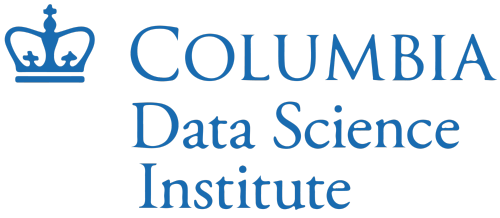

Capstone Final Report - Fall 2020

Reinforcement Learning for Trading

Mariem Ayadi,
Shreyas Saiprasad Jadhav,
Benjamin Weiss Livingston,
Amogh Mishra,
& Kevin Womack

mentored by
Naftali Cohen & Srijan Sood
JP Morgan AI Research

supervised by
Adam Kelleher

Preface

The purpose of this report is summarize the final results of the group's Capstone project; it will synthesize the statistical, computational, social, and ethical challenges involved in constructing an artificially-intelligent stock trader using classic Q-Learning methods from Reinforcement Learning.

1 Introduction

1.1 Background & Motivation

1.1.1 Reinforcement Learning

Reinforcement Learning differs significantly from Supervised Learning and Unsupervised Learning, and is probably closer to a more classical conception of artificial intelligence. In Reinforcement Learning, the algorithm acts much like a human does from birth: it tries different actions out, makes mistakes, sees what works and does not work, and then adjusts its actions accordingly.

In Reinforcement Learning, we call our learner an **agent**. In each time step, this agent chooses an **action** and is rendered a **reward**. The agent's goal is simple: to maximize its rewards over time by optimizing its actions given its state.

Imagine this as an automated vacuum cleaner trying to learn the most efficient route. Successfully picking up a mess could be considered worthy of a reward for the vacuum, whereas running into a wall could be considered worthy of a penalty (or more specifically, a negative reward). Running out of battery before returning to the dock, getting stuck in a corner, or getting attacked by a pet could all be considered worthy of an extremely large negative reward; these would help the agent learn not to make the same mistakes again.

The secret sauce behind these decisions is the **state** the agent finds itself in. The agent does not treat every situation the same - it makes decisions based on a quantitative summary of its surroundings. For the vacuum, this could be several things: how far it is from the wall, how much battery it has left, how many feet away the pet is, how full its bag is, summarized sensory data from a camera, and so forth.

The agent's goal is to ultimately develop an optimal **policy**: essentially, figuring out which action makes sense in each state. This is accomplished through a process of *exploration vs. exploitation*.

In **exploration**, the agent randomly tries out different actions, even if it does not believe them to be optimal. This can be thought of as childhood: the best way for humans to learn what works well is to make mistakes and try out different things. The agent learns the same way. Without giving different routes a try, the vacuum would be stuck making a random guess of the best route.

In **exploitation**, the agent follows its optimal policy. It takes what it believes to be the optimal action in the given state. Think of exploitation as imagined adulthood: we have learned what we *believe* is correct, and we do it. In the vacuum's case, it follows what it believes is the best route based on its surroundings, however optimal that route is, and stops trying new strategies for navigating the house.

However, exploitation is *not* necessarily ideal. Perhaps what the agent believes to be optimal is not optimal; after all, whether you are a 10-year-old, a 65-year-old, a one-day-old automatic vacuum, or a three-year-old automatic vacuum, there is usually more learning to do. Your conception of the world around you may not be entirely correct, or the world around you may change over time.

This is where the challenge of *exploration vs. exploitation* comes in. Ideally, an agent would explore until it learns optimal policies for the various states it can find itself in, and then it exploits its knowledge. However, this is an imperfect process that requires careful management.

1.1.2 Why Reinforcement Learning Is Ideal For An Investment Bank (JP Morgan)

Generally speaking, a financial institution's algorithmic trading approaches would ideally be explainable, transparent, and controllable. It is helpful to avoid instituting a black box algorithm - one that does not meet these requirements - for several reasons.

First of all, utilizing an algorithm that cannot be explained or scrutinized leaves a financial institution unable to ensure it does not conduct illegal activity. Insider trading and market manipulation (such as front running) are serious prohibitions in a trading environment, and a complex algorithm that acts of its own accord could risk violating these rules, depending on what information it is considering. It could also take illogical actions that *seem* to make sense based on results, but are actually based on a poor interpretation of the data and are not particularly robust.

With no understanding of the algorithm's inner workings, JP Morgan might be unable to conclusively state that it is operating within the rules, or justify to stakeholders that it is truly trading logically. Many institutions use algorithmic trading approaches that may not quite meet these requirements, but the design of Reinforcement Learning makes it uniquely poised to achieve high performance while attaining these ideal characteristics.

Reinforcement Learning maintains a simple formulation of the trading problem, and this permits the algorithm to maintain a distinct sense of transparency. The state of the market can be quantified in a direct, simple manner (although the design decisions in how to distill the state of the market through indicators can be tricky), the actions are as simple as buying and selling certain amounts of various securities, and the reward can be as simple as how much money is being made and how volatile (or more specifically, how non-volatile) a portfolio is.

Thus, Reinforcement Learning allows for a simple-yet-powerful formulation of the trading problem that avoids creating a black box, making it ideal for a financial institution that requires transparency.

One other practical consideration is that the state of the market is not necessarily independent of JP Morgan's actions. JP Morgan is a large financial institution that can make major waves in a market, and thus any algorithm considered must allow us to account for the potential *Market Impact* of any moves that JP Morgan makes.

Reinforcement Learning is ideal here as well, as it considers both current and future consequences of the agent's actions, and does not make its decisions rigidly in a vacuum. More generally speaking, it can dynamically adapt to a changing environment in a way that Supervised Learning and Unsupervised Learning would struggle to - and this makes it an ideal candidate for the trading environment.

Many of these considerations will be discussed further in the discussion of ethics later in this report. The general idea here is that a good stock trading strategy should be explicit and manageable. "Buy low, sell high" is the classic solution. We will explore more similar stock trading strategies in our exploration of benchmarks later in this report, but any stock trading strategy JP Morgan explores would ideally follow this general blueprint: intelligent but eminently explicit. Thus, Reinforcement Learning is an ideal candidate here.

1.1.3 Q-Learning

Q-Learning will serve as our chosen Reinforcement Learning algorithm. Q-Learning is a *model-free*, *off-policy* RL algorithm. Many Reinforcement Learning algorithms aim to learn a policy

directly, and/or directly predict how the environment will shift. Q-Learning does neither of these things: it simply *estimates the impact* of them.

Q-Learning models the **expected reward** for each action in a given state, by balancing **current reward** and **expected future reward** (which is done using the *Bellman Equation*, shared later). This is a simple, beautiful alternative to trying to build a complex understanding of the environment. Rather than going to the trouble of modeling *causes*, the agent focuses on *outcomes*.

