REINFORCEMENT LEARNING IN FINANCE
Quant Club, IIT Kharagpur

Pradipto Mondal

1. What is Reinforcement Learning?

Reinforcement learning is an area of machine learning concerned with how intelligent **agents** ought to take **actions** in an **environment** to maximize cumulative **reward**.

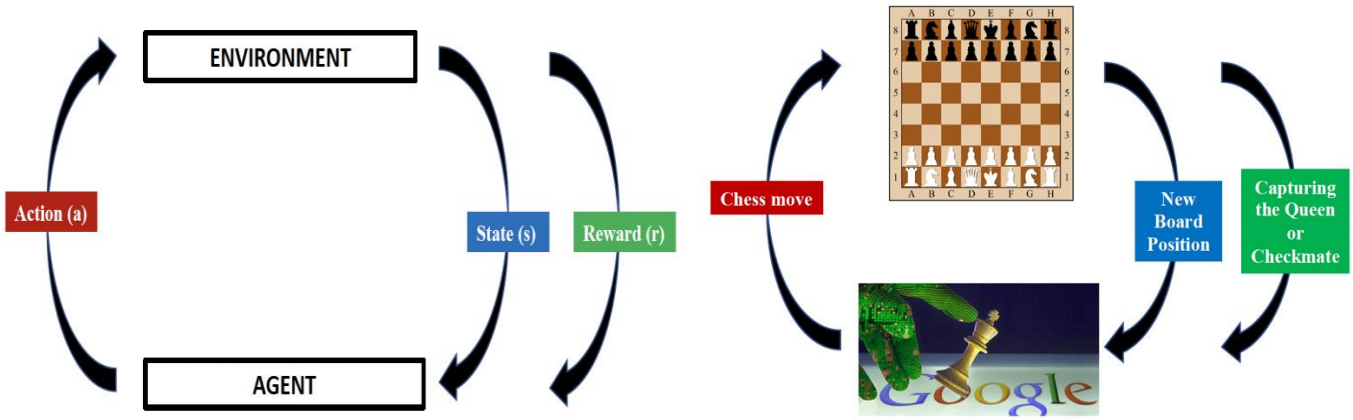
The above definition brings a handful of terms to the table at once. Let us understand them and how they form the bigger picture in detail.

The two main entities involved are *environment* and *agent*. The *agent* performs an *action* 'a' on the *environment* which results in a change in its *state*. The *environment* returns this *state* to the agent in addition to another entity called the *reward*. This *reward* acts like a feedback loop that tells the *agent* whether the *action* it took is simply put, a good one.

2. Chessbots as an example

To understand the underlying principle of reinforcement learning let us go through one of the applications of reinforcement learning in the real world: a chess bot.
You must have heard about different powerful Chessbots out there like Leela, Stockfish, AlphaGo. They have famously even defeated the best human players in the world. So how do they do it? Let's understand it roughly.

These bots are fueled via reinforcement learning. The environment here is the Chessboard, this is where all the action takes place. The agent i.e., the chess bot can perform an action on the environment through a chess move. This action then returns its new state, the new board position to the agent. Along with the new state, it returns a reward that gives the agent an idea of how good its action was or how favourable the current state of the board is for the chess bot. This reward can be just a real number that scores the position, or a flag for checkmate or queen capture. After the opponent plays its move, the agent observes the new state and plans its next action.



3. Markov's Decision Process

All the mathematics behind reinforcement learning is based on Markov decision processes. The underlying assumption behind such a process is that the successor state depends only and only on the current state.
We define a few important sets as follows:

- Set of all possible states: $\mathcal{S}$
- Set of all possible actions: $\mathcal{A}$
- Set of all possible rewards: $\mathbb{R}$
- Policy $\pi \in \Pi$

POLICY: Policy π is a function that yields an action 'a' given the current state 's' as input:

$$\pi(s) = a$$

A policy can also be thought of as a stochastic function, which emits the probability of taking an action 'a' given an action 's'.

$$\pi(s, a) = \mathbb{P}(action = a \,|state = s)$$

VALUE FUNCTION: This function helps to calculate the 'value' of a state 's'. This is synonymous with quantifying how much the current state is favourable to the user. It is calculated as follows:

$$V_\pi(s) = \mathbb{E}\left(\sum_t \gamma^t r_t | s_0 = s\right)$$

In the above expression, γ is called the discount rate, a number between 0 and 1 whereas r_t is the reward at a time instant 't'. Thus, this expression is the expected value of the cumulative value of future rewards given the current state is s_0. Due to the discount rate being raised to the power of 't', the value function tends to favour immediate rewards, and this behaviour can be controlled externally by varying the discount reward.

