WHAT IS THE MOST PROMISING REINFORCEMENT LEARNING MODEL FOR TRADING?

INTERNAL PROMOTOR: WOUTER GEVAERT
EXTERNAL PROMOTOR: YARNE COPPENS

RESEARCH QUESTION CONDUCTED BY

ARNO DEFAUW

FOR OBTAINING A BACHELOR'S DEGREE IN

MULTIMEDIA & CREATIVE TECHNOLOGIEs

Howest | 2020-2021

Foreword

This bachelor thesis is the final work for the course MCT AI Engineer at HoWest University of Applied Sciences. It is also the discussion of the conducted research of the module “Research Project” from the previous semester.

After the analysis I reflect on the practical applications of this research based on professional knowledge and opinions of stock traders.

Based on the research and reflection I will also give some pointers and advice if this research were to be continued or re-conducted.

The research around the subject of AI in financial trading has been visited many times and is often looked at with speculation and doubt. With the knowledge I have Accumulated these past 3 years I too asked the question of the possibilities of AI in trading, and with that came the research question:

“*What is the most promising Reinforcement Learning model for trading?”*

Which I eventually decided to perform my research project and bachelor thesis around.

Acknowledgements

With special thanks to the following people for their help, support, and guidance:

Wouter Gevaert

Yente De Wael

Stijn Vandendriessche

- Arno Defauw, June 1, 2021

Abstract

The research question of this thesis is "*What is the most promising Reinforcement Learning model for trading?*". To test this, a multitude of reinforcement agents were programmed and tested using the correct research methods in order to make useful conclusions from the gathered data.

The most important aspects of the practical research were the manner of data collection and implementations of the agents themselves.

All in all, the results were quite meaningful for the tested agents, but more variants of the most promising agent could be researched and expanded upon.

In this thesis I will go through all the actions and decisions made during and after the research period, and the conclusions I made that affected my choices in development.

Table of Contents

Figures

List of Abbreviations

AC – Actor Critic

A2C – Advantage Actor Critic

A3C – Asynchronous Advantage Actor Critic

API – Application Programming Interface

BB – Bollinger Bands

CSV – Comma Separated Value

DQN – Deep-Q Network

DDQN – Double Deep-Q Network or Duelling Deep-Q Network

DDDQN – Double Duelling Deep-Q Network

EMA – Exponential Moving Average

FMDP – Finite Markov Decision Process

LSTM – Long Short-Term Memory

MACD – Moving Average Convergence Divergence

MDP – Markov Decision Process

NN – Neural Network

PG – Policy Gradient

RSI – Relative Strength Index

RL – Reinforcement Learning

TDAC – Temporal Difference Actor Critic

Glossary

Agent: object that learns to react in the proper way to maximise reward in its environment

Environment: object that defines the external situation for an agent to react to

Symbol: abbreviation of a company name used on the stock market

Hyperparameter: variable used to tune functionality of a neural network

Python: programming language developed by Guido Van Rossem

TensorFlow: AI library for python developed by Google

C++: a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language

Rust: a multi-paradigm programming language designed for performance and safety, especially safe concurrency

Markov Decision Process: is a discrete-time stochastic control process. It provides a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker.[1]

1 Intro

If one could find the solution to automate stock trading it would probably immediately make this person / organisation able to become rich beyond their wildest dreams. Unfortunately (or fortunately, depending on how you look at it) this is not yet possible with the technology of today.

This research project and thesis will cover an attempt to automate trading the stock marking using different Deep Reinforcement Learning agents and all the relevant technologies that are needed alongside.

It will cover the collection of data, choice of technologies for the different aspects of the technical implementation and all the mistakes made on the way. The conclusion of the research and advice for any possible continuation will also be discussed afterwards.

1.1 Background

Financial Trading

A **stock market**, **equity market**, or **share market** is the aggregation of buyers and sellers of stocks (also called shares), which represent ownership claims on businesses. Investment in the stock market is most often done via stockbrokerages and electronic trading platforms, and is usually made with an investment strategy in mind.[2]

This investment strategy is most often based on the combination of an analysis of the previous financial situations of the company and their future potential, and an analysis of the many different indicators that come with the market history of the company.[3]

These **indicators** are comprised of many different collections of data that can show different signs of how the market could possibly evolve. Example indicators are:

- Relative Strength Index:[4] is a momentum oscillator, measuring the velocity and magnitude of price movements.
- Moving Average Convergence Divergence:[5] is designed to reveal changes in the strength, direction, momentum, and duration of a trend in a stock's price.
- Exponential Moving Average:[6] is a type of weighted moving average that gives more importance to recent price data.
- Bollinger Bands:[7] display a graphical band (the envelope maximum and minimum of moving averages) and volatility (expressed by the width of the envelope) in one two-dimensional chart.

Since all these indicators are comprised of the historical data of a share, an advanced enough AI system should be able to figure these out by itself. So, it is debatable whether to use these indicators as extra input data or let the system figure it out for itself.