Think of this as the robot vacuum trying to avoid an attack by the pet. Q-Learning does not require any understanding of where the pet goes at what time. Rather, if the vacuum is consistently attacked by the pet when trying to clean the bedroom at 3 p.m., it would (theoretically) learn something much simpler such as "turning left at 3 p.m. is a bad idea". This often makes for a much more parsimonious approach.

In classic, simple Q-Learning (what we generally refer to as "Q-Learning" in the remainder of this report), the state and action space is discrete. For the vacuum, a specific Q-Learning state would be something such as "it is between 3 and 6 p.m., I am in the living room, and my bag is 50-75% full", and the action space could be "turn left, turn right, go straight, or turn around".

The final result is a **Q-Table**, where the rows are states and the columns are actions, and each cell contains a **Q-Value**. Each Q-Value is the estimated reward for that action in the given state. If we are exploiting in the given time step, we select the optimal action by choosing the largest value in the row - in other words, the action that yields the highest expected reward.

1.1.4 Deep Q Networks

Often times, simple Q-Learning is *too* simple for a given situation. Perhaps we want to consider our current state more precisely - in the robotic vacuum example, the vacuum may want to know *exactly* how much battery it has left, rather than a vague quantization of said value.

This creates a challenge, though. Suddenly, the state space becomes infinite, so the agent cannot learn an optimal policy for each state; it must *approximate* it. This is where **Deep Q Networks (DQN)** comes into play. DQN approximates the values of a Q-table using a Neural Network.

Examples of this in practice have been seen in the works of DeepMind and OpenAI, where it has been shown that Deep-RL has the ability to outperform humans in games. This approach is particularly useful and applicable when the state-action pairs are continuous, such as in self-driving cars.

In order to use DQN in the setting of this project, the input would be the state at the current time step and the output would be the approximate Q-Values of the allowed actions. DQN plays a pivotal role in our problem definition, since for a given time t, we may want to incorporate more variance in our states through *continuous* (rather than discrete) states and experiment with DQN to produce superior results.

1.2 Problem Statement

Our goal is to use Q-Learning and DQN to allow an artificially intelligent stock trader to navigate the market. Specifically, our agent will manage a portfolio containing two assets: JPM stock and cash. On any given day, it will be allowed to choose between maintaining its current split of cash

and stock, buying more stock, or selling stock.

Our goal will be to build Q-Learning and DQN agents that maximize their returns in the market while *also* mitigating volatility (although admittedly, the latter did not factor into the agent's training, so our ability to achieve low volatility will be limited in this experiment). We will measure their success against very simple, run-of-the-mill trading strategies. Should our strategies develop a healthy mix of improved returns and/or mitigated volatility over these baseline strategies, we will consider them a success.

This is admittedly a challeging task - "beating the market" is notoriously difficult. It is often simply a product of luck more than skill, and it is challenging to define *exactly* what success is. With this in mind, we will not view success and failure as binary; we will focus more on methodology, explainability, and insight than on raw results.

1.3 Existing Work & Literature Review

Admittedly, building a basic Q-Learning trading framework from scratch is a highly-involved process. This being the case, simply getting to the point of creating a functioning, explainable framework for Q-Learning and DQN encompassed the bulk of the project given the timetable available. This literature review provides a "horizon" to look towards: a few papers that provide a roadmap for what is possible to accomplish in trading with Reinforcement Learning given enough time and resources, and what may not be.

Théate and Ernst [10] laid out a framework for DQN for trading, built around the Sharpe Ratio (to be discussed later - essentially returns divided by volatility). In this experiment, they found that DQN did not perform demonstrably better than simply buying stock and holding it - which raises the question of whether the reward-maximization problem can be reduced much further than "stock is generally worth more than money, so buy stock". This little issue will rear its ugly head later again in this report, and it's a decidedly nontrivial potential limitation of the Reinforcement Learning solution.

Meng and Khushi [7] more recently completed a literature review that found that Reinforcement Learning showed promise in the stock trading arena, but it still struggled to adapt to the realities of the market: commissions / transaction costs, volatility, changes in the market between the training and testing periods, etc. This also foreshadows some challenges we will see, and further explains why our process is more focused on methodology and explanation than purely generating strong results.

A more recent attempt at a robust framework was attempted by Ponomarev, Oseledets, and Cichocki [8], who used an Actor-Critic method (another Reinforcement Learning approach) coupled with Recurrent Neural Networks (RNN) utilizing Long Short-Term Memory (LSTM). This is well beyond the scope of this project, which will consider methods no more complex than simple DQN. However, that paper leaves out a critical piece that our report *will* consider: direct numerical comparison with benchmark strategies of simple stock trading. This is a crucial aspect of evaluating the marginal benefit of Reinforcement Learning.

This literature review can be summarized as follows: profitability is not terribly difficult to achieve, hence the challenge is more complex than making money. After all, stock is generally more profitable than cash, therefore buying as much stock as possible will result in a profit. The *real* question is if our algorithms can make more money than simple stock trading strategies - or at least make a comparable amount of money while mitigating portfolio volatility. This review suggests that this may be quite a challenge.

1.4 Overall Approach

The team began by collecting the data, constructing code to evaluate benchmark strategies and metrics, and then began building the algorithms. An initial framework was laid out, consisting of an automated visualization / metrics pipeline and a simple API for a stock trading strategy. This created a foundation for measuring the success of the team's Reinforcement Learning strategies.

The next challenge was to select an RL algorithm. This involved mulitiple weeks of research and consultation with JP Morgan. Q-Learning, SARSA, and Policy Gradients were all strongly considered, but Q-Learning was chosen due to its popularity and simplicity.

Following this groundwork, this project required an intense design process. Selecting indicators for states and metrics was a non-trivial process that required weeks of research, experimentation, coding, and collaboration. Admittedly, this process has been iterative: the group made significant progress, but there is always room for further refinement.

Once the exploratory data analysis framework was complete, the team moved into modeling. This was an immensely challenging process, because success was difficult to define. Additional benchmarks were added (namely the "Buy Always" strategy), and new questions arose.

Was it acceptable to have a Q-Learner that bought nearly all the time (since JPM stock is generally a higher-value asset than cash), or is it optimal to have a more balanced set of actions? How do we quantitatively separate a learner that is "lucky" from a learner that is intelligent? Should rewards be rendered as holdings appreciate, or only once the held stock is sold, given that both are fair approximations of the real world?

Admittedly, many of these questions stayed open, and became as much a part of the research as evaluating the results themselves. Many of them will be addressed in the sections that follow. Continual collaborative improvement of the algorithm was the backbone of the team's work in the later phases of the project. This included the extension of simple Q-Learning to DQN that took place in the final few weeks.

