"Q-Trading": Learning to Scalp Using Order Book Features

Kevin J. Wu KJW2157@COLUMBIA.EDU
Department of Computer Science
Columbia University, School of Engineering and Applied Sciences

Abstract

In this report, I explore the feasibility of using Q-learning to train a high-frequency trading (HFT) agent that can capitalize on short-term price fluctuations, using only order book features as inputs. I propose a specific functional form for the agent's value function based on domain-specific intuition and evaluate the performance of Q-learning trading algorithms by replaying historical Bitcoin-USD exchange rate data through a "naive" market simulator.

1. Introduction

High-frequency trading of financial assets presents a rich set of challenges for reinforcement learning. In the context of a financial exchange, a rational trading agent seeks to maximize cumulative profits (rewards) over a given time horizon, within a complex environment whose underlying dynamics are unknown. Depending on the characteristics of the trading algorithm's decisions and the environment, each agent's actions may have an affect not only on his/her own future rewards but also on the future state of the market.

1.1 Trading Agent Overview

The goal of this project is to examine the viability of a trading agent that profits by predicting short-term directional changes in price using only information contained in the order book[1] and "scalping" accordingly — buying high and selling low. To simplify the analysis and simulation, I restrict this agent to act as a market-taker, only placing market orders and getting filled using whatever quantity is available at the top of the order book at the time of the trade. Additionally, I assume that the agent's actions do not directly cause changes in the future state of the order book.[2]

Actions, however, do affect the future state of the environment by causing changes in the agent's net position. Since the objective of the agent will be to trade quickly back and forth rather than accumulate to large positions, having a non-zero position will affect the utility of a particular action; for example, an agent who is already long an asset will be more incentivized to hold on to or sell their position at the next state, all else equal.

2. Related Work

Much of the existing academic work using reinforcement learning techniques on financial market data focuses on longer-term price prediction problems, on the order of minutes or

1. See Appendix A for a more detailed description of the limit order book.
2. If the agent's traded quantities are small relative to the available volume, this may not be an unrealistic assumption

even days. Techniques from deep learning, particularly the use of recurrent neural networks for modeling time series, have been applied at this time scale with some degree of success (Y. Deng and Dai, 2017).

At the market microstructure level, applications of reinforcement learning are less common and have yielded mixed results. For trade execution problems, Kearns and Nevymyvaka have successfully applied Q-learning algorithms to trade execution, where the goal is to minimize the market impact of a given trade (Nevmyvaka et al., 2006). For the task of price prediction and alpha generation, Kearns and Nevymyvaka find that a Q-learning algorithm using heterogeneous order book features is able to learn a momentum-based strategy[3] using equities order book data (Kearns and Nevmyvaka, 2013).

While the overarching goal of this project is similar to the latter study, this project explores the feasibility and interpretability of a *hand-crafted* Q-learning function approximator on order book data using a naive market playback environment. In addition, while Kearns and Nevmyvaka have shown empirical limits to profitability of HFT strategies in highly-liquid US equity markets (Kearns et al., 2010), this project aims to explore whether reinforcement learning-based HFT strategies may have a competitive advantage in newer and potentially less efficient electronic markets, such as cryptocurrencies.

3. Methods

In this section, I describe the HFT environment as a Markov Decision Process (MDP), represented by the tuple (S, A, P, R, H, γ), with state space S, action space A, transition probabilities P, expected rewards R, time horizon H, and discount factor γ.

3.1 Model-Free RL

Model-free RL methods seek to determine an optimal policy for an environment without directly learning the underlying dynamics of the MDP. One of the earliest examples of a model-free RL algorithm is Q-learning (Watkins and Dayan, 1992), which seeks to learn the value of every state-action pair and acts greedily or ϵ-greedily with respect to these values. For every state-action pair (s, a), we define the optimal value function $Q^*(s, a)$ recursively as:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s,a,s') \max_{a'} Q^*(s',a')$$

When the number of possible states and actions is small, $Q^*(s, a)$ for all s, a can be calculated directly using dynamic programming. For most real-world problems, however, with a large (potentially infinite) number of possible states and/or actions, the value of a state-action pair $Q(s, a)$ is approximated by a function $Q_\theta(s, a)$, whose parameters θ are learned using gradient descent.

3.2 "Q-Trading" Definitions

In this report, we consider the finite time horizon MDP setting, in which a trading agent seeks to maximize a discounted total reward function over H time steps.

3. "Momentum" strategies aim to profit from continuing trends in price.