Reinforcement Learning

The concept of Reinforcement Learning is an area of Machine Learning that implements ML into an **Agent** that then learns to make the right action-choice to **optimise** its **reward** received by its environment.[8]

Since this **environment** can be anything that can be represented by an n-dimensional matrix, it is possible to create an agent for almost anything virtual that needs intelligent actions to achieve its end goal. For example an old ATARI-style game such as Pong, or Atari Breakout can be represented in an environment where an **observation** of the environment's **state** can be represented by a matrix of 3 dimensions [res-y, res-x, colour] -> (210, 160, 3)[9], which can then be interpreted by an agent.

This agent can input this matrix in its Neural Net and make a random decision for which action to take, in this example of Breakout; move the paddle left or right. This action then gets sent back to the environment which, in its turn, simulates the next state and passes it back to the agent. At every step the agent makes a decision and the environment updates accordingly.

After each step, the agent observes if the previously action taken was good or bad depending on the reward the environment gave, like a score or points received, and makes a backward-pass to adjust the weights of its neural net to make better decisions in the future.[10]

Application in Financial Trading

Stock trading in itself is just the observation of data and making decisions accordingly, so this would be the perfect application for an attempt to automate using Reinforcement Learning. By translating the price data to a 1-dimensional matrix and then creating a second dimension with the following timesteps we get a nice matrix to feed into a neural net.

And since this data is a time series, an optimised network architecture specifically for time series, such as Long Short-Term Memory layers, can be used to fully exploit this property.[11]

But the network alone isn't the only variable that has an impact on performance. The number of layers and the amount of neurons and connections for each layer, also need to be taken into account. Inside the workings of the agent itself are even more variables to consider.

These variables include but are not limited to:

- The formula to adjust the weights of the taken actions
- The choice between a random action and a calculated action
  (to collect more varying data to learn from)
- Epsilon: The rate at which this decision of randomness is reduced over time
- Learning Rate: The impact one step has on the network
- Gamma: The discount rate of old decisions that might have still had an impact

All this and more is included in the research I conducted, and will be explained in the following chapters.

2 Research

The research consisted of 4 sub-question. These questions are analysed and answered in detail throughout this paper.

Which RL models can be used for trading?

The first step was to look at the different options of neural networks to use for the trading agent because not all are qualified for this type of timeseries data. The eventual set of models to test were DQN, DQN with LSTM, PG and TDAC. To compare performances of the agents I added results of a random agent.[12], [13]

What is the performance difference between the available models?

The performance between the different agents in the research was obvious, in initial tests it was already clear that DQN had the advantage over the other agents. Variants on DQN with an LSTM network gave performance an even bigger boost.[14]

Is there a contextual performance difference between some models?

No, it’s apparent that the models see the data in the same way and performance was purely impacted by the model itself.

Also, in later versions of performance measuring, a better data implementation was added that completely removed all forms of contextual meaning from the training data by randomising every single episode.

Advantages and disadvantages of the tested models?

In the application of this case study, the better the performance of the model the more compute it required. So, the research on the better performing models was harder to achieve with the available compute of a single RTX3070 while the simpler models were generally done within the hour.[15]

3 Technical Research

In the following chapter I will describe all the different technical aspects of the conducted research, with the faced difficulties, problems, and their solutions.

3.1 Language and Libraries

The programming language is an important aspect of any software-based research. The choice was easily made since most data science research is conducted purely in Python with the use of the TensorFlow library by Google.[16], [17]

Although the Python language is relatively slow in processing[18] and not easily multi-threaded it is easy to work with, and understand more complex programs. This also makes Python the most go-to language of the field of AI development. The added use of the extremely popular library TensorFlow, which is an interface to the C++ layer of the CUDA compute libraries, makes it an obvious choice to learn the basics of AI with.

Other options considered were C++[19] and Rust[20] for their low-level and speed advantages, but as of yet I have no experience with these languages so these would have to be learnt during the process together with the practical implementation of neural networks all at the same time of the research.

Rust being too new to practically implement deep learning was quickly dismissed.[21] C++ on the other hand was considered for the remarkable speed advantage it has over Python[22], also the NVIDIA CUDA libraries used by TensorFlow are made in C++ so the integration of this would be lower level.

The use of libraries was logical here but in more recent days I have researched and learnt to program neural networks from scratch.[23] The implementation of this technique would also have been a major factor in reducing compute time, but not preferable to apply in this project due to time constraints.

All in all C++ with a NNFS approach would have been a preferable choice for a complete implementation of this research but, alas, the learning of an entirely new low-level language, multithreading, creating an own library, and CUDA integration together with the research project in 4 weeks would have been unrealistic to say the least.

3.2 Data

Primarily the task was to obtain the data required for use in the environment. Since stock data is quite popular to work with, datasets are easy to find. Unfortunately, these datasets are outdated and often of low resolution. For this project the choice was made to work with 1-minute size data resolution, so this meant most of the historical datasets publicly available did not meet the requirements.

Another solution was to turn to API based datasets; these deliver live updates of the newest stock prices but there the problem was that historical data was limited for free tiers of the API. The maximal available data on any API found was 6 months. For 1-minute data, this is already quite a sizeable dataset. This solved the problem of acquiring a good amount of data.