4. Q – LEARNING

The value function simply gives the expected reward for a current state. Now, we introduce an entity called the Q-value which takes into account a state and an action and gives us the value of the maximum reward that can be achieved if the action 'a' is performed at a state 's'.

$$Q(s_t, a_t) = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \gamma^4 R_{t+4} + \cdots$$

$$\Rightarrow Q(s_t, a_t) = R_t + \gamma(R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots)$$

$$\boxed{\Rightarrow Q(s_t, a_t) = R_t + \gamma \max_a Q(s_{t+1}, a)}$$

The last equation we reached, is the famous Bellman's equation which is used extensively in reinforcement learning systems.

5. DIRECT IMPLICATIONS IN FINANCE

Now, after toying around with the foundations for a while we will finally dive into the meat of this report. Let us try to understand roughly how Q-Learning might be used in a simple trading bot. This is a very rough example just to demonstrate the overall functioning of Bellman's function.

Assume, we have purchased an equity share for Rs. 100. And we have information on the stock prices for the next five days: 120, 130, 110, 125, 150. Here is a table representing the profit we can make if we hold/sell the share on any particular day.

Action/State	120	130	110	125	150
Hold	0	0	0	0	0
Sell	20	30	10	25	50

Now, if this information is fed into a trading bot, on Day 1, it sees that it earns more on selling the stock than holding the stock, so without further ado, the stock is sold on the first day. Hence, it is deprived of the maximum profit on Day 5 i.e., Rs. 50. In some way, holding the stock on the first day should have more value than selling it. Let's see if Q-learning can solve this issue.

Through Bellman's equation, we will attempt to update the 'value' of holding the stock on the days so that the trading bot can complete the transaction in the most profitable manner.

Assume discount rate $\gamma = 0.9$. Beginning from Day 4, the Q value for holding the stock on that day can be calculated as follows:

Original reward + 0.9 × Maximum reward of the successor state = 0 + 0.9×50 = 45

Going backwards with a similar calculation we can evaluate the value for holding the stock for all the other days.

$$\Rightarrow Q(s_t, a_t) = R_t + \gamma \max_a Q(s_{t+1}, a)$$

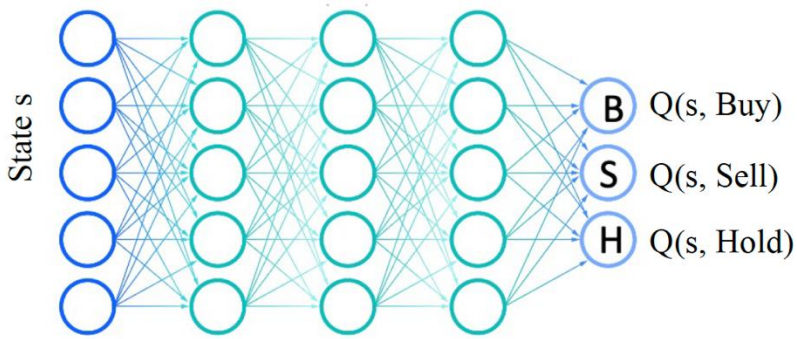
One share purchased at 100

Action/State	120	130	110	125	150
Hold	32.805	36.45	40.5	45	0
Sell	20	30	10	25	50

Now, we can clearly see holding the stock on Day 1 is more valuable than selling it, and likewise for the following days so that it can enjoy maximum profit on Day 5.

6. DEEP Q-LEARNING & EXPERIENCE REPLAY

In the above example, one must have noticed one fundamental issue. It is not possible to know the future prices of stock accurately, nor can one evaluate Q-values for every state-action pair because the state-space is simply too large to deal with in the case of stock prices.
This is where deep learning comes into play.
We use a deep neural network to estimate the Q-values for a given state-action pair. The input to this layer is a state that is not restricted only to the stock price, it can have other components too which might include indicators such as RSI, moving averages, market sentiment scores, portfolio value, remaining funds etc.



Now we will discuss in detail how does experience replay work towards optimizing this model. We have at our disposal historical data of the prices of a particular stock.