State representation: For a trading agent, the state space consists of both the set of standing buy and sell orders represented by the limit order book and the agent's current inventory. At time t, an order book K levels deep consists of bid prices $b_t^{(1)}, ..., b_t^{(K)}$, ask prices $a_t^{(1)}, ..., a_t^{(K)}$, bid volumes $u_t^{(1)}, ..., u_t^{(K)}$, and ask volumes $v_t^{(1)}, ..., v_t^{(K)}$. We denote the midpoint price as p_t, where $p_t = (b_t^{(1)} + a_t^{(1)})/2$. Finally, we use s_t to denote the agent's position or inventory at time t.

Action space: Although in practice, an agent's trading decision at time t may be described by any real-numbered value, I discretize possible actions to a finite set of integer order amounts in order to simplify the action space.

Reward: Finally, the reward for the agent corresponds directly to the gains and losses from trading. This includes both realized profits, computed on a first-in-first-out (FIFO) basis, and unrealized profits at the end end of H time steps.

3.3 Function Approximation for Q-Trading

Many existing applications of reinforcement learning to financial market data make use of deep neural networks (specifically RNNs) as function approximators. However, in a high-frequency setting, a signal on future price direction may become stale in the time it takes to complete a forward or backward pass through a deep neural network. In addition, with limited data, deep neural networks are prone to overfitting. In this section, I walk through the hypotheses and intuitions about the market used to develop a Q-function from the ground up.

First, I hypothesize that the expected future midpoint price change, $\mathbb{E}[p_{t+1} - p_t]$, is a linear combination of the distances of bid and ask prices ($b_t^{(i)}$ and $a_t^{(i)}$) from the current midpoint price, where each order book quote is weighted by its respective volume ($u_t^{(i)}$ and $v_t^{(i)}$).[4] To account for the varying levels of overall trading activity at different times of day, I normalize the volume at each price level by the total quantity available to trade at the first K bid/ask levels. Initially then, we have $\mathbb{E}[p_{t+1} - p_t] = \sum_{i=1}^{K} u_t^{(i)}(b_t^{(i)} - p_t) + \sum_{i=1}^{K} v_t^{(i)}(a_t^{(i)} - p_t)$. Since the influence of a particular price level on future moves may be dependent on its position in the order book in addition to its volume, we can reweight the terms above using a vector of coefficients θ, so that $\mathbb{E}[p_{t+1} - p_t] = \theta^{\mathbf{T}}\mathbf{x}$, where $\mathbf{x} = (\mathbf{u_t^{(1)}}(\mathbf{b_t^{(1)}} - \mathbf{p_t}), ..., \mathbf{v_t^{(1)}}(\mathbf{a_t^{(1)}} - \mathbf{p_t}), ...)$, plus a bias term.

Next, I note that the value of a particular expected price move is dependent on the agent's existing position s_t and its chosen action a_t; an agent that is currently holding a long position will profit from opposite price moves compared to a similar agent that is holding a short position. The expected reward of placing order a for an agent with position s is therefore $\mathbb{E}[R] = (s + a)\,\mathbb{E}[p_{t+1} - p_t]$. In practice, I found that passing this product through a tanh non-linearity helped stabilize Q-learning updates against extreme fluctuations in available volumes.

Finally, I add in an L1-regularization on the agent's position to the value function, to discourage the agent from accumulating large positions. The resulting functional form for

4. This concept is sometimes referred to as "book pressure" in trading parlance.

the value at time t of an action, given the agent's current position s_t and the state of the order book x_t is as follows.

$$Q(\mathbf{x_t}, \mathbf{s_t}, \mathbf{a}) = \tanh[(\mathbf{a} + \mathbf{s_t}) \cdot \langle \theta^{\mathbf{T}}, \mathbf{x_t} \rangle] - \lambda |\mathbf{s_t} + \mathbf{a}| \tag{1}$$

The above value function has the following desirable properties, all of which are grounded in common sense and financial intuition: (1) It encodes the current state of the market using volume-weighted bid and ask price deviations from the midpoint price, $\mathbf{x_t}$; (2) it is linear in $\mathbf{x_t}$ given an action a; (3) the tanh term squashes the expected price movement between -1 and 1 (predicting a general downward or upward directional movement); (4) and the regularization term encodes the agent's preference for short holding periods.

