Deep Reinforcement Learning in Cryptocurrency Market Making

Jonathan Sadighian
École Pour l'Informatique et les Techniques Avancées (EPITA)
jonathan.m.sadighian@gmail.com

November 21, 2019

Abstract

This paper sets forth a framework for deep reinforcement learning as applied to market making (DRLMM) for cryptocurrencies. Two advanced policy gradient-based algorithms were selected as agents to interact with an environment that represents the observation space through limit order book data, and order flow arrival statistics. Within the experiment, a forward-feed neural network is used as the function approximator and two reward functions are compared. The performance of each combination of agent and reward function is evaluated by daily and average trade returns. Using this DRLMM framework, this paper demonstrates the effectiveness of deep reinforcement learning in solving stochastic inventory control challenges market makers face.

Keywords: market making; limit order book; deep reinforcement learning; cryptocurrencies

1 Introduction

In the past 10 years, algorithmic trading has been gaining market share and currently represents approximately 70 percent of trading activity for equity markets in the United States [6]. Algorithmic trading takes place in quote-driven electronic marketplaces, where participants purchase or sell securities by submitting orders to an exchange. Exchanges maintain orders in a centralized limit order book (LOB), which arranges orders by direction (i.e., sell or buy), price, and time-queue priority. Exchange market makers facilitate transactions amongst participants by providing liquidity in the LOB. Traditionally, statistical approaches have been used to model limit order books for market making and inventory control optimization [2, 7]. Understanding that there have been many recent advancements in deep reinforcement learning in other domains such as video games [10, 14], yet very little research published on the application of deep reinforcement learning to market making, the focus of this paper is to apply deep reinforcement to cryptocurrency market making.

This paper is structured as follows: section 1, preliminaries on market making, limit order books, and reinforcement learning; section 2, related research; section 3, contributions in this paper; section 4, experiment design; section 5, experiment methodology; section 6, experiment results and analysis; and section 7, conclusion and future work.

1.1 Market Making (MM)

A market maker's purpose is to provide liquidity to a marketplace and facilitate transactions amongst participants. This requires continuously maintaining two things: bid and offer orders at a range of prices, and an inventory of positions. Market makers generate revenue by capturing the spread (i.e., the difference between the best bid and ask price). To be profitable, market makers must continuously update price quotes to reflect changing market conditions, and manage their inventory of positions to avoid overexposure to a directional move in the market. This stochastic optimization problem of maintaining quotes and inventory levels poses an interesting challenge for reinforcement learning.

Market makers operate differently based on asset class. Equity exchanges in the United States, such as the NYSE or AMEX, have a designated market maker for each security, who in turn has an obligation to provide a specified amount of liquidity to that security. Other exchanges in the United States, such as NASDAQ, have more than one designated market maker per security. However, in cryptocurrency exchanges, such as Coinbase or Bitfinex, there are no designated market makers. In these marketplaces, liquidity is provided by participants, who can be either institutional or retail investors.

1.2 Limit Order Books (LOB)

When participants in cryptocurrency markets want to buy or sell an asset, they must send their order to an exchange. If the participant wants to buy (or sell) an asset at the current market price, they can submit a market order, whereas if the participant has a specific price limit they are willing to pay for the asset, they can send a limit order. Exchanges use a limit order book to track inventory of all limit orders accepted by the exchange.

LOBs are grouped by order direction (i.e., buy or sell), price, and time priority (i.e., FIFO order). The bid side of the LOB contains all the buy orders, whereas the ask side contains all the sell orders. If an order within the LOB is matched with a market order, it is removed from the LOB upon fulfillment. In this scenario, the participant who made the limit order is referred to as the liquidity provider or maker, and the participant who sent the market order is the liquidity remover or taker. Since exchanges profit from trading volume, most electronic exchanges use a maker-taker transaction fee model. Under this fee structure, makers are incentivized to provide liquidity to a given exchange in the form of a rebate, or reduced transaction fee. Market makers usually rely on rebates as part of their trading strategies to prevent transaction fees from eroding profitability.

1.3 Order Flow Imbalances (OFI)

The number and frequency of orders received by the LOB can provide additional information relating to future prices of an asset [22, 23]. In this paper, orders are classified by the following three categories as described in Table 1: market orders, limit orders, and cancellation orders. Under this schema, market and cancel orders remove liquidity from the LOB, while limit orders add liquidity. Throughout the rest of this paper, the order flow statistics for these events are referred to as order flow imbalances.

1.4 Trade Flow Imbalance (TFI)

Trade flow imbalances quantify the ebbs-and-flows of transactions for a given asset. It has been demonstrated by [3] that trade flow data can be used as an indicator to predict price movements.

#	Order Event Type	Description
1	Cancel, C	Order cancellation request C received by the exchange, resulting in liquidity being removed from the LOB
2	Limit, L	New limit order L received by the exchange, resulting in liquidity being added to the LOB
3	Market, M	Market order M received by the exchange, resulting in liquidity being removed from the LOB when matched with a resting limit order

Table 1: Types of order flow arrival events

Typically, the TFI is calculated via the net difference between notional values of buy and sell transactions for a given time period. In this paper, the TFI is calculated differently, as described in equation 5 in section 4.1.1.

1.5 Reinforcement Learning (RL)

Reinforcement learning is a branch of machine learning wherein learning occurs through trial and error, rather than by example, as in supervised learning. As such, RL algorithms are able to learn goals, rather than patterns. RL problems are modeled using the Markov Decision Process (MDP), a framework that enables agents to learn sensations, actions, and goals. Learning is achieved by maximizing a reward signal through trial and error [20].

The MDP framework is expressed by the tuple $\langle S, A, R, P \rangle$ and is depicted in figure 1:

- State space, S, is the set of all valid states (what the agent can see),
- Action space, A, is the set of all valid actions (what the agent can do in response to an observation),
- Reward, R, is the reward (feedback signal from the environment), and
- Policy, P, is the map of actions to a given state.

