Trading via Reinforcement Learning

Alban Sarfati, Kenz Bensmaine, and Salah Azekour

CentraleSupélec

Abstract. Trading is a problem in which an agent continually takes a position on a given security by buying or selling it. Traders are compensated for the risk of holding securities known as the inventory, which may decline in value between the purchase and the sale, the accumulation of these unfavourable positions is known as inventory risk. The purpose of this project is to use reinforcement learning in order to design competitive trading agents for financial markets using mid-frequency historical equities data. Reinforcement learning is often considered one of the most promising approaches to algorithmic trading because it most closely models the task an investor faces. However, there are many obstacles given the noisy nature of financial data, which makes it even more difficult to learn a value function based on delayed rewards.

Keywords: Trading Strategy · Dynamic programming · DQN.

1 Introduction

December 1998, in response to the technology boom sweeping the market, the US Securities and Exchange Commission (SEC) approved new rules that allowed electronic trading systems to register as full-fledged exchanges. Since then, algorithmic trading has become increasingly common as the need to process more data and act on shorter time scales makes the task almost impossible for humans.

In the context of Reinforcement Learning applied to trading, the agent observes the current financial market-state of the environment and uses this information to take a position from an action-space table which affects the next state of the environment and the reward. His goal is to learn to act in a way that will maximize its risk-adjusted return measure over time.

An optimal trader agent would adjust dynamically it's position on a stock in response to the current market conditions and its inventory level, this selection constitutes a control mechanism.

2 Data

We used mid-frequency historical data (1 hour tick) from yahoo finance, the data is requested over the last two years allowing to have 3527 instances per stock. Each instance is indexed by a timestamp and characterized by the open price, close price, highest price, lowest price and volume.

3 Environment

3.1 Action space

The trading agent acts hourly on events as they occur on the environment. An event is anything that constitutes an observable change in the states of the environment, as a change in financial market technical indicator or a change in the agent inventory. The agent can choose from three actions:

Long (2): Buy the stock by investing capital in it

Flat (1): Hold cash only

Short(0): Sell the stock

For the long and short action the agent is restricted to buy or sell the action for a volume of 100.

3.2 Reward function

The "natural" reward function for trading agents is the PnL which represents how much money is made or lost through exchanges with the market. At each time step t_i, the inventory of the agent, $Inv\left(t_i\right)$, is either:

100 for a long position

0 for a flat position

-100 for a short position

Moreover, the price move, $\Delta m\left(t_i\right)$, is the change in the stock price since the last time the agent compute an order. Assuming that the initial amount invested (cash) by the trader is 1000\$, the incremental PnL function $\Psi\left(t_i\right)$ is :

$$\Psi\left(t_0\right) = 1000$$

$$\Psi\left(t_i\right) = \left(1 + Inv\left(t_i\right) \Delta m\left(t_i\right)\right) * \Psi\left(t_{i-1}\right)$$

This basic formulation of PnL, captures the gain from speculation without involving trading fees.

One other PnL function is engineered to take into account this issue. The order cost, η , is set at 1\$ which is a good proxy of real-life brokerage fees. Therefore, this cost is multiplied by the inventory move factor, ζ, for example going from long to short will cost the agent 2\$.

$$\zeta\left(t_i\right) = \left|\Delta Inv\left(t_i\right)\right| / volume_{order}$$

Dampened PnL:

$$r_i = \Psi\left(t_i\right) - \eta * \zeta\left(t_i\right).$$

This new reward function is encouraging to reduce the total number of trades smoothing the variance of inventory.

3.3 State space

The environment is composed of two states, the agent-state and the financial market-state. These states are constructed as set of attributes that describe the agent and the financial market. At each time step t_i :

The agent state is composed of $Inv(t_i)$.

The market state is composed of technical indicators which are derived from the prices and volume of the specific stock. They represent statistical tools and are extensively use to make investment decisions by generating signals. We programmed ten of them to identify trends, regime switches, momentum, and potential reversal points in the stock market, such as:

Exponential moving averages (EMA): These are used to smooth out price data over a period of time and are commonly used to identify trends.

Relative strength index (RSI): This measures the strength of a security's recent price action to determine if it is overbought or oversold.

Stochastic oscillator: This compares a security's closing price to its price range over a period of time to determine if it is overbought or oversold.

Moving Average Convergence Divergence (MACD): This is a momentum indicator that shows the relationship between two moving averages of a security's price.

Three different representations of state could be considered, the agent-state, the market state and full-state (agent and market variables). However, only the full-state is considered as it is rich enough to contain all the meaningful information and sparse enough to enable quick learning, which is crucial to facilitating learning and improving the efficiency of RL algorithms.

Moreover, the action of the agent does not influence the market state, therefore this part of the environment evolves deterministically and the agent learns how to behave optimally—trade profitably—in that environment.

