AI Approach to Bike Rebalancing using Reinforcement Learning

Ian Xiao, Alex Shannon,
Prince Abunku, Brenton Arnaboldi

What is the problem?

Citi Bike has a rebalancing problem due to the growing size of operation and travel patterns ...

- 12,000 bikes in NYC, 752 active stations, average 30,111 rides per day
- Asymmetric shift of of bikes from popular originations to common destinations
- Pressing problem to scale rebalancing workforce as the program expands

Image Sources: see ppt notes.

Can we develop a intelligent solution to show what is the best strategy of moving bikes? Yes, Reinforcement Learning can help.

What is Reinforcement Learning (RL)?

Training a Reinforcement Learning model is similar to how humans and animals learn “good behaviors”.

Agent (e.g. person) selects an action based on the environment, and receives a reward or punishment based on their action.

Image Sources:
https://www.independent.co.uk/life-style/health-and-families/how-to-decide-if-you-want-kids-or-not-a7752011.html
http://www.vetstreet.com/our-pet-experts/teach-your-dog-to-sit

Why use Reinforcement Learning?

Imagine using the traditional supervised and unsupervised learning techniques to achieve the following:

- Learning without explicit human instructions
- Continuous improvement over time
- Capture and adjust to complex and changing system dynamics
- Learn without deliberate feature engineering

The traditional feature engineering, predictive modelling, and cross validation approach are not designed to solve these problems.

How is RL being applied?

RL powers many modern Artificial Intelligence solutions that minic how humans learn and act.

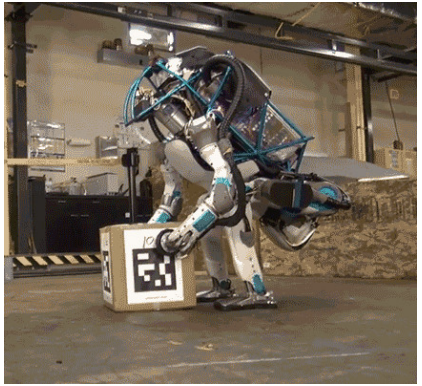


Image Sources:
https://www.dezeen.com/2018/01/10/self-driving-cars-exacerbate-congestion-says-paul-priestman/
https://deepmind.com/research/alphago/
https://postandparcel.info/71426/news/googles-boston-dynamics-releases-video-of-parcel-handling-robot/

What is our set up?

We want to design an intelligent solution that can do the following **without explicit human instructions**:

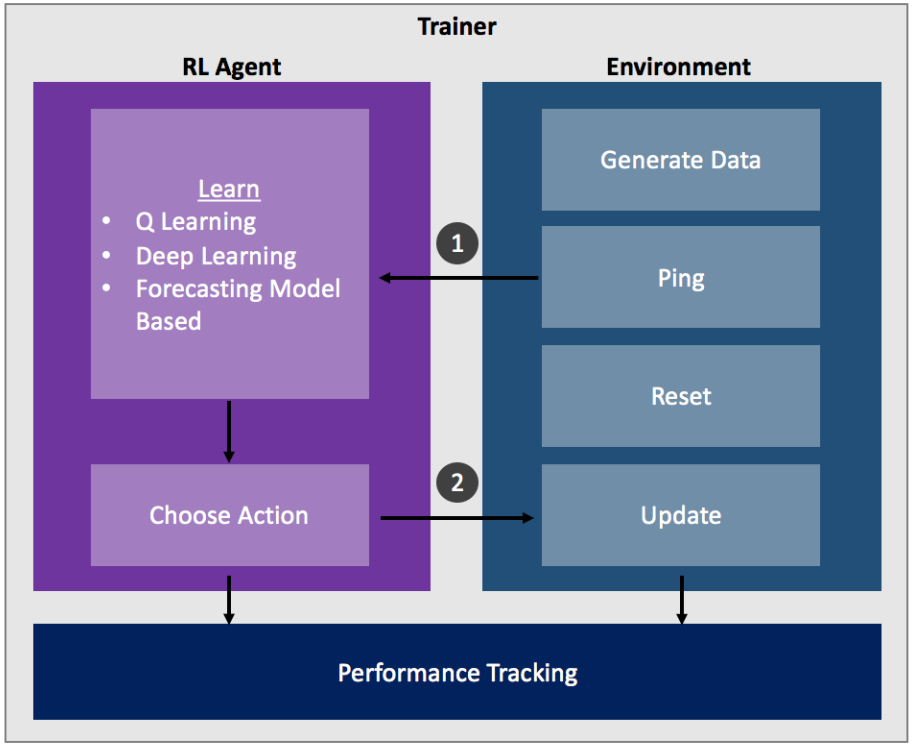
- Maintain bike stocks within a certain range (0 to 50)
- Understand what the bike stock requirements are
- Decide how many bikes to move to meet the requirements
- Determine when to move bikes