- First, the weights of the deep neural network are initialized randomly.
- The first state s is passed into the network, and the Q values for the three actions (buy, sell & hold) is calculated. The action a with the highest Q-value is chosen, and a new state s' is achieved. The reward (~profit) is also calculated corresponding to this action.
- This whole transaction can be identified through four entities $(s, a,\ s', r)$. This tuple is known as an 'experience' and stored in a memory called the replay memory.
  The replay memory has a memory limit beyond which it begins deleting old experiences.
- A random experience is picked up from the replay memory say $e = (s, a, s', r)$.
- The state s is then again passed through the neural network, and now the Q value is obtained corresponding to the action a.
- The expected Q value for this state-action pair (s, a) is however different. The expected Q-value is calculated by using Bellman's equation.

$$Q^*(s,a) = r + \max_a Q(s',a)$$

  To calculate the second part of RHS, max $[Q(s', a)$ needs to be calculated. This is found out by passing the state s' again through the network and retrieving the maximum Q value for all possible actions a.
- Finally, the loss for the neural network is calculated as LOSS = (Q* - Q)
- This loss is then used to update the weights of the neural network.

With sufficient number of iterations and data points, we eventually can create a model which can almost accurately estimate Q values for any state-action pair.

Computers, Materials & Continua
DOI:10.32604/cmc.2022.017069
Article

Tech Science Press

Stock Market Trading Based on Market Sentiments and Reinforcement Learning

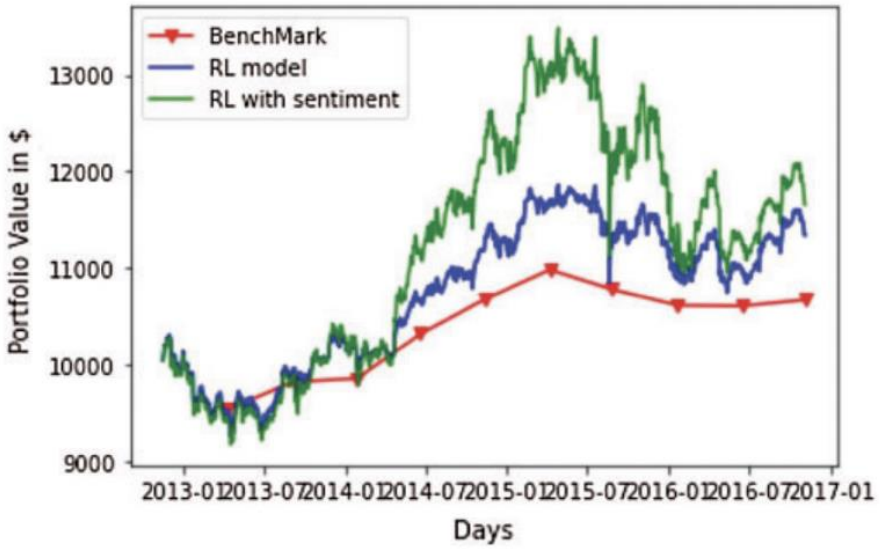
K. M. Ameen Suhail[1], Syam Sankar[1], Ashok S. Kumar[2], Tsafack Nestor[3], Naglaa F. Soliman[4,*], Abeer D. Algarni[4], Walid El-Shafai[5] and Fathi E. Abd El-Samie[4,5]

[1]Department of Computer Science & Engineering, NSS College of Engineering, Palakkad, 678008, Kerala, India
[2]Department of Electronics and Communication Engineering, NSS College of Engineering, Palakkad, 678008, Kerala, India
[3]Unité de Recherche de Matière Condensée, D'Electronique et de Traitement du Signal (URAMACETS), Department of Physics, University of Dschang, P. O. Box 67, Dschang, Cameroon
[4]Department of Information Technology, College of Computer and Information Sciences, Princess Nourah Bint Abdulrahman University, Riyadh, 84428, Saudi Arabia
[5]Department Electronics and Electrical Communications, Faculty of Electronic Engineering, Menoufia University, Menouf, 32952, Egypt
*Corresponding Author: Naglaa F. Soliman. Email: nfsoliman@pnu.edu.sa
Received: 20 January 2021; Accepted: 16 May 2021

