Quantum Reinforcement Learning for High Frequency Trading

Abstract: We introduce a new approach of Reinforcement Learning Application for High Frequency Trading called Quantum Reinforcement Learning as our agent learns to react on 'quantum' individual events in Limit Order Book – single Limit Order Book updates and single trades (and optionally single Orders if provided by Exchange). We claim that such level of learning granularity allows our agent to find optimal trading strategies by on-line modeling of Market Microstructure with a maximum rate and precision.

INTRODUCTION

There are 2 main approaches in Modeling of High Frequency Trading – traditional statistical approaches such as Market Microstructure modeling and more recent Machine Learning approaches detecting such Microstructure models 'on-line' without statistical modeling of various distributions in Limit Order Book data. Machine Learning models gained popularity on higher frequencies for it's dynamic nature as static statistical modeling is much more difficult on such trading frequencies. Machine Learning approaches can be further categorized in 2 main groups – supervised predictive models and unsupervised Reinforcement Learning models. Supervised models attempt to predict short term price spikes based on some history in Limit Order Book events preceding such spikes, manually marked by a supervisor from price history and presented to the model for further learning – typically to make 3 decisions on each tick – buy, sell or stay out of position. Such predictive models have some drawback of poor ability to generalize on unseen market data. On the other hand unsupervised Reinforcement Learning models allow agent to autonomously learn trading strategies by first acting randomly but then correcting itself trying to maximize final PnL. Such approach tends to generalize much better even in strongly stochastic market environment. In this paper we choose to use unsupervised approach and we also push Reinforcement Learning to it's 'quantum' limits by allowing our agent to learn how to trade from individual events in Limit Order Book - such as individual update of some book's level, individual trade or individual book order (if order log is provided by exchange). We claim that such Quantum Reinforcement Learning can find optimal trading strategies with maximal precision. We then investigate three major Reinforcement learning approaches in such learning environment – namely Cross Entropy, Deep Q-Learning and Policy Proximal Optimization. As yet another contribution of the current research we have observed importance of high level information for making trading decisions even on such trading frequencies which to our knowledge has not been used in previous publications on Deep Learning applications for High Frequency Trading. Namely, our trading agent is analyzing both micro events in Limit Order Books along with higher level information represented as time series of price candlesticks on various frequencies up-to 15 minutes. We claim that such information is important for agent as in some sense HFT is itself a computerized form of well know 'scalping' human trading strategies – hence we attempt to model our agent's behavior to closely follow 'scalping' which utilizes high level information as yet another indicator to enter positions. Eg – if there is a combination of 2 factors - 15 minutes based bearish trend and Limit Order Book is currently 'empty' on a buy side – it's a good indicator to enter short position. In future we also plan to add 'news analysis' as part of high level information for trading agent by utilizing some powerful language models such as GPT3. Finally our models also accurately account for both maker and taker's fees on each transaction which makes this work quite practical.

TRADING MODEL

Every Reinforcement Learning Model starts by specifying Environment States and Agent's actions along with description of how Environment State changes after agents undertakes certain actions and which reward agent gains by choosing specific actions.

Below is a description of three types of Microstructure events, that lead to the change in the trading environment.

1. Book Level Update event.
Typically most exchanges provide a Market Data channel that sends such update events:
We represent it as a tuple ('update_type','price_level_absolute','price_level_relative','size_absolute', 'size_delta', 'timestamp')

update_type can be either Insert, Delete or Update. Absolute Price Size specifies actual level's price and relative price's size specifies a number of an updated level in the book (eg best ask would be level 0 at sell side). Size's delta of the update specifies how much level is modified, absolute size specifies new size of the level after update is applied.

As can be noted we also add a timestamp as a part of event information which allows agent to learn potentially important information from current frequency of updates.

2. Trade event.
This is also a common channel where exchanges broadcast information on committed trades.
Trade is represented as a tuple: ('price_absolute','size', 'trade_direction', 'timestamp')

3. Individual trading order
Some exchanges also broadcast information on individual trading orders. In such case our model incorporates such events as well in the form of the following tuple:
('order_type','price_level_absolute','price_level_relative','order_size','timestamp')

Our agent makes trading decisions upon arrival of each individual event from the above list. It makes such decisions by supplying it's model's NN (Neural Network) with a current state of trading environment (coded in a way, described below).

Below is a description of the coding scheme for the environment state.

Environment State is represented as a combination of two States: Market Microstructure State (State_MM) + Agent's Positions State (State_AP).

Let us first describe State_MM.

In turn this state can be split into three parts: State_LOB + State_micro_structure_events + State_high_level_prices

State_LOB is a current state of a Limit Order Book with 10 best sell and buy price levels with sizes + a timestamp.

