JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

High Frequency Trading System using Deep-Q Learning Algorithm

Aditya B
Department of Computer Science Engineering
PES University, South Campus, India
aditya.bhimesha456@gmail.com

H Prajwal Navada
Department of Computer Science Engineering
PES University, South Campus, India
navadaprajwal@gmail.com

Shreyan
Department of Computer Science Engineering
PES University, South Campus, India
shrnshty@gmail.com

Nandha Kumar N
Department of Computer Science Engineering
PES University, South Campus, India
nandakumarngowda@gmail.com

***Abstract*—This project makes use of market dynamics available on the stock market and creates a backtest indicating the market trends for that particular stock in the recent past and the near future, based upon the stocks available. Based on this backtest, one can make a wise decision as to invest into that particular stock by buying it, or to hold on to it, or perhaps sell it if you seem to be in possession of it.**

***Keywords*—DQN, Reinforcement Learning, Backtest, Stocks, Training.**

I. INTRODUCTION

Financial markets are highly complex stochastic systems, Thus it is much more beneficial for us to predict prices on a short term basis rather than predict what is going to happen a few hours, days or even months down the line. This method of trading is known as High Frequency Trading in general.

Through extensive research we have found that the reinforcement learning methods apply best on these markets as they tend to reflect human behaviour through rewards which translate to market profit and loss for humans.

Through our research, we can conclude that the deep Q learning algorithm is the most efficient in this case. In other reinforcement learning algorithms, approximating the action-value function often leads to instability.

As we also see from Fig 1, Deep Q learning is also comparatively the better performer when seen against a normal Buy and Hold strategy(BH) and Recurrent Reinforcement Learning(RRL). Thus, we will be using Deep Q learning as it focuses on the benefit of the Q value rather than modelling the market.

Q learning can also quickly adapt to recent market conditions thus making it less prone to market structure changes. Q learning focuses on long term rewards more, as trading can be a probability game, we need to see consistency in results rather than short wins or lucky positions.

	HSI			S&P500		
	BH	DQT	RRL	BH	DQT	RRL
Accumulated Return(%)	154	350	174	169	214	141
Sharp Ratio	0.28	0.59	0.89	0.34	0.45	1.23
Maximum Drawdown(%)	65	42	55	57	31	43

Fig 1: Comparison between the performance of suitable algorithms

Our main objective here is to:

- Build a fast and sustainable model.
- Implement a risk management strategy for the model.
- Build an analytics tool to help people judge the results through a long term view.
- Compare and test results to real life scenarios.

At any given time (episode) in a Deep-Q network, an agent observes its current state, then selects and performs an action (buy/sell/hold), observes a subsequent state, receives some reward signal and lastly adjusts it's parameters based on the gradient of the loss computed.

II. DATASET

We have found abundant data on financial platforms such as yahoo finance and other dedicated data providers. We can also extract data from trading terminals provided they allow us to write scripts on their platform, thus capturing very recent data and saving it to a file.

Our data mainly concerns the **Open**, **High**, **Low**, **Close** of the concerned timeframe we are currently trading the system on. Thus we would like to capture the closing prices of the candle on multiple timeframes.

III. PROPOSED METHOD

This work uses a Model-free Reinforcement Learning technique called Deep Q-Learning. At any given time, an agent observes its current state, selects and performs a suitable action and observes a subsequent state, and adjusts its parameters based on the reward signal.

Advantages of this approach are as follows:

- Reinforcement learning **won't require large labeled datasets**. This is a massive advantage because as the amount of data in the world grows it becomes increasingly costly to label it for all required applications.
- Reinforcement learning is **Adaptable**, unlike supervised learning algorithms, it doesn't require retraining again and again as it adapts to new environments automatically on the fly.

Existing systems make use of techniques such as supervised learning which may be suitable in some cases, although it may have certain drawbacks. The computation time is vast in the case of supervised learning which may cause a few uncertainties. The efficiency achieved by this algorithm is also below par which is not good. Pre-processing of data is also a huge challenge in this approach and overfitting of the algorithm can be done by anyone which is a concern. All these drawbacks led us to divert into another approach.