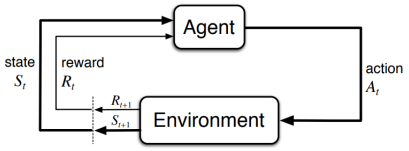

Figure 1: Markov Decision Process represented as a loop

In context of market making, the MDP framework is referred to as a Partially Observable Market Decision Process (POMDP), since the agent cannot have a complete view of the state space. In a POMDP, the partially observable state space is referred to as an observation.

In model-free reinforcement learning, the agent has no prior knowledge of its environment and learns to solve problems through trial-and-error, in which the agent iteratively updates its policy while collecting experience from environmental interactions. There are various ways in which model-free algorithms can update their policies: value-based methods optimize the policy indirectly through estimating action-values; policy-gradient methods optimize the policy directly

without requiring a value function; methods that learn approximations to both policy and value functions are often called actor–critic methods, where "actor" is a reference to the learned policy, and "critic" refers to the learned value function (usually a state-value function) [20]. The algorithms used in this experiment are model-free actor-critic algorithms.

1.6 Deep Reinforcement Learning (DRL)

In Deep Reinforcement Learning (DRL), the agent's policy is an artificial neural network, into which the environment observation is input, triggering an output of the action the agent should take. DRL enables the agent to generalize more effectively than other function approximators for high-dimensional spaces, such as linear approximation [20].

2 Related Work

There has been substantial recent research concerning the successful application of deep reinforcement learning algorithms to complex tasks, such as the game of Go [17] or Atari [4,10]. The aforementioned experiments have implemented value-based algorithms to optimize the agent's policy indirectly through an action-value function, whereas this paper focuses on policy-gradient methods.

In financial services, it has been demonstrated that a model-free approach with policy gradient methods can be effective at trading or portfolio management and that a policy gradient method is more effective at learning the dynamics of trading than Q-learning, a value-based method [1, 5, 8, 11, 12]. Other findings show that using risk-based metrics, such as the Sharpe Ratio (or other variations like Differential Sharpe Ratio), for the reward function (i.e., feedback signal) outperform algorithms that use the more intuitive profit-and-loss reward function [5,8,12]. However, none of these papers use data from limit order books or trade and order flow data as an input into the agent's observation space, but rather price returns.

Within the research community, there is a dearth of published work exploring the application of DRL to automated market making. [19] created a framework for market making and evaluated several value-based reinforcement learning algorithms to achieve a stable performance *without* using a neural network. They used the top five levels of the LOB in combination with a few other hand-crafted features to represent the agent's observation space and a linear combination of tile codings as a function approximator. They attribute success to their custom reward function and linear combination of tile codes, with the help of eligibility traces. [15] proposed a macro-micro duel agent architecture for solving the market making problem, wherein the macro-agent views higher-resolution data and generates trade signals, and the micro-agent views the LOB and is responsible for order execution. Although [15]'s agent outperformed their baseline, the lack of inventory constraints on the agent makes the results impractical for live trading. Unlike these approaches, this paper uses more levels from the LOB, trade and order flow imbalances, and advanced policy-gradient algorithms to achieve stable performance for cryptocurrency market making.

There has been ample research in modeling the LOB using statistical, machine learning, and deep learning approaches. [2, 7, 13, 22] use a marked point process to indirectly predict midpoint price changes and inventory control for market making. [24] use order flow imbalances and linear regressions to predict midpoint changes. [25] use a custom stack of convolutional and LSTM neural networks with an inception module to predict midpoint price changes. Unlike these stochastic and supervised machine learning approaches, which rely on heuristics to create a trading system, this paper uses reinforcement learning with neural networks to learn an end-to-end automated trading solution for market making.

3 Contributions

The main contributions of this paper are the analyses of the DRLMM frameworking using advanced policy gradient-based algorithms. In contrast to existing research [19], this experiment features an expanded scope, including 15 LOB levels, in addition to trade and order flow arrival data for the agent's exploitation. This paper aims to provide the first published research using LOB, TFI and OFI to create an informed deep reinforcement learning market making agent.

The main contributions are detailed below:

1. **Analysis of advanced policy gradient-based algorithms:** The performance and generalization capability across three cryptocurrency pairs are compared for Advantage Actor Critic and Proximal Policy Optimization algorithms.
2. **Reward functions for market making:** Two reward functions, positional profit-and-loss and trade completion, are evaluated in context of deep reinforcement learning applied to market making.
3. **Framework for a DRLMM agent:** Agents that are able to effectively learn the end-to-end process of market making with no prior knowledge are demonstrated.

4 Experiment Design

4.1 Environment Design

The agent's observation space is a combination of three sub-spaces: the environment state space (ESS), consisting of LOB, TFI and OFI snapshots with a lookback window size w; the agent state space (ASS), consisting of handcrafted risk and position indicators; and the agent action space (AAS), consisting of a one-hot vector of the agent's latest action.

4.1.1 Environment State Space (ESS)

In this paper, the ESS consists of eight indicators: 1) LOB levels rendered as stationary prices; 2) cumulative notional value at each price level; 3) imbalances for cumulative notionals; 4) order inflow imbalances at LOB levels; 5) the spread between the best bid and ask prices; 6) custom price momentum indicator; 7) custom trade flow imbalance indicator; and 8) environmental reward signal.