To collect the data, a script was written to obtain the maximal amount of available data provided by the chosen API. 6 months of data for the 1-minute timeframe make about 300 000 datapoints per symbol. Since final choice of the API only allowed 5000 datapoints per request and a maximum 1 request per minute, it would be hard to collect all data for a multiple of symbols. So, the solution for this was to collect a multitude of API keys with temporary emails and cycle through the keys during requests to not be hindered by this limitation.

The choice of time resolution was made to collect as much data as possible, since the resolution could be aggregated into a larger timeframe afterwards. During the research, it became clear that the timeframe of 1 minute is rather small to accurately predict meaningful indicators. More research should be conducted to validate the impact of these timeframes on the results. A larger timeframe was never implemented, but I will touch on this subject later on in the reflection.

The script collected all data for any symbol (company) given in a list obtained from a dataset. This made the available choice out of 37 671 different symbols.

It was also possible to collect indicator data for all these symbols. However, this would decrease the amount of data that could be collected per API request and would slow things down massively. An alternative solution would be to calculate the indicators locally, since these are just formulas after all, and use them as environment data for the agent. But alas, the inclusion of indicators would multiply the input size of the neural network by $n_{\text{indicator}}$. So, it would be impractical to include this in the project without the availability of large amounts of compute with enough graphical memory to fit the size of array this would make. This could vastly improve the performance of the agent. The system that was used, however, did not allow these kinds of calculations.

Due to the limited timeframe of this research and the limited access to compute power, the amount of data used is rather small. The research, however, gives an indication about what results could be. To get more accurate and reliable results, this research could be reiterated in the future with different data, more data and multiple time-resolutions.

Post-Research Fixes

Due to the restrictions mentioned above, which were later solved, I made the decision to re-train the agents without any change to the agents or hyperparameters themselves. This gave me a better view of the performance difference between the agents and gave a better feel of how good the agents were learning the indicators. This, however, could have resulted in the agents learning the data by heart in the previous training, since they all learned from the exact same piece of data from the same stock.

The implementation of the new dataloader made sure that every single episode had a new set of random chosen data. Inside the training loop of the agent class the dataloader gets called to initialise the environment. The loader itself chooses a random company stock and extracts a random timeframe of data from the available data. The dataloader's function then initialises an environment instance with this randomly chosen data and passes it back to the agent for training. This procedure happens after every single episode, so the agent has no choice but to only learn indicators instead of the data itself.[24]

After running the DataCollector script for a multitude of companies, the available collected data should be more than sufficient for representative results of the agent's capabilities after training.

Another implemented solution was in the collection of the episode performance data. Since the environment changed every single episode the parameters that were being collected no longer represented meaningful data. So, all agents had to be adjusted with new performance data metrics, which also gave me the chance to collect more data in general.

Since the maximal achievable performance is dependant on the environment instance, I had to relativise the data to accurately represent performance, since some episodes can have a completely different achievable profit, e.g. max_multiplier: 1.4 versus max_multiplier: 5.2, I divided the achieved performance by the maximal performance to be able to more accurately compare different episodes and agents.

The code to write the data to a csv file was also moved from outside the agent so the data could be written after each episode instead of at the end of the entire training. This unforeseen situation was responsible for the loss of a lot of training data on the original runs, so the change was needed and allowed me to see the results while the training was still in progress.

A new method was also added to calculate the duration of episodes to know how long a certain number of episodes will take, how much time has passed, and an estimation of time left. This gave a better feel to know how long training runs are going to take and to know whether I had time to completely run these, or to even attempt to run them for 500 episodes. As the prediction of the LSTM agent was 42 hours, I was aware that it wasn't going to finish any time soon, so I was able to make an informed decision to stop the training at a certain time.

3.3 Agents

The choice of agents to test was clear from the start, research suggested multiple agents and their feasibility for trading.[12], [13] The final choice of agents was the following:

Random

The random agent was included to compare the results of agents to an unconscious choice of action and to see how much impact the algorithms and neural nets had on the performance. We will be comparing all performance measurements against this agent, since the difference between a Reinforcement Learning (RL) agent and a random agent is the main observation goal. This data is the performance baseline that needs to be improved upon.

This agent consists of just the basic framework and the action method, simply being a random choice of action to take, with no use of context or calculation whatsoever.

The graphs below show the profit multiplier for each episode ran. over a total of 500 episodes. A profit multiplier of 1.0 would mean the same resulting capital as the beginning of the episode. The following data shows that a random agent would achieve a loss of, on average, 99.9991% every episode, you might as well burn your money in a nice campfire and roast some marshmallows.[25]