4. Experiments: BTC-USD Orderbook Data

4.1 Dataset

The dataset for this project comes from Level-2 orderbook data for US dollar-denominated Bitcoin prices (BTC-USD) collected from the Global Digital Asset Exchange (GDAX)[5]. The scraped data includes the top 20 bid and ask prices and volumes, and contains over 1.7 million timestamped order book snapshots spanning a period of 20 trading days from 04/11/2017 to 04/30/2017. See Appendix B for further details.

4.2 Simulation Environment

To simplify the scope of this project, I make a set of simplifying assumptions regarding the interaction between trading agent and the market environment. First, I assume that the trading agent only makes market orders, which execute at the best bid (if the agent's action is a sell order) and at the best ask otherwise. The agent is always either fully filled at the top-of-book price, or filled up to the available volume at the best price. Future order book updates are replayed as is. As mentioned in the introduction, this is a reasonable simplification to make if the agent's order quantities are small compared to the overall volume available in the market.

4.3 Training

To train a Q-learning agent in an episodic MDP setting, I randomly sample episodes of experience from the training dataset. Each episode contains 300 order book snapshots (corresponding to a 5-minute long period in the order book history). For each set of hyper-parameters, I train for a total of 1000 episodes, using an ϵ-greedy policy with $\epsilon = 1$ for the first 50 episodes and gradually annealed to a minimum level of 0.001. There was no use of a target network; in other words, the same parameters were used to calculate Q-values for the exploration policy and for the target values. The gradient of the TD-loss was used to update the Q function after every 15 observations. I used a discount factor $\gamma = 0.8$ and a gradient update step size $\alpha = 0.0005$.

5. Market data available at https://www.gdax.com/trade/BTC-USD

4.4 Baseline

For each set of actions A, the baseline agent takes a random action in A at the beginning of the episode, and hold its position until the end of the episode. This can be thought of as a “random buy/sell-and-hold” strategy with short holding periods (where each holding period is exactly equal to the episode length for the Q-learning agent).

4.5 Results

I simulated the performance of Q-learning trading agents under several different environments. I tested two different action spaces, A_s and A_l, where $A_s = \{-1, 0, +1\}$ (a small action space where the agent may only choose to buy or sell the asset, or make no trade), and $A_l = \{-2, -1, 0, +1, +2\}$ (an expanded state space where an agent may choose to make a large or a small order in either direction). I also try limiting the dimensionality of the feature space to the top K best bids and asks, where $K = 5$, 10, and 20. Table 1 shows the full set of results for two different values of regularization, $\lambda = 0.05$ and $\lambda = 0.1$. Figure 2 shows the evolution of the Q-learning agent under various settings of the environment variables for $\lambda = 0.1$.

Overall, the Q-learning agent did not consistently outperform the random baseline in terms of average reward except for a few cases. Specifically, using $K = 10$ and A_s resulted in the best performance for multiple values of regularization; however, the standard error of the final average reward statistic is relatively large, and we cannot say with certainty that Q-learning does better than “break even” on its learned trading strategy.

However, though the Q-learning performance shows mixed success relative to the baseline in terms of average final reward, in nearly all cases, the standard deviation of rewards between episodes improved significantly from that of the baseline strategy.

4.6 Interpretability

Here, I look at the parameters and behavior of the learned trading agent and offer some financial intuition for the results. In Figure 3, I show the coefficients on bid and ask state variables for long and short positions, respectively. For the “bid states” (the price-volume variables corresponding to bids in the order book), we see that the effect of price/volume in the order book on the potential *upward* movement in price decreases deeper into the order book. Similarly, for the “ask states,” we see that the effect of pressure in the order book on the expected *downward* movement in price decreases deeper into the order book. This confirms the initial intuition that one can predict price movements from book pressure on the bid and ask. In addition, since our order book state representations already consist of volume-adjusted price differentials, these coefficients indicate there is some additional adjustment to the weighting necessary due to being a particular distance away from the top of the order book.

Finally, Figure 4 shows the behavior of the trading agent at the end of the 1000 training episodes. The uppermost time series shows the Q-values of the three available actions. We can see that placing a buy or sell order (shown in the middle graph) causes a discontinuity in the Q-values of the three actions. We can similarly align the changes in Q-values with changes in the position of the agent due to a buy or sell order being placed.

5. Conclusion and Directions for Further Study

This work shows that model-free methods from reinforcement learning may be used to generate modest returns from an aggressive scalping strategy with short holding periods. Although without explicitly accounting for transaction costs, a Q-learning agent with this particular functional form appears to offer only a slight improvement over baseline strategies, it is important to note that there is still much more work to be done.