Price Level Distance to Midpoint Distance ξ is the difference between price p at LOB level i from midpoint m at time t. The LOB's first 15 price levels are extracted from *bid* and *ask* sides of the LOB, thus creating 30 values for this feature.

$$\xi_{t,i}^{bid,ask} = \frac{p_{t,i}^{bid,ask}}{m_t} - 1 \tag{1}$$

Cumulative Notional Value at Price Level Cumulative sum χ is the sum of the product of price p and quantity q at LOB level i at time t for both *bid* and *ask* sides. The LOB's first

15 price levels are extracted from *bid* and *ask* sides of the LOB, thus creating 30 values for this feature.

$$\chi_{t,i}^{bid,ask} = \sum_{i=0}^{I-1} p_{bid,ask}^{t,i} \times q_{bid,ask}^{t,i} \tag{2}$$

Notional Imbalances The order imbalances ι are represented by notional value and normalized to the scale $[-1, 1]$. Since there are 15 price levels included, there are 15 values for this feature.

$$\iota_{t,i} = \frac{\chi_{t,i}^{ask,q} - \chi_{t,i}^{bid,q}}{\chi_{t,i}^{ask,q} + \chi_{t,i}^{bid,q}} \tag{3}$$

Order Flow Imbalances (OFI) The sum of notional values for cancel C, limit L, and market M orders is captured between each LOB snapshot. Since i is 15 in this experiment, this feature is represented by a vector of 30 values, 15 per each side of the LOB.

$$OFI_{t,i}^{bid,ask} = OFI(L)_{t,i}^{bid,ask} - OFI(C)_{t,i}^{bid,ask} - OFI(M)_{t,i}^{bid,ask} \tag{4}$$

where $OFI(C)_{t,i}^{bid,ask} = C_{t,i}^{bid,ask,p} \times C_{t,i}^{bid,ask,q}$ and $OFI(L)_{t,i}^{bid,ask} = L_{t,i}^{bid,ask,p} \times L_{t,i}^{bid,ask,q}$ and $OFI(M)_{t,i}^{bid,ask} = M_{t,i}^{bid,ask,p} \times M_{t,i}^{bid,ask,q}$.

Trade Flow Imbalances (TFI) The Trade Flow Imbalances TFI indicator measures the magnitude of buyer initiated BI and and seller initiated SI transactions over a given window w. This indicator is scaled to the range of $[-1, 1]$ and is applied to 3 different windows (5, 15, and 30 minutes), thus is represented by a vector of 6.

$$TFI_t = \frac{gain_t - loss_t}{gain_t + loss_t} \tag{5}$$

where: $gain_t = \sum_{n=0}^{w} BI_n$ and $loss_t = \sum_{n=0}^{w} SI_n$.

Spread The spread is the difference between the best bid p^{bid} and best ask p^{ask}.

$$\varsigma_t = p_t^{bid} - p_t^{ask} \tag{6}$$

Custom RSI The relative strength index indicator (RSI) measures the magnitude of prices changes over a given window w. This custom implementation $CRSI$ replaces the default *mean* calculation with a *sum*, and scales the output to the range $[-1, 1]$. This indicator is applied to three different windows (5, 15, and 30 minutes), and is thus represented by a vector of 3 features.

$$CRSI_t = \frac{gain_t - |loss_t|}{gain_t + |loss_t|} \tag{7}$$

where $gain_t = \sum_{n=0}^{w} \Delta m_n$ if $\Delta m_n > 0$ else 0 and $loss_t = \sum_{n=0}^{w} \Delta m_n$ if $\Delta m_n < 0$ else 0 and $\Delta m_t = \frac{m_t}{m_{t-1}} - 1$.

Reward The reward r from the environment, as described in section 4.1.4.

4.1.2 Agent State Space (ASS)

The ASS provides transparency into the agent's operations. In this paper, the ASS consists of five indicators: 1) percentage of inventory utilized; 2) total profit-and-loss; 3) unrealized profit-and-loss; 4) distance for each open order to midpoint price; 5) a custom function to measure the progress of an order's position in the queue and partial executions.

Inventory Ratio The agent's inventory ratio υ is the inventory count ic represented as a percentage of the maximum inventory IM, scaled between $[0, 1]$, and represented as a vector with 2 values.

$$\upsilon_t^{long,short} = \frac{ic^{long,short}}{IM} \tag{8}$$

Total PnL The agent's total profit-and-loss $TPNL$ is the sum of realized and unrealized profit and losses. In this experiment, the $TPNL$ is scaled by a scalar value ρ, which represents the daily PnL target. The current midpoint m at time t marks the unrealized position to market value. Total PnL is represented as a scalar.

$$TPNL_t = \frac{PNL_t^{realized} + PNL_t^{unrealized}}{\rho} \tag{9}$$

Unrealized PnL The agent's current unrealized PnL $PNL_{unrealized}$ is the sum of unrealized PnL across all open positions. The unrealized PnL feature is represented as a vector with 2 values, containing both long and short positions.

$$PNL_{unrealized}^{long,short} = \frac{p_{average}^{long,short}}{m} - 1 \tag{10}$$

where m is the midpoint price at time t.

Limit Order Distance to Midpoint The agent's limit order distance to midpoint is the distance ζ of the agent's open *bid* and *ask* limit orders L to the midpoint price m at time t. The feature is represented as a vector with 2 values.

$$\zeta_t^{long,short} = \frac{L_t^{bid,ask}}{m} - 1 \tag{11}$$

Order Completion Ratio Order completion η is a custom indicator that incorporates an order's relative position in the LOB queue q and partial executions ex relative to the order size sz. The feature scales the value to the range $[-1, 1]$ and is represented as a vector with 2 values.

$$\eta_t^{long,short} = \frac{ex_t^{long,short} - q_t^{long,short}}{q_t^{long,short} + sz_t^{long,short}} \tag{12}$$

4.1.3 Agent Action Space (AAS)

The agent action space A consists of 17 possible actions, as outlined in Table 4.1.3. The idea is that the agent can take four general actions: do nothing, place its orders asymmetrically in the LOB, skew its orders to either *buy* or *sell* sides of the LOB, or flatten its inventory of positions.

Action ID	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bid	0	0	0	4	4	4	4	9	9	9	9	14	14	14	14
Ask	4	9	14	0	4	9	14	0	4	9	14	0	4	9	14
Action 1	No action														
Action 17	Market order M with size ic														

Table 2: The agent action space with 17 possible actions. The numbers in the *Bid* and *Ask* rows represent the price level at which the agent's orders are set to for a given action.