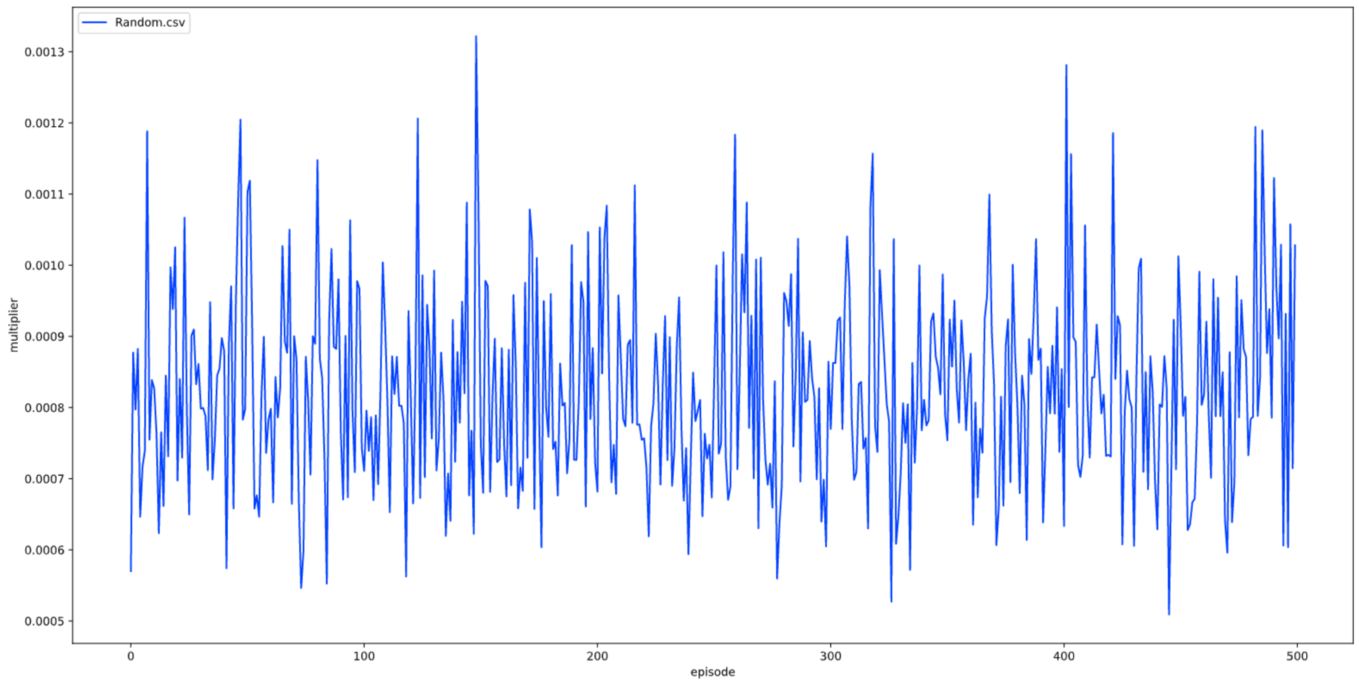


Figure 2: [Graph] Random agent performance

Policy Gradient

Policy Gradient (PG) is a policy-based agent that primarily works to optimise the policy to maximise the return value in any state and with any initial state. For this application this agent would be suboptimal because there is a near infinite amount of states. This makes it hard to approximate or generalise states in this context so its results might be a bit too simplistic and generalising. The PG agent is also not all that efficient with training data.[26], [27]

This agent is especially adapted to allow for continuous action spaces, which is not at all necessary in this use-case since trading has only a few discrete actions to take (buy, sell, short, etc.). If the amount of money to be invested at every step would be integrated into the project as a variable, this approach would have more value. This would make the entire research a lot more complicated and thus falls outside of the scope of the project.

A couple of sources indicated the use of a PG agent for trading, but these suggested an ensemble strategy with multiple agents doing different tasks.[12], [28] This strategy was too complex to implement during the research period. An attempt was made at a singular PG agent, but these results should be analysed with care since this is a relatively simple algorithm to implement. However, was decided to include it in the research to see what it had to offer. To no surprise the performance results indicate a capital multiplier not far from the random agent, even after 500 training episodes the results are marginally better than random. Since this training took 69 times longer than the random agent the effort is certainly not worth it.[24]

After these results I decided not to pursue the PG any further.

