DYNAMIC FLEET MANAGEMENT VIA DEEP REINFORCEMENT LEARNING

Yichen Ruan
yichenr@andrew.cmu.edu

Jinhang Zuo
jzuo@andrew.cmu.edu

Jakob Cassiman
jcassima@andrew.cmu.edu

Xiaoyi Duan
xiaoyid@andrew.cmu.edu

ABSTRACT

In this project, we use deep reinforcement learning to solve the dynamic vehicle dispatching problem for ride-hailing fleet networks. We explore two different Deep Q-Network (DQN) frameworks: CNN-based DQN and Diffusion Convolution (DC) based DQN. Compared to CNN-based DQN, DC-based DQN captures more specific graph-based properties in road networks. We implement both frameworks using the same traffic simulator. We select one week of the NYC taxi trip record data to evaluate their performance. Experiment results show that the DC-based DQN converges faster than the CNN-based DQN. It also works better in terms of waiting time during rush hours, and exhibits a different distribution pattern of the idle cruising time.

1 INTRODUCTION

Ride hailing is an efficient way for people to travel between locations. In ride-hailing fleet networks, such as Uber, Lyft and taxi, drivers cruise vehicles on the street and head to locations where pickup requests appear. Although drivers can move to regions with higher current demand to reduce their idle cruising times with the help of apps, they cannot predict future pickup demand heat maps of regions to optimize their cruising strategies. Also, it is difficult to find a global optimal solution to coordinate thousands of drivers due to the real-time change of passenger demands. Moreover, the uncertainty of future demand and vehicle trip time makes it harder to model and make dispatch decisions.

In this project, we use deep learning, especially deep reinforcement learning techniques, to learn the optimal vehicle dispatch policy for ride-hailing fleet networks. We use Deep Q-Network (DQN) introduced by Mnih et al. (2015), a well-known reinforcement learning framework where Q-functions are modeled by neural networks. We explore two different DQN frameworks for the dynamic fleet management problem: CNN-based DQN and Diffusion Convolution (DC) based DQN. We first study on MOVI, a CNN-based DQN framework for vehicle dispatch management proposed by Oda & Joe-Wong (2018). It uses CNN layers to extract features from the demand and supply maps to calculate the Q-value for each possible action. Although such a CNN-based DQN achieves fairly good performance in terms of rejection rate and average waiting time, we still want to find better frameworks than simple CNN-based DQN. The reason is that CNN scans a square of parameters across the grid-structured input, but it cannot fully capture the graph-based properties in road networks (see section 3.2). As a result, we introduce the diffusion convolution technique that was proposed by Atwood & Towsley (2016). Diffusion convolution builds a latent representation by scanning a diffusion process across each node in a graph-structured input. Compared to CNN, it is suitable for non-Euclidean, directional graph-based architectures such as road networks. Based on Li et al. (2017) and Oda & Tachibana (2018), we build up a DC based DQN framework with the same reward function and vehicle-customer matching policy as MOVI so that we can have fair performance comparison between these two frameworks.

We implement both CNN-based DQN and DC-based DQN using the same traffic simulator from Oda (2018) to compare their performance. We use NYC taxi trip record data as the dataset, which includes pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, etc. For our evaluation, we use the trip records from Monday 6/6/2016 to Sunday, 6/12/2016. Experiment results show that: the DC-based DQN converges much faster than the CNN-based DQN; the DC-based DQN works better in terms of waiting time during rush hours; the total amount of revenue for drivers is relatively fixed whatever which DQN is used; average cruising time is similar for both DQNs, but the DC-based method could be unfair for drivers, i.e., some drivers benefits more while some do not.

2 Related Work

In ride-hailing fleet networks (e.g., Uber), drivers cruise vehicles on the street and head to locations where pickup requests appear. A dynamic fleet management should proactively dispatch vehicles to locations where pickup requests are likely appeared in the future. In order to reduce drivers' idle cruising times or passengers waiting times, many previous works on fleet management use model-based or model-free methods to address this vehicle dispatch problem.