Upon reflecting on the drawbacks of the existing system, we can conclude that deep Q reinforcement learning is more efficient for our purpose. Reinforcement learning does not need huge datasets. This is advantageous because it becomes increasingly expensive to label the data as the amount of data increases.

Goal-oriented, Reinforcement learning can be applied for a certain series of actions but supervised learning is generally applied in an input-output way. Reinforcement learning can be utilized for assignments with goals, for example, a calculation amplifying profit from venture on ads spent.

Reinforcement learning is highly versatile, on the grounds that it is not like supervised/unsupervised learning techniques, it will not need retraining more than once as it adjusts to new conditions naturally as the market changes.

The logical data flow of the system will include us making a trading environment and based upon the indicator data, one can send the data for training or testing with the help of the DQN agent. This DQN agent would use a classic reinforcement learning method (reward/punishment). It would reward if the decision taken turns out to provide a positive outcome and it would punish it otherwise. This is explained more clearly in Fig 2.

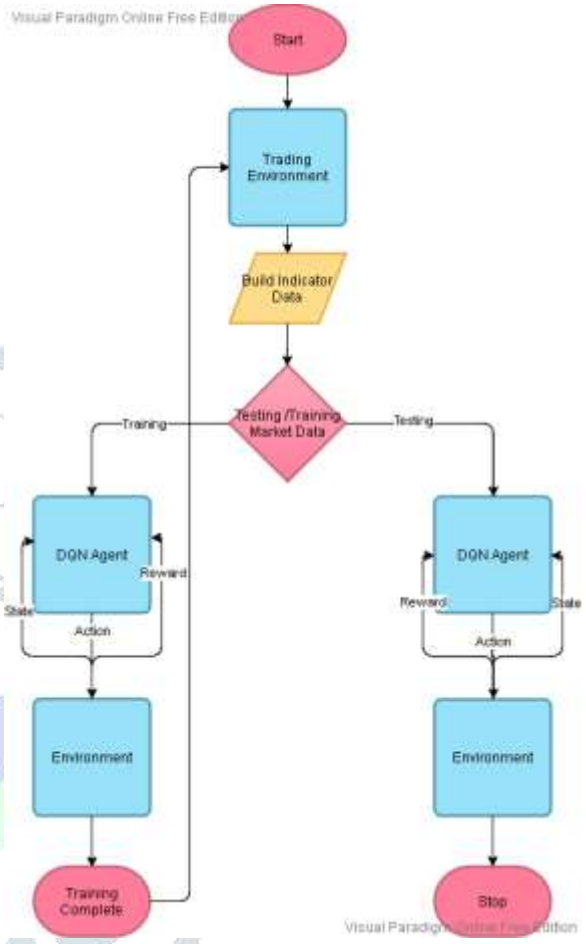

Fig 2: Logical data flow of the system

Our system will also include a dashboard wherein the user can have a quick glance about the stock that he/she is trying to backtest. Based upon this backtest statistics, he/she can deposit/withdraw more money. One can use this information later to make a wise decision as to invest or not. This can be clearly interpreted from Fig 3.

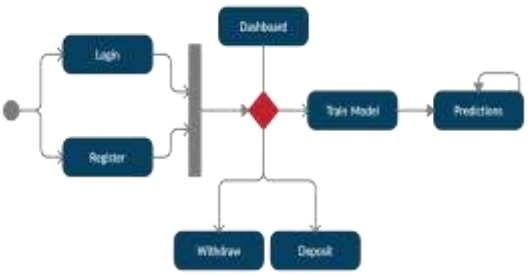

Fig 3: State Diagram of the system

The swimlane diagram below (Fig 4) indicates the manner in which the model is trained. i.e, the market data is collected and training is done on that. This trained data can be later used for a backtest.

