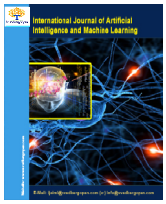
https://doi.org/10.51483/IJAIML.2.1.2022.61-70 ISSN: 2789-2557

SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https: / / www.svedbergopen.com /



Research Paper **Open Access**

Deep Q-Network Interpretability: Applications to ETF Trading

Bryan Yekelchik[1] and Zachary Coriarty[2*]

[1]P.C. Rossin College of Engineering & Applied Science, Lehigh University, 19 Memorial Drive West, Bethlehem, PA 18015, United States. E-mail: bxy@comcast.net

[2]Lehigh University, 19 Memorial Drive West, Bethlehem, PA 18015, United States. E-mail: zackcor27@gmail.com

Article Info

Volume 2, Issue 1, January 2022
Received : 10 November 2021
Accepted : 25 December 2021
Published : 18 January 2022
doi: 10.51483/IJAIML.2.1.2022.61-70

Abstract

We present an interpretability infrastructure for Reinforcement Learning (RL) based trading strategies. For all audiences to be able to answer the question of 'how does the algorithm work?', we provide a visual and user-friendly approach, in contrast to a more quantitative approach. This allows not only a technical audience to consume insights derived from an RL-based trading approach. In this application, we introduce a three module approach in understanding value-based RL, specifically Deep Q-Learning. We demonstrate this infrastructure and possible derived outcomes of using this infrastructure when applied to trading a market ETF in a given time interval.

Keywords: *Deep Learning, Reinforcement Learning, Artificial Intelligence, Machine Learning, ETF Trading, Visualization, Dashboard*

1. Introduction

In recent years, the finance industry has become no stranger to the applications of Artificial Intelligence (AI) learning. Successful applications have come in the form of automating credit request models, robo-advisors, portfolio allocation, decision making, and algorithmic trading. Specifically within the trading space, and for fairly liquid products, algorithms have taken over the market during stable times.

With machine learning models making themselves ever more prominent in the last decade and with computational power becoming more affordable, there has been a rise in the use of energy-intensive, complex models that are known as “black-box algorithms,” where even the creators of these algorithms do not have a full understanding of how inputs are manipulated in producing an estimated output. Lack of insight into the “how” can be problematic within the context of financial markets. Whether a credit desk is trying to understand if an algorithm is biased toward a certain demographic, or a portfolio manager is trying to understand what an algorithm is doing given certain market conditions, having insight into the “how” is critical.

* *Corresponding author: Zachary Coriarty, Lehigh University, 19 Memorial Drive West, Bethlehem, PA 18015, United States. E-mail: zackcor27@gmail.com*

With the question of "how" in mind, in this paper we introduce a visual methodology to help answer the "how" for a black box AI algorithm within the trading space. Specifically, we investigate a Reinforcement Learning (RL) based trading strategy, and provide a visual interpretability methodology. RL has been around for sometime now, famous for being trained as an agent to play various Atari games way above human level performance. The model used to achieve above-average performance for these games, and discussed in this paper is a model-free RL algorithm, know as Deep-Q Networks (DQN).

Introduced in the early 1990s, Q-learning utilizes an agent that looks to maximize the discounted future reward. In the context of trading, the agent looks to maximize Profit and Loss (P&L) in a given 'environment' (the market). Through iterative exposure, 'training', the agent learns from its previous exposures to maximize reward. As described by Wang *et al*. (2019), DQN interpretation is critical in understanding if the agent is taking an action because it is "intentional or is it just random?" Questions such as this and the "how?" cannot be directly observed from simple calculations of model outputs.

In this work, we propose a modular visualization system for interpreting DQN in the context of trading. This system helps users of DQN-based trading strategies to understand how inputs map to outputs, in both the training and go-live phases of the algorithm, through friendly visualizations in a customizable dashboard (Figure 1). Further, this infrastructure can help machine learning developers diagnose and improve an established model by easily showing where the agent may not be doing what a developer anticipated for a given state of the environment. In summary, the add-value of this work includes: (1) A modular visualization system for interpreting DQN; (2) A method for generating synthetic actions for states that do not occur naturally in the environment of the agent. This can help to address the question of "What would the agent do if a certain state would occur?"