As this improvement took place, the team also placed a heightened focus on explainability. The question was not simply how the RL algorithms were performing, but *why* they did what they did. Could we see patterns in the algorithm's actions, and perhaps discern an artificially intelligent trading strategy from it? This process produced several fascinating visualizations: Q-Table heatmaps, state transition matrices, views of actions over time, and more.

This work continued until the final hours of the project as this report was authored, and is the culmination of copious research, collaboration, imagination, and persistence. There is always room for improvement; this work methodically scratches the surface of what is possible in stock trading with RL.

2 Data & Processing

2.1 Data

The underlying data for this project was extremely simple: it was nothing more than the daily stock price for our chosen stock to manage, JPM.

To be more specific, JPM's daily *Adjusted Closing Price* for each trading day was considered. This is a version of a stock price that accounts for corporate actions such as splits and dividends,

providing more stability over time in price movements than the raw closing price.

Date	JPM Adj. Close
Feb 1, 2018	$107.47
Feb 2, 2018	$105.09
Feb 5, 2018	$100.05
Feb 6, 2018	$103.09
Feb 7, 2018	$103.79
Feb 8, 2018	$99.20
Feb 9, 2018	$101.19
Feb 12, 2018	$102.75
Feb 13, 2018	$103.38
Feb 14, 2018	$105.78
Feb 15, 2018	$106.22
Feb 16, 2018	$105.45

This data was accessed using the Yahoo Finance's Python API, primarily through the package *yfinance*. While this data is quite simple, the calculations done with it were complex, and will be detailed throughout this report.

Our Reinforcement Learning algorithms were built by *training* them in an in-sample period, and then *testing* the trained algorithms in an out-of-sample period. The in-sample (training) period was defined as every trading day from 2007 to 2016, inclusive. The out-of-sample (testing/evaluation) period was defined as every trading day from 2017 to 2019, inclusive.

2.2 Preprocessing

Preprocessing was not a major consideration for our pipeline. A large amount of processing was done inside the algorithm to generate daily states (e.g. calculating each day's return and Bollinger Band Percent, among other things), but this was done in parallel to training and testing. No further alterations were made to the aforementioned adjusted closing prices.

However, this is certainly not to say that several daily indicators were not considered. Rather, all states, indicators, and metrics were calculated directly from the daily stock price.

3 Analytical Methods

3.1 Methodology

To begin this section, we will outline the "rules of the game": the guidelines for how an agent is allowed to sell or trade stock. This framework is completely implementation-agnostic; it simply lays out the trading environment.

To begin the testing (out-of-sample) period of 2017 to 2019, the stock trading "agent" (deploying its chosen strategy) will be given a portfolio worth $100,000 on the first day of 2017. All strategies start with 0 shares and $100,000 in cash. Hold, on the other hand, only holds stock and never touches cash (which, as we will discuss in our results, gives it a somewhat unfair advantage that should be noted).

On a given trading day, the agent will be allowed to take one of three actions: **buy**, **sell**, or **hold**. If the agent chooses **hold**, nothing changes. If the agent chooses **buy**, it will buy a set number of shares. If the agent chooses **sell**, it will sell that same set number of shares. The number of shares the agent can buy or sell on a given trading day is set to a constant amount of shares (in our experiment, this number of shares was 30).

It should be noted that buy and sell actions were only permitted if an agent had enough stock to sell or cash to buy. If the agent opted to sell but had fewer than 30 shares, or opted to buy but did not have enough money to purchase 30 shares of stock, it was forced to hold.

3.2 Benchmarks

The success of the team's Reinforcement Learning strategies were measured against five simple stock trading strategies, which provide a baseline level of performance as a gauge for evaluating success.

It is important to note that these methods are *not* trained, since they employ simple strategies that are not tuned using machine learning. Unlike the Reinforcement Learning algorithms, these strategies involve no information from the in-sample period (2007 to 2016) - they simply carry out trades in the out-of-sample period (2017 to 2019) according to their individual simple strategies.

- **Hold Consistently:** do nothing except hold stock (and zero cash) for the entire trading period, regardless of what happens.
- **Random Action:** randomly choose between 'buy', 'sell' and 'hold'.
- **Rule-based Action:** the agent chooses 'sell' if yesterday's return was positive, 'buy' if yesterday's return was negative, and 'hold' if the stock price saw no change on the previous trading day.
- **Ordinary Least Squares (OLS):** use simple linear regression to predict the upcoming day's return, using the last five trading days as data; 'sell' if the predicted return is negative, 'buy' if it is positive, and 'hold' if neither.
- **Buy Always:** buy stock every day as long as there is still money left to spend. This benchmark is designed based off the well-known principle that, in the long-term, stock is *generally* worth more than cash, meaning any intelligent stock trading strategy should yield better results than simply buying as much stock as possible (and should not be labeled a success simply because it buys more stock than cash).

It should be noted that the "Hold" benchmark strategy *technically* violated the rules of the game as we outlined them earlier, so its performance should be analyzed with a grain of salt.

3.3 Metrics

3.3.1 Goals of Metrics

The relative success of the strategies (both Reinforcement Learning and benchmarks) will be measured based on a wide range of analyses. The goal of the metrics is simple: to measure how well the trading strategy generates improved returns *while also* mitigating volatility.

If a Reinforcement Learning trading strategy generates returns that are gauged to be no better than the baselines, it is fair to question its worth. Similarly, if it generates improved returns, but with an in-kind increase in volatility, it is reasonable to doubt whether or not the increased returns are worthwhile.

We will use our metrics to determine the relative returns, relative volatility, and the relative volatility-normalized returns of each strategy. The final item will use classic indicators that implicitly weigh the returns of a strategy's managed portfolio against that portfolio's volatility.

3.3.2 Metric Definitions

For the purposes of this project, a stock's **return** can be calculated as follows:

$$Daily\ Return[t] = \frac{Portfolio\ Value[t]}{Portfolio\ Value[t-1]} - 1$$

With that in mind, the metrics and associated hypothesis tests are as follows:

1. **Buy, Sell, and Hold percentages:** the percentage of the time each action was taken under the strategy
2. **Mean Daily Return:** the arithmetic average of multiplicative daily returns; this is expressed as a percentage for readability purposes
3. **Volatility (Standard Deviation of Daily Return):** captures the volatility of a stock (this is calculated on the raw return, not the percentage)
4. **Information Ratio:** measures and compares the active return of an investment compared to a benchmark index relative to the volatility of the active return defined as follows: $IR = (Portfolio\ Return - Benchmark\ Return)/Tracking\ Error$, where the benchmark return is the S&P 500, and tracking error is defined as the standard deviation of the difference between the stock's daily return and the benchmark's daily return. We note that these *tend* to be negative, due to the fact that they are measured against full investment in the S&P 500, which generally posts a higher return than a more even cash-stock split in the cases of most stocks (which is our default for testing/evaluation). What matters for now is the relative value of these IRs - not their raw values.
5. **Sharpe Ratio:** measures the performance of an investment compared to a risk-free asset, after adjusting for its risk defined as: $Sharpe\ Ratio = (Rp - Rf)/\sigma$ where Rf is the risk-free return, defined here as the U.S. five-year treasury rate for our problem. Here, the denominator is defined as the standard deviation of the difference between the stock's daily return and the risk-free asset's daily return.
6. **Average Return After Buying, Selling, and Holding:** a measure of the strategy's situational effectiveness, examining mean percent return the day following each strategy being deployed; note that this only measures *immediate* reward and not future reward, so it is of limited utility
7. **T-test for Significance in Difference of Returns:** in our comparison, we perform one-sided t-tests used to determine if the mean of the daily returns from two separate trading strategies or assets are significantly different statistically (we will assume equal variances for simplicity, which functionally did not have much impact on these tests). We do this between each strategy and the benchmark return (the S&P 500) defined above. The result consists of two portions: (1) a p-value, which if less than .05, would traditionally lead to the assumption that the difference is significant, and (2) a symbol to denote if the stock performs better than the benchmark return ($>$) or worse ($<$). Differences found to be significant are noted with a '*'.
8. **Levene Test for Significance in Difference of Volatility:** used to determine if the volatilities in the returns of the trading strategy and the benchmark return (the S&P 500) are significantly different. The result and p-value are reported in a similar fashion to the t-tests.

In addition to these methods, we will present visualizations that track the portfolio's returns, daily actions, and chosen proportions of stock versus cash over time, as well as the final appearance of the stock's Q-Table.

While these metrics are valuable, often times a common-sense check of the visualizations is equally as valuable. These alternative evaluations are critical. For instance, as mentioned numerous times thus far, many stocks (such as JPM in our out-of-sample period) are worth more on average than cash. Ergo, a strategy that buys stock wantonly until it runs out of money will inherently be more effective than most more balanced strategies. However, this does not demonstrate any real sense of learning, intelligence, or reacting to surroundings, even if it beats multiple benchmark strategies in every single metric.

With this in mind, it is important to seek out a Reinforcement Learning strategy that both performs well in the above metrics and testing, *and* exhibits a pattern of actions that demonstrate an intelligent, measured, reactive strategy. Granted, this is a steep goal, essentially akin to asking a stock trader to be successful without being lucky, heedless, or simplistic. We acknowledge that this is a lofty expectation that may not be realistic, and that a successful strategy should exhibit *some* of these characteristics - not necessarily all of them entirely.

3.4 Outlining the Q-Learner

3.4.1 State Space

The simple Q-Learner's state space was comprised of two simple elements, namely:

- The ratio of the chosen stock's Adjusted Close to its 3-day Simple Moving Average a.k.a. its **SMA Ratio**
- **Bollinger Band Percent** – an indicator for quantifying the value of the stock asset relative to the upper and lower Bollinger Bands

$$\%B = (Price - LowerBand)/(UpperBand - LowerBand)$$

- **Cash Percentage** - an indicator that represents the percentage of the cash held to the total portfolio value at each time step.

  At each time step,

$$\%Cash = Cash/(Cash + Shares * CurrentAdjustedClosePrice)$$

Per the suggestion of the team mentors, the team also explored an indicator for volatility called the **Average True Range (ATR)**. This metric is calculated by taking the maximum of:

$$(\ Current\ day's\ high - the\ current\ day's\ low)$$
$$(Current\ day's\ high - the\ previous\ day's\ close)$$
$$(Previous\ day's\ close - the\ current\ day's\ low)$$

...and creating a 14-day exponential moving average from this value. A larger ATR indicates higher trading ranges and therefore increased volatility. Low readings from ATR are generally consistent with periods of quiet or uneventful trading.

The team also explored utilizing a metric known as **Market Relative Daily Return**, a measure of the stock asset's return relative to the S&P 500. The idea is that if the asset's return is not better than the general market performance, then one may conclude that there is no particular advantage to owning shares of this stock. This indicator is defined as:

$$\%MRDR = \frac{(\ Day\ N\ Stock\ Adj\ Close\ )\ /\ (\ Day\ N - 1\ Stock\ Adj\ Close\ )}{(\ Day\ N\ S\&P\ 500\ Adj\ Close\ )\ /\ (\ Day\ N - 1\ S\&P\ 500\ Adj\ Close\ )}$$

While the ATR and MRDR indicators were explored, implemented, and lightly tested in the weeks leading up to this report, the team opted not to fully integrate them into the state space for the agent.

Each of these indicators was arranged into a certain number of discrete quantiles based on patterns the in-sample (training) period (the number of quantiles varied by indicator). The borders of these quantiles for some states will be visible in the Q-Table visualizations shared later in this report.

A precise "state" was defined as the current mix of quantiles: a specific state would be, say, currently being in the third quantile of SMA Ratio and the fourth quantile of Bollinger Band Percentage based on the stock's recent adjusted close prices.

3.4.2 Action Space

The definition of the simple Q-Learner's action space was much simpler. As mentioned earlier, the agent can choose between **buy**, **sell**, and **hold**. The agent aimed to learn which of these three actions was optimal in a given state (i.e. it learned which of these three actions yields the highest expected reward given the mix of quantiles the agent finds itself in).

3.4.3 Rewards

To keep the concept simple, the reward was the change in the portfolio value between two consecutive time steps depending on the action taken by the agent.

When buying or selling, the agent transacts a fixed amount of shares (30). The reward for a specific trading day (i.e. a time step) was defined as the following based on the different actions.

$$Shares\,traded\,per\,action\ :\ 30$$

Action: BUY

$$Cash[t+1]\ = Cash[t]\ -\ 30\ *\ AdjustedClose[t]$$
$$Shares[t+1]\ = Shares[t]\ +\ 30$$

Action: SELL

$$Cash[t+1]\ = Cash[t]\ +\ 30\ *\ AdjustedClose[t]$$
$$Shares[t+1]\ = Shares[t]\ -\ 30$$

Action: HOLD

$$Cash[t+1]\ = Cash[t]$$
$$Shares[t+1]\ = Shares[t]$$

$$Reward[t] = Cash[t{+}1]{+}(Shares[t{+}1]{*}AdjustedClose[t{+}1]){-}(Cash[t]{+}Shares[t]{*}AdjustedClose[t])$$

This is where the concept of expected future reward becomes critical. The immediate reward for the agent's move was defined as the return the next trading day, but long-term success mattered too - i.e. what happens to the holdings in the *future*, in the days and weeks that follow. The onus to estimate these future rewards lies on the Q-Learner.

One addition worth noting is that in some experiments, a *commission* and/or *sell penalty* were added to these calculations. These will be discussed in the next section.

