BIKE REDISTRIBUTION

A Project Report
Presented to
The Faculty of the College of
Engineering

San Jose State University
In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Software Engineering

By
Ashika Anand Babu
Anusha Gangasani
Ramya Mahesh
May 2023

TABLE OF CONTENTS

1. Executive Summary:

For many transit providers, including Citi Bike, rebalancing is a significant service bottleneck. Citi Bike relies on manual coordinations between dispatchers and field agents to rebalance bikes. The RL system maintains bike stock under control while requiring the least amount of resources.

2. Introduction:

Reinforcement learning can be used to solve strategic problems. The problem of Citibike redistribution can be framed in the context of a strategic game. Currently redistribution is done manually but to optimize the movement of vans, we could use reinforcement learning techniques such as Q-learning with forecasting, deep Q network models can be used.

Using an autonomous approach made possible by distributed reinforcement learning (DiRL) can result in cost savings in two ways: 1) better resource optimization with a timely and accurate representation of the system dynamics, and 2) a more reliable model as a result of the solution's ongoing adaptability to shifting system dynamics.

3. Problem Statement:

One of the major drawbacks of the largest American bike-share programs as of today is an unbalanced flow of bikes from station to station. This results in some stations filled with an overflow of bikes and others short of bikes. These issues cause the users inconvenience and have led to dockless bike-share programs, where the users are allowed to park their bikes anywhere. However, dockless bike-share programs have their own issues. Piles of bikes on the sidewalks and sometimes the users hide their bike for future use. This has motivated some of the companies like Citibike (New York) to currently maintain a close eye of the issue of evenly allocating bikes among their pick-up/drop-off locations. This comes with a price of using vans to continuously redistribute the bikes among their stations. The company has also begun experimenting with incentivizing users to move bikes from full stations to empty stations with a points-based reward system in its NYC-based BikeAngels4 program. Bike redistribution is one of the major problems of the bike-share programs.

4. Purpose/Motivation:

Currently, bike-redistribution is done manually by a team of over 100 employees, monitoring the stations from the company’s headquarters and operating the vans to redistribute the bikes on the ground. There are regular patterns in which the stations fill up. These patterns include particular time of the day, particular days, during certain types of weather and so on. The team working on bike redistribution is undoubtedly familiar with many of the intricacies of these systems. RL algorithms can be applied on these patterns to predict the rate of redistribution of bikes. This would greatly accelerate the successful redistribution of bikes within the network and eliminate the current bottlenecks.

5. Differentiator/Contribution:

Compared to the current situation in the market where the monitoring is done manually with a random estimation, this project focuses on using RL techniques for monitoring.

Some of the previous researches are, "Benchmarking Deep Reinforcement Learning for Continuous Control" aims to establish a benchmark for the assessment of reinforcement learning tasks by evaluating how well different algorithms perform on a range of tasks, including basic tasks (like the "cart-and-pole" problem), locomotion tasks, partially observable tasks, and hierarchical tasks. We were given a framework for performance evaluation and a road map for methodologies that could be employed to broaden the scope of our study by Duan et al.'s survey of the efficacy of several algorithmic approaches in reinforcement learning at solving a range of challenges.

Other works, such as Yu et al’s "Deep Reinforcement Learning for Simulated Autonomous Vehicle Control" and Mania et al’s "Simple Random Search Provides a Competitive Approach to Reinforcement Learning," offer insight into applications of deep-Q learning techniques in the physical environment. In an effort to teach an autonomous car to navigate a simulated environment, Yu et al. research a range of reward functions for deep-Q networks. Although these techniques aren't directly relevant,they do show how well deep reinforcement learning can be applied to real-world issues and how well they function with complex systems.

6. Methodology:

This solution includes three learning methods:

- Q-Learning
- Enhanced Q-Learning with Forecasting
- Deep Q-Learning with Neural Networks.