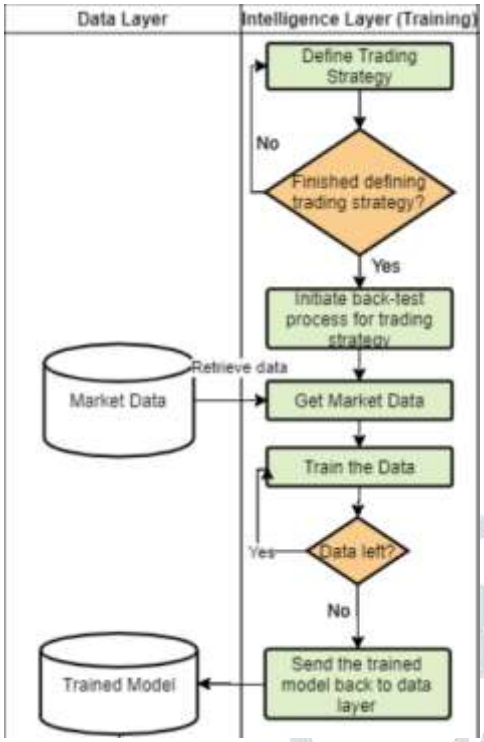

Fig 4: Training the model

The deployment of the system can be done with the help of AWS S3 storage. The model can also be trained using AWS SageMaker. This is explained in detail here (Fig 5).

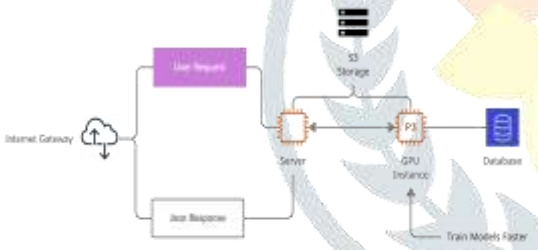

Fig 5: Deployment Diagram

IV. IMPLEMENTATION

We prepare a DQN Agent for our system. The DQN (Deep Q-Network) algorithm was developed by DeepMind in 2015. It was able to solve a wide range of Atari games (some to superhuman level) by combining reinforcement learning and deep neural networks at scale. The algorithm was developed by enhancing a classic RL algorithm called Q-Learning with deep neural networks and a technique called *experience replay*. [1]

Q-Learning

It is a model-free off-policy RL method in which the agent aims to obtain the optimal state-action-value function Q(s,a) by interacting with the environment. It maintains a state-action table Q[S, A] called a Q-table containing Q-values for every state-action pair.

At the start, Q-values initialize to zeros. Q-learning updates the Q-values using the Temporal Difference method.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

where a is the learning rate and (gamma) ranging from 0 to 1. Q(s,a) is the actual Q-value for the state-action pair (s,a). The target Q-value for state-action pair (s,a) is R_{t+1} + (gamma)$\max_a Q(S_{t+1},a)$ i.e. immediate reward plus discounted Q-value of the next state. The table converges using iterative updates for each state-action pair. To efficiently converge the Q-table - greedy approach is used.

The greedy approach: At the starting of the training, Q-values of the Q-table initialize to zeros. It means all actions for a state have the same chance to be selected. So to converge the Q-table using iterative updates, exploration–exploitation trade-off is used. The exploration updates the Q-value of the random state-action pair (s,a) of the Q-table by randomly selecting the action.

The exploitation selects greedy action (a_t) for the state (s_t) from the Q-table having maximum rewards. So, to converge Q-table from an initial condition (all Q-values of the Q-table initialize to zeros), at the initial level of iterative update exploration is needed and at a later time of the iterative updates, requires more exploitation. It can use probability , in which random action selected using a probability , and action is chosen from the Q-table using probability 1. At the beginning, the value of is one, and it reduces with time, and once the table converges, it becomes nearly zero. [2]

Experience Replay

To avoid computing the full expectation in the DQN loss, we can minimize it using stochastic gradient descent. If the loss is computed using just the last transition s,a,r,s′, this reduces to standard Q-Learning.

This DQN work introduced a technique called Experience Replay to make the network updates more stable. At each time step of data collection, the transitions are added to a circular buffer called the *replay buffer*. Then during training, instead of using just the latest transition to compute the loss and its gradient, we compute them using a mini-batch of transitions sampled from the replay buffer. This has two advantages: better data efficiency by reusing each transition in many updates, and better stability using uncorrelated transitions in a batch. [1]

V. RESULTS AND OUTCOME

Initially, we start the training procedure by running the train.py file by giving the stock that you want to train as a parameter. Then the training commences for the episode duration that you have specified. In the current scenario, we have given a period of approximately 250 days which is basically a year including only the working days. (Fig 6). The training takes place, step by step, in such a manner that whichever out of the three signals (Buy, Hold, Sell signals) is higher, that particular action is performed and a neat summary of the same is shown.