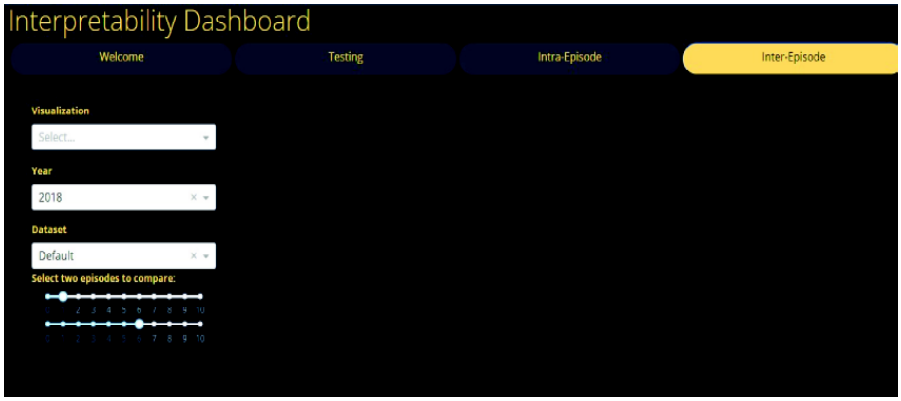


Figure 1: Control Center and Tab Structure

1.1. Technical Introduction

In engineering a visualization system for DQN models, we first train a DQN model and collect three types of data for each training observation in each episode: (1) the agent's actions; (2) the set of Q-values; and (3) the timestamp for each action for all possible actions. In this specific application, we define our agent as the trading agent that makes Buy, Sell and Hold (BSH) actions, based on the current state. BSH is encoded as 1, 2, and 0, respectfully. Our reward function is defined as the daily realized (P&L). We define the state as a standardized two-dimensional vector of the sigmoid of the differences of the ETFs' past price and volume. Formally, let pt and vt be vectors of the ETFs' price and volume at time *t*, respectively. State $St = (dt–1, dt)$, where dT = [$sigmoid(pt – pt–1)$, $sigmoid(st – st–1)$]. Simply, the *St* is a two-dimensional point representing the one day change of the price and volume of the ETF, respectfully. In the application of RL to trading, we use a DQN model to deal with the complexities of calculating Q-values as a result of high-dimensional states. Hence, solving the Bellman equation given this input size proves to be a computational nightmare. The solution to this dimensionality issue rests in estimating Q(s,a) using a DNN Neural Network (DNN) (Wang *et al.*, 2019). In our case specifically, we use between 1-3 hidden layers (2 is our benchmark) and using an Artificial Neural Network to do the approximations. Once data is extracted following training, pre-processing is done on the q- values to calculate useful summary statistics such as average Q-values within and across episodes. With the time-series, set of training Q-values and actions prepared, a modular design of the visualization system allows users to use the User Interface (UI) to display

desired training interpretation visualizations either for a specific episode or across episodes. From a software perspective, two modules are developed, one for specific episodes, and another for analysis across episodes (Section 3.1 and 3.2). Similarly, we extract the same data while testing the algorithm on out-of-sample data. A third module is developed for testing analysis (Section 3.3). In addition to these three modules, a fourth module is introduced as a control center. Via callbacks, this allows a user to control what episodes to be analyzed in the context of training, what trading time-frame display, and what interpretability instrument to show.

2. Literature Review

Gautam Reddy *et al.* (2016) explore the possibility of using atmospheric currents to soar. To simulate a turbulent environment, they use numerical models of convective flows and test against model-free, experience-based, reinforcement learning algorithms. The purpose is to approach the problem of flight as "learning to navigate complex, highly fluctuating turbulent environments." To visualize decisions made by the reinforcement learning agent, Reddy's team plotted the optimal action on a bank angle μ based off of a given vertical acceleration and torque (α_z, τ). To visualize this, a plot is made having +, –, and 0 denote positive high, negative high, and low values for α_z and τ. The red upward arrow, blue downward arrow, and orange square indicate that the optimal policy is to increase, decrease, or maintain the same bank angle, respectively. From these plots, Reddy's team was able to understand better how their reinforcement learning agent learns from outside stimuli, which is helpful in designing both a more effective simulation environment and a more adept agent.

Wang *et al.* (2019) attempts to visualize a deep q-network's training process in a way that will enable users to dive into the experience space of the reinforcement learning agent. Their work emphasized use cases such as Atari games with simple action spaces, so the action/reward patterns are streamlined. They also reached out to experts to collaborate on visuals that would more effectively help domain experts understand their DQN models. From those experts, the following project goal was outlined: create an overview of the training process and demonstrate the action/movement/reward patterns of the agent.