3.4 Environment

Given all the elements below, a class representing the specific environment is developed with two main methods:

*Reset(): Allows to constructs the initial attributes such as the features.

*Step(action): Allows to convert the action took by the agent into orders. Uses the method forward step and update the attributes according to the evolution of data. Calculates the reward accordingly to the action took, the current state and the next state. Check for success or failure: a success is achieved when the agent is able to correctly trade through the whole split data set, a failure is defined when the agent's PnL becomes negative. However, this is only checked for after a certain number of steps to avoid the high initial variance of this metric. This method allows the agent to explore the environment.

4 Agent

4.1 Policy

In our framework, the financial market is a great source of state complexity as it is in constant evolution, a multi-agent system and data can be unrepresentative (black swan). One solution is to find a function $Q_\theta(s, a)$ that approximates the Q-Value of any (s, a).

Neural networks with their universal approximation capabilities are a natural choice to accomplish this task. We use a deep neural network composed of Linear layers to estimate Q-Values, known as DQN.

In this trading context, the goal is to maximize the long-term performance, hence using delayed reward. The approximate Q-Value should be close to the reward the agent gets after playing action a in state s plus the future discounted value of playing optimally from then on.

$$Q_{target}\left(S_t, A_t\right) = R_{t+1} + \gamma \max_a Q_\theta\left(S_{t+1}, a\right)$$

S_t is the state at time-step t, A_t is the action taken at state S_t, R_{t+1} is the direct reward of action A_t, γ is the discount factor, and $\max_a Q_\theta\left(S_{t+1}, a\right)$ is the maximum delayed reward given the optimal action from the current policy Q_θ.

The loss to be minimized is the mean squared error between the estimated Q-Value, Q(s, a), and the target Q-Value. To select an action using this DQN, we pick the action with the largest predicted Q-Value, indeed we want to maximize the Q-Value of the agent.

4.2 Performance criteria

The agent's net PnL over time will be compared to the market PnL (represented by a long position only). In addition, the Sharpe ratio of the agent's strategy will be compared to that of the market. The Sharpe ratio is a financial metric that measures the risk-adjusted return of an investment. A higher ratio indicates that an investment has generated higher returns for its level of risk, while a lower ratio indicates that an investment has generated lower returns for its level of risk. It is calculated by dividing the standard deviation of the investment's returns from the return of the investment.

4.3 Agent

For DQN and for fixed/random algorithm, a respective agent is created taking as input the learning and testing environments, the main methods are:

*GetAction(state): For non learning algorithm it will return the fixed or random action. Otherwise, it will select an action on the learning environment using the model based on exploration or exploitation. The agent starts with exploration only, in the beginning it could not have learned anything, and slowly but steadily decreases the exploration rate epsilon until it reach a minimum level.

*PlayOneStep(state): Method to play with the action generated from GetAction(state). Call the method Step(action) of the environment which returns four elements: the next state, the resulting reward, a Boolean indicating whether the episode ended at that point and some information. These data will be append into a memory object such as deque.

*Replay(): For learning algorithms, method to retrain the model based on batches of memorized experiences. The agent replays a number of experiences (state-action combinations) to update the policy function Q regularly. The discount factor γ would be applied to the scalar sum of future rewards. It will changes how much a single measure of future reward will contribute to an update step.

*Learn(): Train the agent for each step of the learning environment and for n episodes. At each step, we first call GetAction() which compute the epsilon value for the ϵ-greedy policy and use the ϵ-greedy policy to pick an action. Then we call PlayOneStep(state) which execute it and record the experience in the replay buffer. If the episode is done we evaluate the agent on the testing environment, using exploitation only, and save some interesting data (PnL, Sharpe ratio, total number of position, etc). If we are past the batch size we call Replay() to train the model on one batch sampled from the replay buffer. Then we exit the second loop to move one episode forward.

5 Results

We evaluated each agent on Apple stock, the intraday data range from 2021-04 to 2023-01 with a training environment representing the first 70% of the time series. Hence, the testing environment range from to 2022-08 to 2023-01.

5.1 Fixed and random agent

A fixed agent is represented by taking only one kind of action i.e. either being long, neutral or short all the time. A random agent takes action randomly according to randint() and seed 42. As these agents are not learners, we compute directly the metrics on the testing environment.

Type	pnl	sharpe	trades
long	-1993	nan	1
neutral	0	nan	0
short	1991	0.9	1
random	-982.8	nan	775

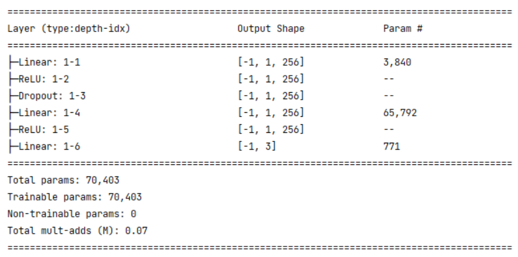
Table 1. Agent's testing characteristic.