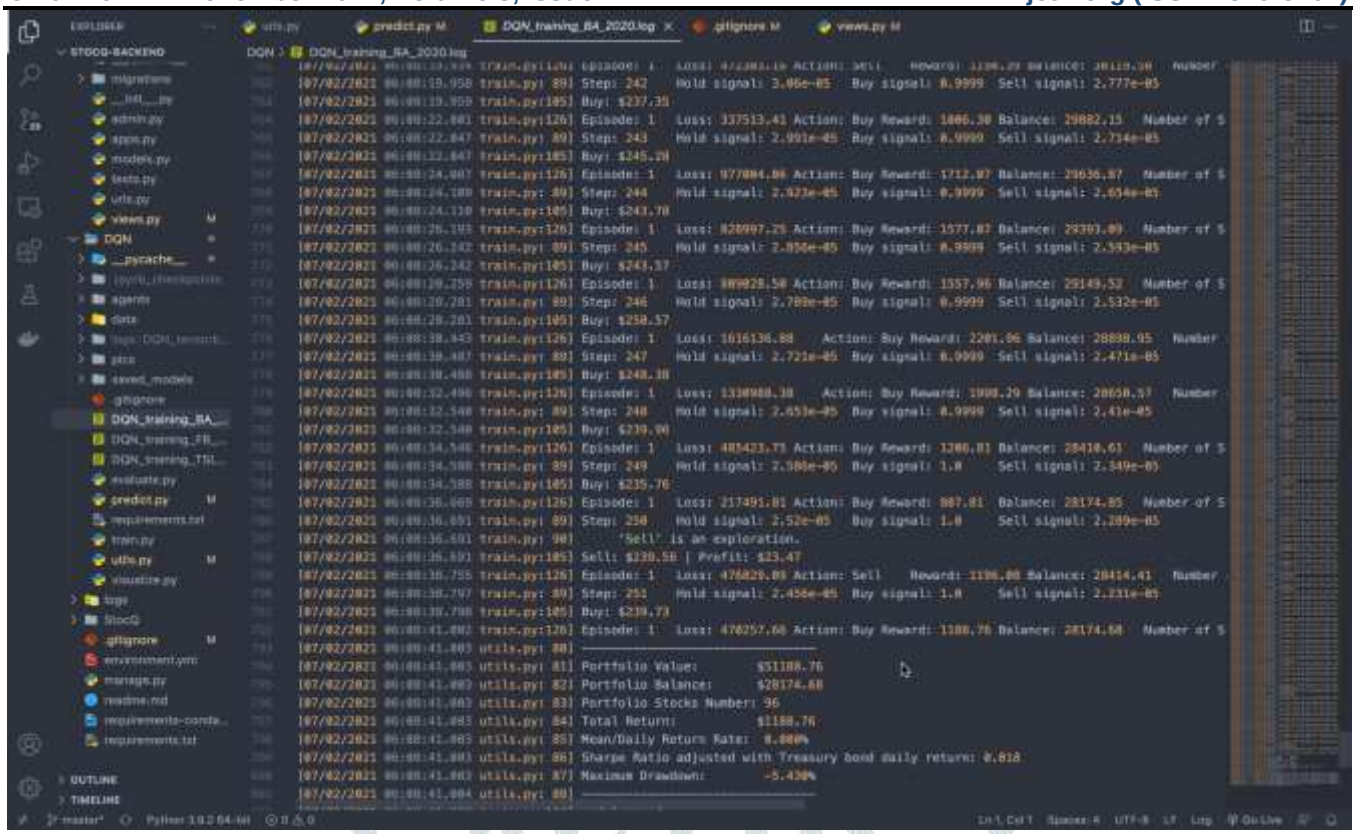

Fig 6: Training Log

Similarly, we can train for several other stocks upon the desire of the user. We can later start a backtest upon the trained stocks with the help of this page. The user can choose to deposit or withdraw money based on the backtest results and start a new backtest. Upon doing so, a neat backtest statistical dashboard is displayed before you containing information such as Buys to Sells ratios, open, high, low, close prices for that stock, return rates, portfolio value and profit data. (Fig 7 - 12)

