Citibike Redistribution with Reinforcement Learning

Machine Learning for Cities, Final Project (Spring 2018)

Ian Xiao (ixx200@nyu.edu), Alex Shannon (acs882@nyu.edu),
Prince Abunku (pa1303@nyu.edu), Brenton Arnaboldi (ba1303@nyu.edu)

GitHub Repository: https://github.com/ianxxiao/reinforcement_learning_project

Project Overview

Over the past half-decade[1], bike-share has become an increasingly viable option for transportation and recreation in urban areas. It offers a potentially pleasant[2] and often convenient mode of moving about a city; bike-share also helps mitigate concerns of getting one's bike stolen, and provides the option of one-way trips. One major drawback of the largest American bike-share programs as of 2018 is an uneven flow from station-to-station, leaving some stations with an overflow of bikes, and others lacking bikes altogether. These issues inconvenience users, and have lead to a number of efforts to create 'dockless' bike-share programs, which solve this problem by allowing users to park their bikes anywhere, but create a number of externalities of their own[3], such as piles of bikes on sidewalks, and users hiding bikes for future use. Motivate, the company which oversees bike-share programs in New York (Citibike), Washington D.C. (Capital Bikeshare), Portland, Oregon (Downtown PDX), the Bay Area (Ford GoBike), among others, currently maintains a close eye on the issue of evenly allocating bikes among their pick-up/drop-off stations, and has a fleet of vans which drive around to redistribute the bikes; 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 *BikeAngels*[4] program, attesting to the importance the company sees in resolving the issue.

At the present moment, the bulk of bike redistribution is done manually by a team of over 100 employees, monitoring stations in the company's headquarters and operating the vans to redistribute the bikes on the ground[5]. Regular patterns emerge in which stations fill up at which times, on which days of the week, during certain types of weather, etc. The team working bike redistribution is no doubt familiar with many of the nuances of these systems, but a problem such as this lends itself readily to the prediction techniques of machine learning, and we strongly believe that a system

[1] Citibike began operation in May, 2013

[2] The authors speak from personal experience in stating that 'pleasantness' is contingent on factors such as weather, traffic, and road conditions, but is, in fact, possible.

[3] https://www.washingtonpost.com/local/you-can-park-a-dockless-bike-share-bicycle-anywhere-but-you-shouldnt/2018/02/19/adbcd996-1585-11e8-8b08-027a6ccb38eb_story.html

[4] https://help.citibikenyc.com/hc/en-us/articles/115000246291-What-is-Bike-Angels-

[5] https://help.citibikenyc.com/hc/en-us/articles/115007197887-Redistribution

designed to predict and optimize the pattern of bike redistribution in New York City (and eventually trained in other cities) could greatly accelerate the successful redistribution of bikes within the network, and alleviate the inconvenience to users of winding up at a full or empty station, ultimately increasing ridership and satisfaction.

After researching possible machine learning techniques for such redistribution, we ultimately decided to take the approach of using Reinforcement Learning (RL), as these methods have had great success in solving strategic problems[6], and we believe that Citibike redistribution can be framed in the context of a strategic game without significant information loss. In our analysis, we examined and compared the success of a number of Reinforcement Learning methods, including Q-tables, deep Q-networks, and Q-learning with forecasting models.

Literature Review and Context

We are not the first people to investigate the issue of bike-share redistribution, however to our knowledge, our approach of gamifying the system and applying reinforcement learning methods is novel.

Most analyses of the bike-share data have come at the issue from attempting to classify station types or provide data on system as they are, which may be helpful in the development of strategies to improve redistribution, but do not attempt to optimize or apply predictive methods to the systems themselves. For example, the Moscow-based urban analytics firm, Urbica, has shared analytics on which stations are the most heavily trafficked (and thus likely to over-fill) and used cluster analysis to group stations with similar usage-statistics together[7]. This analysis can certainly be effective from a planning perspective, but ultimately still leaves real-time decision-making up to human actors.

A more behavioral approach has been taken by Citibike's 'BikeAngels' team, which seeks to incentivise users to redistribute the bikes by providing incentives to move bikes from full or close-to-full stations to less-populated stations. While this program has shown some effective results[8], it also has its faults by serendipitously rewarding 'reverse-commuters' with no real added effort or behavior-change on their part. The approach is also not comprehensive, and is, critically, beyond any central control of the team attempting to redistribute the bikes. While predictive methods of how BikeAngel members will move bikes may be used to and incorporated into redistribution strategies, they can not be relied upon in times of particularly heavy traffic.

[6] E.g. DeepMind's 'AlphaGo': https://www.nature.com/articles/nature24270
[7] https://medium.com/@Urbica.co/city-bike-rebalanced-92ac61a867c7
[8] http://www.slate.com/blogs/moneybox/2017/02/09/new_york_s_citi_bike_pays_riders_to_make_it_run_better.html

In *An Intelligent Bike-Sharing Rebalancing System*[9], Diogo Lopes investigates an array of machine learning techniques for predicting how bikes will move throughout bike-share systems on an hourly basis. Forming training and test sets by splitting data gathered from Washington D.C. and Chicago bike-shares, Lopes examines Bayesian Networks, Extra Trees, Gradient Boosting Machines, Linear Regression, and Poisson Regression as potential algorithms for predicting bike-movement throughout the network; he graphs error-rates for each algorithm over 1-, 2-, and 3-hour intervals from given start-times. In each of these three hourly intervals, Gradient Boosting Machines prove to be the most effective at predicting bike movement, however there is a noticeable upward shift in error rate with each additional hour. Although we chose to address the problem from a different angle (namely gamifying the system and building a decision-making agent at the timescale of 24 hours), Lopes serves as a benchmark for the application of a variety of standard machine learning methods to the problem of bike-share rebalancing.