We can observe that the Long Agent has a negative PnL of -1991$, hence Apple had a downward trend from 2022-08 to 2023-01. Symmetrically, the Short

Agent has a positive PnL of 1991$, by selling Apple stock with a volume of 100 on the window made a subsequent profit given the starting stake (1000$). On the other hand, the Neutral agent has a neutral PnL as it only holds cash. Finally, the Random Agent has a negative PnL of -983$ and has a total of 775 trades i.e. he has changed his position often.

5.2 DQN agent

The DQN agent is based on a feed forward neural network whose architecture is below. The batch size has been set to 512, the learning rate to 0.01, the dropout rate to 0.2 and we used Adam optimizer to minimize the MSE loss.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Linear: 1-1                            [-1, 1, 256]              3,840
├─ReLU: 1-2                              [-1, 1, 256]              --
├─Dropout: 1-3                           [-1, 1, 256]              --
├─Linear: 1-4                            [-1, 1, 256]              65,792
├─ReLU: 1-5                              [-1, 1, 256]              --
├─Linear: 1-6                            [-1, 3]                   771
==========================================================================================
Total params: 70,403
Trainable params: 70,403
Non-trainable params: 0
Total mult-adds (M): 0.07
==========================================================================================
```

Fig. 1. DQN architecture.

As for the replay parameters, we choose $\epsilon = 0.99$, $\epsilon_{decay} = 0.99$, $\epsilon_{min} = 0.01$, $\gamma = 0.97$, $n_{episode} = 1000$ and $memory = deque(maxlen = 10e4)$.

All these parameters have not been tuned (no validation environment) but were set according to the literature.