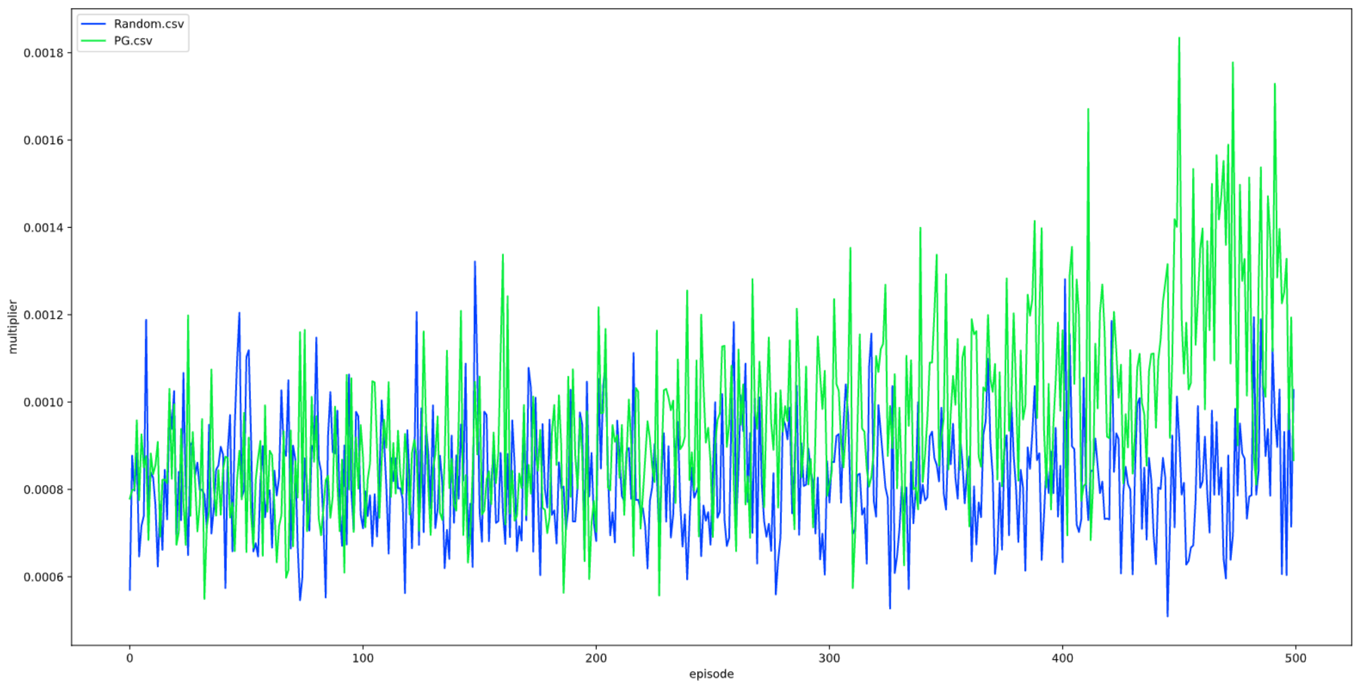


Figure 3: [Graph] Policy Gradient agent performance

Temporal Difference Actor Critic

Temporal Difference learning is a model-free method that utilises estimates of the final result to pre-emptively update values in an attempt to make more accurate predictions the more data is available. The algorithm adjusts the weights, instead of adjusting the values associated with the individual state-action pairs.[13]

The Actor Critic-part in the name of this agent points to the fact that the neural network is split into two different networks. The Critic part takes the observation and result of previously taken actions and makes an estimation on the "value" of the current state. This value being the importance of the current action that needs to be taken. The Actor then chooses an action based on the observation state and the predicted value of this state. In this way the agent criticises its own decisions and learns by correcting them.[29]

This is still a pretty simplistic approach to the complicated problem of financial trading but as seen in the achieved results it's the biggest leap in performance we have seen so far, but still on a marginal scale in relation to the desired outcome of multiplier >1.0.

This result is good to see, but since it is still too far off from the goal, it was decided to test other agents before trying to optimise TDAC with different hyperparameters.

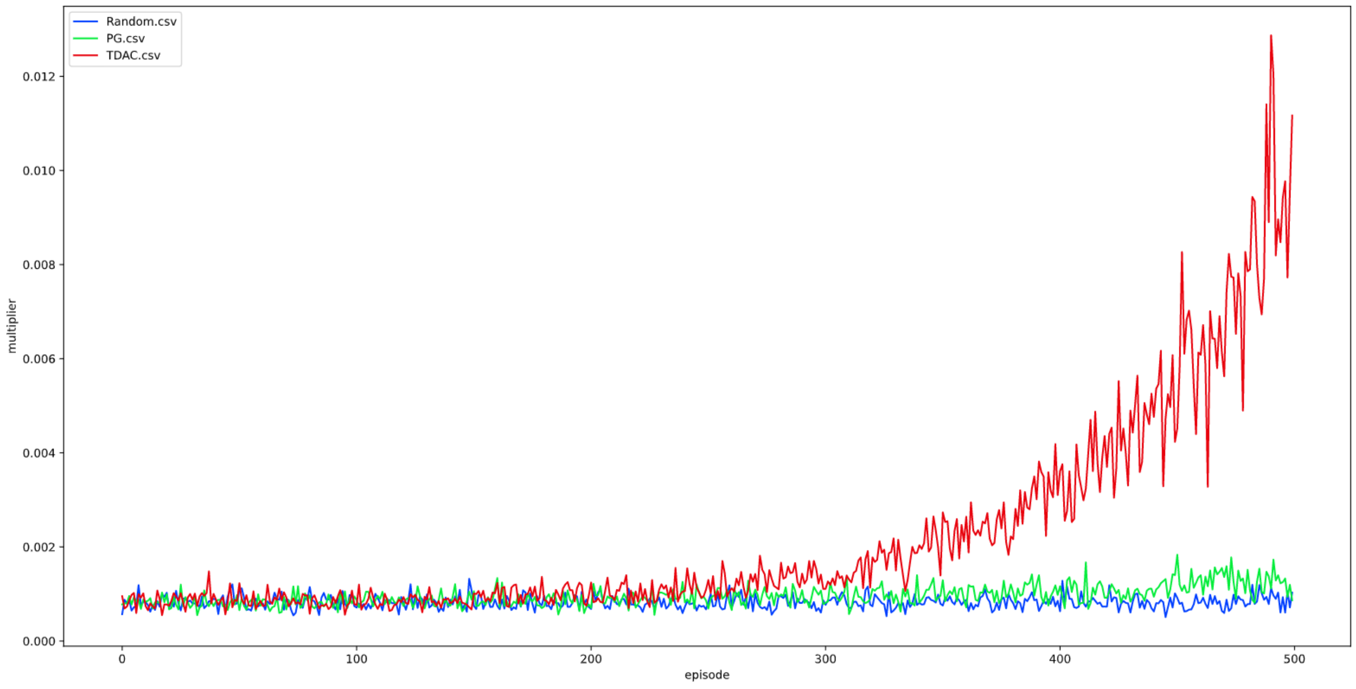


Figure 4: [Graph] Temporal Difference Actor Critic agent performance

Deep Q-Network with Convolutional Network

The next implemented agent is the Q-learning method, which, just like the previous TDAC agent, is a model-free approach to RL. This agent has the ability to handle problems with stochastic transitions and rewards, which is perfectly suited for the problem of financial trading.[12]