For our review of reinforcement learning techniques and their applications, we relied heavily upon the Sutton & Barto's canonical textbook on the subject, *Reinforcement Learning, An Introduction*[10] for a theoretical background in the subject, and an understanding of possible use cases and methodologies. David Silver's thesis[11] on reinforcement learning in the context of the traditional board game *Go* provided guidance in how to frame the problem, and Morvan Zhou's reinforcement learning tutorials on YouTube[12] and GitHub[13] provided practical advice for implementing these techniques in a python-based framework.

A further dive into recent literature in reinforcement learning lead us to Duan et al.'s *Benchmarking Deep Reinforcement Learning for Continuous Control*[14], which attempts to lay out a benchmark for the evaluation of reinforcement learning tasks by observing the performance of a variety of algorithms on tasks covering the spectrum of basic tasks (e.g. the 'cart-and-pole' problem), locomotion tasks, partially-observable tasks, and hierarchical tasks. Duan et al.'s general survey of the effectiveness of different algorithmic approaches in reinforcement learning at addressing a variety of problems provided us with context for performance evaluation and a roadmap for methodologies that could be used to expand our study to a broader context.

---

9 https://estudogeral.sib.uc.pt/bitstream/10316/35509/1/An%20Intelligent%20Bike-Sharing%20Rebalancing%20System.pdf

10 http://incompleteideas.net/book/bookdraft2017nov5.pdf

11 http://papersdb.cs.ualberta.ca/~papersdb/uploaded_files/1029/paper_thesis.pdf

12 https://www.youtube.com/watch?v=NVWBs7b3oGk&list=PLXO45tsB95cJYKCSATwh1M4n8cUnUv6lT

13 https://morvanzhou.github.io/tutorials/machine-learning/reinforcement-learning/

14 http://proceedings.mlr.press/v48/duan16.pdf

Other literature, such as Mania et al.'s *Simple random search provides a competitive approach to reinforcement learning*[15] and Yu at al.'s *Deep Reinforcement Learning for Simulated Autonomous Vehicle Control*[16] provide insight into applications of deep-Q learning methods in the physical environment; Mania et al. investigate model-free methods of controlling physical systems, attesting to the success of such systems relying on random search in non-physical systems, and propose that they can compete with many of the pre-tuned methods often deployed for systems operating in the physical world. Yu et al. investigate a variety of reward functions for deep-Q networks in attempts to train an autonomous vehicle to navigate a simulated environment. While these methods are not directly applicable to the our effort, they demonstrate the robustness of deep reinforcement learning applications to real world problems, and the effectiveness of such methods when working with complex systems, such as those which we attempt to address in our own work.

Solution Design

The solution includes four modules: RL agent, Environment, Performance Tracking, and Training. To better develop a robust solution with flexibility for future expansion, the code was written using an Object Oriented Programming (OOP) framework.

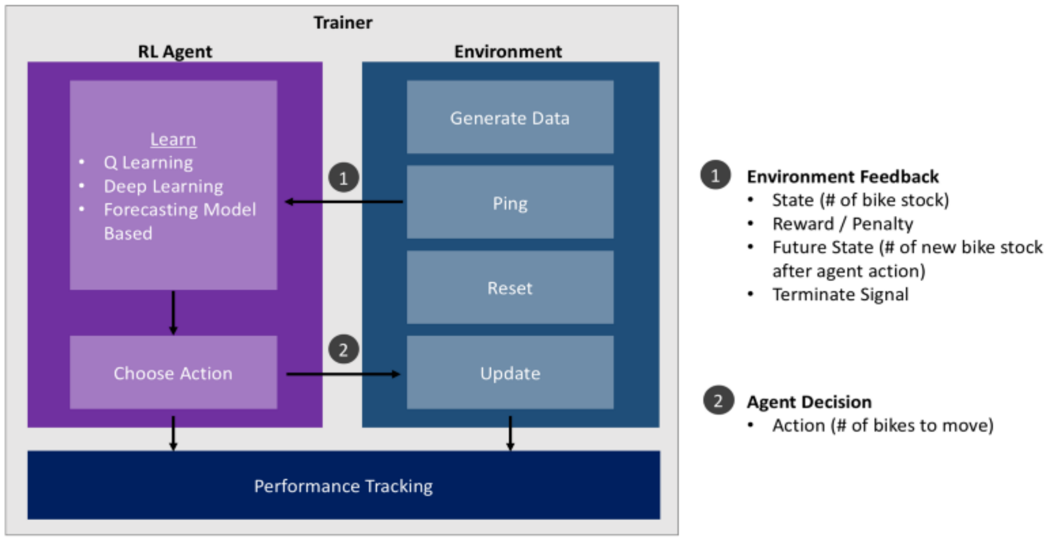


Figure 1: A High Level Conceptual Solution Architecture Diagram

The Environment module generates simulated and actual Citibike data, sends feedback to the RL agent (e.g. current bike stock, reward and penalty, new bike stock, and

[15] https://arxiv.org/pdf/1803.07055.pdf
[16] http://cs231n.stanford.edu/reports/2016/pdfs/112_Report.pdf

termination signal), updates the bike stock based on the action received from the RL agent, and resets the environment for new training session.