From Table 3, we can observe that the training performance (accumulated PnL) has an upward trend and shows a high variance, which is due to the exploration that is going on in addition to the exploitation of the currently optimal policy. The upward trend is a good news as it shows that the algorithms worked, our DQN agent is making some profit, but it also shows some overfitting. In comparison, the validation performance has a much lower variance because it only relies on the exploitation of the currently optimal policy but is characterized by catastrophic forgetting. Indeed, the accumulated PnL from one episode to another can go from 0$ to 4000$.

Table 4 shows that the 5 best episodes, i.e. those with the highest cumulative PnL on testing set, they were run after 900 episodes. The DQN agent on episode 925 has been making a profit of 4933$ with a total of 125 trades and a SHARP ratio of 1. These metrics clearly support the fact that our DQN agent is a good trader.

Moreover if we look at the plots of Table 5, which represents our DQN agent at the best episode i.e. 925 on the testing set, we see that 2/3 of its actions are long and 1/3 are short. At no time did the agent have a flat position, perhaps

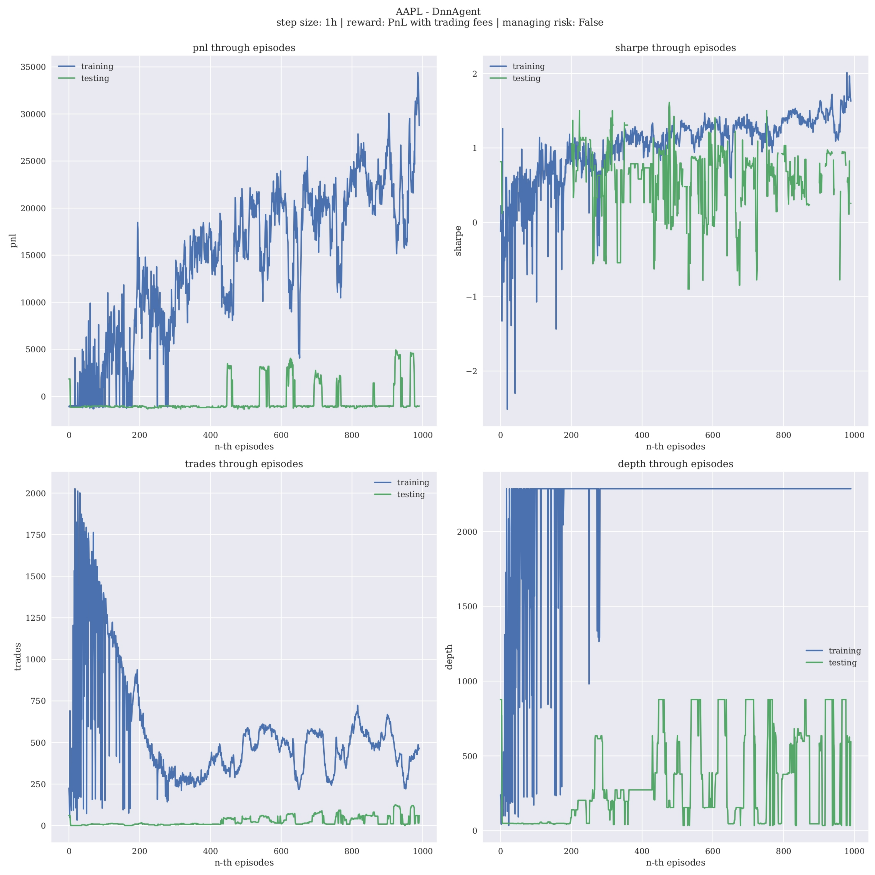
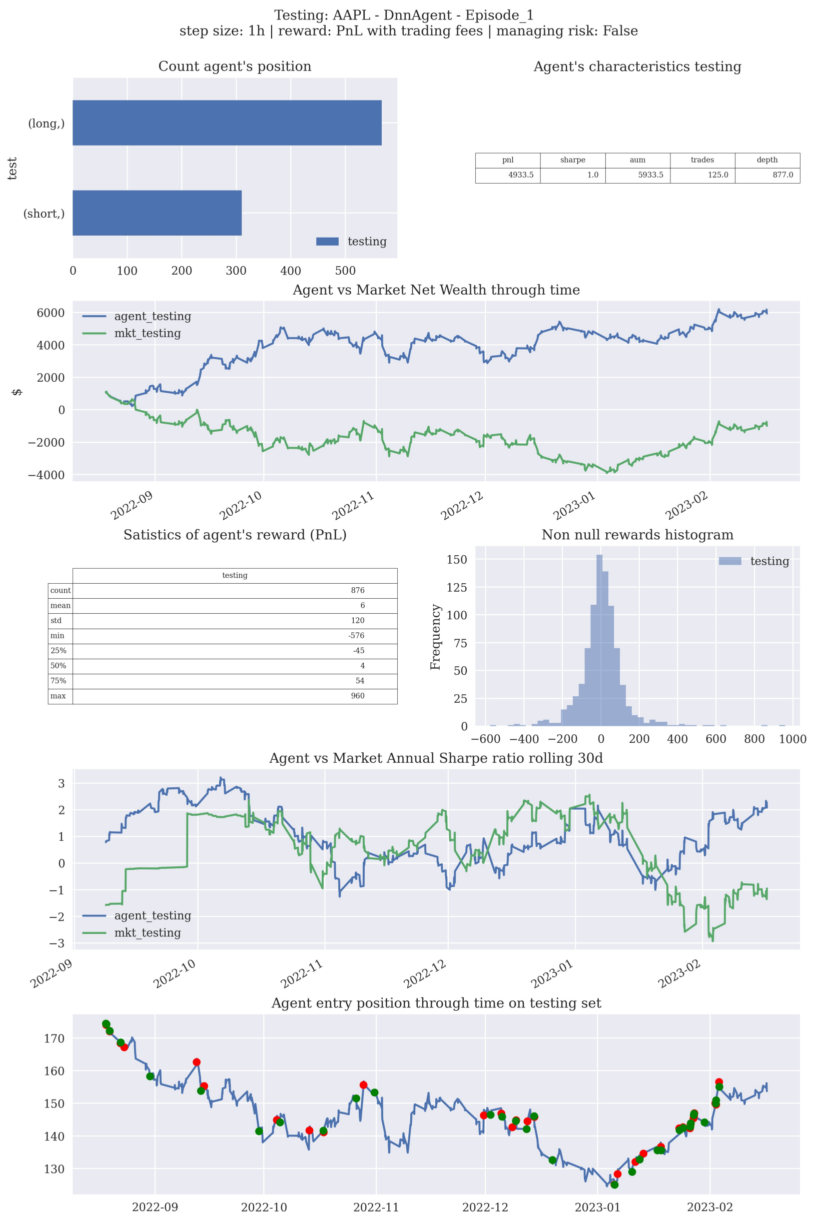


Fig. 2. Training and testing metrics according to episodes.

Episode	pnl	sharpe	trades
925	4933	1.0	125
927	4804	1.0	121
967	4669	1.0	105
971	4571	0.9	121
973	4559	0.9	121

Table 2. Agent's testing characteristic on top 5 episodes.

because we are maximising the PnL function and it is not profitable to have a flat position. In addition, we observe that our agent unambiguously outperforms the market. The hourly rewards have a mean of 6 with a standard deviation of 120, they represent a bell curve with a positive skew and left fat tail. Last but not least, one can show the entry positions (green for a long position and red for a short position) of our agent, red dots are well positioned at some high pike just before a bear market.



pnl	sharpe	aum	trades	depth
4933.5	1.0	5933.5	125.0	877.0

	testing
count	876
mean	6
std	120
min	-576
25%	-45
50%	4
75%	54
max	960

Fig. 3. Agent's testing metrics for best episode.