4.1.4 Reward Function

There are two reward functions evaluated in this paper as part of the DRLMM framework: positional PnL and trade completion. Each function returns a reward r, which is the agent's feedback signal the agent uses to learn.

Positional PnL The reward r is the change in the agent's unrealized position value, plus any realized gains or losses from the current step t. If no positions have been closed between time steps t and $t-1$, the realized gains input is set to 0.

$$r_t = \Delta m_t \times ic_t + PNL_t^{realized} \tag{13}$$

where the midpoint price return is $\Delta m_t = \frac{m_t}{m_{t-1}} - 1$. and inventory count ic is the number of positions held in inventory at time t.

Trade Completion The reward r is a value between $[-1, 1]$ that is derived from a clipped goal-based custom reward function. This reward function only generates a value other than 0 when a position is closed and realized PnL is generated. If the realized PnL is greater (or less) than a predefined threshold, the reward r is 1 (or -1) otherwise, if the realized PnL is in between the thresholds, the realized PnL in percentage terms is the reward. Using this clipped approach, the agent is encouraged to open and close positions with a targeted profit-to-loss ratio, and is not rewarded for longer term price speculation.

$$r_t = \begin{cases} 1, & \text{if } PNL^{realized} \geq \epsilon * \varpi \\ -1, & \text{if } PNL^{realized} \leq -\varpi \\ PNL^{realized}, & \text{otherwise} \end{cases} \tag{14}$$

where ϵ is a constant used for the multiplier and ϖ is a constant used for the threshold.

4.2 Function Approximator

The function approximator is a multilayer perceptron (MLP), which is a forward feed artificial neural network. The architecture in this implementation consists of 3-layer network with a single shared layer for feature extraction, followed by separate policy and value networks.

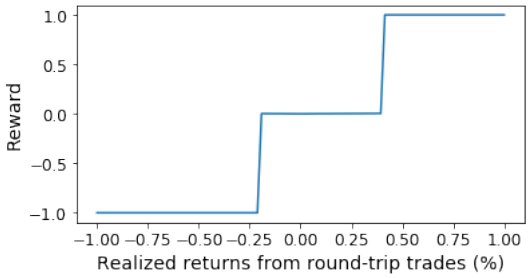

Figure 2: Trade Completion reward function for returns ranging from -1% to 1% when ϵ is set to 2 and ϖ is set to 0.2%.

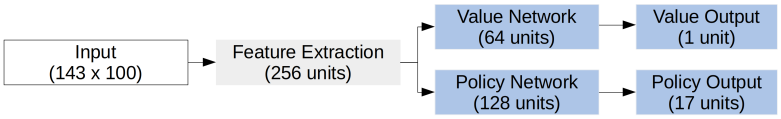

Figure 3: Architecture of MLP neural network used in experiment. The input space is a matrix of 143 features with 100 lags. *Gray* represents shared layers. *Blue* represents non-shared layers.

4.3 Agents

Two advanced policy gradient methods are used as market making agents: Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO). Since both learning algorithms run on multiple processes, they require nearly the same amount of time to train. The same policy network architectures are used in each experiment, and hyperparameters are listed in the appendix (section A.1). It is worth noting that Actor-Critic with Experience Replay (ACER) was originally in scope for this paper, but was de-scoped due to poor performance after demonstrating instability and slow learning in comparison to A2C and PPO.

4.3.1 Advantage Actor-Critic (A2C)

The A2C is an on-policy model-free actor-critic algorithm that is a descendent of policy gradient methods. It interacts with the environment asynchronously while maintaining a policy $\pi(a_t|s_t;\theta)$ and estimate of the value function $V(s_t;\theta_v)$, and synchronously updates parameters to improve GPU efficiency compared to sibling algorithm A3C [9]. Being a more evolved algorithm than a vanilla policy gradient, the A2C algorithm uses its value function to reduce variance and improve stability when learning through calculating the advantage $A(s_t|a_t)$ at each state. The advantage is the difference between the action-value $Q(s_t|a_t)$ and value function $V(s_t)$ (i.e., the advantage of taking a particular action at a given state). The A2C algorithm also uses n-step returns to update both policy and value-function, which results in more stable learning than a vanilla policy gradient, which uses 1-step returns.

The A2C update is calculated as

$$\nabla_\theta J(\theta) = \nabla_{\theta'} log\pi(a_t|s_t;\theta')A(s_t,a_t;\theta,\theta_v) \tag{15}$$

where $A(s_t,a_t;\theta,\theta_v)$ is the estimate of the advantage function given by $\sum_{i=0}^{k-1} = \gamma^i r_{t+i} + \gamma^k V(s_{t+k},\theta_v) - V(s_t,\theta_v)$, where k can vary by the upper bound t_{max} [9].

4.3.2 Proximal Policy Optimization (PPO)

The PPO is an on-policy model-free actor-critic algorithm and a descendent of policy gradient methods. It interacts with the environment asynchronously while indirectly optimizing a policy $\pi_\theta(a_t|s_t)$ through a clipped surrogate function L^{CLIP} [16] that represents the difference between the new policy after the most recent update $\pi_\theta(a|s)$ and the old policy before the most recent update $\pi_{\theta_k}(a|s)$. Although this algorithm descends from policy gradient methods, it is different in that the surrogate function, not the policy, is optimized directly. This surrogate function removes the incentive for a new policy to depart from the old policy, thereby increasing learning stability. Like the A2C, the PPO algorithm updates its parameters θ synchronously.

The PPO Clip update is calculated as

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s_t, a_t)\right) \tag{16}$$

where ϵ is a hyperparameter constant [16].

5 Methodology

5.1 Data Collection

LOB data for cryptocurrencies is free to access via WebSocket, but not readily downloadable from exchanges, and therefore requires recording. The data set for this experiment was recorded using level-3 tick data from the Coinbase exchange, persisted into a Mongo database, and replayed to fully reconstruct the LOB and derive TFI and OFI metrics. During data replay, snapshots of the LOB, TFI and OFI statistics were taken every n-intervals and exported to a CSV file to reduce the computational burden of parsing millions of ticks per trading day. In this experiment, the snapshots were taken every 1-seconds, resulting in approximately 84,600 records per trading day.