Reinforcement Learning

RL generally consists of an agent, a set of states, and a set of actions for each state. Agents transition from state to state and perform actions up to an end state. Actions in certain states bring rewards or penalties. The agent's overall goal is to maximize long-term rewards. This reward is the weighted sum of all expected future rewards from the current state. In the context of the Citi Bike Rebalancing problem, the state is the number of bikes in stock at the station. Actions are the number of bikes an agent can move in his hour. An agent can choose to move 0, 1, 3, or 10 bikes per hour. Rewards and penalties are structured as follows:

- -30 if bike stock falls outside the range [0, 50] at each hour
- -0.5 times the number of bikes removed at each hour
- +20 if bike stock in [0, 50] at 23 hours; else -20

This reward structure encourages the agent to keep the bike stock within an acceptable range while moving as few bikes as possible.

Q-Learning

QLearning is the most basic and traditional learning method for learning associations between actions in a given state and weighted future rewards. This technique does not require a model of the environment. Q-Learning can handle stochastic transition and reward problems without the need for adaptations. Finally, for any finite Markov decision process (FMDP), Q-Learning is the optimal find guidelines. Q-Learning can identify the best action selection strategies for a particular FMDP. The following equation, inspired by Bellman's equation, is used to create a mapping between states, actions, and future expected values.

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Where State = s Action = a Reward = r Time step = t

Q-Learning with Forecasting

A potential flaw of the Q-learning model is that the agent chooses an action (number of bikes to move) based solely on the current bike inventory. In theory, the agent should also consider the expected bike inventory for the next hour. During rush hour, the number of bikes at the station can change dramatically from hour to hour. At such a busy time, we thought it made sense for the model to review the expected bike inventory an hour in advance before the agent made a decision. For example, consider a scenario where at 8am the midtown bike station has 30 bikes and 20 empty spaces. With so many people commuting, 40 rides are expected to end at the station in the next hour, so the expected balance at 9:00 am is 70. The model chooses the best action based on the average of current inventory and hourly expected inventory. A random forest model was created to calculate the "expected inventory of bicycles" for the next hour.

Deep Q Network

Neural networks are very efficient at learning features from structured data. Neural networks have a great ability to generalize and handle unexpected data. To compute the scores that we look up in the Q-table, we make a forward pass through the network to get results more quickly. For citibike, the data is well organized and we know how many bikes are in the station at any given time, so this problem is a good fit for his DQN.

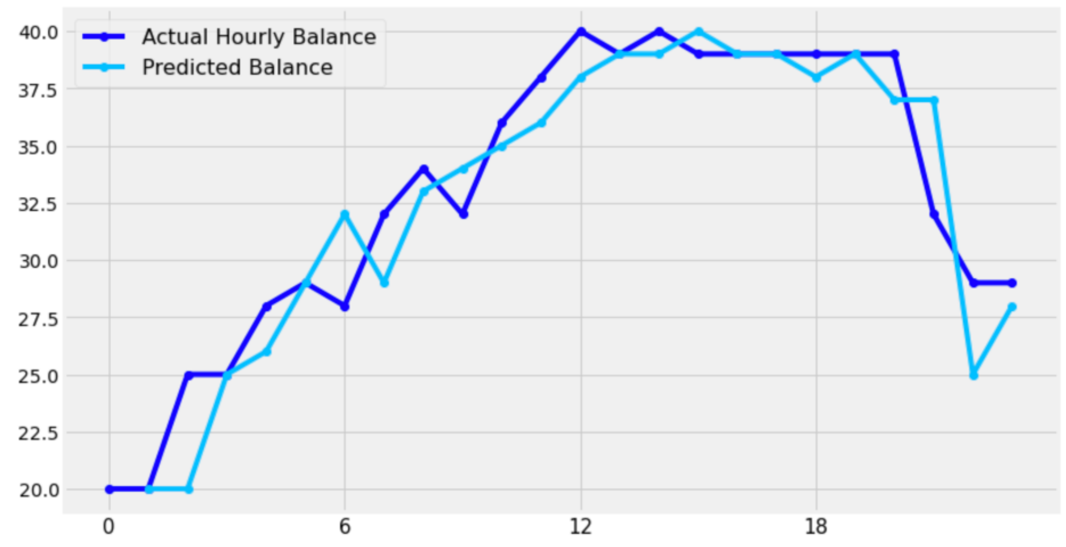
In our Citibike redistribution model, each state is represented by the time of day and the number of bikes at the station. In the simple case of a station that can accommodate up to 80 bikes, we have 80 x 23 states. After getting the states and possible actions as outputs, the model returns the action that produces the highest q-value for the possible states. In later work, we wanted to explore the situation of observing multiple bike stations simultaneously. If each station can hold n bikes, the number of states will be tx,i * nx,j. where x is the station, i is the time and j is the number of bikes at that station.