By trial and error, the solution should meet the following **objectives:**

- Higher success ratio: keep bike stock within a certain range (0 - 50 bikes by hour 23)
- Lower cost: move the least amount of bikes possible

Environment: bike station, current number of bikes at station, and the current hour
Action Space: move bikes, or do nothing
"Reward" Per Hour: -30 if bike stock is below 0 or above 50 ,
-0.5*(number of bikes moved)

What is our solution?

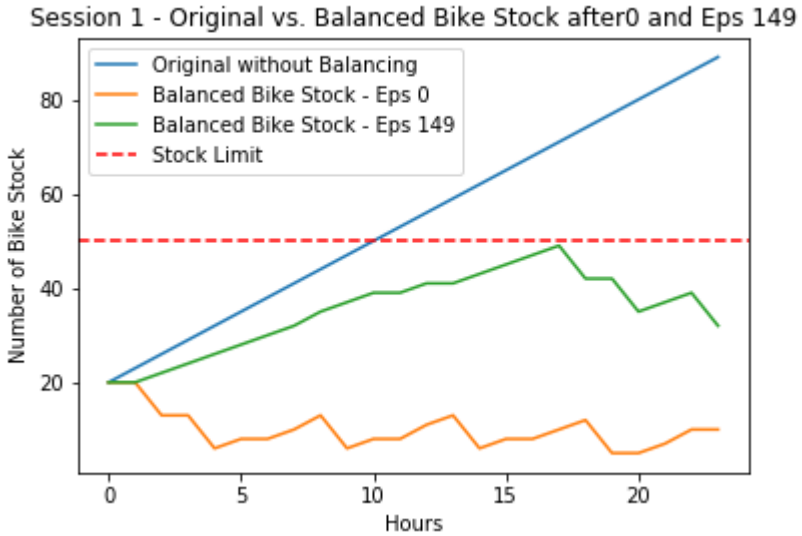


1 **Environment Feedback**
- State (# of bike stock)
- Reward / Penalty
- Future State (# of new bike stock after agent action)
- Terminate Signal

2 **Agent Decision**
- Action (# of bikes to move)

What are our results?

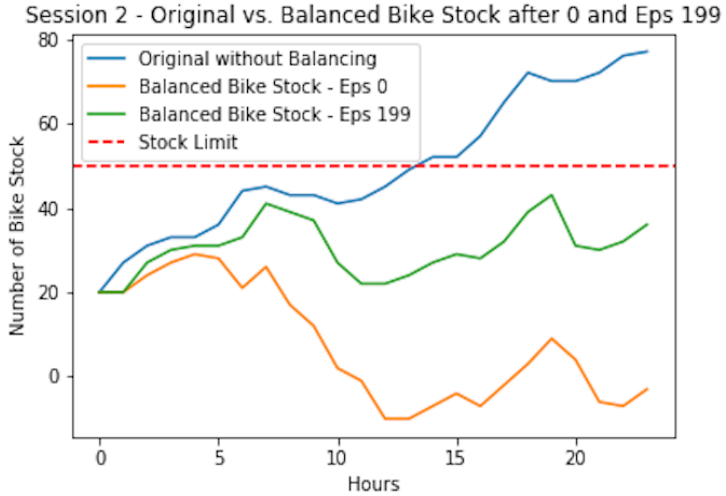
- Can the RL agent recognize a upper bike stock bound? Yes.
- Can the RL agent decide when to move and not to move bikes to save cost? Yes.
- Does the RL agent find a better strategy by learning over time? Yes.