Oda & Joe-Wong (2018) propose MOVI, a model-free Deep Q-network (DQN)-based framework that learns the optimal dispatch policy. Since the complexity of the DQN grows exponentially with the number of dispatches, MOVI takes a distributed approach: it streamlines the DQN training and let each individual vehicle independently learn its own optimal policy. It builds a large-scale realistic simulator based on 15 million taxi trip records and shows that the DQN dispatch policy outperforms a centralized receding-horizon control (RHC) policy proposed by Miao et al. (2016).

Lin et al. (2018) also consider the large-scale fleet management problem, but they use multi-agent reinforcement learning algorithms. They propose contextual deep Q-learning and contextual multi-agent actor-critic to tackle this problem. They also build a traffic activity simulator to evaluate the their algorithms. The framework is however incompatible with MOVI which only supports single-agent dispatching. We will defer the implementation of multi-agent algorithms for future works.

Zheng & Wu (2017) study the urban taxi dispatching problem and model the interests of passengers and taxi drivers. They propose a stable marriage approach to dispatch taxis. Inspired by their work, we can also introduce the interests of passengers to MOVI, since now it only considers the feedback from drivers' side. Qu et al. (2014) also take a vehicle-centric approach as MOVI, but provide route recommendations that aim to maximize individual drivers profits. Drivers profits can also be related to the reward function of Q learning. Miao et al. (2016) design a centralized Receding Horizon Control (RHC) framework, which serves as the baseline in MOVI. However, their framework is model-based, so cannot really capture the uncertainty of fleet management problem.

3 Methods

3.1 CNN-based Deep Q-Network

In fleet networks, the reward function for a vehicle n is given by:

$$r_t^{(n)} = \sum_{t'=t-\delta}^{t} \lambda b_{t'}^{(n)} - c_{t'}^{(n)}, \tag{1}$$

which is the weighted sum of the number of rides, $b_{t'}^{(n)}$ and the total dispatch time, $c_{t'}^{(n)}$. The reward function is not directly expressed as a function of the actions because the relation between actions and reward will be learned by the network. Given the reward function for vehicles, the Q-value function can be written as:

$$Q^*(s,a) = \max_{\pi} \mathrm{E}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}^{(n)}] \tag{2}$$

In the DQN for fleet management problem, the state means the current location and occupancy of each vehicle, while the action is where the vehicle should approach to pick up customers. Deep neural networks are used to model the relationship between input data and output Q-values. For CNN-based DQN, it uses a deep convolutional net as displayed in Figure 1. The input to this DQN includes the demand map that predicts where future requests will appear and the supply map that indicates current locations and occupancy of all vehicles. Auxiliary features include current position of the input vehicle, current time, day of week, etc. (see Table 2 in Oda & Joe-Wong (2018) for more details) The output of the network is a map of Q values, which represents the long-term expected reward gain of moving to different locations. It can give us a probability distribution over all possible actions the taxi can take to maximize the collected reward at each time step. The loss is calculated by an estimate of the optimal Q-value as calculated with the current network:

$$L_i(\theta_i) = \mathrm{E}[(r_t + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \tag{3}$$

with θ_i^- the parameters from the previous iterations. That easily gives rise to unstable learning as the network is learning from its own predictions. Next to that, reinforcement learning with a non-linear approximator for the Q-function is also known to cause unstable conditions. This can be countered by leveraging RMSProp optimization and using an experience replay to remove correlations. Both like the original DQN-paper by Mnih et al. (2015) suggests. This DQN is distributed in the sense that every taxi agent in the simulation makes its own decisions unaware of the decisions being made by the other taxis. The only information shared is the current state of the environment.