Abstract: Stock market is a place, where shares of different companies are traded. It is a collection of buyers' and sellers' stocks. In this digital era, analysis and prediction in the stock market have gained an essential role in shaping today's economy. Stock market analysis can be either fundamental or technical. Technical analysis can be performed either with technical indicators or through machine learning techniques. In this paper, we report a system that uses a Reinforcement Learning (RL) network and market sentiments to make decisions about stock market trading. The system uses sentiment analysis on daily market news to spot trends in stock prices. The sentiment analysis module generates a unified score as a measure of the daily news about sentiments. This score is then fed into the RL module as one of its inputs. The RL section gives decisions in the form of three actions: buy, sell, or hold. The objective is to maximize long-term future profits. We have used stock data of Apple from 2006 to 2016 to interpret how sentiments affect trading. The stock price of any company rises, when significant positive news become available in the public domain. Our results reveal the influence of market sentiments on forecasting of stock prices.

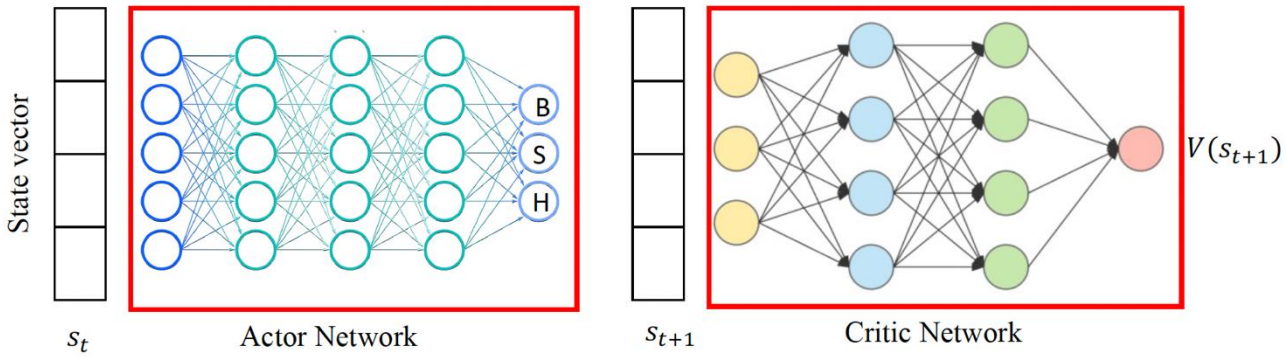
The above work by Suhail et. al. implements Deep Q-learning and experience replay to create a trading bot that performs really well on real-world data. The input state vector in this implementation contains 5 quantities: Open, 5-day SMA, Stocks held, Cash held, and sentiment score. The market sentiment score is calculated by processing relevant market news through a robust NLTK library called VADER (Valence Aware Dictionary and Sentiment Reasoner).

The evaluation results are represented in the following graph:



7. ACTOR-CRITIC NETWORKS

This is a slightly different kind of reinforcement learning model, which estimated the Value function for the successor state. It implements two networks: the actor and the critic network. The actor-network decides which action to take, whereas the critic-network on the other hand estimates the value function for the successor state thereby 'criticising' or keeping the actor-network in check.



