estimate advantage functions $\hat{A}(s_{t,i,k}, a_{t,i,k})$ by (3.7).

  Maximize surrogate objective function (3.4) w.r.t. θ. Update $\theta_j \leftarrow \theta$

end

$$\sum_{k=1}^{K} \sum_{t=1}^{H} \sum_{i=1}^{I_{t,k}} \|V_\psi(s_{t,i,k}) - \hat{V}_{t,i,k}\|^2.$$

PPO framework

Estimate Advantage Function

Algorithm 1: The PPO algorithm

Result: policy π_{θ_J}

Initialize policy function π_{θ_0} and value function approximator $V_{\psi_{-1}}$;

for *policy iteration* $j = 1, 2, \dots, J$ **do**

- Run policy $\pi_{\theta_{j-1}}$ for K episodes and collect dataset (3.3).
- Construct Monte-Carlo estimates of the value function $V_{\theta_{j-1}}$ following (3.5).
- Update function approximator V_ψ minimizing (3.6).
- Estimate advantage functions $\hat{A}(s_{t,i,k}, a_{t,i,k})$ by (3.7).
- Maximize surrogate objective function (3.4) w.r.t. θ. Update $\theta_j \leftarrow \theta$

end

Next, we obtain the advantage function estimates

$$\hat{A}(s_{t,i,k}, a_{t,i,k}) := \begin{cases} c(s_{t,i,k}, a_{t,i,k}) + V_\psi(s_{t,i+1,k}) - V_\psi(s_{t,i,k}) & \text{if } i \neq I_{t,k}, \\ c(s_{t,i,k}, a_{t,i,k}) + V_\psi(s_{t+1,1,k}) - V_\psi(s_{t,i,k}) & \text{otherwise,} \end{cases} \tag{3.7}$$

PPO framework

Maximize Surrogate Objective, Update Policy Net

Algorithm 1: The PPO algorithm

Result: policy π_{θ_J}

Initialize policy function π_{θ_0} and value function approximator $V_{\psi_{-1}}$;

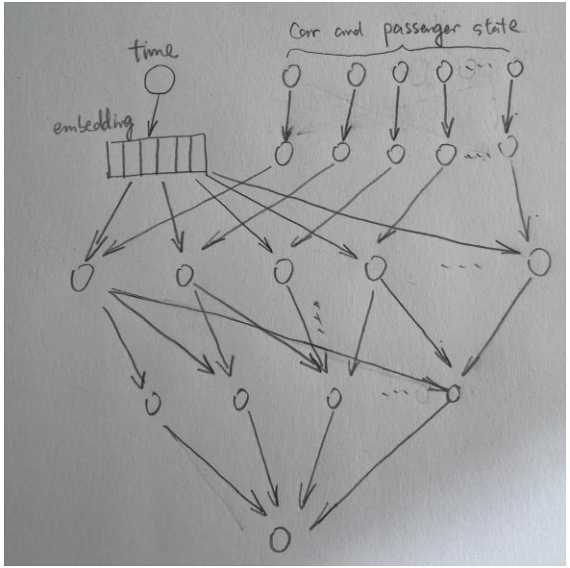
for *policy iteration* $j = 1, 2, \ldots, J$ **do**

- Run policy $\pi_{\theta_{j-1}}$ for K episodes and collect dataset (3.3).
- Construct Monte-Carlo estimates of the value function $V_{\theta_{j-1}}$ following (3.5).
- Update function approximator V_ψ minimizing (3.6).
- Estimate advantage functions $\hat{A}(s_{t,i,k}, a_{t,i,k})$ by (3.7).
- Maximize surrogate objective function (3.4) w.r.t. θ. Update $\theta_j \leftarrow \theta$

end

$$\hat{L}(\theta, \xi, D_\xi^{(K)}) := \frac{1}{K} \sum_{k=1}^{K} \Bigg[\sum_{t=1}^{H} \sum_{i=1}^{I_{t,k}} \min\Big(r_{\theta,\xi}(s_{t,i,k}, a_{t,i,k}) \hat{A}_\xi(s_{t,i,k}, a_{t,i,k}), \quad \mathrm{clip}(r_{\theta,\xi}(s_{t,i,k}, a_{t,i,k}), 1-\epsilon, 1+\epsilon) \hat{A}_\xi(s_{t,i,k}, a_{t,i,k}) \Big) \Bigg]. \tag{3.4}$$

Value function approximators



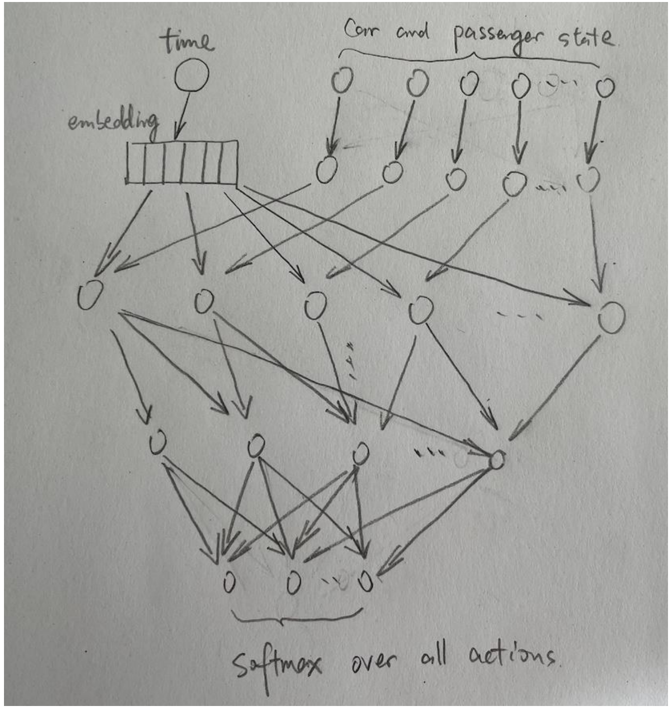
1. Fully connected neural network with embeddings