Figure 8 in the paper demonstrates how epoch data can be visualized and even predicted based off its trajectory. Additionally, the finalized view can be found in Figure 3 of the paper where epoch, episode, segment data can all be seen from a single dashboard.

The final work included four levels of detail: overall training, epoch-level, episode-level, and segment-level. From these levels, they were able to identify action patterns of the agent which were used to help control the random actions of the agent. Overall, Wang et al. produced insightful findings and were able to achieve improvements to their own model from the visual feedback.

Arthur (2016) provides an open source platform for visualizing a RL algorithm's actions, in a very dynamic way. As the agent makes decisions, Juliani's platform displays the decision against all other decisions, along with the respective Q-value. This mode of interpretation breaks each episode of training down to the individual step the agent takes. From this, we concluded it would be best to take those individual steps and approach them from a macro-level, as a way to detect trends in an agent's action-space, rather than viewing each action as an independent datapoint.

The work done in our project is directly inspired by the techniques used in each of these papers, with the primary takeaways being, from Junpeng *et al.* (2018) partitioning the dashboard by testing, inter-episode, and intra-episode (episode) data and, from Gautam Reddy *et al.* (2016) using an altered version of the optimal action plot to visualize the agent's actions using two state dimensions at a time.

3. Data and Methodology

We use a liquid S&P ETF's price and volume as trading data fed into our DQN, which generates our trading signals. Specifically, we use the Vanguard 500 Index Fund ETF (VOO), from 2016-2020. As of 8/31/2021, the VOO ETF was comprised of 507 equities and had a market capitalization of $195.8 bn. We chose this security as a traded market proxy due to both its liquidity and near identical performance to the S&P Market Index, since its inception on 9/7/2010. Over the 10+ year life span of the VOO, the ETF performed only 4 basis points under the S&P on average, annually (Vanguard, 2016-2020). By choosing the VOO as the trading instrument, we have a reputable market proxy to generate our signals, resulting in reputable "market" signals to interpret.

In answering the question of "how", based on the work of Wang *et al.* (2019), visualizations tend to be the most user friendly and presentable medium in conveying sometimes complex ideas in the RL space. To make the user experience both friendly, yet as custom as possible, a multi-page UI infrastructure provides the best method of providing insight

into different aspects of the algorithm. With two layers of the training process and, more importantly, performance of the agent on out-of-sample data, there are various contexts and moments of the algorithm that can be interpreted and analyzed. Developed from the work in Wang *et al.* (2019), a multi-strata approach is most appropriate in interpreting the training of an agent, while a single level design is sufficient in understanding the testing of a model. In summary, the UI is organized into a two-level, bottom-up training order, and a separated testing section.

At the most granular level of the algorithm, a specific episode of the training phase can provide insights for a specific pass through of the data; we call interpretation at this level, inter-episode analysis. For example, in this layer, we may track Q-values and respective Buy-Sell-Hold (BSH) signals for a given trading day, for a given episode during the training phase. More importantly, we expect our agent to learn from reward overtime; hence, we introduce the notion of intra-episode analysis. In this layer, analysis over a series of episodes provides visualization of training overtime. Insights such as a decrease in greedy decision making can easily be derived from this type of analysis. With the ability of seeing into specific episodes of the training, and an evolutionary perspective across training episodes, a model can be deployed to trade on out of-sample, or real-time data. In traditional machine learning applications, data that the model is tested on is referred to test data. Hence, we develop the idea of test analysis, where interpretation analysis is done when the agent is acting on out-of-sample information.

3.1. Intra-Episode Analysis

With the ideas of inter and intra episode, and test analysis developed, a UI with three tabs containing the respective information provides clear insights and distinction into the various layers of the agent, from a granular episode during training, to go-live application.

Analyzing a specific episode during training of an RL agent provides insights into how the agent is learning from its environment, for a single pass-through of the data. More specifically, understanding why certain actions are taken by the agent on certain days, provides interpretation for why a certain BSH signal is chosen. Given that the signal at trading day i is a function of Q-values (application of the Bellman Equation), we suggest visualizing all the values that $Q(s^0,a^0;\theta_i)$ can take on for a specific trading day i. In doing so, all possible values that the algorithm is selecting from can be visualized. From here, trends and magnitudes of Q-values for each action can be easily extracted and analyzed.

The UI provides a clean and customized interface for users to visualize the idea mentioned above. Because this type of analysis is done for a particular episode during the training phase, the "Q-Values Over Time" plot is accessible as a drop-down item in the "Visualizations" selection in the "Intra-Episode" tab. From Figure 2, a slider provides easy visualization of each episode during the training. To make the visual distinctions between Q-values and trends more clear, Q-values for respective actions are color coded.