The RL Agent module mainly consists of learning and decision capabilities. One of the objectives of this project is to benchmark performances of different learning implementation, such as Q Learning, Deep Q Learning, and Model Based Learning (Q-Learning with Forecasting). Users can specify which method to use for learning or simply select all.

The Performance Tracking module captures all the detailed metrics and visualizes and stores the results automatically. Users can find the results in their local directory for analysis and future reference. The key metrics the program tracks are action history, reward history, bike stock comparison between first and last training episodes, and success ratio of all training sessions.

The Trainer module facilitates the end-to-end process of training set up, RL and environment module set up based on user inputs, and exception handling.

Data

To train the RL agent, the program relies on two types of data: simulated and actual Citi Bike data. Simulated data can be 1) a simple bike stock with fixed increment of 3 additional bikes per hour and 2) an increasing bike stock of 3 additional bikes per hour with random fluctuation. The random fluctuation is produced by a random integer generator, which has a mean of zero and range between -5 and 5.

The program also generates bike stock data based on real historical Citi Bike ridership. Using the Citi Bike Open Data site, an API called is made to pull a month of trip data from September, 2017. A sequence of data processing steps are performed to translate trip data to hourly bike stock by station (~600 stations) over the month based on the net flow of arrival and departure trips. Because the initial bike stock is unknown, the team made an assumption that all stations start with 20 bikes on September 1st, 2017. This is a parameter users can change dynamically as they see fit.

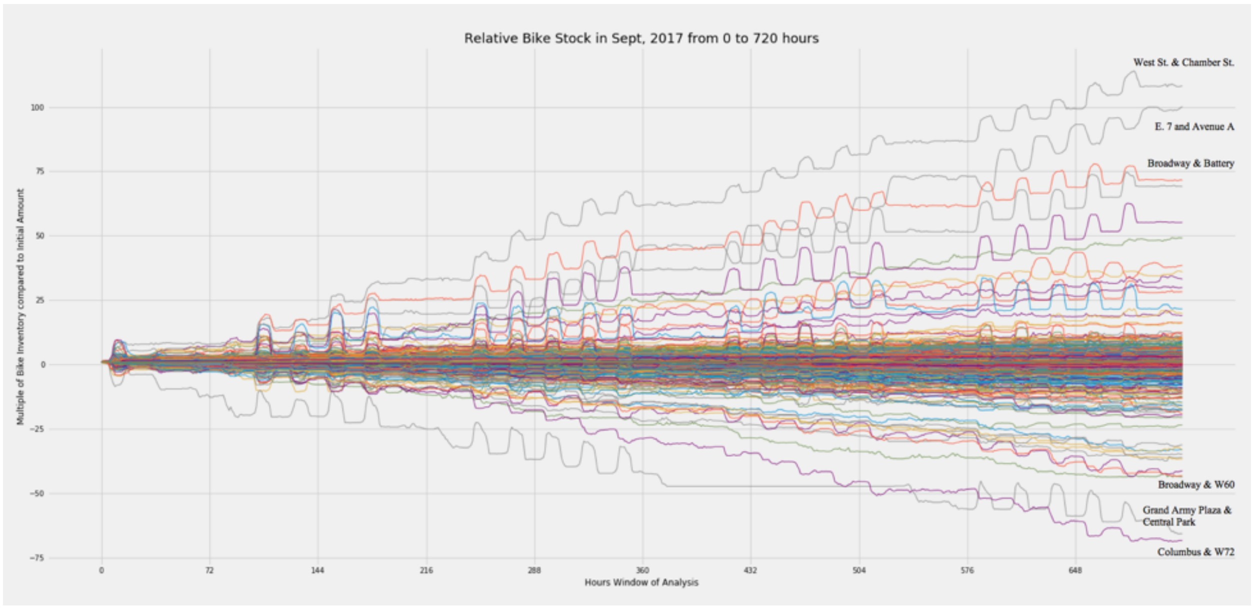


Figure 2: bike stock change for all stations from September 1st to 30th, 2017

<u>Method</u>

The solution includes three types of learning methods: Q-Learning, Enhanced Q-Learning with Forecasting, and Deep Q-Learning using Neural Network. This section provides a more detailed overview of Reinforcement Learning and each method.

Reinforcement Learning

In general, RL consists of an agent, a set of states, and a set of action per state. The agent transitions from state to state by performing an action until a terminating state. The action in a specific state leads to a reward or penalty. The overall goal of the agent is to maximize long term reward. This reward is a weighted sum of all future expected rewards starting at the current state.

In the context of the Citi Bike Rebalancing problem, a state is the number of bike stock at a station. Action is the number of bikes the agent can move at each hour. The agent can choose to move 0, 1, 3, or 10 bikes at a given hour. Reward and penalty is structured as the following:

- -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

To learn the mapping between action and weighted future reward at a given state, Q-Learning is the most basic and traditional learning method. The technique does not require a model of the environment. *Q*-learning can handle problems with stochastic transitions and rewards, without requiring adaptations. For any finite Markov decision process (FMDP), *Q*-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable. *Q*-learning can identify an optimal action-selection policy for any given FMDP. The following equation, inspired by the Bellman's Equation, is used to develop a mapping between state, action, and future expected value [17].