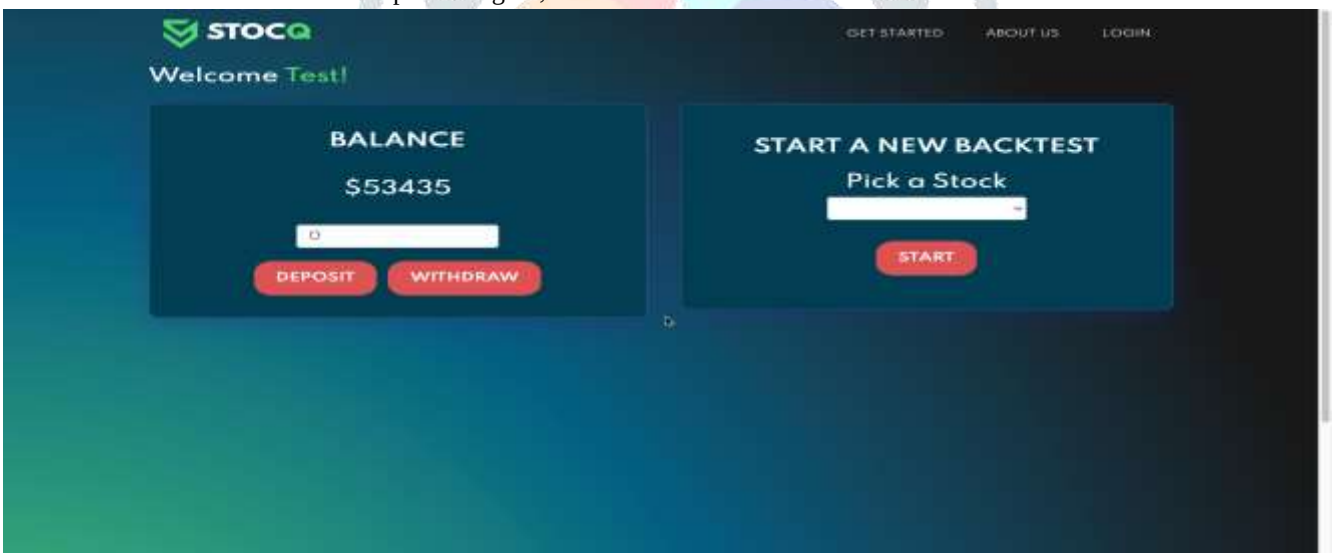

Fig 7: Get started

Fig 8: Backtest Statistics (Buys/Sells)

Fig 9: Backtest Statistics (Open, High, Low, Close Prices)

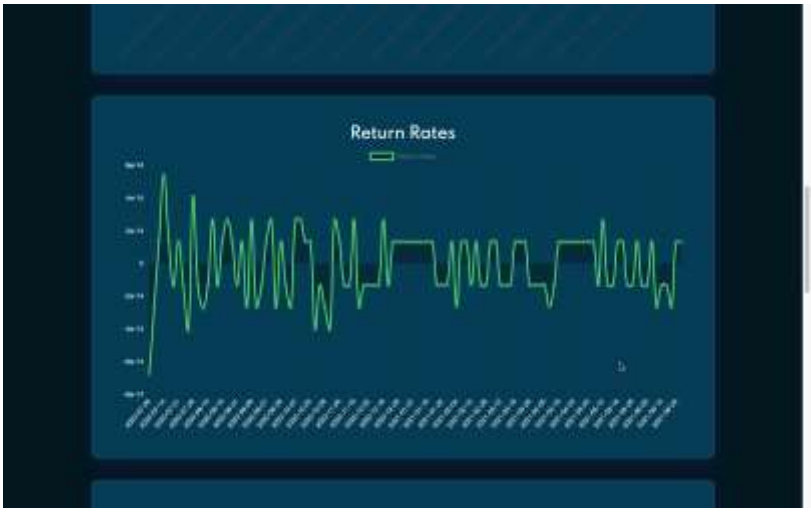

Fig 10: Backtest Statistics (Return Rates)