State_micro_structure_events is represented as 3 time series of events of each type (book update, trade, individual order). Each time series contains a 100 of last events of corresponding type, coded in a way described above.

State_high_level_prices represents 3 time series, composed of 10 candlesticks each on the following time frequencies (15 minutes, 1 minute, 1 second) + 3 pairs of Support, Resistance Levels – more specifically Min and Max prices for each series. Support and Resistance levels information is considered to be a hint to the model to act differently if price is approaching one of such levels compared to times when price is in the middle of the corresponding price range. Candlestick information is coded as a following tuple: (‘price_open’, ‘price_max’,’price_min’,’price_close’)

Let us now describe State_AP – a state, representing agent’s current trading position

This state is split into two following parts: State_Open_Position + State_Limit_Orders

State_Open_Position represents a total current open position as an open position’s size – positive for buy or negative for sell.

In general it would be tempting to represent State_Limit_Orders as a vector of certain length of orders on every book level less than vector’s length. This way agent could maintain multiple limit orders on different book levels. However in this work simplifying assumption was made to maintain only 2 limit orders for best ask and best bid assuming that maintaining a net of limit orders maybe beneficial only in actual trading environment where it is important to add an order in advance to prevent price ‘slipping’ effect. In this work however we do not analyze such effects and hence limit our orders to just best bid and best ask. Hence State_Limit_Orders is represented as a pair (‘size_best_bid’, ‘size_best_ask’).

Now, once a Trading Environment is specified let us proceed to the description of agent’s possible actions and rewards for those actions.

We assume that agent’s action can be represented as a tuple:
(‘best_ask’,’best_bid’,’episode_close_indicator’).
Specifying these delta’s instructs and agent to properly maintain his two current limit orders. Depending on the output from the model’s NN – he can decide to keep same order sizes or possibly increase some of them, or decrease or completely cancel.

Episode_close_indicator when set to 1 instructs an agent to close his open position and finalize episode of Reinforcement Learning.

Let us now describe how agent is being rewarded for actions.

There are two possible ways to simulate how Book Level is being traded. It is common to assume Poisson distribution of orders on best bid / ask levels and hence estimate when potentially agent’s order can be consumed partly or fully. Potentially parameters of Poisson distribution could be calibrated from information on Limit Order Book updates presented above, however in this work another approach is used for estimation of time when agent’s order is consumed. Namely internally simulation maintains approximate location of agent’s order inside level’s orders FIFO and thus it is assumed that first trade, that consumes level’s size up-to agent’s location would also consume agent’s order. In this case agent’s reward is calculated as follows:

R = maker_fee – trade_price*size, maker_fee > 0

If size is positive and that is a limit buy – we assume that agent spends money to buys specified size, if size is negative and that is limit sell – we assume – agent is payed back for specified trades size.

In both cases agent is rewarded with maker_fee.

The moment when position is filled is not directly related to trader's current action – rather it depends on his previous actions and that is a typical scenario for Reinforcement Learning.

Another scenario when agent is reward in our model is when episode is closed. In this case we compute reward as follows:

R = taker_fee + trade_price * open_size, taker_fee < 0

In this case when open_size is positive and episode is closed – agent sells his current position and gets rewarded with the sold amount. If open size is negative – agent is closing his short position and pays corresponding amount for that. In both cases agent pays a taker's fee as he is using a market order.

Finally, let us describe how environment is modified upon every step.

There are two changes that happen in the environment upon event's arrival: first of all a time series of corresponding event's type is shifted by 1 event in State_MM. In addition – State_MM's Limit Order Book is properly updated. Finally, possible changes in State_high_level_prices state is monitored based on timestamp of the event – eg next second's candlestick may be closed if event's timestamp crosses previous second. In addition to that State_AP can be modified as follows: if agent's action specifies changes in his limit orders – that is being done. Also if it is estimated that agent's order is consumed – his limit order is updated properly as well as his open position.

AGENT'S NEURAL NETWORK

Depending on the Reinforcement Learning Approach used – weather it is DQN or Proximal Policy Optimization or Cross Entropy – output of the network differs. For DQN – it is estimated value of a pair (S,a) – where S is environment's state and a is an agent's action. Agent selects and action which maximizes Q(S,a). If it is Proximal Policy Optimization or Cross Entropy – output is a probability distribution of agent's possible actions in a given state. In both cases however there is a common part of network architecture which consumes state of the environment as an input and is described below.