3.4.4 Parameters Guiding Q-Learning

In this section, we will lay out some of the key hyperparameters that guide the Reinforcement Learning process. Some of them will be specific to the stock trading environment.

These parameters only pertain to training in the in-sample period, unless otherwise noted. The Q-Table was treated as a static object during testing in the out-of-sample period, and was not updated further.

Epsilon refers to the "exploration rate". It is defined as the percentage of time an agent should take a random, non-optimal action instead of taking what it believes to be an optimal action. Setting epsilon to a high value early is key - otherwise, the agent will pay too much attention to the Q-Table's initial randomization.

On the other hand, we do not want to continue taking random actions frequently once the agent has developed a keen sense of expected rewards and has learned a good policy. For this reason, we add an **epsilon decay** - an amount (between 0 and 1) that we multiply epsilon by every trading day so it shrinks.

The Q-Learner will train for a certain number of **episodes**. It will not simply iterate through the in-sample (training) period once (unless the number of episodes it set to 1). It will do this over and over, learning progressively, with epsilon forcing random actions on different days each time passing through the data, leading to slightly different results each time and progressive learning.

We also add a *learning rate*, which we call **alpha**: the proportional weight we put on our updates to the Q-Table. Say our alpha is .01. If we take a certain action in a certain state, we calculate the new value for that Q-Table cell as .99 times its existing value plus .01 times the current expected reward. In our experiment, we keep alpha at constant value of 0.1.

Gamma is another key element in the aforementioned Bellman Equation, referring to a *discount factor*: the proportion of the expected future return that should be added to the current return. It is also set between 0 and 1.

In some experiments, agents were charged a fixed-price **commission** for each buy/sell action. This applied in both training and testing, and was also levied upon the benchmark strategies.

Finally, in some experiments, the Q-Learning agent was charged a **sell penalty** if it decided that selling was optimal when it did not have any shares to sell. The goal was to disincentivize these "impossible" sales.

For reference, the Bellman Equation that is used to update the Q-Table in the simple Q-Learner can be viewed below (Sebastiano).

$$\underbrace{\text{New } Q(s,a)}_{\text{New Q-Value}} = Q(s,a) + \underbrace{\alpha}_{\substack{\text{New Q-Value}}} [\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\substack{\text{Discount rate}}} \overbrace{\max Q'(s',a')}^{\substack{\text{Maximum predicted reward, given} \\ \text{new state and all possible actions}}} - Q(s,a)]$$

3.4.5 DQN Process

Whereas Q-Learning is a tabular method for determining the optimal action strategy, an extension of it is to leverage Neural Networks for the same. DQN was invented by Mnih et al.[11]. Neural Networks are known non-linear function approximators. They are especially beneficial compared to a Q-learner when the number of unique states and actions becomes large, as maintaining a huge Q-Table becomes infeasible.

In DQN, we use two different Deep Neural Networks (DNN) composed of stacked dense layers. The choice of using two networks, both having the same network, is useful for stabilizing the network during frequent updates. The first network, the **Policy Network**, is used to estimate the Q-value for the current state **s** and action **a**: $Q(s, a : \theta)$. The second network, the **Target Network**, is used to estimate the Q-values of the next state s' and action a'. At every 10^{th} epoch, the weights of the policy network are copied to the target network.

4 Evaluation

4.1 Results

We begin by examining the quantitative performance of our various trading strategies, using the metrics outlined in section 3.3.2.

	Hold	**Buy Always**	**Random**	**Rule-Based**	**OLS**	**Q-Learner**	**DQN**
Buy %	0%	6%	36%	51%	50%	34%	33.5%
Sell %	0%	0%	31%	47%	46%	29%	33.5%
Hold %	100%	94%	33%	2%	4%	37%	33%
Sharpe Ratio	+0.73	+1.25	+0.73	+0.18	+0.52	+0.6	+0.38
Information Ratio	-0.34	0.9	-0.1	-1.48	-0.72	+0.31	-0.9
Mean Daily Return	+.081%	+.077%	+.048%	+.015%	+.034%	+.064%	.029%
Mean Return Days After Buying	N\A	+.087%	+.056%	+.022%	+.057%	+.051%	-.01%
Mean Return Days After Holding	+.081%	+.077%	+.038%	-.052%	-.186%	+.12%	+.071%
Mean Return Days After Selling	N\A	N\A	+.049%	+.011%	+.025%	+.008%	+.026%
Volatility	.007	.012	.007	.004	.009	.01	.006
T-test (For Returns)	>0.17	>0.18	>0.27	>0.44	>0.33	>0.26	>0.37
Levene Test (For Volatility)	>*0.0	>*0.0	>*0.03	<*0.0	>0.62	>*0.0	<*0.0

Figure 1: Table of trading results across strategies

We can also examine a graph of daily portfolio values, as well as daily share and cash holdings (to examine the cash/stock splits) for each strategy.