Fig 11: Backtest Statistics (Portfolio Value)

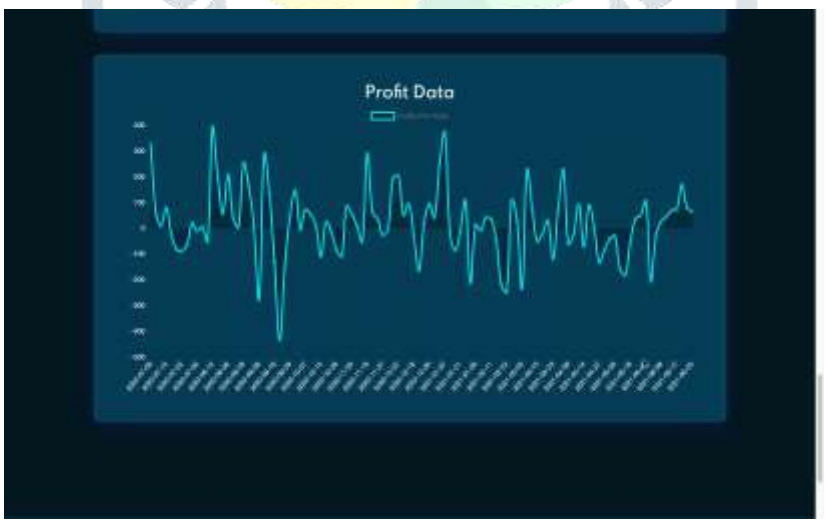

Fig 12: Backtest Statistics (Profit Data)

One can refer to the above backtest statistics and take up a wise decision as to whether to invest in a particular stock or not and hence, fulfilling the purpose of this project.

VI. CONCLUSION

The reinforcement learning agent receives rewards based on the performance of its trading decisions, given some environment. The environment evolves in discrete time steps according to some dynamical rules, and at each time step the agent takes an action that may influence the next state. After taking each action, the agent receives feedback in the form of a reward signal.

Deep Q-Network also allows us to trade directly without taking further optimization steps like other supervised learning methods. Using only a few hundreds samples, reinforcement learning algorithms variants based on Q-learning can generate the strategies that on average earn a positive profit.

So from the above discussions, we can conclude that the Deep Q Reinforcement Learning approach is the most suitable solution to our problem statement and has helped us in achieving satisfying results.

VII. REFERENCES

[1] <https://www.tensorflow.org/agents/tutorials/0_intro_rl>

[2] Jagdish Bhagwan Chakole, Mugdha S. Kolhe, Grishma D. Mahapurush, Anushka Yadav and Manish P. Kurhekar (2020), "A Q-learning agent for automated trading in equity stock markets".

[3] Akhil Raj Azhikodan, Anvitha G. K. Bhat and Mamatha V. Jadhav (2019), "Stock Trading Bot Using Deep Reinforcement Learning".

[4]Yang Wang, Dong Wang, Shiyue Zhang, Yang Feng, Shiyao Li and Qiang Zhou (2017), "Deep Q-trading"

[5]Michael Kearns and Yuriy Nevmyvaka (2013), "Machine Learning for Market Microstructure and High Frequency Trading".

[6]Jianqing Fan, Zhaoran Wang, Yuchen Xie and Zhuoran Yang (2020), "A Theoretical Analysis of Deep Q-Learning".

[7]Jagdish Chakole and Manish Kurhekar (2019), "Trend following deep Q-Learning strategy for stock trading".

[8]Quang-Vinh Dang (2020), "Reinforcement Learning in Stock Trading".

[9]Salvatore Carta, Anselmo Ferreira, Alessandro Sebastian Podda, Diego Reforgiato Recupero and Antonio Sanna (2020), "Multi-DQN: An ensemble of Deep Q-learning agents for stock market forecasting".