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

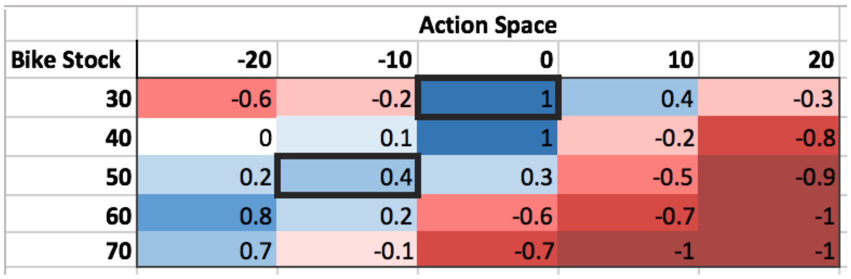
Where

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

Q-Learning with Forecasting
One potential shortcoming from our Q-Learning model is that the agent chooses an action (the number of bikes to move) based only on the ***current bike stock***. In theory, the agent should also consider the ***expected bike stock*** in the next hour. During rush hour, the number of bikes at a station might change dramatically from hour to hour. For these high-traffic periods, we thought it might be useful for the agent to consider the expected bike stock one hour in advance before making a decision.

For example, consider a scenario where we have a bike station in Midtown with 30 bikes and 20 open slots at 8:00am. Because many people are commuting to work, we expect that 40 rides will end at the station in the next hour, so the expected balance at 9:00am is 70. A hypothetical Q-Table for the particular station is below. What is the optimal action at this time?

[17] Q Learning Equation: Source: https://en.wikipedia.org/wiki/Q-learning



	Action Space				
Bike Stock	-20	-10	0	10	20
30	-0.6	-0.2	1	0.4	-0.3
40	0	0.1	1	-0.2	-0.8
50	0.2	0.4	0.3	-0.5	-0.9
60	0.8	0.2	-0.6	-0.7	-1
70	0.7	-0.1	-0.7	-1	-1

Figure 3: A sample Q-Table from Q-Learning

If we consider only the current stock (30), the Q-table tells us that the optimal action is to do nothing. (Given row=30, the action that maximizes expected reward in one hour is "0"). But because we expect the station's bike stock to increase from 30 to 70 in the next hour, selecting an action of "0" is probably not a good strategy.

Instead, we experimented with an approach that picks the best action based on ***an average of the current stock and expected stock in one hour.*** In this case, the average of the current stock (30) and expected stock (70) is 50. We then take the best action specified by bike stock=50, which is to remove 10 bikes. So at 8:00am, we would reduce the number of bikes in the Midtown station from 30 to 20 to account for the expected increase.

To compute "expected bike stock" in the next hour, we built a Random Forests model. For more details about the Random Forests model, please refer to the appendix at the end of the paper.

Deep Q Network

There are many times when one wants to perform reinforcement learning that a q-table will be too inefficient. A typical example of this situation is using a reinforcement learning algorithm to teach a program to play videogames. In the seminal paper by Mnih et al *Human-level control through deep reinforcement learning*[18] they discuss their work teaching a program to play different Atari video games. The possible states were represented by the number of pixels in the image screen and they used 4 such screens to detect the direction and speed of objects in the game as well. Taking into consideration that the screen used a 128 color palette and the size was 84 x 84 pixels the number of possible states would equal to 128^{28224}. That means that are q-table would have to have the same number of rows to represent each state. Lookups in this table would be slow and highly demanding of memory so the alternative presented was a neural network.

[18] https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

A neural network is very efficient in learning features from structured data. Though the reason why neural networks work so well is not fully understood we know they they have a high ability to generalize and deal with unexpected data. In order to calculate scores that we would look up in a q-table we make one forward pass through the network and more quickly get our results. In the previous case where we mentioned the screens from a video game as the input has no missing data which allows the model to learn from data well. In the case of citibike the data is also well organized in the sense that we know the amount of bikes at a station at any particular time which makes the problem well suited for a DQN.

In our model of Citibike redistribution, each state is represented by the hour of the day and number of bikes at a station. In a simple case where we are looking at 1 station that could host a maximum of 80 bikes, there are 80 x 23 states. After taking the state and possible actions as output, the model will return an action that gives the highest q-value for the possible state. In later work we planned on considering a situation where we observe multiple bike stations at the same time. If each station can hold n bikes we would have the number of states equal to $\Pi t_{x,i} * n_{x,j}$ where x is the station, i is the hour of the day, and j is the number of bikes at that station.

<u>Results</u>

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.

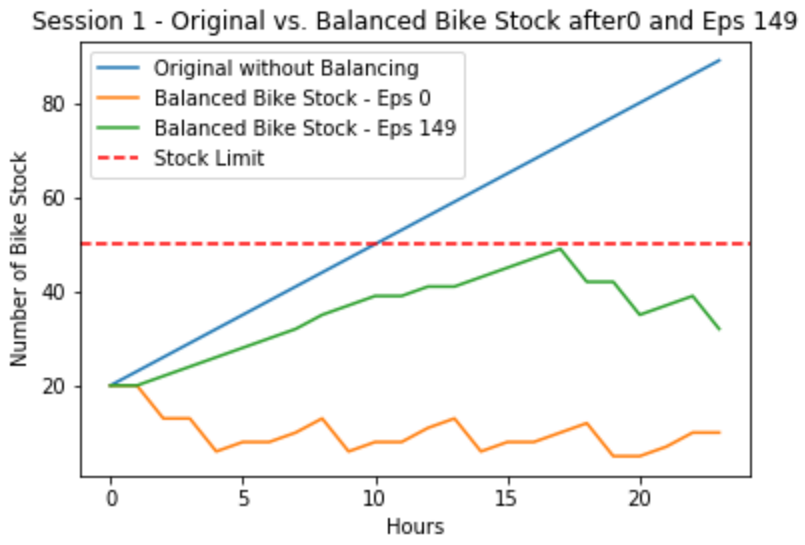


Figure 4: Comparison of bike stock without balancing (blue), bike stock by a newly trained RL agent (orange), and bike stock by a better trained RL agent (green)

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 penalty. Without changing the code or having additional human instruction, the RL agent was able to adapt and develop better bike rebalancing strategy and recognize the non-zero boundary.