7. Implementation & Results:

Results:

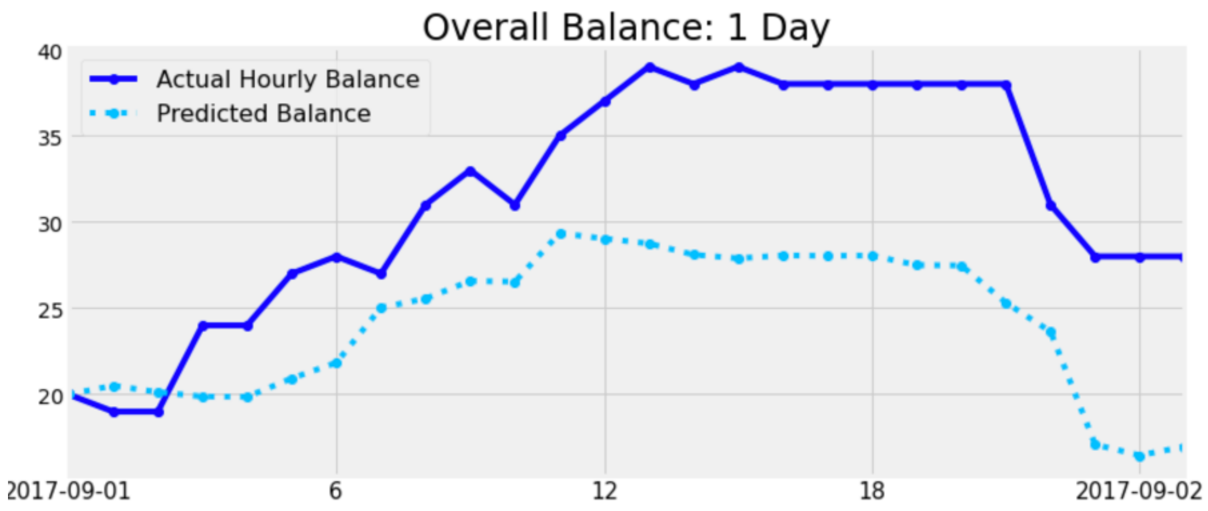
A Simple Case with Linearly Increasing Bike Stock:

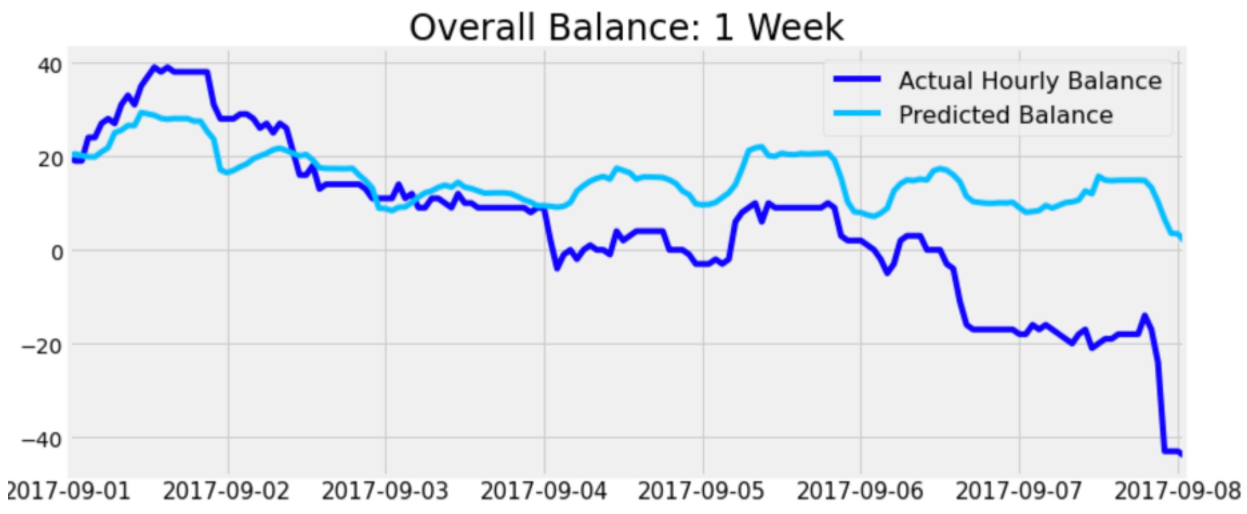
The team tested the RL agent with a simple case. Based on the graph below, the RL agent proved to be able to 1) recognize the bike stock limit of 50 without explicit coding and 2) learn a more cost-effective way to move bikes. The orange line represents how the RL agent moved bikes in the first interaction with the environment. It simply moved a random number of bikes at each time step. The green line represents how the RL agent moved bikes after interacting with the environment and learning after 150 rounds. It developed a smarter strategy: the agent waited until the bike stock was about to reach 50, which was the artificial constraint, before taking some actions. The agent learned to meet the objective with a cost-effective strategy that it developed by trial-and-error without having human instructions.