1. Input layer includes time, car state and passenger state. Time has very different scale than the other two.

1. Estimate the state value with Monte-Carlo method

1. Use Adm as optimizer and Mean squared error as the loss function to train the neural network

Policy Network



1. Fully connected neural network with embeddings

1. Input layer includes time, car state and passenger state. Time has very different scale than the other two.

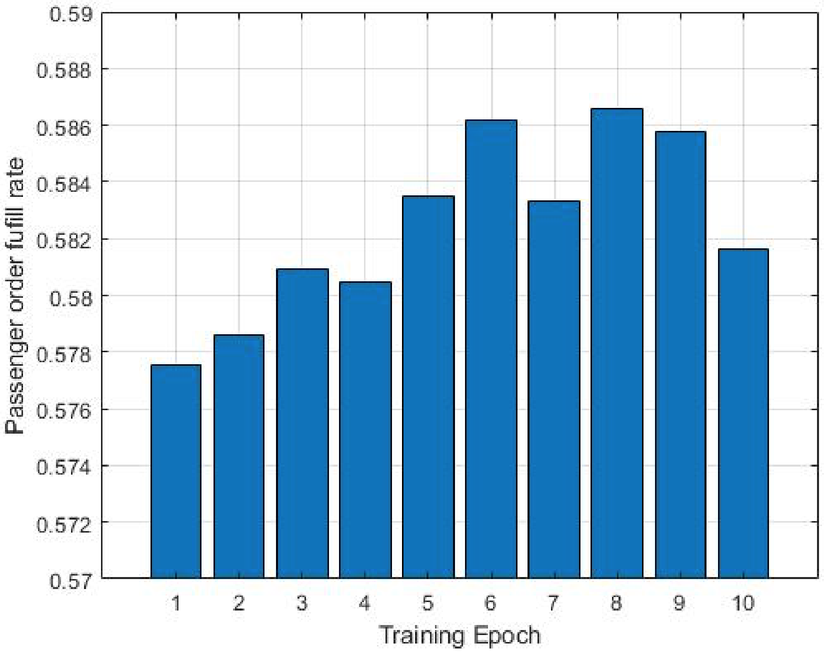
1. Output is a distribution over entire action space

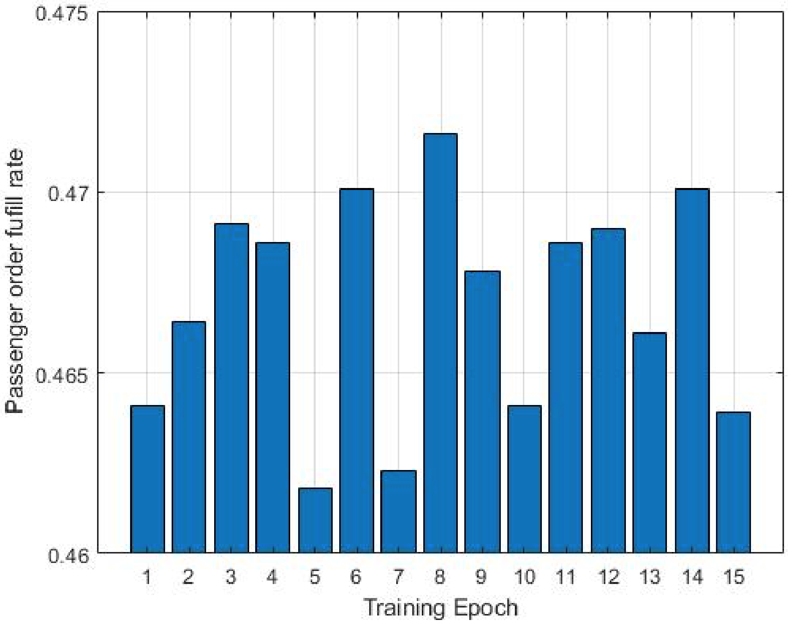
1. Use Adm as optimizer and objective is specified by (3.4)

Key hyper-parameters

Key hyper-parameters	Feng et. al's implementation	Our implementation
# of episodes per training iteration	300	25
Clipping epsilon (with decay)	0.2	0.05
# of passes for value net training	10	20
# of passes for policy net training	3	3
(Initial) learning rate for policy net training	0.00005	0.00001

Numerical results





Future work

The atomic action space is linear in R^2, where R is the number of grids considered in the whole area.

Consider a multi-agent setting where we take each grid as an agent so that each agent can have a R-dimensional action space (way smaller than the original model)