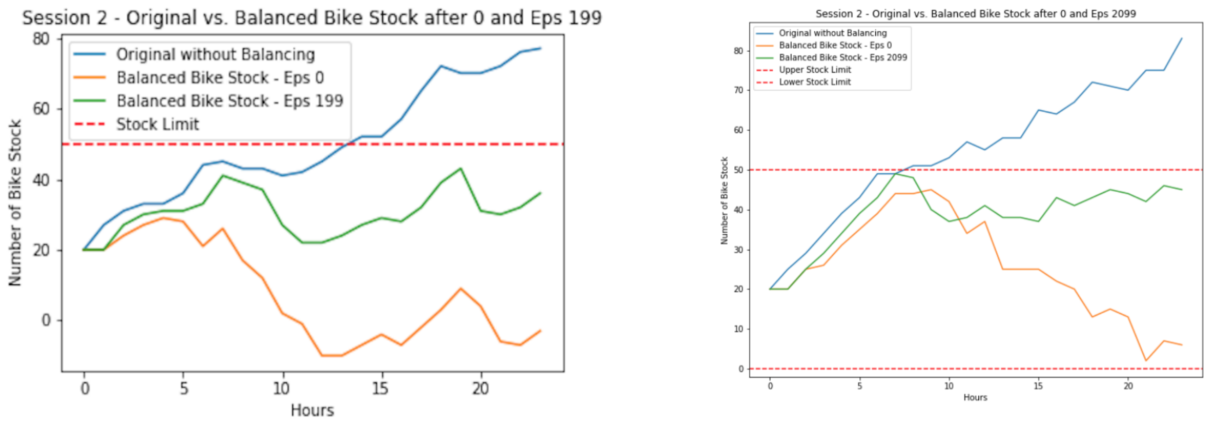


Figure 5: Comparison of a randomly generated bike stock without (left) and with non-zero constraint (right)

A Comparison between Different Learning Methods with Real Citi Bike data

To test the real-world effectiveness of the three RL models (Q-Learning, Q-Learning with forecasting, and Deep Q Networks), we analyzed how they performed for a Citi Bike station. More specifically, we tried to balance the bike stock for the Citi Bike station at 17th and Broadway (Union Square) over a 24-hour period on September 1, 2017. Union Square is peculiar because there are two convenient north-south bike lanes (Broadway and Lafayette Street) that feed into the plaza, but no good outbound bike lanes. As a result, the station on 17th and Broadway tends to accumulate bikes over the course of the day, as more bikes are arriving than departing.

The first set of plots illustrate the ability of the models to balance bikes at the 17th and Broadway station. From left to right, we have the regular Q-Learning model, the Q-Learning plus forecasting model, and the Deep Q Network method. Each plot contains: 1) the station's actual bike stock (blue); 2) rebalanced stock by a "dumb" untrained agent (orange); and 3) rebalanced stock by a trained agent (green). The trained agent has undergone 10,000 days of training. We assume that the bike stock starts at 20, and that the station has 50 total spaces for bikes.

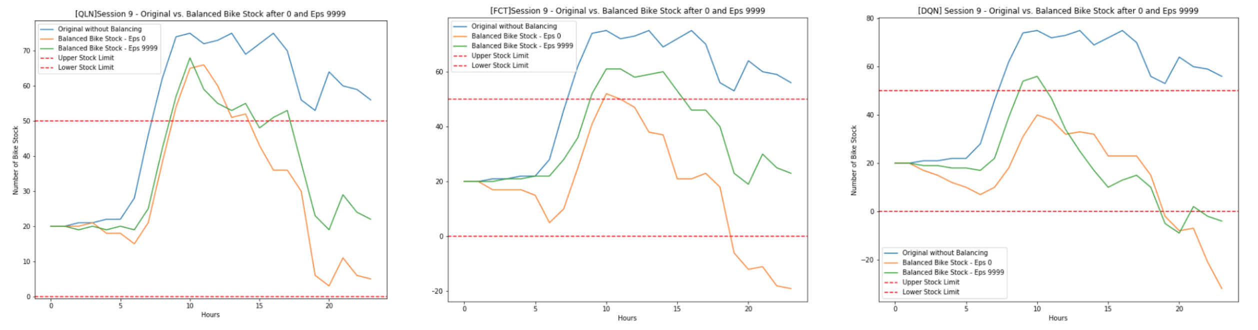


Figure 6: Comparison of Bike Stock Outcomes based on Q-Learning (left), Enhanced Q-Learning with Forecasting (middle), and Deep Q-Learning (right)

We want the agent to keep the bike stock in the range [0, 50] while minimizing the number of bikes moved. If the agent does nothing (blue line), it will suffer a huge penalty, since the bike stock remains above 50 for almost 18 hours. Conversely, an untrained agent (orange line) may pursue inefficient transfers of bicycles, sometimes even causing the stock to fall below 0. At least in the first two plots, the trained agent seems to do the best job of balancing the priorities of: i) keeping bike stock within [0, 50] and ii) moving as few bikes as possible.