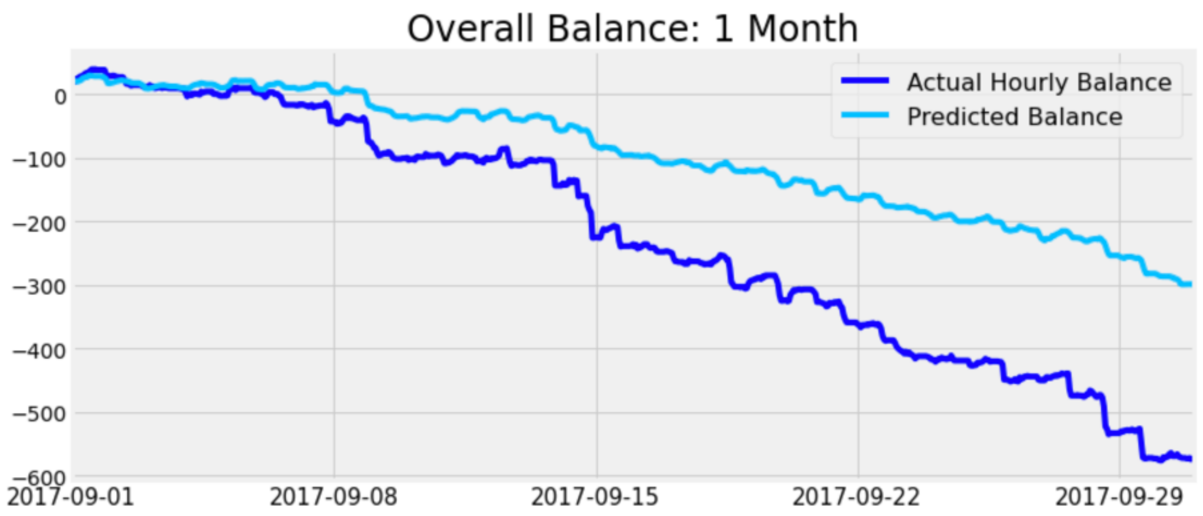


A more Realistic Simulation with Random Variation and New Constraints:

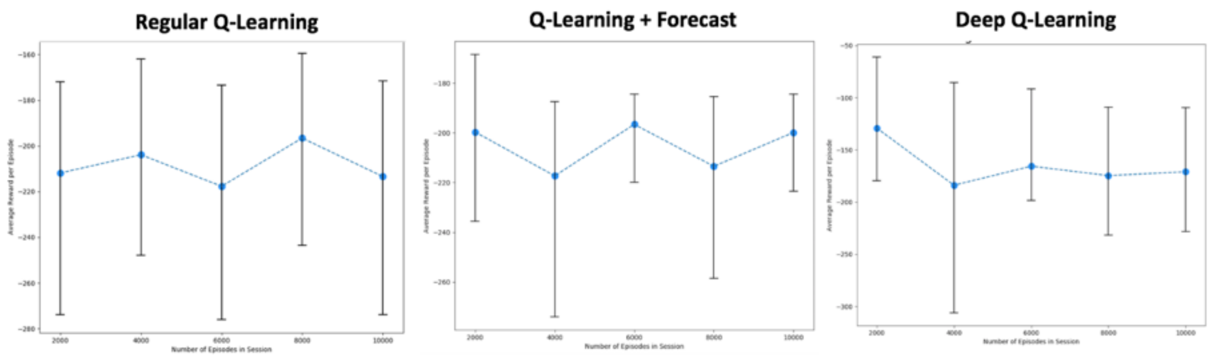
Once the team proved the basic mechanics of the RL agent, random dynamics was introduced to the bike stock. In addition, new constraints of non-zero bike stock was also reinforced using heavy penalties. Without changing the code or having additional human instruction, the RL agent was able to adapt and develop a better bike rebalancing strategy and recognize the non-zero boundary.