5.2 Data Normalization

Since LOB data cannot be used for machine learning without preprocessing, it is necessary to apply a normalization technique to the raw data set. For this experiment, the data set was normalized using the approach described by [21], which transforms the LOB from non-stationary into a stationary feature set, then uses the previous trading day to fit and z-score normalize the current trading day's values, and in which data point x is σ standard deviations from the mean $\bar{x}$.

$$z_x = \frac{x - \bar{x}}{\sigma} \tag{17}$$

5.3 Environment Rules

The environment invokes business rules to make the simulation as realistic as possible. For example, agents are permitted to hold only one open order at a time on each side of the LOB (the ask and bid sides). Once an order is filled, the agent must wait until the next environment step to select an action (such as replenishing the filled order in the order book). Additionally,

Reward function	A2C			PPO		
	BTC	***ETH***	***LTC***	***BTC***	***ETH***	***LTC***
Positional PnL	0.0034%	-0.0001%	-0.0002%	-0.0274%	0.0001%	0.0005%
Trade Completion	0.0017%	0.0005%	0.0017%	0.0001%	0.0002%	0.0018%

Table 3: Average of the average per-trade PnL of the *same currency* for two testing days: November 1 and 2, 2019.

agents are able to update partially filled open orders with new price levels, and upon order completion, the average execution price is updated in all PnL or risk calculations. If the agent has an open position on one side of the book and fills an open order on the opposing side, the existing position is netted in FIFO for PnL calculations.

Each time a new order is posted to the LOB, its price and standing in the queue is updated according to relative notional value (e.g. price, quantity). If the agent posts a new order at the same price as existing resting orders, the new order price jumps a one-tick increment ahead of the queue by default, assuming there is no liquidity posted at that price level. However, it is important to note that this hard-coded order posting strategy is less relevant for live trading, wherein other market participants have the ability to repost their own orders ahead of the agent's, therby triggering a series of requotes.

In general, transaction fees for limit orders are not taken into account. Since limit order fees or rebates vary by exchange, transaction fees are only applied to market orders (0.20% per side), which the agent can engage by selecting the "flatten inventory" action.

5.4 Training and Testing

The market-making agents (A2C and PPO) were trained on 8 days of data (September 27 to October 4, 2019) and tested on 2 trading days (November 1 and 2, 2019) using three cryptocurrency pairs: Bitcoin (BTC-USD), Ether (ETH-USD), and Litecoin (LTC-USD). Each trading day consists of 84,600 snapshots, which reflects 1 snapshot per second in a 24-hour market. Each agent's performance is evaluated on both total daily and average trade returns.

In each experiment, agents are trained for 10 million steps using 5 action repeats, enabling agents to accelerate learning and take 50 million environmental steps. It is important to note that during action repeats, the agent's action is only performed once on the first step, with all subsequent steps consisting of "no action," thereby avoid performing illogical repetitive actions multiple times in a row, such as "flatten all inventory." Each agent uses the hyperparameters specified in the appendix A.1.

6 Results

In this section, the performance of each agent (PPO and A2C) are compared on the testing data set. The performance of each data set is evaluated in terms of daily returns (Table 6), as well as the average trade return (Table 6). Additionally, the agents are tested on different cryptocurrency pairs to evaluate their capability to generalize. Both agents learned how to generate daily profits and manage inventories using the DRLMM framework.

Reward function	A2C			PPO		
	BTC	***ETH***	***LTC***	***BTC***	***ETH***	***LTC***
Positional PnL	0.0934%	-0.0851%	-0.1836%	-0.1094%	0.1320%	0.3462%
Trade Completion	3.3063%	0.0643%	1.5078%	0.4051%	0.0650%	1.4223%

Table 4: Average daily returns of the *same currency* for two testing days: November 1 and 2, 2019.

6.1 Same Currency

The agents achieved positive returns for the evaluation metrics on the testing data sets. BTC and LTC trained agents generated more stable returns than the ETH agent. Interestingly, the performance of each algorithm varies based on the number of action repeats performed during testing, as seen in Figure 4. For example, the A2C algorithm achieves greater returns with a smoother equity curve using smaller action repeat values (i.e, 1 or 5), whereas the PPO algorithm performs better with larger action repeats values (i.e., 5 or 10). In the case of A2C with fewer action repeats, the agent appears to manage its position inventory better, thereby reducing the holding period of each position. For the PPO agent, the opposite effect occurs: The agent accumulates a large position quickly, and holds onto the positions until they become profitable.

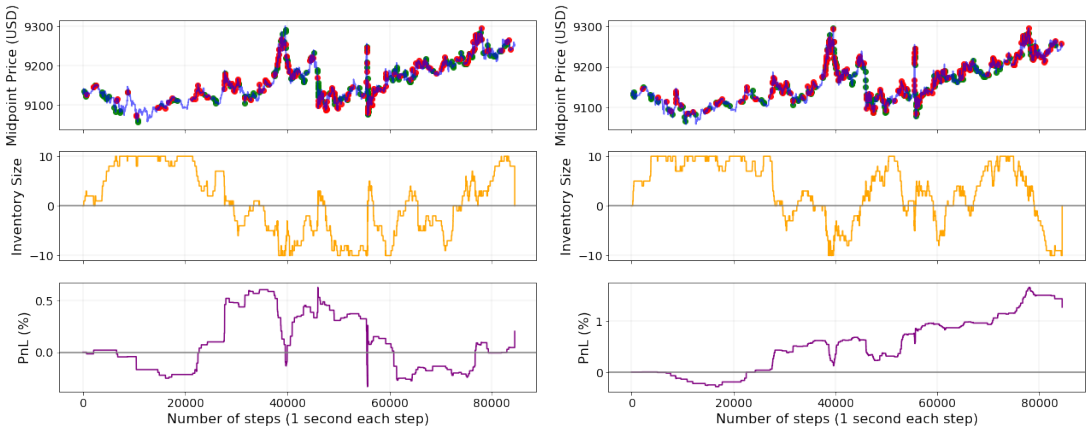

Figure 4: Comparison of action repeat settings for PPO agent trading BTC-USD on November 1, 2019 using trade completion reward function. Buy and sell orders executed by the agent are marked in *red* and *green*. *Left:* Agent with action repeat set to 1. *Right:* Agent with action repeat set to 10.