Our main evaluation metric for "success" was the percentage of episodes (days) where the end-of-day balance (hour 23) fell in the range [0, 50]. For each model, we trained sessions with 2000, 4000, 6000, 8000, and 10000 episodes. The results are plotted below.

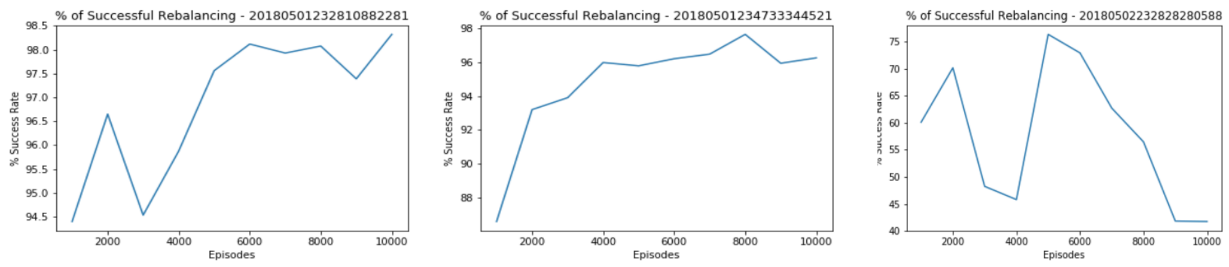
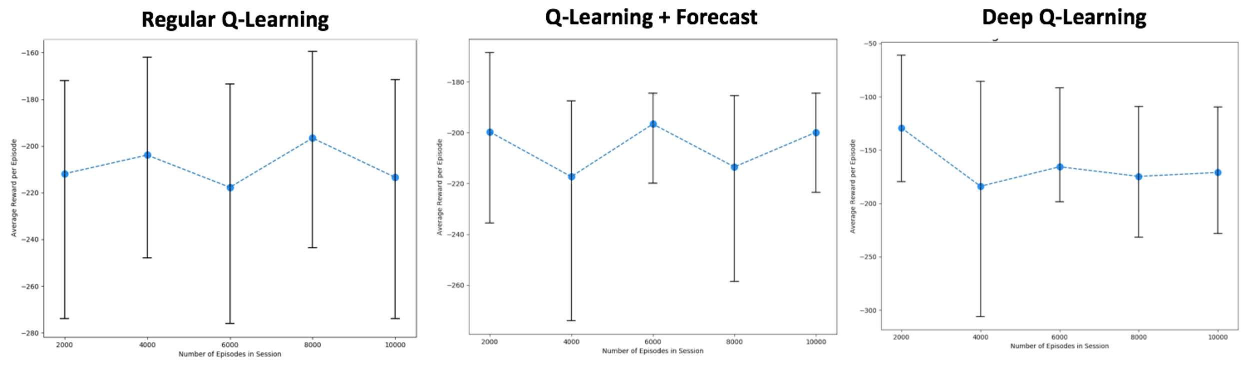


Figure 7: Comparison of Success Rate of Q-Learning (left), Enhanced Q-Learning with Forecasting (middle), and Deep Q-Learning (right)

After sufficient training, the two Q-Learning models were able to balance the bike stock by hour 23 over 95% of the time. However, the Deep Q Network method was not as proficient according to this evaluation metric, averaging a success rate of around 60%. We suspect that with additional parameter tuning, the performance of the DQN method would have improved.

Another evaluation metric we analyzed was "Average Reward Per Episode". As a refresher, the agent incurs a cost of -30 for each hour it fails to keep the bike stock within [0, 50], as well as -0.5 for each bike moved. The results for 2000, 4000, 6000, 8000, and 10000 episodes are illustrated below. The y-axes are on different scales, but the most important finding is that the **Deep Q-Learning method outperforms the Q-learning models by average reward.** Specifically, the average reward for the Q-Learning methods is approximately -210. By contrast, the DQN model has an average reward of roughly -170.



Note: blue dots denote "mean reward"
Tick marks represent 75th percentile and 25th percentile rewards

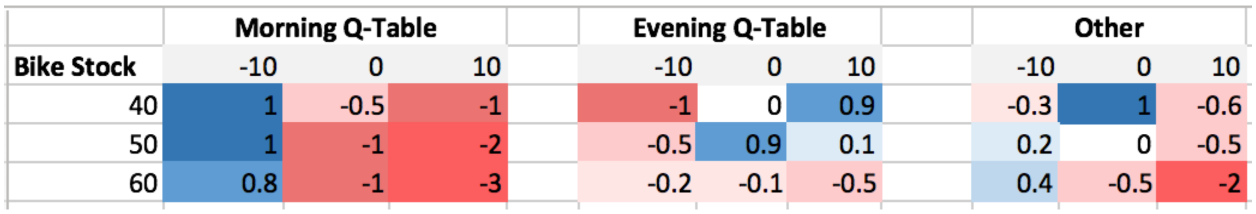
Figure 8: Comparison of Reward History of Q-Learning (left), Enhanced Q-Learning with Forecasting (middle), and Deep Q-Learning (right)

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 Q-Table that we updated corresponded to the *averaged stock*, but you could argue that we should have updated the *current bike stock* row instead.