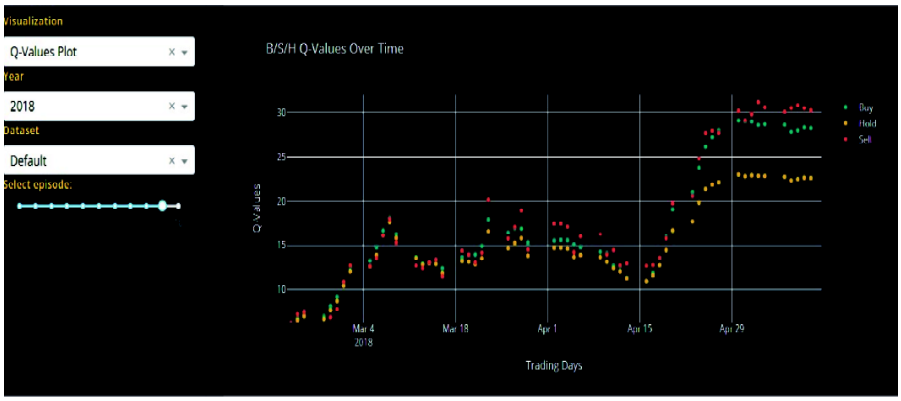


Figure 2: Q-Values Overtime Plot for Training Episode 9

In addition to looking at the decision making engine of a DQN agent, the Q-value, insights can be derived by looking at the values that the state variables take on for each actions over the time horizon of an episode. For each state variable provided to the agent (in our case daily price and volume change of the VOO are provided), the time horizon is plotted on the x-axis, and value of the state for that day is plotted on the y-axis, the point is color coded with the action the agent made for that day. Figure 3 provides a visualization of this idea for training episode 10.

Similarly to the "Q-Values Overtime" plot, the UI allows for a slider to change which episode is being analyzed. With all forms of interpretation, the goal is tell a "Story" of whats happening with the agent. Using these plots, users can easily infer, for a given state and time of the market, what tendencies the agent has during training. Having an idea about these tendencies can help answer training questions such as "if the volume changes by *x*%, what action did my agent