As for reward functions, the trade completion approach yielded greater and more stable returns than positional PnL. This is due to trade completion agents generating more trades than positional PnL agents, while also maintaining a lower inventory count, as seen in in Figure 5. Another difference between the two reward functions is the agents' use of the "flatten inventory" action, which instantly liquidates the agent's position. The positional PnL agents used this action more frequently than trade completion agents, which is expected since future changes in midpoint prices directly impact the reward signal.

6.2 Different Currencies

When applied to currencies other than the respective training data set, each agent's performance was relatively similar to results in section 6.1. Some agents trained on different currencies performed better than the *same currency* approach counterparts, as seen in Figure 6. For

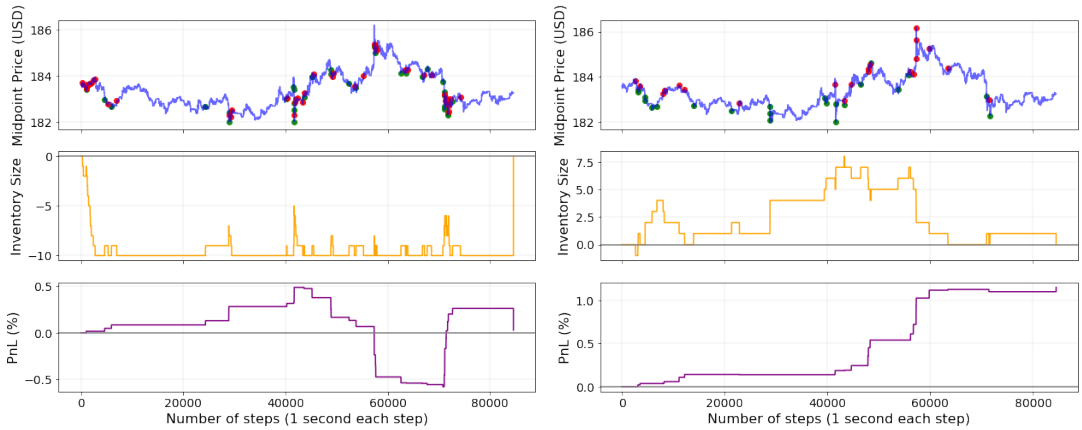

Figure 5: Comparison of reward function effects on trading activities for A2C agent trading ETH-USD on November 2, 2019. Buy and sell orders executed by the agent are marked in *red* and *green*. *Left:* Agent with positional PnL reward function. *Right:* Agent with trade completion reward function.

example, the PPO agents trained on BTC-USD yielded better performance on ETH-USD data than the agent trained on ETH-USD data. Similar results occurred with the A2C algorithm, where agent trained on BTC-USD outperformed the agent trained on ETH-USD. These results support the findings of [18], where LOBs exhibit universality.

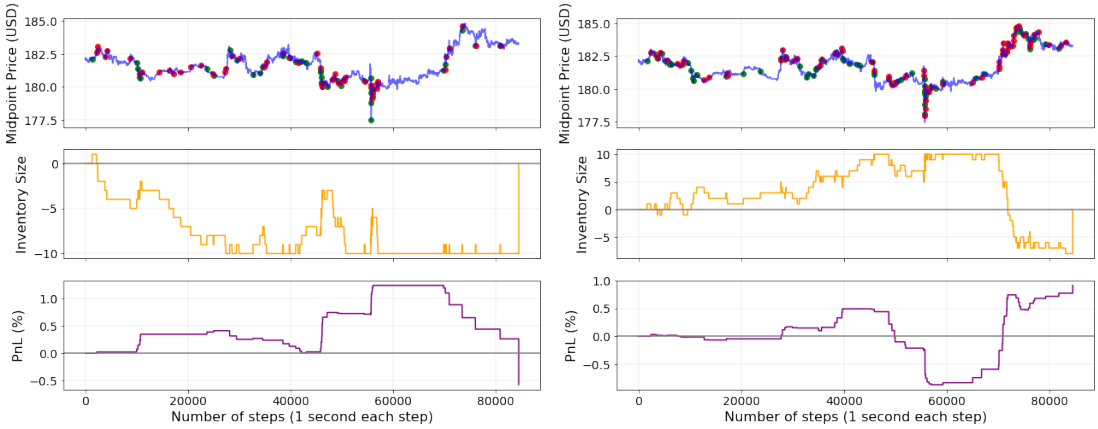

Figure 6: Example of improved performance when PPO agent trades different currency (ETH-USD) from training currency (LTC-USD) on November 1, 2019 using trade completion reward function and action repeat 10. Buy and sell orders executed by the agent are marked in *red* and *green*. *Left:* Agent trained on ETH-USD. *Right:* Agent trained on LTC-USD.

7 Conclusion

This paper applies deep reinforcement learning to cryptocurrency market making using advanced policy gradient methods. The proposed DRLMM framework demonstrates the effectiveness of policy gradient methods in solving stochastic control problems in market making. Using only limit order book data and trade and order flow indicators, the agent is able to generate profitable daily returns without prior knowledge of how the market making process is performed. Moreover, the DRLMM framework enables the agent to generate profitable daily returns on

different cryptocurrency data sets, highlighting the agent's ability to generalize effectively and learn the non-linear characteristics of its obervation space.

In future research, there are many other features to explore in the DRLMM framework. It would be beneficial to compare additional reward functions, such as Differential Sharpe Ratio [11], or Asymmetrical Dampining [19]. Since there have been many advancements in the supervised learning domain for LOBs [25], it would be worthwhile to test those techniques in DRLMM.