The first and most immediate next step would be to explore other, potentially more complex function approximators for the Q-function. The current Q-function, best described as a single-layer perceptron with a tanh non-linearity and regularization, offers easy interpretability and fast gradient calculations, but it is limited to learning linear representations of order book features. Other possible parameterizations of the Q-function with relatively low calculation latencies include tree-based methods and support vector machines.

The next step would be to construct a more sophisticated and realistic market simulation environment, modeling ways in which other traders and market participants might change their behavior upon observing the actions of the machine learning agent. The dynamics of this environment would be unknown to the Q-learning agent, and they would be based on either a set of heuristics or models taken from empirical studies of HFT behavior.

Finally, periodic order book snapshots are an artificial view of the market microstructure environment; in a live market environment, state updates (or "ticks") occur upon the execution of any trade or any update to an existing order, at the frequency of milliseconds and even microseconds. Replaying (and simulating) order book data at this scale poses an additional engineering challenge, but by generating a more realistic simulation and training environment, we can not only capture more potential trading signals, but also more closely align back-testing and paper-trading results with actual performance in live trading.

References

Michael Kearns and Yuriy Nevmyvaka. Machine learning for market microstructure and high frequency trading. 2013.

Michael Kearns, Alex Kulesza, and Yuriy Nevmyvaka. Empirical limitations on high frequency trading profitability. 5, 09 2010.

Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. Reinforcement learning for optimized trade execution. *Proceedings of the Twenty-Third International Conference (ICML 2006)*, pages 673–680, 01 2006.

C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

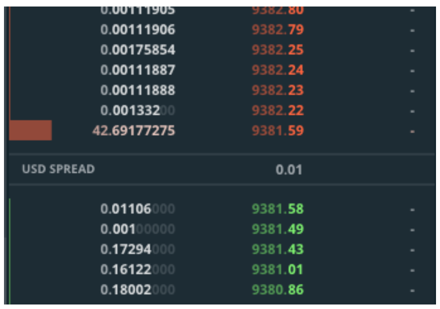
Y. Kong Z. Ren Y. Deng, F. Bao and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):653–664, 2017.

Appendix A: A Brief Glossary of Relevant Financial Terms

Electronic trading of most financial assets takes place via a continuous double-sided auction in which buyers and sellers simultaneously submit their orders to a centralized exchange. An interested party may submit two basic types of orders to an exchange: a *market order*, which is immediately executed at the best available price, or a *limit order*, which is an order to buy (sell) at or below (above) a specified price.

Limit orders which are not immediately executable because there is no matching counterparty at the specified price are aggregated in the form of a *limit order book*, an example of which is provided below in Figure 1. Orders to buy are known as *bids* and orders to sell are known as *asks* (or *offers*). Market orders and executable limit orders will be matched against the list of available quantities in the limit order book according to price-time priority.

The difference between best bid price and best ask price is called the *bid-ask spread*, and the *midpoint* or *mid* price is the theoretical fair price, calculated as the average of the best bid and ask prices.



0.00111905	9382.80	-
0.00111906	9382.79	-
0.00175854	9382.25	-
0.00111887	9382.24	-
0.00111888	9382.23	-
0.00133200	9382.22	-
42.69177275	9381.59	-
USD SPREAD	0.01	
0.01106000	9381.58	-
0.00100000	9381.49	-
0.17294000	9381.43	-
0.16122000	9381.01	-
0.18002000	9380.86	-

Figure 1: Snapshot of the limit order book for BTC-USD (Source: GDAX)

Appendix B: Data Collection and Preprocessing

Order book data used for the experiments in this project was collected by querying the Global Digital Asset Exchange (GDAX) web API[6] for the Level 2 order book snapshot at every second in time. Only the top 20 bids and ask prices, along with the volumes and number of orders available at those levels, were kept. Due to slight latencies in the API calls and responses, there were occasional lags or missing data points in the snapshot time-series. Timestamps were rounded to the whole second and order book snapshots were deduplicated on the time index, while missing snapshots were forward-filled with the last available snapshot. Data points affected by this processing step represent less than 0.1% of the total dataset.

6. https://docs.gdax.com/.