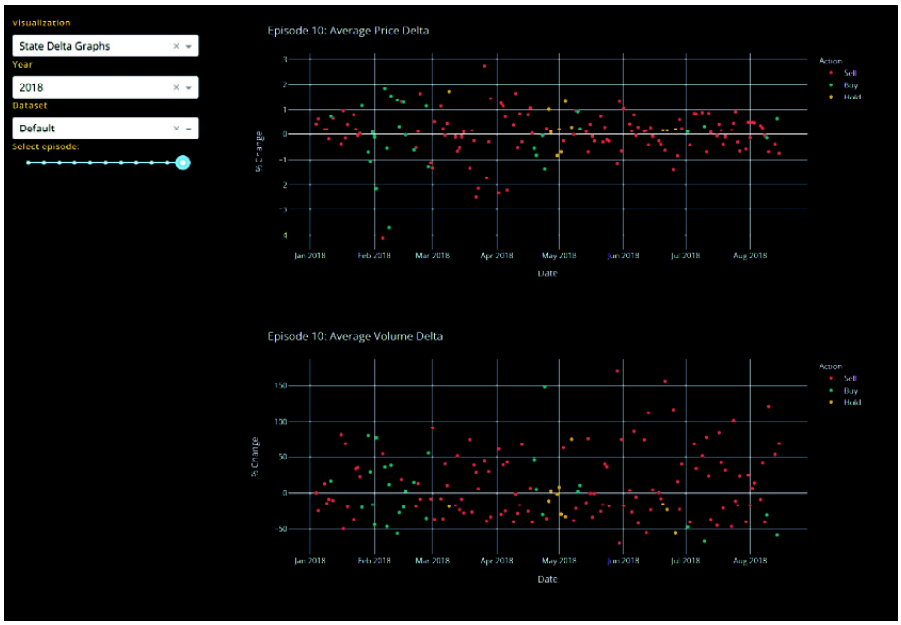


Figure 3: State Values Plots for Training Episode 10

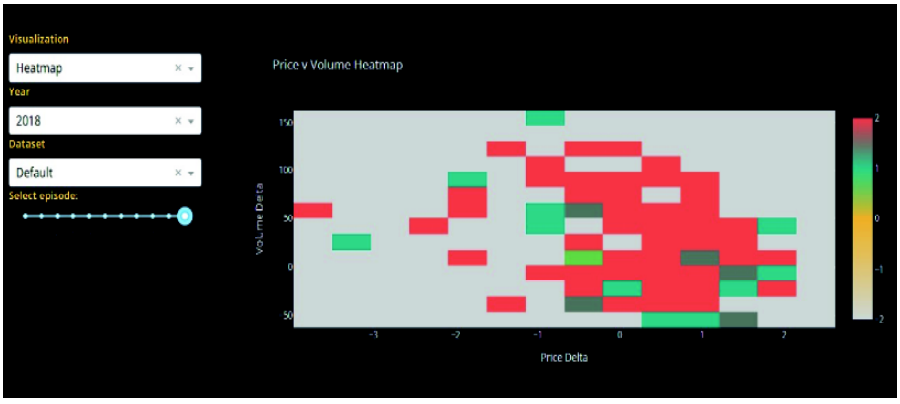


Figure 4: Heatmap for Training Episode 10

tend toward?" or "for a given level of price change, when did my agent buy, sell, or hold?". By visualizing actions across the training phase for the range of each state variable, we begin to see how the agent makes decisions within its environment from a marginal perspective (each input state). But how do we understand the actions of the agent from a holistic view?

To understand the agent from a holistic view, we introduce the notion of a state variable heatmap. Within the context of model training, the state variable heatmap provides a visual representation of what action the agent takes with respect to the values the state variable takes on across the training episode. Derived from Figure 4 in Gautam Reddy *et al.* (2016), where the optimal action, the bank angle change, is plotted against the bird's vertical acceleration and torque, the heatmap in Figure 4 provides a similar visualization in the context of trading. This visual is adapted to the states and

actions within the trading context and allows for changing the heatmap for a given training episode via the slider on the left hand. In essence, the heatmap blends the State Value Plots in Figure 3, into one figure, while the State Value Plots provide a marginal looks at agent outcomes at specific trading days.

The "Heatmap" selection under "Visualizations" provides a holistic view of the decisions that the agent is making for given values of the states across the specific training episode. In Figure 4 the heatmap for episode 10 tells the story of "what price and volume change combinations do certain actions of the agent occur in?" It is important to note the grey areas of the heatmap. Here, a specific state combination does not exist in the training set, hence we fill this area of the map with a filler color, grey. During training and for the purposes of simplicity, we leave these combinations empty. In applying this heatmap to test analysis, we create artificial state spaces that the agent would have taken, should that state combination occurred, resulting in a more filled in heatmap (Section 3.3).

3.2. Inter-Episode Analysis

Understanding of the agent through an episode provides a snapshot understanding of how an agent is learning about its environment. As we increase episodes, we expect the agent to adjust its view of the environment more finely. Inter-Episode analysis provide insights and interpretability at a less granular level, showing evolution over the entire training length. In this tab of the UI, two new interpretation visualization are introduced in addition to showing heatmaps for each episode in the range selected.

Having a categorical variable as the output of our RL algorithm allows us to represent the difference between two processes in a two-way table. In Figure 5, our two processes are two episodes of the training selected via the selection slider on the left side of the interface. Along the diagonal, the table populates count of days where the action is the same for two chosen training episodes. For the $(i, j)^{th}$ cell of the table, where $i_6 = j$, the value n represents the times where one episode did action i, n times on days when the other episode did action j.

Figure 5: Two-Way Tables for Training Episode 1-8

In Figure 5, we would interpret that only 3 times did both Episodes 1 and 8 decide to hold on the same day. Similarly, Episode 1 made 24 sell decisions on days when episode 8 was holding. In addition to the two-way table, tables for individual episodes at the maximum and minimum of the selected episode range, provide average values of the states for a given action in that episode. In context, for episode 1, the average volume change when the agent decided to buy was –13.19%. While for that same episode, the hold action had an average volume change of only –5.5%, across the entire episode. Using the two tables, the user can easy extract a relationship of how the average states for a given action evolves over a range of episodes. Applied to the data in Figure 5, over the course of 8 training episodes, the average volume when selling did not change, when both the holding and buying average volume decreased.

Similar to understand the average state values at the minimum and maximum of the selected analysis range, another visual referred to as the "State Delta Graphs", shows this evolution for every action across the entire range, not just the minimum and maximum of the selection.