The classic Q-learning method is able to work with any finite Markov Decision Process[1] (MDP), requiring a finite state space. This agent operates by finding an optimal policy to maximise the expected reward over all subsequent steps in the episode, starting from the current state. Q-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly random policy.[30]

Since this classic Q-learning method consists of a matrix for a finite state space where every number is an optimised value to see which action is best taken, it would be impossible to make a network for the infinite state-space of financial trading. This problem, however, can be solved by replacing this Q-table with a regular convolutional neural network that will approach an optimal value for each given combination of input data.[31]

For training this agent the Monte Carlo method is used to randomly sample every possible state-action pair possible and use this data to determine the optimal score for every discrete action that can be taken in that state.[32]

As seen in the figure below this gave an exceedingly large jump in performance for the agent. Although the results are still far below 1 it gives promising results to follow through with this agent and experiment with more variations of this network.

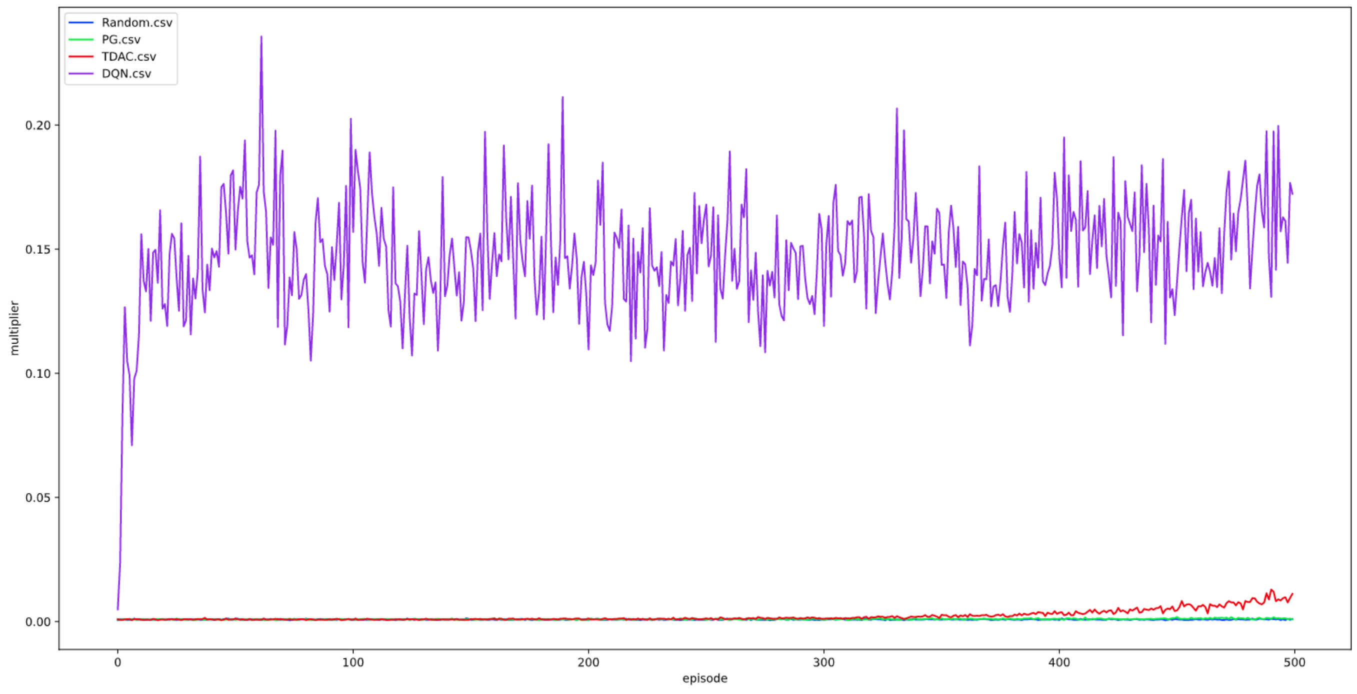


Figure 5: [Graph] Deep Q-Network agent performance

Deep Q-Network with Long Short-Term Memory network

Continuing the promising results of Q-learning, some further research was conducted that suggested a Long Short-Term Memory neural network would be perfect for the use-case of financial trading. So an implementation was made on a variation of the DQN.[11]

This neural net is a lot more complicated (Figure 5) in respect to calculations and logic. This made the implementation a lot more complex and increased the duration of the training cycle

After a small 200 hundred episodes, the model already started to stagnate, which led to the decision to not further invest resources into this network.