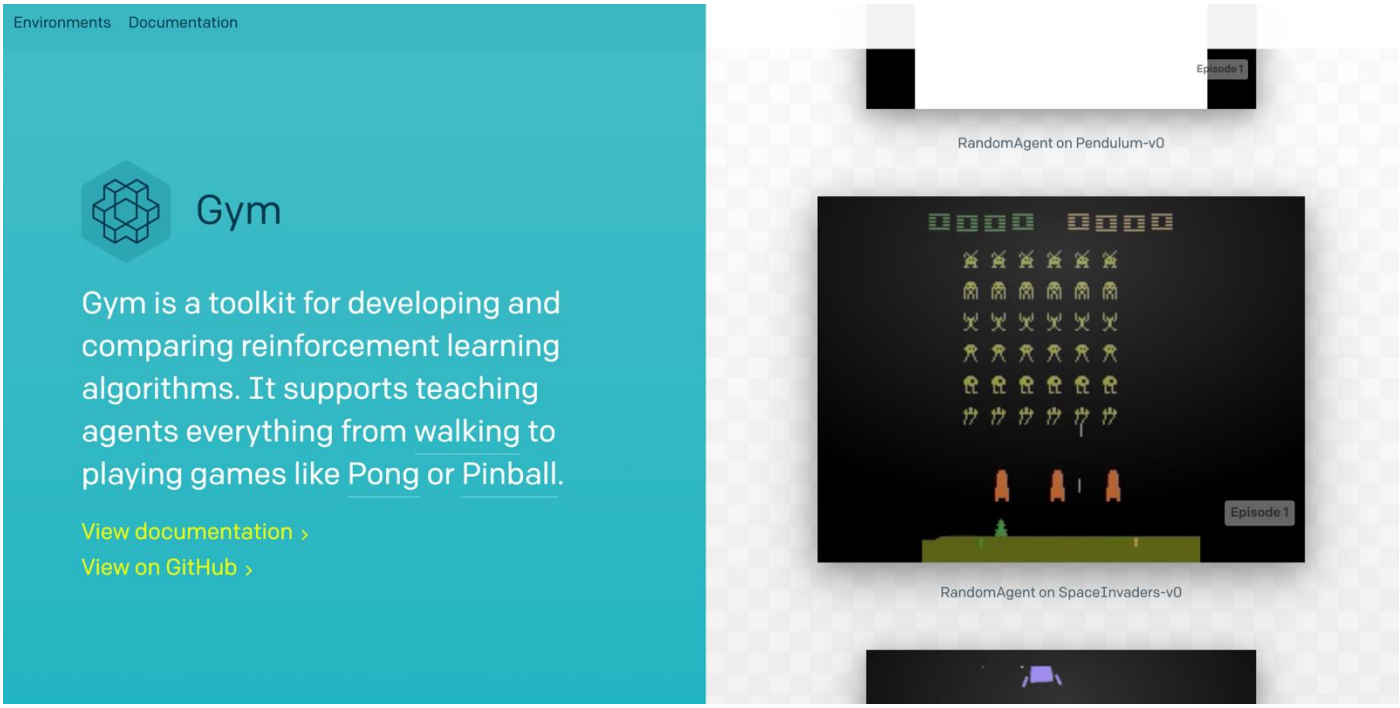
The 'advantage' value is calculated as follows:

$$\boldsymbol{\delta = R_t + \gamma V(s_{t+1}) - V(s_t)}$$

The above expression can be expressed in terms of Q-value as $Q(s_t, a_t) - V(s_{t+1})$
The losses for the above two networks is calculated as follows:

$$\text{Actor Loss} = \delta^2$$
$$\text{Critic Loss} = \delta \cdot \ln \pi(a_t|s_t)$$

Such actor-critic networks can be easily implemented through a dataset using the famous open-source library Gym. This implements many other useful models based on reinforcement learning.



8. IMPLEMENTATION OF A2C NETWORK ON MARKET DATA

Using the gym open-source library, the A2C network can be implemented on real-world stock data, in this case, the $AAPL stock is used for the period of 1 year: 2019-2020.

Setting up indicators (SMA, RSI, OBV):

```
[ ] from gym_anytrading.envs import StocksEnv
    from finta import TA

[ ] df1 = yf.download('AAPL', start = '2019-01-01', end = '2020-01-01')
    df1['SMA'] = TA.SMA(df, 12)
    df1['RSI'] = TA.RSI(df)
    df1['OBV'] = TA.OBV(df)
    df1.fillna(0, inplace=True)

    [**********100%**********]  1 of 1 completed

[ ] df1.head()
```

Date	Open	High	Low	Close	Adj Close	Volume	SMA	RSI	OBV
2019-01-02	38.722500	39.712502	38.557499	39.480000	38.326290	148158800	0.0	0.000000	0.0
2019-01-03	35.994999	36.430000	35.500000	35.547501	34.508705	365248800	0.0	0.000000	-365248800.0
2019-01-04	36.132500	37.137501	35.950001	37.064999	35.981857	234428400	0.0	29.357079	-130820400.0
2019-01-07	37.174999	37.207500	36.474998	36.982498	35.901772	219111200	0.0	28.861016	-349931600.0
2019-01-08	37.389999	37.955002	37.130001	37.687500	36.586174	164101200	0.0	38.434700	-185830400.0