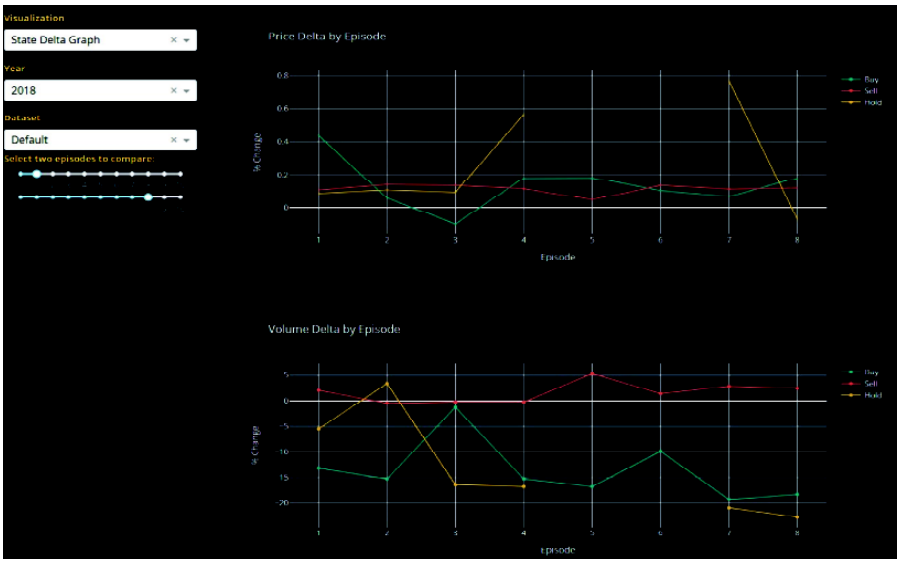


Figure 6: Intra-Episode State Graphs for Training Episode 1-8

Like the state tables in Figure 5, Figure 6 provides the information across the range of selected episodes. Similar in design to Figure 3, a plot is generated for each feature fed into the RL algorithm. These plots visualize explicit trends in the average state and clearly show the evolution of decisions made by the agent, across a given training segment. Interpretability from these plots comes from insights about the average state across a training horizon, and in turn helps answer the question of "during a given training interval and action, how did the average price or volume change?"

3.3. Test Analysis

For any model, the true test of its performance and value-added is on data the model has not seen before. As mentioned prior, a test analysis tap provides interpretability insights for out-of-sample actions. During the test analysis, the user is interested in answering the question of "what is the algorithm doing given unseen data?" To begin to tackle this answer, the "Test Analysis" tab provides a "State Value Plot", along with histograms of state value for each action/state pair. The third and most important visual in the test analysis is the modified heatmap that fills in missing state/action pairs with synthetic actions.

As mentioned prior, the heatmap visualization provides an idea of what actions the agent tends to take in a given environment. In the trading context, not all environments can or have been observed in the markets. Hence, we introduce a methodology to mine artificial state spaces that the agent would do, should a specific state of the environment occur

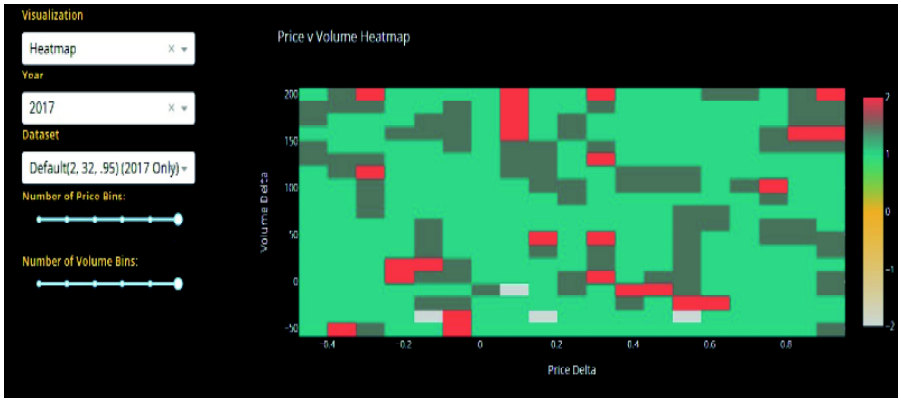
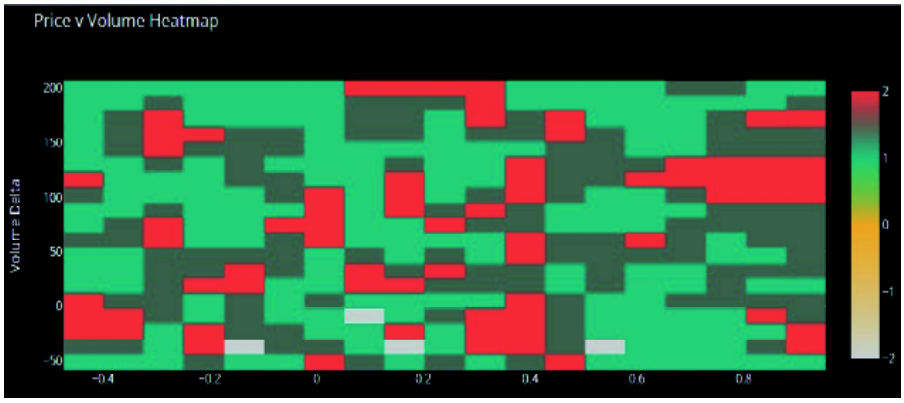