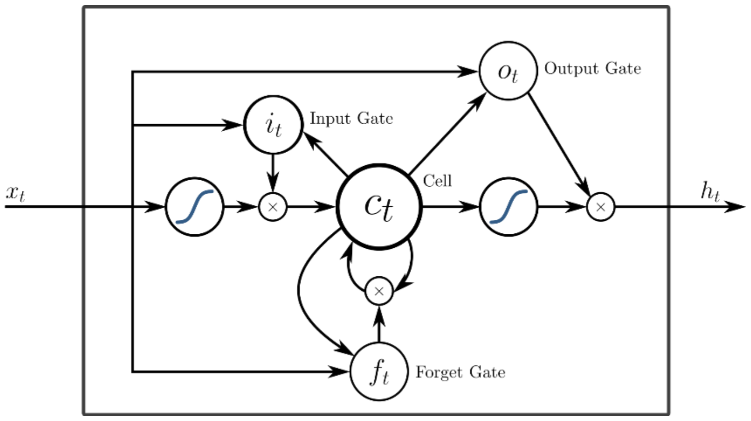


Figure 7: [Diagram] Inner workings of an LSTM cell

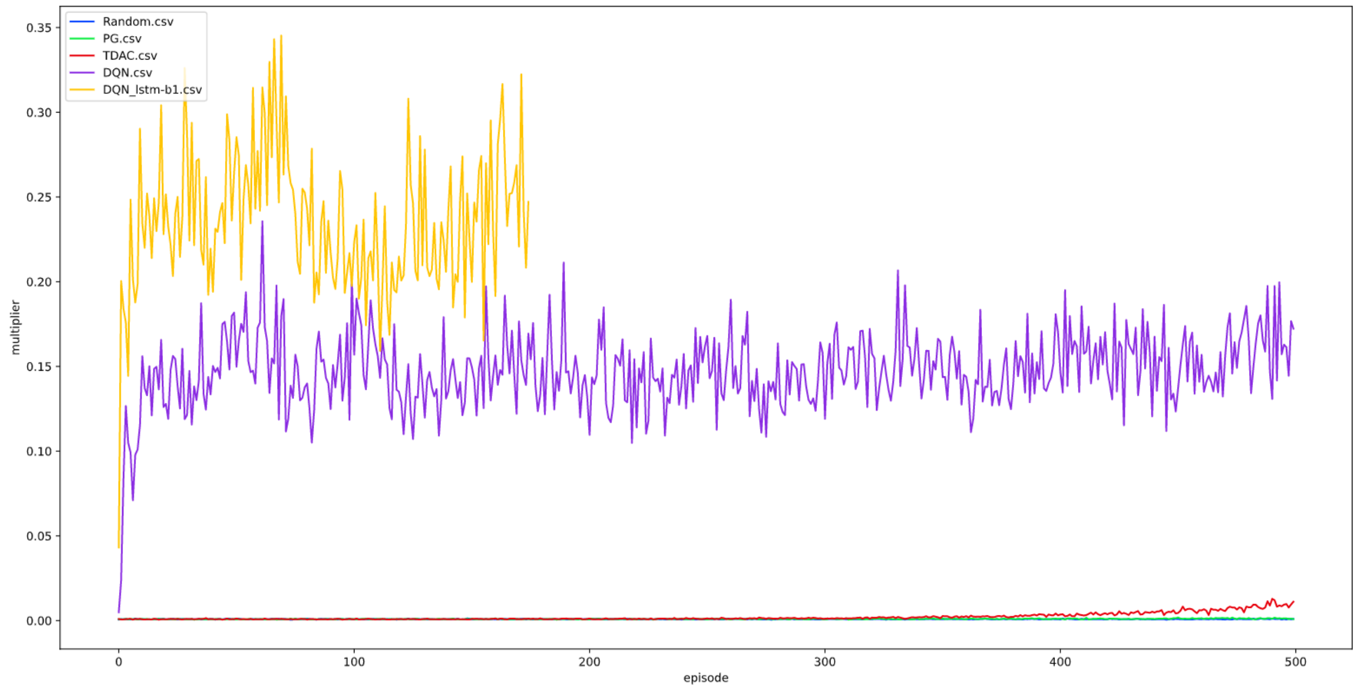


Figure 6: [Graph] Deep Q-Network with LSTM agent performance

Max profit

To place the results of the agent's performance onto a spectrum the 2 extremes need to be apparent. The random agent having the lowest possible performance metric, and against that, the maximal achievable profit that the agents could have possibly achieved in a perfect performance. Since every episode of every agent was random and had a different maximum profit, it would be too chaotic to visualise. So, the choice was made to take a random example value of ± 1.4526 to represent the concept.

All these previous results look promising and it is nice to see the performance progress but when the maximal profit of the episode is included, the results become quite a bit more disappointing.