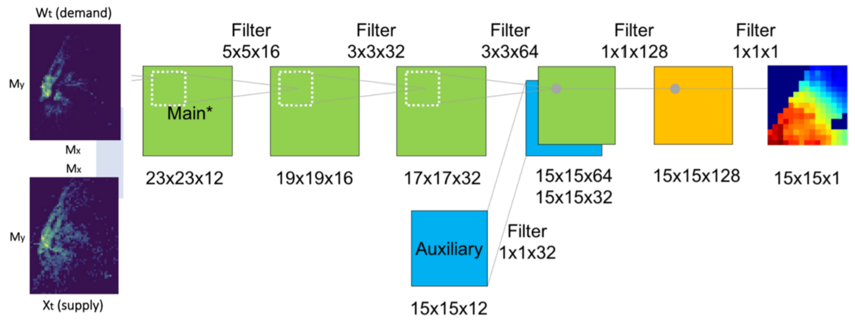

Figure 1: Architecture of CNN-based Q network

3.2 Diffusion Convolution-based Deep Q-Network

Convolutional neural networks (CNNs) scan a square of parameters across a grid-structured input as the convolution operation, while diffusion-convolutional neural networks (DCNNs) Atwood & Towsley (2016) is for graph-structured data by scanning a diffusion-convolution operation over every node. In contrast with standard CNNs, DCNN parameters are tied according to search depth rather than their position in a grid. Road network is a non-Euclidean and directional graph-based structure as shown in 2. Road segment $road$ 3 is close to $road$ 1 in distance but may not have similar speed or direction as $road$ 2.

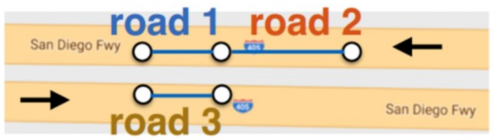

Figure 2: Road Network

The diffusion convolution-based q-network architecture is illustrated in Figure 3. The diffusion convolution (DC) operation learns latent representations of graph-structured data. The diffusion-convolutional activate Z_{tijk} for node i, hop k, and feature t of graph a is given by

$$Z_{aikt} = (W_{kt})^c \cdot \sum A^*_{aik} \tau_{ait} \tag{4}$$

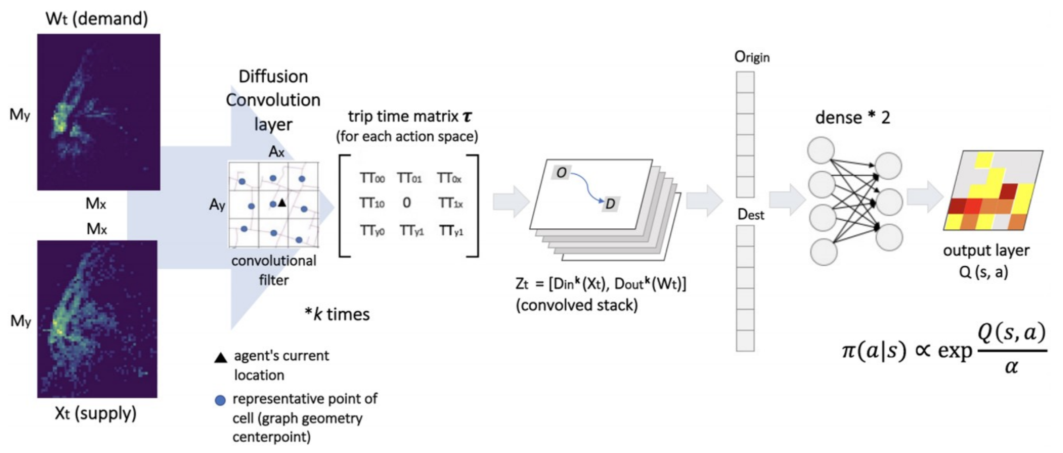
Wt (demand)
My
Mx
Mx
My
Xt (supply)
Diffusion Convolution layer
Ax
Ay
convolutional filter
*k times
▲ agent's current location
● representative point of cell (graph geometry centerpoint)
trip time matrix τ (for each action space)
TT00 TT01 TT0x
TT10 0 TT1x
TTy0 TTy1 TTy1
O
D
Zt = [Din^k(Xt), Dout^k(Wt)] (convolved stack)
Origin
Dest
dense * 2
output layer Q (s, a)
π(a|s) ∝ exp Q(s,a)/α