Figure 7: Heatmap for Testing

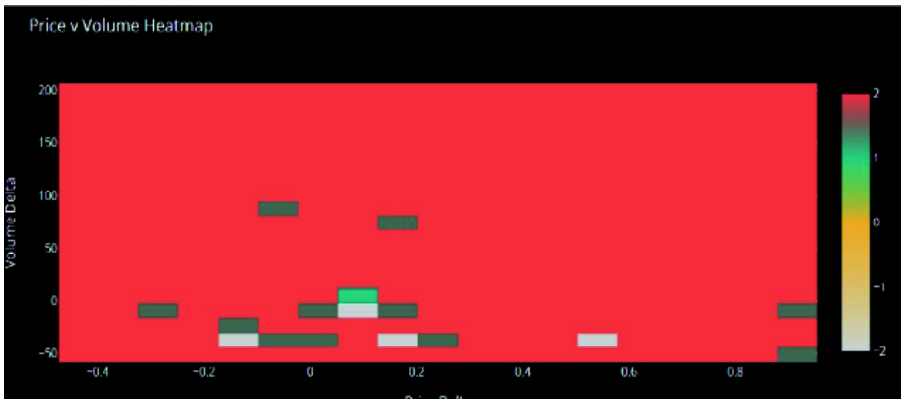
that is not observed in our test set. Mining these theoretical actions fills in the grey areas as we see in Figure 4, making the visual more complete and informative, and provides *would-be* scenarios in yet-to-be observed market conditions.

To begin mining theoretical actions of the agent, it is first necessary to identify market conditions that do not exist in our test data. More specifically, we identify combinations of states that do not occur in our test set. With the dataset of non-observed state combinations set aside, we now utilize our DNN from out training to approximate a mapping of states to actions. We calculate $\max Q_{mined}(s_{mined}, a; \theta_{train})$, where θ_{train} are the parameters of the DNN used in training, s_{mined} are the combinations of states that do not occur in our test set, and a is as defined previously. The result is a synthetic action that the agent *would* take should $s \in s_{mined}$ occur.

Once the synthetic action has been engineered for states that do not naturally appear in test set, we add these outcomes to the test outputs already known. With both synthetic and true action, the resulting heatmap (Figure 7) is more filled in than the similar result in Figure 4. In addition to incorporating synthetic actions into the heatmap, we recommend allowing users to select heatmaps for various combinations of hyperparameters in order to interpret the effects of changing hyperparameters on the actions of the agent.



(a) Test Heatmap: DNN 1 Hidden Layer



(b) Test Heatmap: DNN 3 Hidden Layer

Figure 8: Testing Heatmaps for Two Different Hyperparameter Configurations

In this application, we experiment with various numbers of hidden layers and learning rates, among other hyperparameters.

4. Results

Thus far, we have introduced and developed a visual methodology of understanding the decision making of an RL agent in the context of securities trading. From using our system to train and test, and RL trading agent on the VOO during two

different time frames, two distinct outcomes are clearly illustrated: users can clearly understand the training evolution of the agent over episodes, and the effects of changing hyperparameters on the agent's decision making.

During training of any iterative algorithm, developers hope that overtime, their system has a commanding understanding of the pattern in the data. One way to gauge the progress of this evolution is to observe a decrease in the utilization of the greedy decision making strategy, over training episodes. With the help of the "State Delta" visualizations in the Intra-Episode analysis tab, confirmation of this desired decrease can be confirmed. In our dummy case, daily trading of the VOO during the calendar year of 2018 using just the price as a state variable with ten training episodes, notable decreases in epsilon greedy usage were observed. Specifically, within the first three episodes, trading days where random actions were chosen went from 123, 51, and then 14 days by the third episode. By the eighth and consequent episodes, greedy was not used and the agent was making experienced-based decisions only. Using the visualizations in the intra-episode module, a user was able to quickly and visually confirm that the agent is indeed iteratively learning, and that most of the exploration of the environment by an RL trading agent via an greedy strategy takes place within the first few training episodes, decreasing at an exponential rate and eventually vanishing.