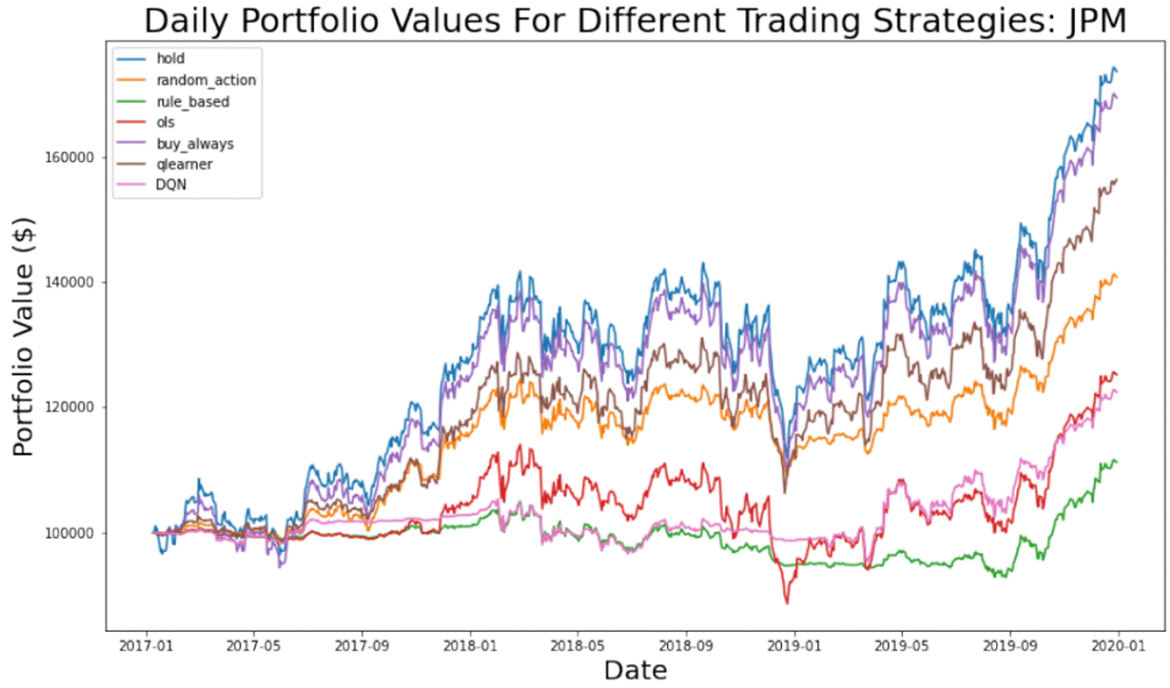

Figure 2: Daily portfolio values for each strategy

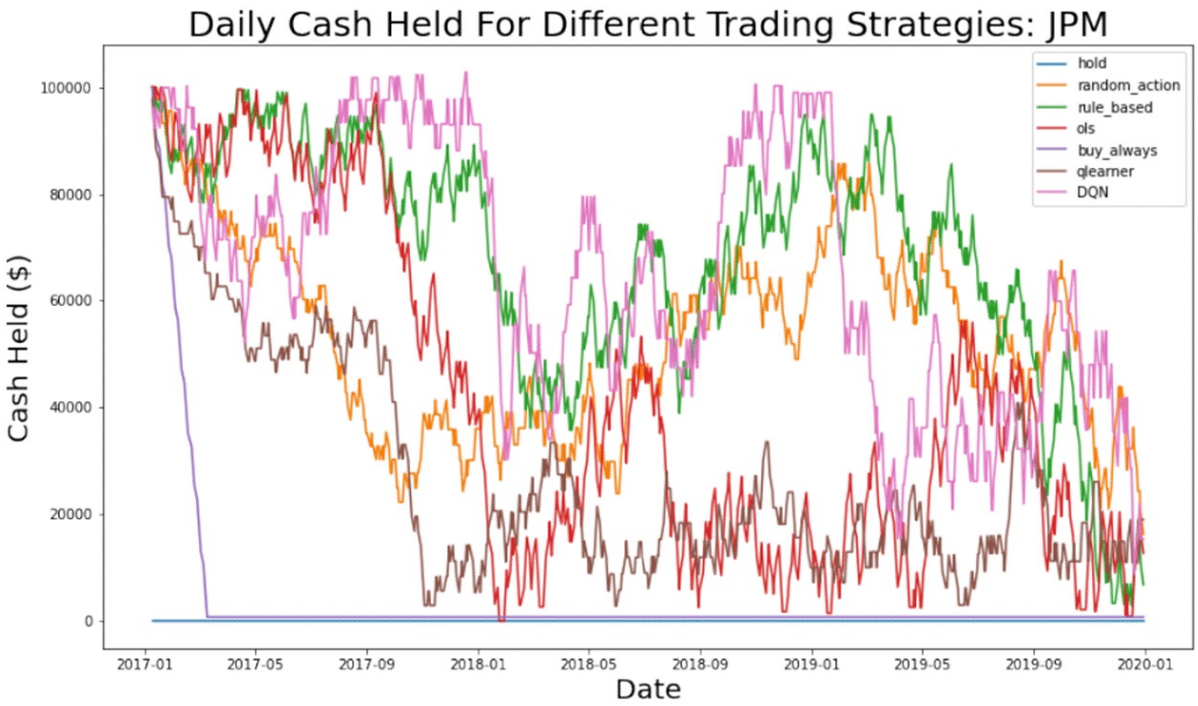

Figure 3: Daily cash holdings for each strategy (the more cash, the less stock)

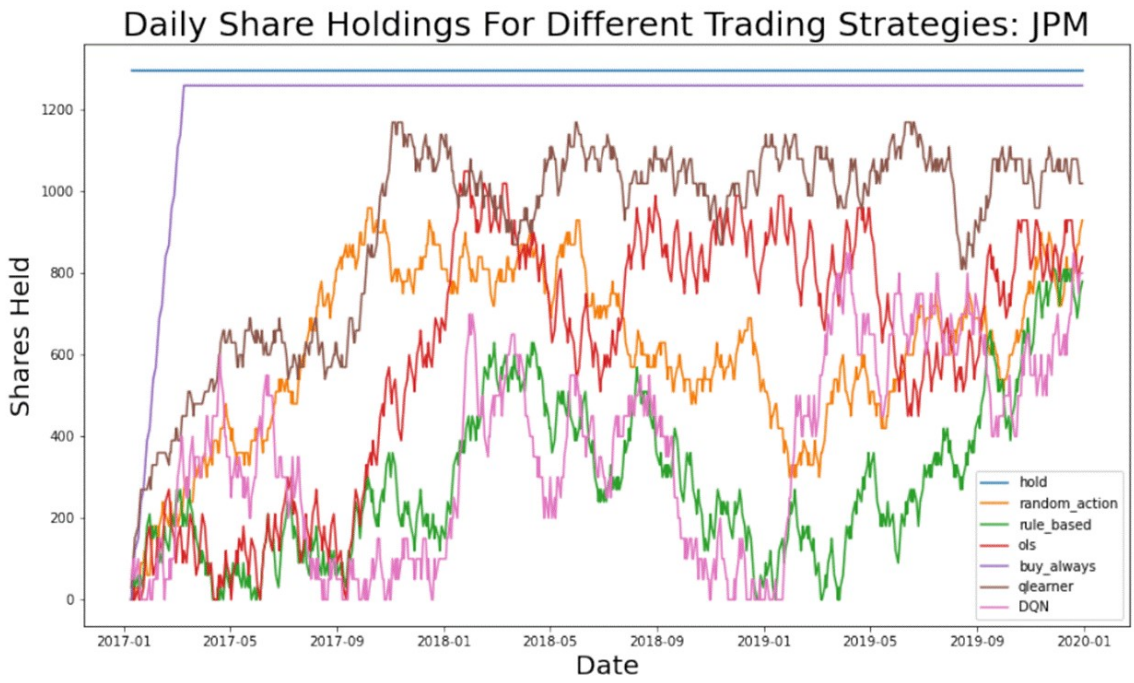

Figure 4: Daily share holdings for each strategy (the more stock, the less cash)

We glean a little more by examining heatmaps of a couple of the Q-Tables from the Q-Learner, for the Bollinger Band % and cash held as a percentage of portfolio value states. These are normalized by row and marginalized across all other states, meaning that the darkest value in each row indicates the generally-most optimal action for that state.