Figure 3: Diffusion convolution-based q-network architecture

Our input matrices are demand and supply graph, W_t and X_t. For each center node in the grids, the action apace is 15 from the node in the center. Therefore, the convolutional filter for every node is a 15 matrix A with the node in center. The feature map τ for each node is a 15 trip time matrix. All features pass through a DC layer where graph diffusion convolution is carried out k times with a filter of dimensions $M_x \times M_y \times A_x \times A_y$ Each node in W_t and X_t is transformed to a diffusion-convolutional representation, which is $k \times t$ matrix defined by k hops of graph diffusion over t features. The feature vectors from output Z of demand matrix and supply matrix represent origin and destination. Input them in fully-connected layers and output action values $Q(s, a)$ for an action a. In addition to Z_t, we also feed into the FC layer squeezed auxiliary features as in the baseline model (see Figure 1).

4 Experiments and Results

This section will provide a brief description of the experimental results. It will start with a brief overview of the data set and framework used. Next, it will continue with plots of the most important graphs related to the problem setting. The analysis and of these results follows in section 5.

4.1 Data Set

We used the simulator and the data of NYC Taxi and Limousine Commission taxi trip records from May and June 2016. The data set contains the pick-up and drop-off dates/times, pick-up and drop-off locations, and travel distances, real time rates etc. For our evaluation, we use the trip records from Monday 6/6/2016 to Sunday, 6/12/2016.

4.2 Simulator

The environment will provide rewards and respond to the actions that the algorithm takes. At each time step the simulator will create ride requests that will be matched to taxis by the Matching Policy. The Matching Policy is a simple greedy policy that takes the closest taxi available. If matched, the taxi calculates the route to the destination and how long it will be busy acting out this action before becoming idle again.

A second policy that interacts with the simulated environment is the Dispatch Policy. The task of this learned algorithm is to predict where future requests will happen and to dispatch taxis proactively to make sure that it maximizes the reward defined above. The entire setup happens in a single agent so there is no explicit competition between the different taxis. Every taxi makes its decision without knowledge of the actions that others choose except from the current demand and supply heat maps. There is one common Q-network that is cloned and used by each taxi separately.

The demand and supply inputs to the network will be generated by the simulator at every step. For the DC based model, it also computes the estimated trip time maps. All input maps are divided into 51 by 51 grids, i.e. there will be a total of 2601 different locations. In order to reduce the searching space, we limit the maximum distance a vehicle can move to 7, resulting in a 15 by 15 action space.

4.3 Configuration

Here we list the parameters we used in the experiment. Unless specifically mentioned, two models share the same settings.

- Trained for a total of 20,000 steps and used a replay memory of the 10,000 most recent transitions.
- The original ϵ greedy value is 1.0, and decays linearly every 3,000 steps all the way to 0.1.
- A batch size of 128 experience records randomly drawn from the replay memory.
- RMSProp Optimizer with learning rate equals 0.00025, and momentum equals 0.95.
- Stabilizing learning with the double Q learning technique, updating the target network every 50 steps.
- The discount coefficient γ for future rewards equals 0.98.
- As per Oda & Joe-Wong (2018), for the baseline model, in the first 5000 steps, only α of vehicles are allowed to move, where α increases linearly from 0.7 to 1.0.

4.4 Results

The following graphs compare the results for the two Q-network structures: the CNN-based and the DC-based.

Figure 4 shows the evolution of the average maximum Q-value during training of the CNN-based network. Figure 5 is the plot for the counterpart network. The DC-based model reached 16 at around 600 steps and started decreasing, while CNN-based model reached 14 at around 700 steps.