References

[1] Rakesh Agrawal, Paul E. Stolorz, and Gregory Piatetsky-Shapiro, editors. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), New York City, New York, USA, August 27-31, 1998*. AAAI Press, 1998.

[2] E. Bacry and J. F Muzy. Hawkes model for price and trades high-frequency dynamics, 2013.

[3] R. Cont, A. Kukanov, and S. Stoikov. The price impact of order book events. *Journal of Financial Econometrics*, 12(1):47–88, Jun 2013.

[4] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *ArXiv*, abs/1802.01561, 2018.

[5] Carl Gold. Fx trading via recurrent reinforcement learning. *2003 IEEE International Conference on Computational Intelligence for Financial Engineering, 2003. Proceedings.*, pages 363–370, 2003.

[6] Coherent Market Insights. Global algorithmic trading market to surpass us$ 21,685.53 million by 2026, 2019.

[7] Baron Law and Frederi Viens. Market making under a weakly consistent limit order book model. 2019.

[8] David W. Lu. Agent inspired trading using recurrent reinforcement learning and lstm neural networks, 2017.

[9] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[11] John Moody and Matthew Saffell. Reinforcement learning for trading. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, pages 917–923, Cambridge, MA, USA, 1999. MIT Press.

[12] John E. Moody and Matthew Saffell. Learning to trade via direct reinforcement. *IEEE Trans. Neural Networks*, 12(4):875–889, 2001.

[13] Maxime Morariu-Patrichi and Mikko S. Pakkanen. State-dependent hawkes processes and their application to limit order book modelling, 2018.

[14] OpenAI. Openai five. `https://blog.openai.com/openai-five/`.

[15] Yagna Patel. Optimizing market making using multi-agent reinforcement learning, 2018.

[16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[17] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016.

[18] Justin Sirignano and Rama Cont. Universal features of price formation in financial markets: perspectives from deep learning, 2018.

[19] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. Market making via reinforcement learning, 2018.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[21] Avraam Tsantekidis, Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Using deep learning for price prediction by exploiting stationary limit order book features, 2018.

[22] Peng Wu, Marcello Rambaldi, Jean-François Muzy, and Emmanuel Bacry. Queue-reactive hawkes models for the order flow, 2019.

[23] Ke Xu, Martin D. Gould, and Sam D. Howison. Multi-level order-flow imbalance in a limit order book. 2019.

[24] Ke Xu, Martin D. Gould, and Sam D. Howison. Multi-level order-flow imbalance in a limit order book. *Market Microstructure and Liquidity*, Nov 2019.

[25] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, Jun 2019.

A Appendix

A.1 Agent Hyperparameters

The hyperparameters used to train agents in the experiment.

#	Parameter	Value
1	action_repeat	5
2	window_size	100
3	transaction_fees	0% for Limit, 0.2% for Market
4	max_positions	10
5	gamma γ	0.99
6	learning rate α	3e-4
7	LOB levels	15
8	n_steps (PPO)	256 (PPO) / 40 (A2C)
9	training_steps	10,000,000
10	action_space	17

Table 5: Hyperparameters for experiments.

A.2 Testing Results - Same Currency

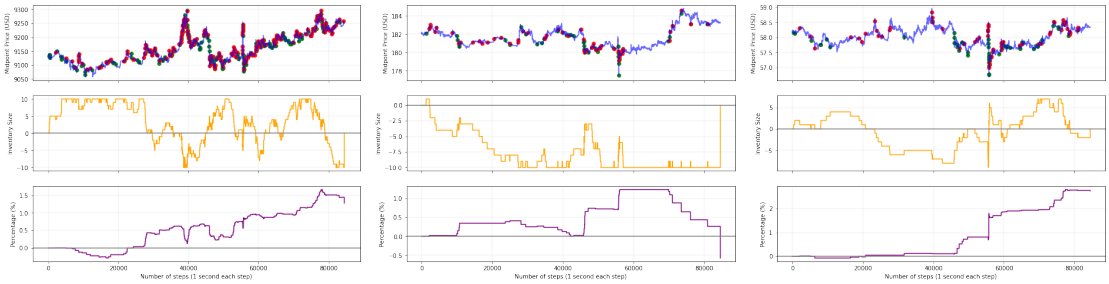

Figure 7: Test results of PPO agents on November 1, 2019. Buy and sell orders executed by the agent are marked in *red* and *green*. *Left:* BTC-USD with action repeat set to 10. *Center:* ETH-USD with action repeat set to 10. *Right:* LTC-USD with action repeat set to 5.

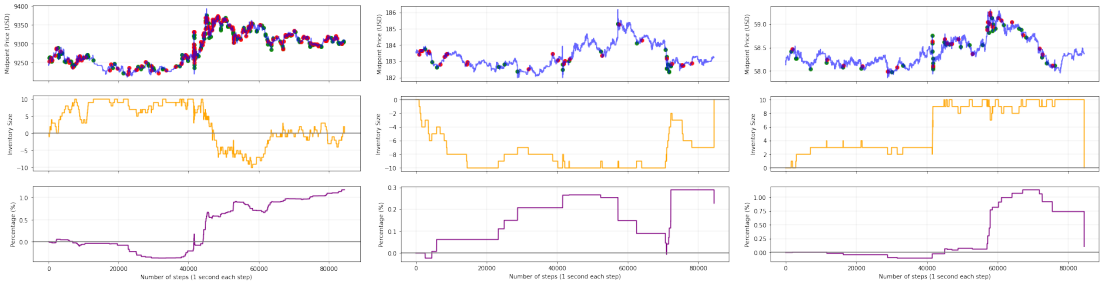

Figure 8: Test results of PPO agents on November 2, 2019. Buy and sell orders executed by the agent are marked in *red* and *green*. *Left:* BTC-USD with action repeat set to 5. *Center:* ETH-USD with action repeat set to 10. *Right:* LTC-USD with action repeat set to 10.