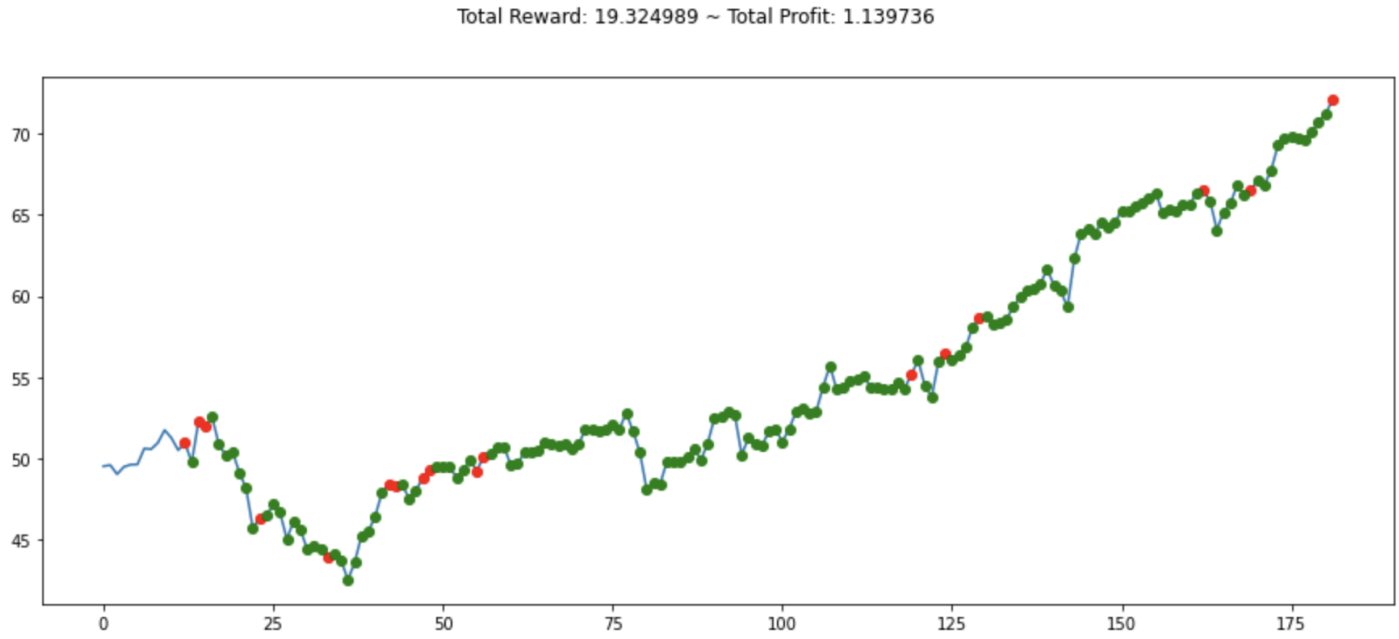
Setting up the trading environment and adding the indicators to it:

```
[ ] def add_signals(env):
        start = env.frame_bound[0] - env.window_size
        end = env.frame_bound[1]
        prices = env.df.loc[:, 'Low'].to_numpy()[start:end]
        signal_features = env.df.loc[:, ['Low', 'Volume','SMA', 'RSI', 'OBV']].to_numpy()[start:end]
        return prices, signal_features

[ ] class MyCustomEnv(StocksEnv):
        _process_data = add_signals

    env2 = MyCustomEnv(df=df1, window_size=12, frame_bound=(12,50))
```

Training phase:

```
[ ] env_maker = lambda: env2
    env = DummyVecEnv([env_maker])

[ ] model = A2C('MlpLstmPolicy', env, verbose=1)
    model.learn(total_timesteps=1000000)
```

Final results:



As we can see, this trained bot earned a net profit of about 13.97% over the period of one year. This performance can be further improved through Deep Q-Learning, using better indicators and integrating market sentiment scores with the model.

Link to notebook:
https://colab.research.google.com/drive/1j1h3q_oxNax_f-2OLdH8H9JAn0hyEjD-?usp=sharing

9. CONCLUSION

Reinforcement Learning is just an artificial way of imitating how living organisms learn to do various tasks: by rewarding good actions and penalizing bad ones. The field of finance has made great use of this emerging concept in portfolio optimization, optimized trade execution, market-making etc. These new RL solutions have proved to be better than many pre-existing solutions to the above-mentioned problem areas. We can expect reinforcement learning to take the world of artificial intelligence by a storm soon and perform tasks like or even better than the most skilled human beings.