LOB part of the input state is processed by 2 Linear Layers of 64 neurons each, producing output of length 10

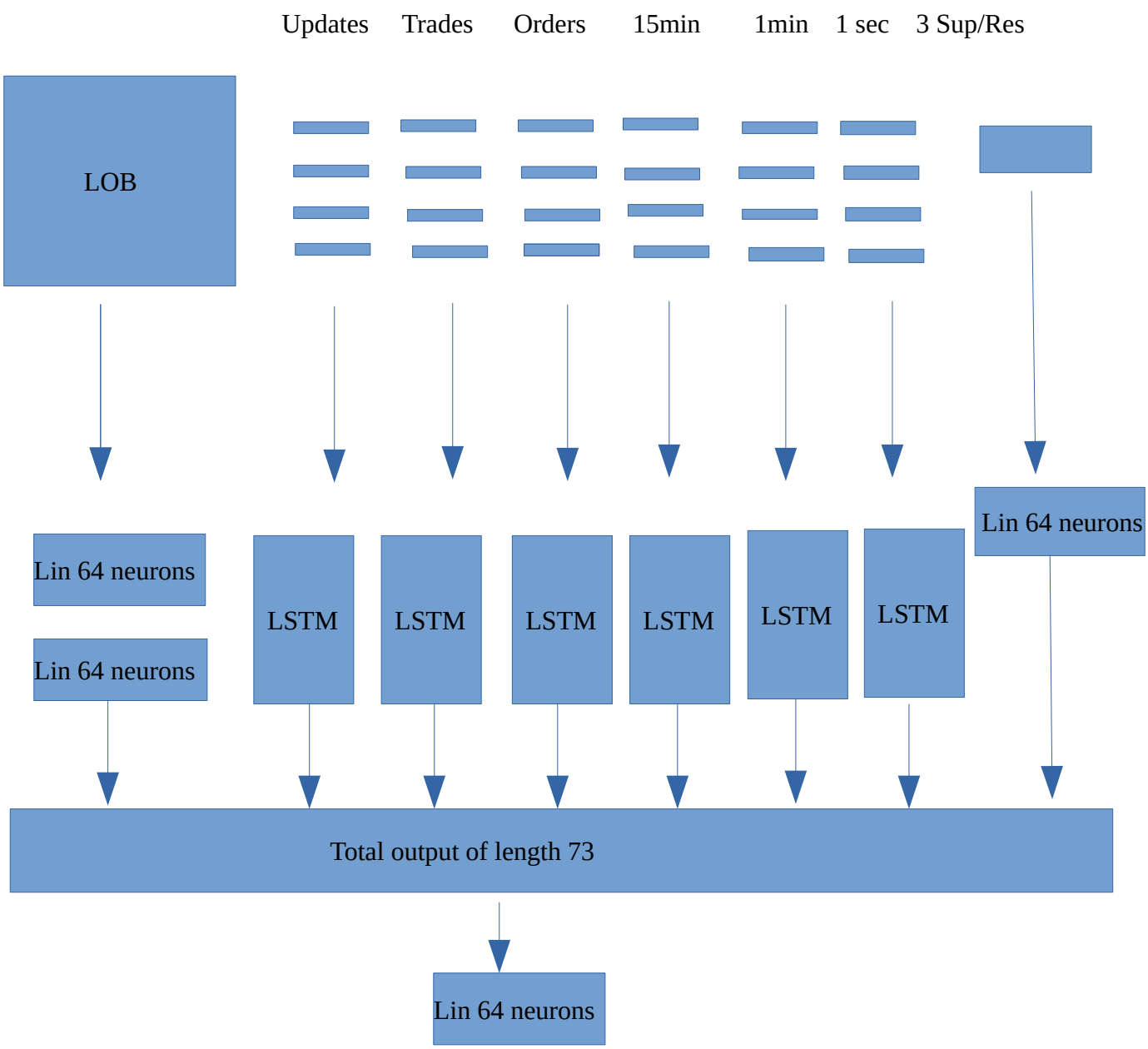
Each of 3 events time time series is processed by LSTM with 64 neurons – each producing an output vector of length 10

Each of 3 candlestick time series is processed by LSTM with 64 neurons – each producing an output vector of length 10

3 pairs of Support / resistance Levels are processed with single Linear Layer with 64 neurons with out put of length 3

All output vectors are then concatenated to produce a single output of length 73 which is then processed by another Linear Layer with 64 neurons to produce either Q-value for DQN or a probability distribution for Proximal Policy Optimization and Cross Entropy approaches.

Graphically architecture is depicted below:



EXPERIMENTAL SETUP AND RESULTS

In order to evaluate the above Quantum RL scheme, data was collected using WebSocket BitMex Api for 7 consecutive days for XBTUSD. Bitmex does not have OrderLog channel – so only Market data and Trades channels were used.

The trained model was evaluated on data from the 8’th day following 7 training days.

Results for each RL method – DQN, PPO and CE are presented below