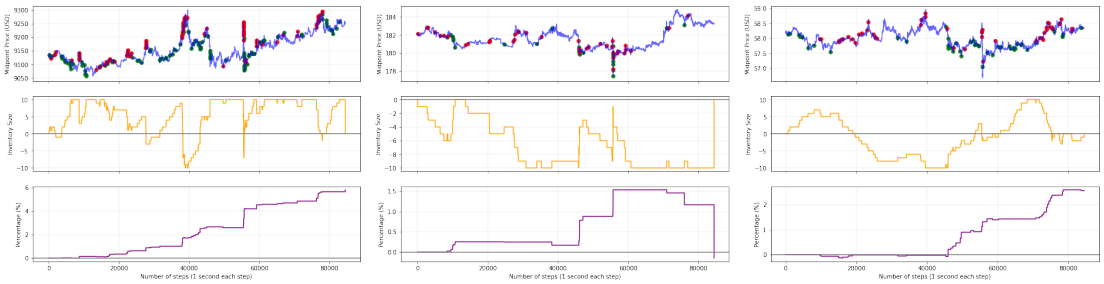

Figure 9: Test results of A2C agents on November 1, 2019. Buy and sell orders executed by the agent are marked in *red* and *green*. *Left:* BTC-USD with action repeat set to 1. *Center:* ETH-USD with action repeat set to 5. *Right:* LTC-USD with action repeat set to 1.

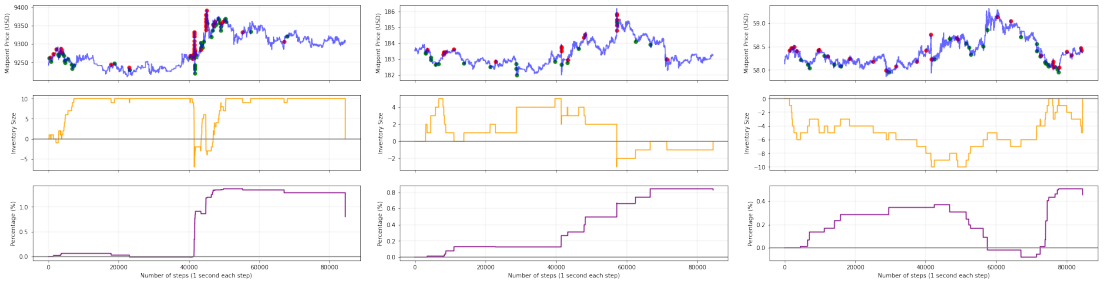

Figure 10: Test results of A2C agents on November 2, 2019. Buy and sell orders executed by the agent are marked in *red* and *green*. *Left:* BTC-USD with action repeat set to 1. *Center:* ETH-USD with action repeat set to 1. *Right:* LTC-USD with action repeat set to 1.

A.3 Testing Results - Different Currencies

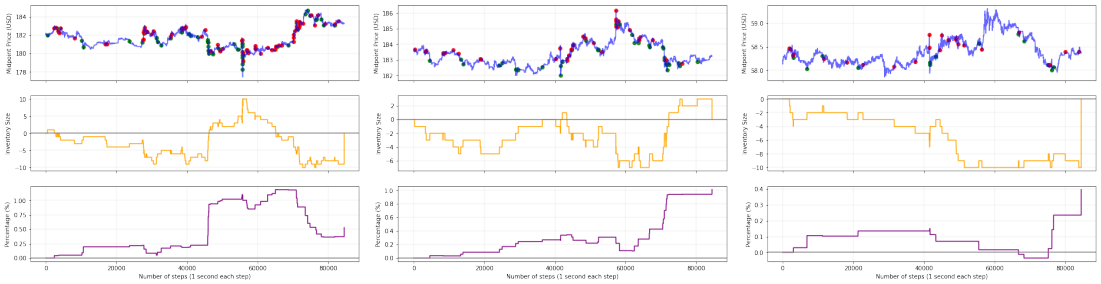

Figure 11: Test results of PPO agents on various test data sets. Buy and sell orders executed by the agent are marked in *red* and *green*. *Left:* BTC-USD trained agent tested on ETH-USD's data from November 1, 2019 with action repeat set to 10. *Center:* BTC-USD trained agent tested on ETH-USD's data from November 2, 2019 with action repeat set to 5. *Right:* ETH-USD trained agent tested on LTC-USD's data from November 2, 2019 with action repeat set to 5.

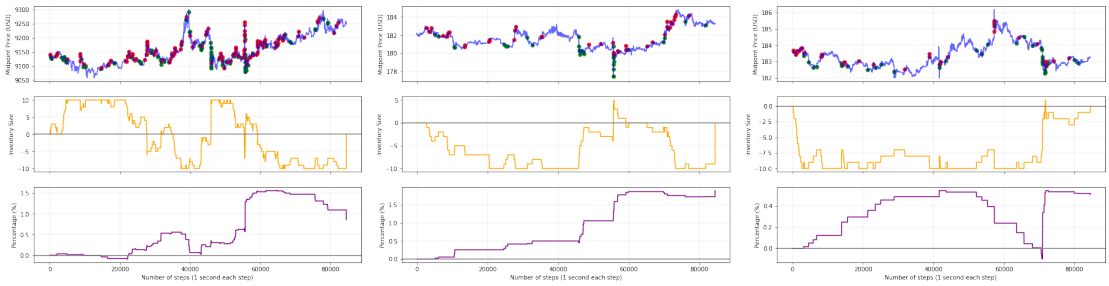

Figure 12: Test results of A2C agents on various test data sets. Buy and sell orders executed by the agent are marked in *red* and *green*. *Left:* ETH-USD trained agent tested on BTC-USD's data from November 1, 2019 with action repeat set to 1. *Center:* BTC-USD trained agent tested on ETH-USD's data from November 1, 2019 with action repeat set to 10. *Right:* LTC-USD trained agent tested on ETH-USD's data from November 2, 2019 with action repeat set to 10.