Appendix C: Q-Learning Performance

		$\lambda = 0.1$			$\lambda = 0.05$		
		μ	σ	*s.e.* (μ)	μ	σ	*s.e.* (μ)
A_s	*baseline*	*-0.01*	*16.18*	–	*-0.01*	*16.18*	–
	K = 5	-0.25	17.01	1.38	-1.32	13.13	0.92
	K = 10	0.43	10.58	0.85	2.03	15.88	1.40
	K = 20	0.39	10.59	0.62	0.83	16.45	0.86
A_l	*baseline*	*0.04*	*28.18*	–	*0.04*	*28.18*	–
	K = 5	-1.29	14.93	1.27	-0.66	13.90	1.11
	K = 10	-0.71	14.33	1.03	-0.65	14.85	1.36
	K = 20	-1.08	14.35	1.29	0.12	11.51	0.77

Table 1: Final experiment results. For each training run, I take the average reward accumulated in the last 20 episodes of the agent as the final reward. The mean (μ) and standard error (σ) of this final reward value across 10 different agents ($N = 10$) are displayed above. σ is the standard deviation of rewards accumulated in the last 20 episodes of all training runs.

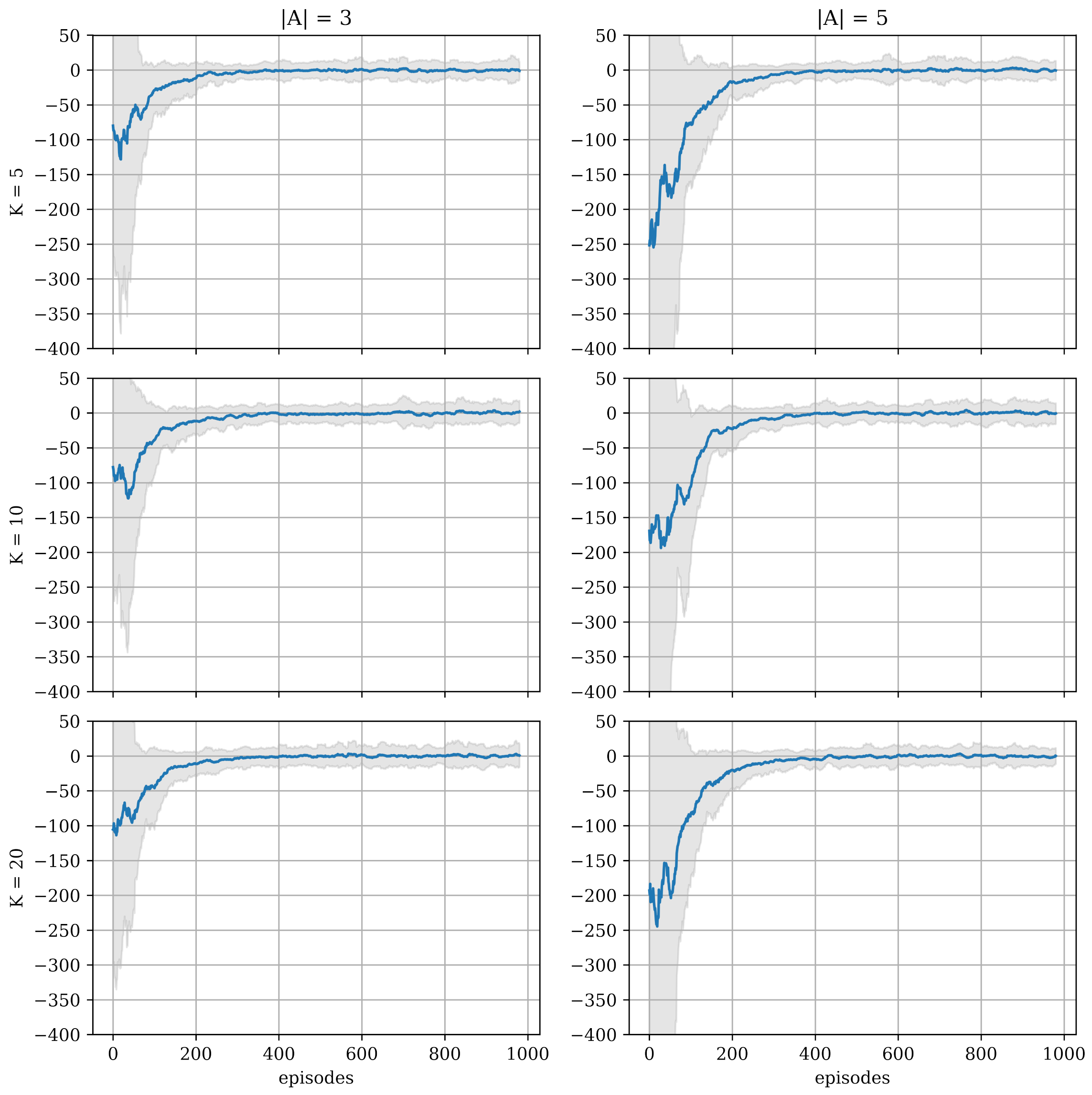


Figure 2: Final reward (as measured by a 20-episode moving average) obtained by Q-learning agents under different settings of the state and action space, averaged over 10 training runs. $\lambda = 0.1$, $\gamma = 0.8$ for these figures. The gray bars represent one standard deviation from the average reward obtained at each episode.