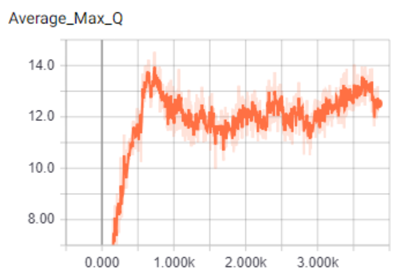

Figure 4: The average maximum Q-value for CNN Q-network

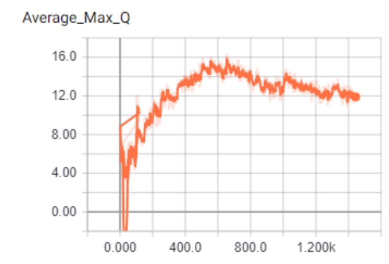

Figure 5: The average maximum Q-value for DC Q-network

In Figure 6 the waiting time is plotted per hour for the CNN-network. When comparing to Figure 7, which is the plot for the DC-network, the two peaks around the rush hours are similar.

In Figures 8 and 9, the two models are compared on the metrics of revenue and cruising time. The orange color represents the CNN network and the blue the DC-network. Where both graphs overlap shows grey. The two graphs show a lot of similarity between both networks and will be discussed further in Section 5.

5 Discussion

We have successfully implemented the CNN-based baseline model and the diffusion convolution model. The outputs of the baseline model is consistent with that reported in the original paper (Oda

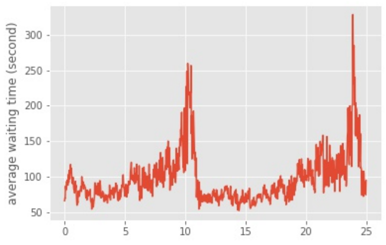

Figure 6: The waiting per hour for CNN-based Q-network

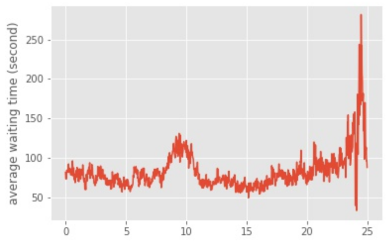

Figure 7: The waiting per hour for DC-based Q-network

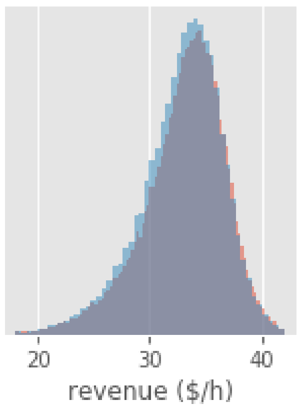

Figure 8: The distribution of the revenue per hour. Orange represents CNN, blue represents DC.

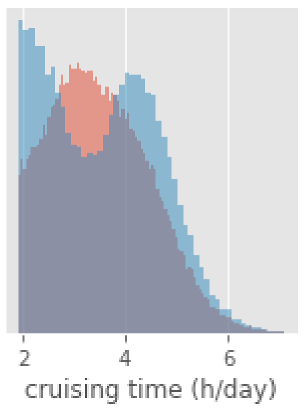

Figure 9: The distribution of the idle cruising time per day. Orange represents CNN, blue represents DC.

& Joe-Wong (2018)). In this section, we will compare their performance using metrics defined above, and provide insights into future works.

5.1 CONVERGENCE

As is shown in Figure 4 and 5, the DC model converges much faster than the CNN based baseline. DC's maximum Q value stays stable after around 400 steps, yet the CNN model takes roughly 1000 steps to converge. This phenomenon is to be expected, as the former uses significantly less parameters for training. Note that the diffusion filter is built directly from the demand and supply map, so the only learnable parameters come from the dense layer. In the meanwhile, we also observed a slightly shorter (10%) wall clock training time for the DC model. This makes sense as diffusion operation requires less matrix calculation. Thus, for training efficiency consideration, DC model may be preferred. This property is desirable when applied in an online learning scenario.

5.2 PERFORMANCE

As is discussed in Section 3, we evaluate the performance of a dispatching policy using three metrics: users' waiting time; drivers' revenue and idle cruising time.