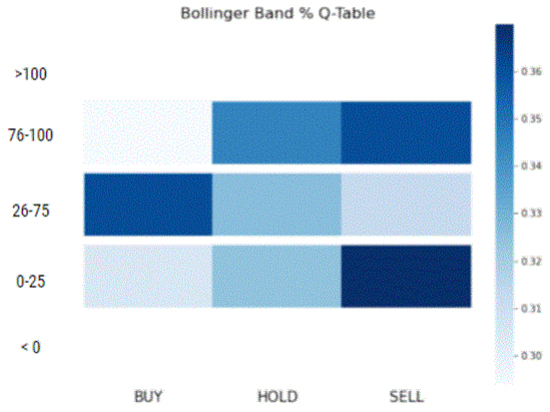

Figure 5: Bollinger Band % Q-Table

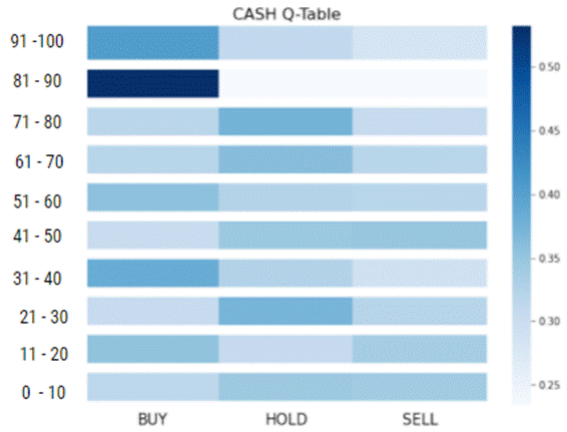

Figure 6: Cash Held % Q-Table

We can also see how the Q-Learner tended to transition between states, by taking a peek at a Markov-like transition matrix. Here, we can see a very slight propensity to stay in the same state.

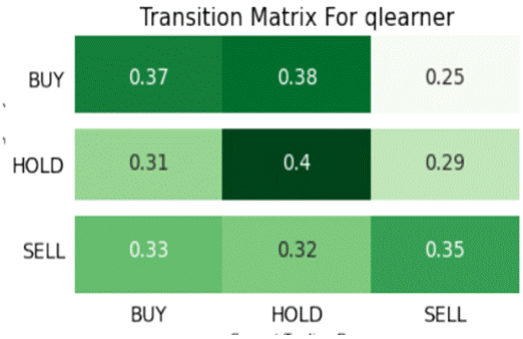

Figure 7: Transition matrix for Q-Learner (rows are action at time t-1, columns action at time t)

It is also worth noting that the team compiled a dashboard illustrating daily Buy/Hold/Sell actions in relation to daily portfolio actions and the Bollinger Bands, a screenshot of which can be viewed here. Please note that this Q-Learner was trained in a similar manner, but is not the same Q-Learner from the other figures.

Figure 8: Preview of Interactive Dashboard

4.2 Analysis of Results

Generally, neither the Q-Learner nor DQN appears to perform better than the other strategies, as we can see in Figure 1. This is not terribly surprising considering what we saw in our literature review, or based on common sense; considering the Efficient Market Hypothesis (which states that prices reflect all information on a stock), it would seem somewhat farcical to expect an algorithm based solely on price to beat simple trading strategies. Still, we can glean insight from these results.

The Q-Learner outgained every strategy other than Buy Always (which, as mentioned earlier, is a gold standard that is nearly impossible to surpass in this context) and Hold (which is equally difficult to beat because it only holds stock), as evident in Figure 2. It did so with a high increase in volatility, which is a clear downside that negatively affected its Sharpe Ratio and Information Ratio. Even random actions led to a better Sharpe Ratio. But the returns were there, even if they came with much volatility.

However, the Q-Learner did opt to buy more stock as we see in Figure 3 and Figure 4, which particularly makes sense. For some reason, though, it stopped buying as it gained more stock. This may be a sign that including the cash held amount as a state was a poor choice; perhaps the agent needs a little more freedom to buy when it holds a high number of shares. We see promise here; clearly, some alterations are needed, but we see the makings of a sensible strategy.

DQN was implemented towards the latter end of the project, and it seems that it would benefit from more tuning. Based on Figure 1, it seems to show little preference for any action, and performs worse than almost every other strategy. It appears to be highly reactive, oscillating between holding more cash and more stock much more wildly than the other strategies. This seems to suggest it is reacting to states attentively (which is likely a sign that the pieces are there and the framework is good), but perhaps needs further fine-tuning.

We notice that Figure 8's Q-Learner has a preferred repeated pattern for holding. This model is also, in the majority of cases, buying and selling at the appropriate moments; buying when the price is low and selling when the price is high. However, a few instances are erroneous and more importantly there are specific instances where the model could have waited slightly longer prior to a sell or buy action. It is important to note that the latter is generally only noticed at unprecedented price peaks.

One final point we want to highlight is that minor variations to the model can create significant shifts in action patterns and performance. One particular example we would like to expand on is epsilon, which involved a lengthy hyperparameter tuning process. Shifting epsilon by very small amounts made a significant impact, as did shifting epsilon decay, or even the random seeding of epsilon, because the placement of random actions on trading days with large returns (whether positive or negative) can drastically impact expected reward, and thus drastically impact a Q-Table. This instability is one of the most challenging aspects of applying Reinforcement Learning to stock trading.

5 Discussion

5.1 Summary

In short, unsurprisingly, the two stock-heavy strategies fared the best returns, but the Q-Learner was remarkably close, and DQN showed signs of reactivity. These results were far from stellar, but there are signs of intelligence that can be built upon.

The aim of this project was not to outperform the stock market, but to develop a framework for an explainable artificially intelligent stock trader. The methodology and outline we have established can set the foundation for this. The pieces of a successful strategy, "rules of the game", and limitations of the current framework are clearly defined here; future growth of DQN and other Reinforcement Learning strategies can stand on the shoulders of what has been done here.

5.2 Conclusion & Take Home Messages

1. Explainability is pivotal, and visualization helps tremendously in this regard
2. Environment representation is everything - without a proper distillation of a time step's true state, it is difficult to make a truly optimal decision (admittedly, this will have to involve more than just a stock price)
3. Applying RL to finance is extremely non-trivial, and even establishing a very basic framework is a significant undertaking

5.3 Future Work

Our algorithms dealt with trading fixed increments of a single stock; they merely create an explainable, theoretical framework for the wider concept of using artificial intelligence to trade stocks.

The first obvious extension of this algorithm would be to move into portfolio management: using Reinforcement Learning to not only buy or sell a single stock, but to manage how to balance holdings in many different stocks. A couple of the ideas discussed in the group involved a multi-agent approach (having a different agent for each stock and weighing their decisions) and rebalancing a portfolio continuously based on the expected rewards for holding each stock.

Another critical improvement would be extending the state space to involve more states. Admittedly, the states in our current iterations of the algorithm are simple indicators, and capture only a small dimension of a stock's movement. One key (and very popular) area the team would like to explore is sentiment analysis: using public chatter about a company to help predict movement's in its stock price.

