**Crypto Quant Trading**

**1. Brief Intro of Crypto project**

The CQT package is a package for crypto quant trading. It includes the following component:

1. **DataGen**: Unified interface for data query and processing. Specific data objects are designed to handle various trading data for crypto assets. It contains utility functions to query data from different exchanges/data providers, and restore them in a standard OHLCV data format indexed by query dictionary
2. **Env**: Built from 'spot', 'forward' and 'vol' asset env components, the env class is used to hold the data from different types of assets. In each env (component), we attach the corresponding env configurations as guideline to postprocess and analyze the input data, performance statistical inference.
3. **Analyze**: Generate analysis from environments and provide interface functions for a large set of market signals (e.g. bear/bull, moving average crossing, and etc)
4. **Ledger**: Standardized holding class, which provides the basic logics of buy/sell and set aside functions to track the changes on the asset holdings and cash
5. **Strategy**: Valuation engine that combines Env and Ledger. It takes the signals generated from Env, and follow the strategy logic to update the Ledger. Back testing is provided to test the strategy against the historical data in the Env. New strategies be derived from the base class simply and one can implement their own trading logic
6. **Execution**: It provides linkage with certain exchange API, which can execute the strategy based on the ledger information

**2. Strategy Intro**

Utilize several technique signals: short-term moving average, long-term moving average, double deep, Relative Strength Index, Average True Range

Raw data, process the time series->get indicator (transformed time series data, including moving average, return or volatility indexes ect.)

Combine indicators, apply mathematical operations, get signal

From signals to strategy:

Single signal strategy

Composite signal strategy

Machine learning strategy: take all the signals and apply dynamically to optimize return.

**3. Brief Intro of auto trading strategy based on reinforcement learning (Deep Q Network)**

The success of Google DeepMinds victory over Lee Sedol in the board game Go brought the attention to Reinforcement learning. The technique DeepMind used was a combination of deep neural networks and reinforcement learning. The reason for combining a neural net with reinforcement learning is that a neural net will be able to handle a large amount of possible states. In plain reinforcement learning you often use a lookup table, and as long as the amount of possible states are finite and not too large this is fine. But when the number of possible states grows or continuous inputs are used then something that can handle a large state space is needed.

So by applying the reinforcement to trading signales, basically we are trying to create a simple self learning quant (or algorithmic trader).

* State S, this is a representation of the current world as the algorithm sees it
* State S’, a new state one time step later than S.
* Action A, one of the possible actions than can be taken at time step S.
* Q, a function that approximates the reward for action A at time step S’. Can be written as Q(s,a). In our case Q is a neural network.
* Reward R, the actual reward at state S’ given action A.

We also have a Q function that should learn to approximate the reward. In a simple world we could just let Q be a table of all possible states and then find a way to explore all possible states, actions and rewards, save these to the table and then look up the best action for a given state when needed. In a more complex world we need a way to generalize our knowledge and to be able to handle a very large number of different states.

The self learning comes from a concept of looping through a number of different states and actions many times, and each time update the Q function a little bit. So in each loop the Q function will know a little bit more about the world around it and should be able to approximate the real reward a little bit better for each possible action. Also, one very important thing in the learning process is to add a bit of randomness in order to explore as much as possible of the world. In our case we do this by adding a chance of selecting a random action instead of the action suggested by the Q function.

Our neural net is a simple three layer neural network with just 4 neurons in each layer that should be sufficient to learn what a straight line looks like.

**4.** **Results and Improvements**