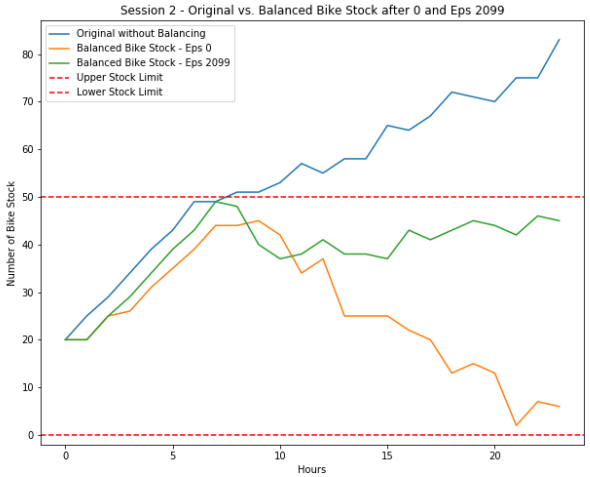


What are our results?

Let’s make it more realistic and complicated …

- Can the RL agent relearn when there are unpredictability? Yes.
- Can the RL agent recognize the new lower bound and respond accordingly? Yes.



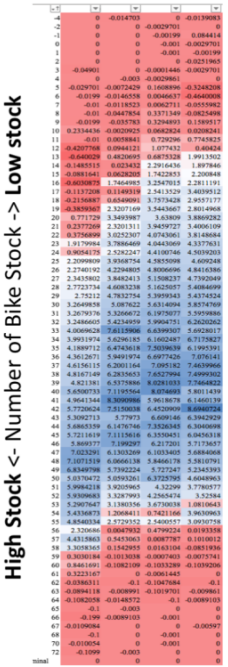


What are our results?

Can we understand how the RL agent makes decisions?

Actions [-10, -3, -1, 0]

High Stock <- Number of Bike Stock -> **Low stock**



Explanation

This is the Q Table that tracks expected reward of an action taken at a given state. Red represents low reward and blue means high reward relatively.

After 98000 episodes of interacting with the environment, the agent was able to construct a re-balancing strategy using this table.

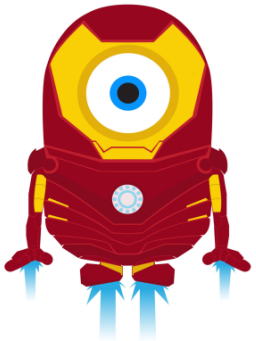
Top Red: the agent knew that it was too costly, and low return, to move bikes when the stocks were low. There is no need to do anything when the stock is way below the threshold of 50.

Middle Blue: the agent started to manage bikes actively when the stock started to reach the 30s (darker blue in actions -3 and -1); and became more aggressive by moving 10 bikes at once when it reached the 40-50s (darker blue in action -10).

Bottom Red: it was likely to exceed the limit of 50 by end of day when the stocks reached the mid 60 – 70s. The best strategy the agent developed was to “do nothing” by moving zero bikes to avoid additional penalty.

Can we improve the learning performance?

Q-Learning (what you saw)



- Markov Decision Process using a matrix to keep track of learning
- The most simple form of learning
- Does not scale

Q-Learning + Forecasting

- Forecast bike stock in the next hour using a Random Forests Model
- Incorporate the forecast into decision and learning

Deep Q Learning

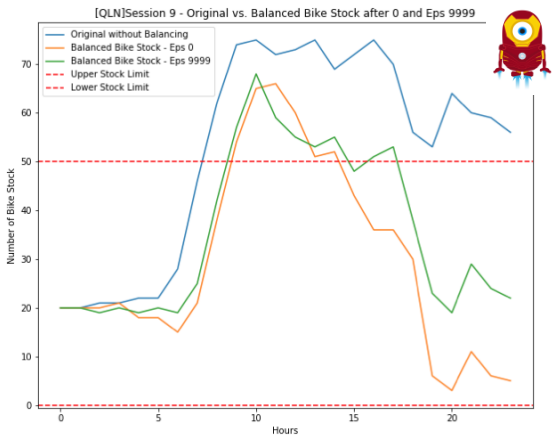
- Use Recurrent Neural Network to remember the learning
- Scale for large system parameters
- Minimizes computation and memory use

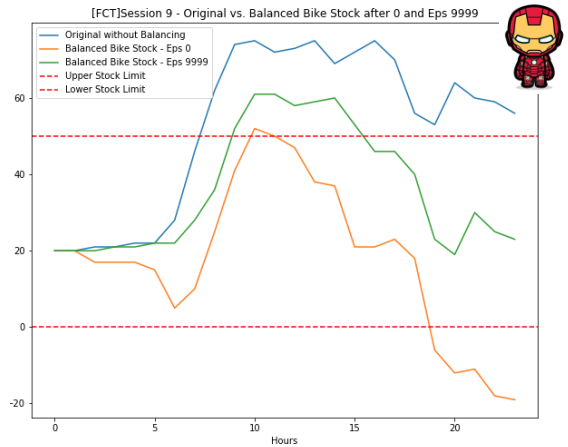
What are our results?