Appendix D: Q-Learning Interpretation

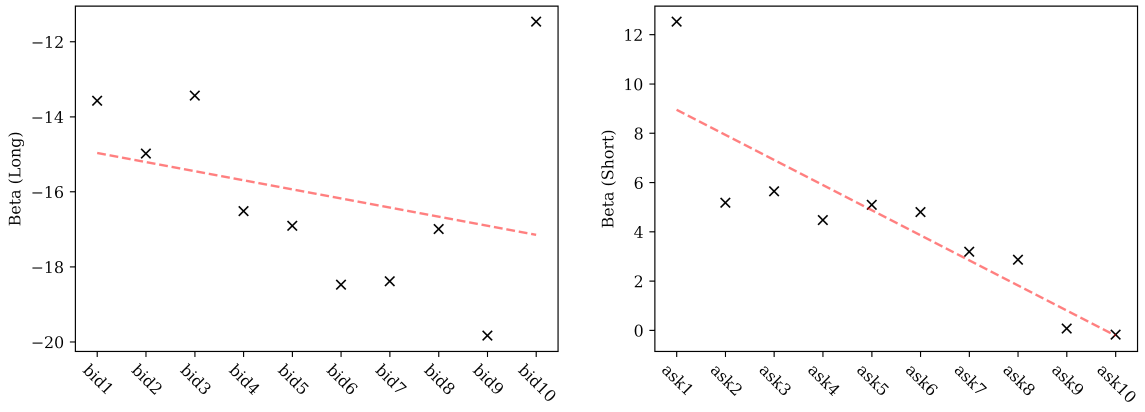


Figure 3: Coefficients on bid variables for positive-valued ("long") positions (left); coefficients on ask variables or negative-valued ("short") positions (right).

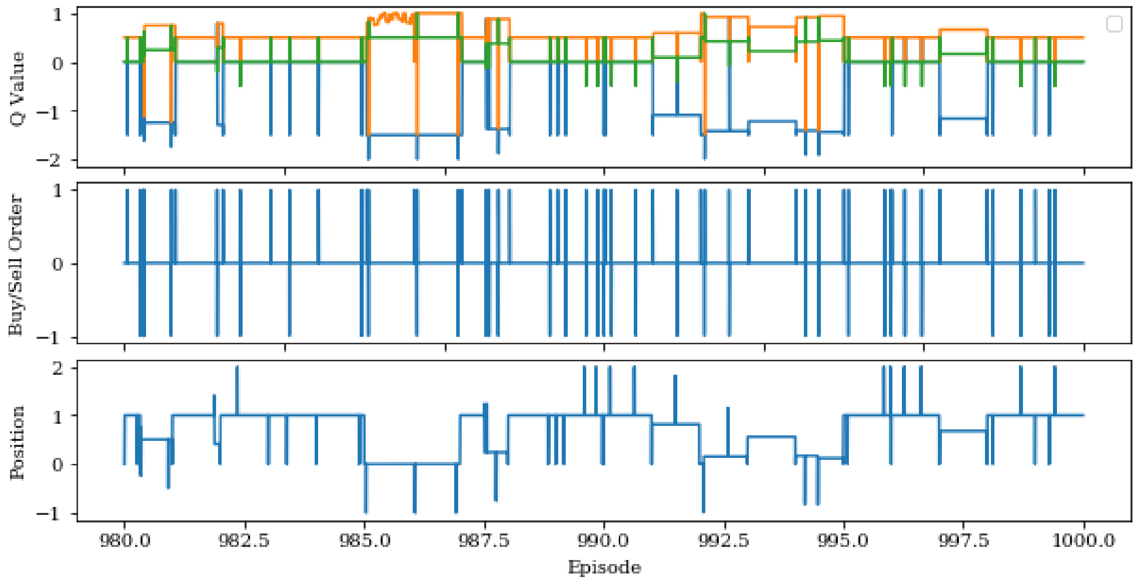


Figure 4: Q-values (top), actions (middle), and positions (bottom) of the agent over the last 20 episodes of a training run.