The gap between the DQN and Q-Learning models is about 40 points. Since each "bad" hour -- with bike stock outside [0, 50] -- results in a -30 point penalty, one could say that the DQN model, on average, keeps the station balanced for 40/30 = 1.33 hours longer than the Q-Learning models.



Strangely, across all 3 models, the average reward does not change significantly with more episodes of training. This result was somewhat surprising given our first two sets of plots. More research needs to be done on determining how long these models need to be trained. We tested in 2000-episode increments, but perhaps the agent can learn sufficiently by episode 1000.

Finally, we note that the Q-Learning + Forecasting method did not result in tangible gains over the regular Q-Learning method. Part of the problem was that we trained and evaluated the models at the same time. Ideally, we would have distinguished between a "training" mode and an "evaluation" mode, and we would have only incorporated the forecasted values during the evaluation mode. Why was this a major issue? For this model, we took an action based on the "averaged stock" -- the average of the "current stock" and the "expected stock". During the subsequent training step, the row in the QTable that we updated corresponded to the averaged stock, but you could argue that we should have updated the current bike stock row instead.

Individual Contributions:

- Ashika: Overall solution design, development of base code, and code integration and testing; Built random forests model to predict a station's hourly net flow; Analyzed results for Q-Learning and Deep q network; Supported presentation and report writing.
- Ramya: Implemented Q-Learning and Q-Learning with Forecast Model; Analyzed results for Q-Learning (basic + Forecasting); Attempted to transfer training methods onto HPC to decrease training time; Compiled literature review; Assisted with the writing of report and presentation.

- Anusha: Wrote the code for the deep q network, as well as the logging functions for this method. tried to incorporate predictions into the Q-Learning method (Q-Learning + Forecasting); Assisted in writing the report and presentation.

8. Conclusion:

We have successfully shown the initial use of the RL method to bike rebalancing. In different bike stock dynamics, from simple linearly growing, randomly generated, to actual bike stock, and new bike stock limits, the RL agent was able to design, adopt, and improve bike balancing strategy on its own. The researchers also compared the performances of several learning techniques, including Deep Q Learning, Q Learning with Forecasting, and Q Learning in general. Finding the ideal approach is not simple. The Q-Learning algorithms perform better according to one evaluation criterion (% of episodes when stock is rebalanced by hour 23). The DQN approach performs better when measured by another metric (average reward each episode).

The ideal model ultimately depends on the priorities of CitiBike, in their opinion. Do CitiBikes efforts to balance bikes only extend over a 24-hour period? DQN is then the undisputed winner. What if CitiBike wishes to guarantee that every bike station is correctly balanced for the following day? The Q-Learning approaches might therefore merit further thought. Concerned about balancing motorcycles over a period of 24 hours? DQN is then the undisputed winner. What if CitiBike wishes to guarantee that every bike station is correctly balanced for the following day? The Q-Learning approaches might therefore merit further thought.

9. Appendix:

Colab and Project Code Link - https://github.com/gangasani-anusha/RL-FinalProject

10. References:

https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

https://www.geeksforgeeks.org/q-learning-in-python/

https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learnin
g-14ac0b4493cc/

https://ojs.aaai.org/index.php/AAAI/article/view/20618

https://arxiv.org/pdf/1803.03916.pdf

https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc#:~:text=D
eep%20Q%2DLearning%20uses%20Experience,to%20train%20after%20each%20step.

https://www.geeksforgeeks.org/deep-q-learning/

https://www.mlq.ai/deep-reinforcement-learning-q-learning/

https://towardsdatascience.com/understanding-random-forest-58381e0602d2