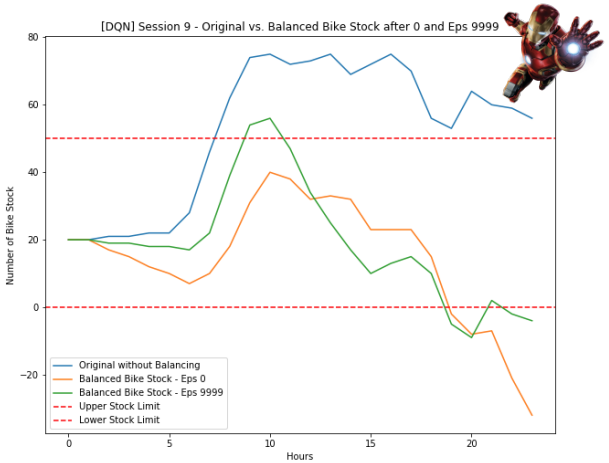
Let’s make it even more realistic by using actual Citi Bike data and benchmark across all three learning methods.

Goal: keep bike stock balanced between [0, 50] while moving the fewest bikes possible.

Bike Stock at 17th and Broadway (North Edge of Union Square) on 9/1/2017





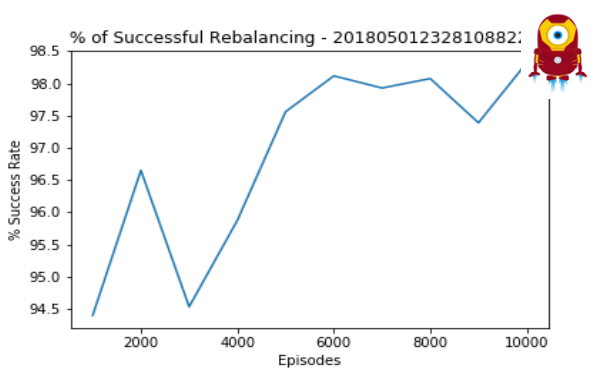


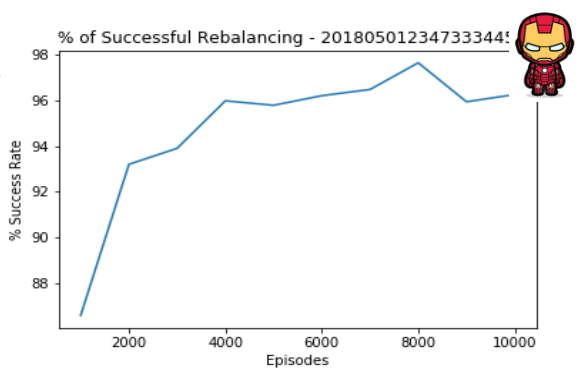
What are our results?

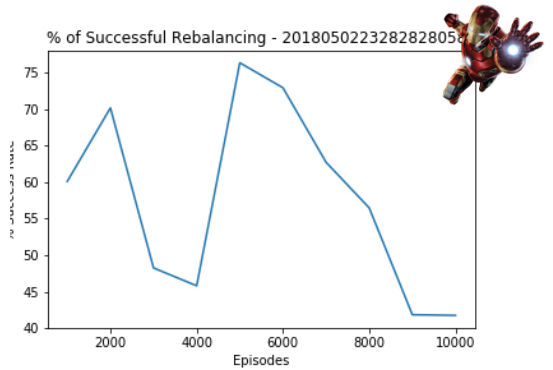
How does each learning method improve over time after more interaction with the environment?

- Some methods shown improvement in success rate (keeping bikes between 0 and 50 at hour 23)
- Some variation in performance consistency

Evaluation Metric: % of episodes where final bike stock is in range [0, 50]







Next Steps

- Augment the solution to rebalance bicycles across multiple stations
- Expand the action space from just removing bikes to adding bikes
- Enable parallel computing to shorten training time
- Develop better tuned forecasting and Deep Learning models
- Apply trained model to different (future) dates