Furthermore, a more complex and lengthy hyperparamater tuning process (perhaps paired with greater computing power) could allow the current iteration of the algorithm to perform significantly better, even without any changes in design. This could greatly benefit DQN, or even the simple Q-Learner (which has no fewer than ten key parameters that could be tuned depending on how one counts, meaning testing five different levels of each parameter in a grid search would require nearly 10 million runs of the algorithm).

6 Social & Ethical Considerations

Due to the nature of this project and the type of data it used, social and ethical considerations were not of particular concern. The project used entirely public data, and did not utilize any insider information about companies to inform its decisions.

That being said, some critical concerns could arise if these algorithms were to be productionalized and extended. Some of these were discussed earlier in the report, and we will add more detail below.

6.1 Market Impact

One concern raised during the development of these algorithms was *Market Impact*. This refers to the potential changes in the price of a stock that could occur due to our Q-Learner's trading activity. A good example of this is how the proliferation of High Frequency Trading has impacted information production (Baldauf & Mollner), but Market Impact need not be that complex - it could simply be, say, a large purchase of stock leading to an increase in the stock's price.

Were a large financial institution to make large, sweeping trades on a daily basis depending on this Q-Learner's state, it could actively disrupt the price of the stock. This could alter the price of the stock. Furthermore, this could allow another actor with awareness of the Q-Learner's structure to anticipate the trades, and short the stock. Q-Learning adjusts well to changes in its environment, but it is still susceptible to this.

This caution applies to all large stock trades, not simply those driven by Reinforcement Learning. That being said, while a large financial institution would be unlikely to implement a Reinforcement Learning algorithm relying heavily on states simple enough for an outside actor to guess them, it is important to avoid enacting a Q-Learner that makes large and predictable trades for the aforementioned reasons.

6.2 Insider Trading

While this project utilized only public data, any extension of this work would have to show caution in what non-public data it might utilize. Extending the project to involve company internals, purchased private data, or anything else non-public would need to involve protections that prevent insider trading.

This could happen very easily, and perhaps even without the algorithm's user knowing. Acquiring non-public data and placing it into an automated pipeline that feeds a state could very easily lead to inside information leaking into the algorithm (especially when it comes to something as high-level as sentiment analysis), and this may happen under-the-hood without the user realizing it.

For this reason, caution must be shown in selecting which non-public data might be used to extend these Reinforcement Learning algorithms.

6.3 Guardrails

As with any stock trading approach, human-designed guardrails should be enacted to prevent illegal activity (insider trading or otherwise), or activity that the company would consider unethical or reckless.

This could be as simple as preventing any trades that are prohibited by law for any reason, limiting dealings in companies that may offer a higher expected reward but do not meet the moral standards of JP Morgan, or limiting the amount of stock that can be invested in one company.

Granted, these concerns are unlikely to arise in the single-stock trading environment we have researched, but they are important considerations in extensions that consider managing a portfolio of a significant variety of stocks.

7 Contributions & Acknowledgements

Each member of the team made critical contributions to the success of this project, and these individual inputs are noted below.

Mariem Ayadi and **Shreyas Saiprasad Jadhav** worked as a team to adapt and operationalize the prototype of the Q-Learner. This involved a lengthy process of state creation, state quantilization, parameter tuning, and debugging. They served as both a first line and last line of defense in identifying and solving algorithmic issues, and attending to the development and performance of a robust Q-Learner. They also worked closely on the final integration of both RL approaches into a unified code base.

Benjamin Weiss Livingston spearheaded the generation of metrics, baselines, and visualization prototypes, as well as code that automatically generated all of the preceding items. He also served as a first line of support for the two teammates developing the Q-Learner, and he compiled the final report.

Amogh Mishra spearheaded the team's DQN research and development, extending part of the simple Q-Learning approach into a deep learning framework and weaving the different approaches together into a single analysis. He also built and managed the GitHub repository, and supported the development of metrics and baselines.

Kevin Womack served as team captain, managing collaboration between the project mentors and the team, as well as ensuring that all project deadlines were met. He also compiled the first two progress reports, developing a framework that laid the foundation for this final report, and he led a presentation to JP Morgan's AI Research team.

The team would also like to extend its heartfelt appreciation to Naftali Cohen, Srijan Sood, Thomas Spooner, and Zhen Zeng at JPMorgan, as well as Columbia University faculty advisor Adam Kelleher. Every one of them graciously took time out of their schedules every week to serve as enthusiastic collaborators and mentors. They offered invaluable energy, support, knowledge, and warmth during a particularly challenging fall for research, often outside of their normal working hours. This project owes its success to their investment.

References

1. Baldauf, Markus and Mollner, Joshua, High-Frequency Trading and Market Performance (January 14, 2020).https://ssrn.com/abstract=2674767

2. Sutton, R. S., & Barto, A. (2018). *Reinforcement learning: An introduction*. Cambridge, MA: The MIT Press. Retrieved 2020http://www.incompleteideas.net/book/RLbook2020.pdf

3. Nguyen, T.C. (n.d.) . SARSA vs Q-Learning. *Tran Canh Nguyen' notes*. Retrieved from https://tcnguyen.github.io/reinforcement_learning/sarsa_vs_q_learning.html

4. Liu, Qian (2019). Stock Trader with Q-Learning. *Medium*. Retrieved from https://medium.com/@nyxqianl/stock-trader-with-Q-Learning-91e70161762b

5. Shyalika, Chathurangi (2019). A Beginner's Guide to Q-Learning. *Towards Data Science*. Retrieved from https://towardsdatascience.com/a-beginners-guide-to-Q-Learning-c3e2a30a653c

6. Karunakaran, Dhanoop (2020). Q-Learning: a value-based reinforcement learning algorithm. *Medium*. Retrieved from https://medium.com/intro-to-artificial-intelligence/Q-Learning-a-value-based-reinforcement-learning-algorithm-272706d835cf

7. Meng. T and Khushi, M. Reinforcement Learning in Financial Markets (2019) https://www.mdpi.com/2306-5729/4/3/110

8. Ponomareva, E. S., Oseledetsa, I. V., Cichockia, A.S. Using Reinforcement Learning in the Algorithmic Trading Problem (2019) https://arxiv.org/ftp/arxiv/papers/2002/2002.11523.pdf

9. Tex Stack Exchange user Sebastiano (for Bellman Equation) https://tex.stackexchange.com/questions/491915/how-can-i-properly-write-this-equation-in-latex

10. Théate, T. and Ernst, D. An Application of Deep Reinforcement Learning to Algorithmic Trading (2004) https://arxiv.org/abs/2004.06627

11. Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533. https://www.nature.com/articles/nature14236?wm=book_wap_0005