- **Waiting time:** Figures 6 and 7 show the waiting time in a typical day. We can clearly see two peaks from both figures, those correspond to the late peak (18 PM) and early peak (8 AM). Compared to the baseline, the DC model significantly reduced the waiting time in rush hours. At the late peak, waiting time reduces from 250 seconds to 125 seconds (50% reduction), and morning waiting time drops from 300 seconds to 250 seconds (16%

reduction). However, for other time, two curves do not have much difference. This can be explained as follows: during low-traffic hours, the taxi's supply is greater than the users' demand, thus any reasonable dispatching policy will keep the waiting time low at certain level. As there comes more traffic, the DC model can more sensitively sense the change of user demand, thus the waiting time increases relatively slower than the baseline model. In rush hours, DC model clearly outperforms the baseline. And since there are more vehicles to dispatch at the late peak, this improvement is more significant in the afternoon.

- **Revenue:** Applying DC does not help increasing the drivers' revenue. As is illustrated in Figure 8, two curves are almost overlapping. This can be explained as the taxi supply is in general greater than the users' demand. Thus, the total amount of revenue is relatively fixed whatever dispatching policy is used.
- **Cruising time:** Figure 9 compares the distribution of taxi's idle cruising time. While the baseline yields a single peak pattern where most taxi stay idle for 3.5 hr/day, there exists two peaks for the DC model. Some taxi will now have shorter cruising time ($\leq$ 3 hr), however, some will get idle for even longer ($\geq$ 4 hr). As a result, the mean cruising time actually stays the same at around 3.5 hours. Thus, when it comes to the cruising time, both models have the same performance. One relevant observation is that the DC model prefers short-distance dispatching actions, thus, it is possible that taxi in low-demand valleys are not get moved for efficiency consideration. Though the average cruising time is similar, one may argue that the DC model is more or less unfair for drivers, i.e., some drivers benefits more while some do not. This can be fixed if we also incorporate fairness into the reward definition (1).

5.3 Tuning the networks

- **Maximum move:** Under the current setting, a vehicle can move up to 7 steps from its current position, resulting in a 15 by 15 state-action table. In the experiment, we find most dispatching actions tend to pick nearby destinations. Further expanding the action space does not improve the performance.
- **Auxiliary features:** For the baseline model, auxiliary features are necessary as the main input (demand and supply) does not contain vehicle specific features. This is fine for the DC model whose diffusion filter differs with respect to vehicles. But we do find time and location relevant auxiliary features can help accelerate the convergence.
- **Reward discount:** As in conventional reinforcement learning, we used a γ coefficient to shrink the weight of future rewards. In our experiment, we find relatively greater γ (0.98) helps stabilize the convergence of Q values. This is probably because the states (i.e., demand and supply) are likely to stay stable for most time of a day. Individual vehicle's action does not significantly change the environment as well. Thus, we can be more sure about future rewards.

REFERENCES

James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems 29*, pp. 1993–2001. 2016.

Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In *Large Scale Kernel Machines*. MIT Press, 2007.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting, 2017.

Kaixiang Lin, Renyu Zhao, Zhe Xu, and Jiayu Zhou. Efficient large-scale fleet management via multi-agent deep reinforcement learning. *arXiv preprint arXiv:1802.06444*, 2018.

F. Miao, S. Han, S. Lin, J. A. Stankovic, D. Zhang, S. Munir, H. Huang, T. He, and G. J. Pappas. Taxi dispatch with real-time sensing data in metropolitan areas: A receding horizon control approach. *IEEE Transactions on Automation Science and Engineering*, 13(2):463–478, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Takuma Oda. Fleet management simulation framework. `https://github.com/misteroda/fleet-sim`, 2018.

Takuma Oda and Carlee Joe-Wong. MOVI: A model-free approach to dynamic fleet management. In *INFOCOM*, 2018.

Takuma Oda and Yulia Tachibana. Fleet management simulation framework. `https://openreview.net/pdf?id=SkxWcjx09Q`, 2018.

Meng Qu, Hengshu Zhu, Junming Liu, Guannan Liu, and Hui Xiong. A cost-effective recommender system for taxi drivers. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 45–54. ACM, 2014.

H. Zheng and J. Wu. Online to offline business: Urban taxi dispatching with passenger-driver matching stability. In *IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017.