In addition to confirmation of iterative learning and other training results, our visual system provides easy access into understanding the effects of changing hyperparameters on agent decisions. In our specific investigation, we adjusted the number of layers and learning rate, among other hyperparameters, and used the various visualizations introduced in this paper to see effects of the changes. Using the two state variables of daily trading volume and price, we can visually conclude that increasing the number of hidden layers in the DNN approximation of the Q-value results in overfitting. Figure 8 illustrates that with three hidden layers (sub-plot b), one more than our benchmark developed with two layers, the agent is not able to distinguish between appropriate buy-sell-hold actions, and always sells.

5. Conclusion

In this work, we introduce a visual method to interpret a DQN agent in the context of securities trading. Inspired by previous works in the realm of RL interpretability, this paper extrapolated and added understanding of an RL agent using a modular visualization dashboard to understand various layers of the algorithm. We present the decision making agent at two levels of the training phase and the testing phase: within a training episode (Intra-Episode Analysis), across a user-chosen amount of training episodes (Inter-Episode Analysis), and testing (Test Analysis). From visualizations in each of these analysis types, the user has a clear, customizable, and visual insight into the decision making process of the trading agent, both for a given day and across a trading horizon. Ultimately, this visualization system successfully conveys a RL trading agent in an understandable and consumable way to its end user, regardless of prior machine learning and RL experience.

6. Future Work

Given the ideas introduced in this work, we recommend three distinct next steps. First, we suggest introducing more inputs into the model. Incorporating additional features provides more information to the RL agent, possibly resulting in better performance of the agent. More relevant to interpretation, using heatmaps introduced in this paper, the marginal effects of adding additional features to the algorithm may open to investigating feature importance in the context of DQN trading. Specifically, adding in sentiment analysis may be an interesting intersection of using Natural Language Processing along with Reinforcement learning.

Second, as briefly introduced in this paper, continuing to study the effects of changing hyperparameters on the actions of the agent may begin to answer which hyperparameters result in significant changes of the trading action. Ideally, given sufficient computational power, users would be able to train and test models, via the control center, using parameters set in the control center, and automatically and in real-time generate our interpretation visualizations.

Finally, it may be of interest to dissect the heart of the DQN algorithm, the DNN mapping states to Q-values. Understanding the components of Q-value estimation can provide value in understanding which feature contributes most to the estimates, and in-turn what is driving the decisions of the agent. This task can be accomplished by including an additional module within the dashboard framework, and work supplement the current training and testing modules. Computationally, interpreting the DNN can come from SHAP or LIME values. Visually, Wang *et al.* (2019) cite numerous recent works in their introduction for providing visual analytics of DNN.

References

Arthur, Juliani. (2016). *Reinforcement Learning Control Center*. https://github.com/ awjuliani/RL-CC

Gautam, Reddy., Antonio, Celani., Terrence, J. Sejnowski., and Massimo, Vergassola. (2016). Learning to Soar in Turbulent Environments. Proceedings of the National Academy of Sciences, 113 (33), E4877-E4884. DOI:10.1073/ pnas.1606075113

Hariom, Tatsat., Sahil, Puri., and Brad, Lookabaugh. (2020). Machine Learning & Data Science Blueprints for Finance. In: 1st Edition. 1005 Gracenstein Highwway North, Sebastopol, CA: O'Reily Media, Inc., 2020. Chapter 9.

Wang, J., Gou, L., Shen, H.W., and Yang, H. (2019). DQNViz: A Visual Analytics Approach to Understand Deep Q-Networks. *IEEE Transactions on Visualization and Computer Graphics*, 25 (1), 288-298. DOI: 10.1109/ TVCG.2018.2864504.

Vanguard. (2016-2020). *Vanguard S&P 500 ETF (VOO)*. url: https://investor. vanguard.com/etf/profile/overview/voo

Cite this article as: Bryan Yekelchik and Zachary Coriarty (2022). Deep Q-Network interpretability: Applications to ETF Trading. *International Journal of Artificial Intelligence and Machine Learning,* 2(1), 61-70. doi: 10.51483/IJAIML.2.1.2022.61-70.