As a simpler alternative to the Q-Learning + Forecasting model, we could have created multiple Q-Tables for different times of the day. Practically speaking, the optimal action for a given bike stock depends heavily on the time of day. In response, we could have trained separate Q-Tables for morning rush hour (8-10am), evening rush hour (4-6pm), and all other times. The tables for a bike station in Midtown, for example, might look like this:



	Morning Q-Table			Evening Q-Table			Other		
Bike Stock	-10	0	10	-10	0	10	-10	0	10
40	1	-0.5	-1	-1	0	0.9	-0.3	1	-0.6
50	1	-1	-2	-0.5	0.9	0.1	0.2	0	-0.5
60	0.8	-1	-3	-0.2	-0.1	-0.5	0.4	-0.5	-2

Figure 9: Comparison of a subset of Q-Table by Time of Day

Consider the row for bike stock = 40. In the morning, the Q-table tells us that the optimal action is to **remove** 10 bikes. In the evening, however, the optimal action is to **add** 10 bikes to the station (since the model might learn that bike stock falls dramatically in the evening). Finally, during off-peak hours, the optimal action is to do nothing. In the example illustrated above, we used our prior knowledge of Citi Bike trips to specify how many Q-tables should be created. A more “pure” reinforcement learning approach would entail the creation of 24 Q-tables (one for each hour), and require the agent to learn which hours are truly relevant for rebalancing.

Conclusion

The team has proven the preliminary application of RL approach to bike rebalancing. The RL agent was able to develop, adopt, and improve bike balancing strategy autonomously in various bike stock dynamics - from simple linearly increasing, randomly generated, to actual bike stock - and new bike stock limits. In addition, the team benchmarked the performances using different learning methods, such as Q Learning, Q Learning using Forecasting, and Deep Q Learning. Determining the best method is not trivial. By one evaluation metric (% of episodes where stock is rebalanced by hour 23), the Q-Learning methods are more successful. By another metric (average reward per episode), the DQN method achieves better performance.

Ultimately, from CitiBike’s perspective, the best choice of model depends on their priorities. Does CitiBike only care about balancing bikes during a 24-hour window? Then DQN is the clear winner. What if CitiBike wants to ensure that each bike station is properly balanced for the next day? Then the Q-Learning methods may warrant additional consideration.

Discussion and Possible Next Steps

In our project, we focused on taking historical data from a particular station in NYC and balancing the bikes to be between 0 and 50. While our methods were generally effective for this approach, the dynamics will most likely change when we observe multiple bike stations. When bikes are removed from one station, they generally must be added to another. This means that the action space will also have to increase to represent the addition of bikes. Specifically, we would probably need to include one action space *between every possible pairing of stations.* Future complications occur when we decide to not only remove bikes from one station but then divide those bikes amongst multiple stations. This situation is more reflective of real world experiences when Citibike picks and retrieves their bikes for optimization.

These additional requirements for our model will also mean there will be an increase in the state space. Previously, our state space was the number of possible bikes at a station * 24 hours. (In our Citi Bike station example, the bike stock peaked at roughly 80, so this would equate to 80*24 = 1920 possibilities. However, even adding one more station with the same state space could increase the total state space to over 3,000,000. Due to the size it would no longer be optimal to use a q-table and we would transition into using the DQN. Since we are using neural networks we could optimize the speed by running on a GPU. Finally, our goal with this model would be to accurately predict future bike stock for different stations so that Citi Bike could implement and improve their optimization methods.

Project Contributions

Ian: Overall solution design, development of base code, and code integration and testing; Supported presentation and report writing.

Alex: Attempted to transfer training methods onto HPC to decrease training time; compiled literature review; reached out to Citibike for info regarding current distribution methods and the applicability of our approach; assisted with writing of report and presentation

Prince: Wrote the code for the deep q network, as well as the logging functions for this method. Assisted in writing the report and presentation.

Brenton: Built random forests model to predict a station's hourly net flow; tried to incorporate predictions into Q-Learning method (Q-Learning + Forecasting); assisted with writing of report and presentation.

Appendix:

Random Forests Forecasting Model

To derive a station's "expected bike stock" in the next hour, we used a Random Forest algorithm. Specifically, we built Random Forests models to predict the **hourly net flow** of bicycles, with one model per station. "Net flow" equals the number of bikes arriving minus the number of bikes departing a station. We simply added the net flow predictions to a station's current balance to predict the expected balance in one hour.

Our first goal was to predict a station's hourly net flow in September 2017. To train the model, we collected Citi Bike trip data from August-October 2016 (to capture seasonal effects) and July-August 2017 (to capture more recent temporal trends), with September 2017 as our test set. Overall, for each station, we had 3696 training records (24 hours * 154 days). The variables for the model were a mix of autoregressive features and weather data scraped from Weather Underground's API.

The main autoregressive features we engineered were:

- **Average Net Flow for Hour in Past Week**: average net flow for that hour over the seven previous days. For example, to predict net flow for 9/27/2017, we computed the average net flow at 9:00am from 9/20-9/26.
- **Average Net Flow for Hour-Weekday Combination**: average net flow for the hour/weekday combination over the three previous weeks. For example, to predict net flow for Wednesday, 9/27 at 9:00am, we would compute the average net flow at 9am across the previous three Wednesdays (9/6, 9/13, and 9/20).
- **Net Flow, hour t-1:** net flow at station in previous hour
- **Net Flow, hour t-2**
- **Cumulative Net Flow, past 12 hours**: total net flow at station over previous 12 hours

We created a similar set of autoregressive features for the number of departing bikes:

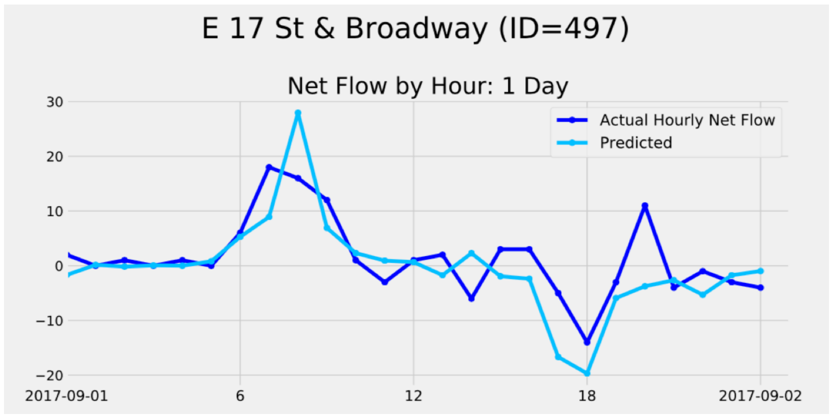
- Average Departures for Hour in Past Week
- Average Departures for Hour-Weekday Combination
- Departures, hour t-1
- Departures, hour t-2
- Cumulative Departures, past 12 hours
- Cumulative Departures, past 24 hours

We included the "departures" features because we felt that they provided information that the "net flow" variables cannot provide. For example, a "net flow" of 0 is ambiguous, as it could either mean: 1) the station has no activity or 2) the station is busy, but the departures and arrivals are balancing each other out. To avoid overfitting, we set the parameter of "minimum samples per leaf" equal to 5.

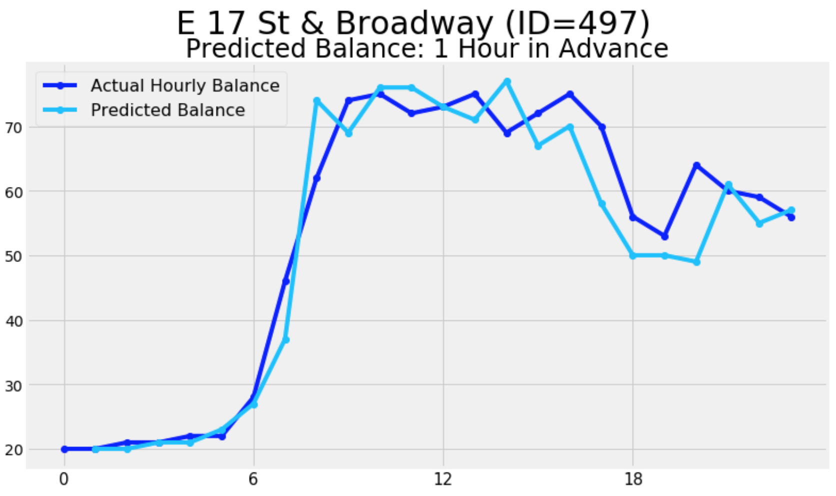
The effectiveness of the Random Forests model varied greatly from station to station. For some stations, the R^2 value on the test set (September 2017) was extremely high, at over 80%. For other stations, however, net flow was far less predictable (see table below). One confounding variable in the analysis was that we lacked information on when Citi Bike moved bicycles. For some really busy stations, Citi Bike probably has a set schedule in place to move bikes. As such, it was easier to forecast net flows at busy stations. By contrast, stations with low bike flow (e.g. 3 Ave & 72 St) were tougher to predict. (Note: test set R^2 calculated over all 30 days in September).

	StationID	Name	Test R^2	Train R^2
0	519	Pershing Square North	0.46	0.708
1	392	Jay St & Tech Pl	0.476	0.71
2	426	West St & Chambers St	0.729	0.832
3	497	E 17 St & Broadway	0.473	0.799
4	3164	Columbus Ave & W 72 St	0.507	0.753
5	281	Grand Army Plaza & Central Park S	0.636	0.873
6	3443	W 52 St & 6 Ave	0.874	0.922
7	304	Broadway & Battery Pl	0.815	0.895
8	3375	3 Ave & E 72 St	0.085	0.554
9	3283	W 89 St & Columbus Ave	0.323	0.659

The plot below illustrates the model's hour-by-hour predictions for net flows at Station 497 (E 17 & Broadway) on September 1, 2017. This is the same station we used to test our RL models. Note that the prediction (light blue) follows a conventional trend, with peaks in the morning and evening, while the actual observations are slightly noisier.



The next plot illustrates the predicted bike stock at the same station. To generate this plot, we simply added the “net flow” predictions from the Random Forests model for each hour to the current balance.



We were concerned with the model’s accuracy rather than interpretability, but for the sake of thoroughness, a list of feature importances for the RF model is below. The autoregressive variables are quite significant. Note that the weather features have low Gini coefficients. It is likely that the autoregressive variables “deps t-1” and “nets t-1” are capturing much of the information provided by weather data.

	Feature Name	**Gini Coeff.**
0	hour_average_nets_weekday_past3	0.315553
1	hour_average_nets_past_week	0.150429
2	hour_average_deps_past_week	0.120441
3	nets_12h	0.0468884
4	hour	0.0426451
5	deps t-1	0.0389631
6	weekday	0.038613
7	deps_12h	0.0339003
8	hour_average_deps_weekday_past3	0.02967
9	nets t-1	0.0295306
10	nets_24h	0.0281394
11	deps_24h	0.026442
12	Temp	0.0222762
13	nets t-2	0.0219192
14	deps t-2	0.0198369
15	Humidity	0.0197379
16	WindSpeed	0.00904039
17	month	0.00597435
18	Precip	0