Remembering that a score of 1.0 on the y-axis is a no-profit, no-loss capital multiplier, the highest performance achieved by any agent is around 0.32, meaning a loss of 68% of capital. Which is certainly less than ideal.[24]

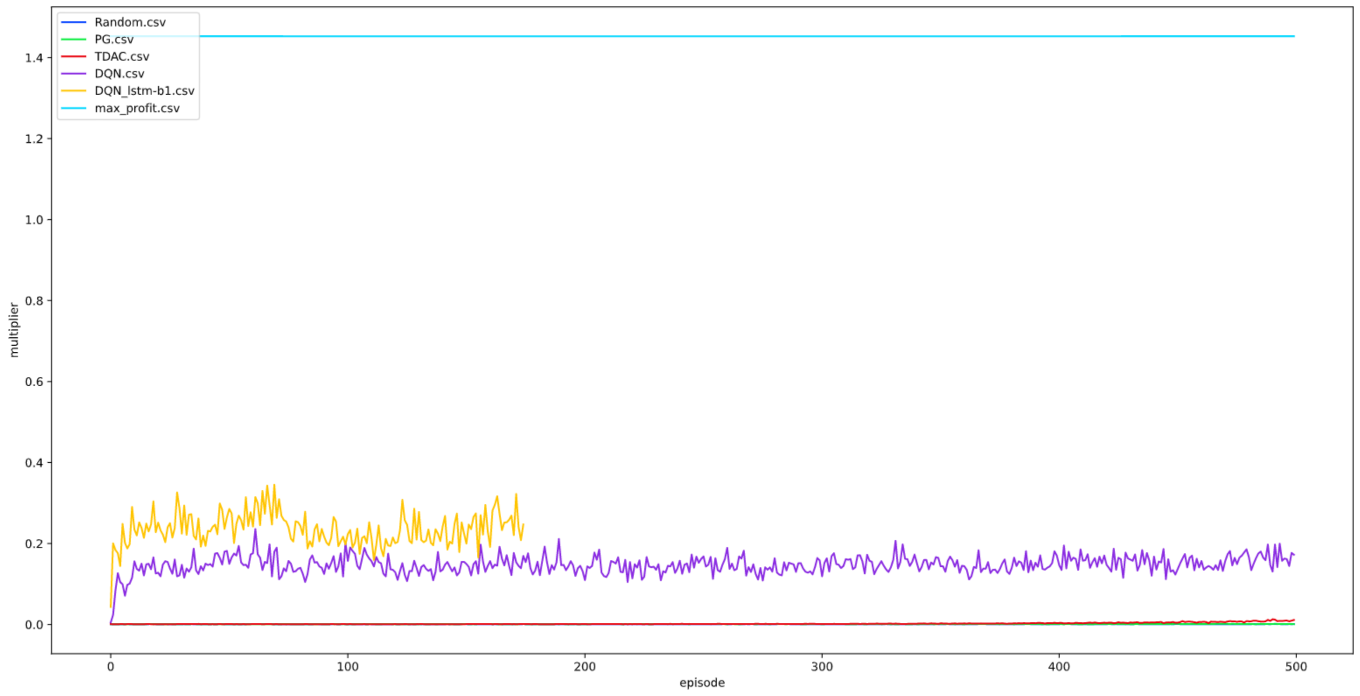


Figure 8: [Graph] Comparison to maximum achievable performance

3.4 Models

Since project structure is important, it was already pre-allocated to develop all agents in separate classes. This allowed for the code structure to be less repetitive and reusable for multiple agents. It also made the collected data more reliable since all agents are trained and tested using the same methods.

A general method was made that could be applied to all agents, but since some agents required different operations and different steps it was not possible to make an all-in-one solution. To do this it would require too many separate small functions and more complex control structures and function arguments which would make the code too complicated and harder to read without any real added value. So, in the end, every agent class had their own training function built-in with most hyperparameter variables as arguments.

Another problem faced was the dimensions of the data. The final choice of environment gave a 2-dimensional array as an observation to the agent. This was no problem for the simpler agents, but as neural networks became more complicated it became harder to reshape the data in a correct way to be of any practical use to the agent. Therefore, the choice was made to take only one of the two dimensions of data to work with, which annulled the value of previously collected training data and required to rerun training of the simpler agents for data uniformity.

For this problem, the choice of data was between the closing prices of the past *n* minutes (with *n* a hyperparameter set to 60) and the difference between the closing price and the previous closing price in that set (so a list of price changes). Since the price itself was not really of any value between different companies and the price change is more applicable to different sets of data, the 2D array was eventually reduced to a 1D array containing only the price changes.

3.5 Environment

The choice of environment was harder than expected. Since AI development around the stock market is mostly done in a corporate context, most publicly available open source environments are of lower quality and outdated. A lot of research was done to choose the best one because creating an environment for stock trading was far outside the scope of the research project.

After a couple of weeks of development the realisation was made that the chosen environment also had it’s deficiencies, because of the vagueness of the documentation, the exact meaning of final profit number was unclear and the actions did not contain a hold position, forcing the agent to always choose between buy or sell on every step. Also, the aforementioned 2-dimensional observation space problem could have been avoided with a different or self-made environment.

To improve the accuracy of the results in this paper, further research could be conducted with a custom-made trading environment that circumvents or corrects the above-mentioned problems. This could increase the probability of the agent successfully learning / recognising usable patterns in the financial data.

3.6 Training

The training time of the agents was incrementally longer the more advanced the agent itself became. Resulting in following timing results for each agent.

```
random:     0.042824721336364
PG:         2.897515153884888
AC:         16.12540385723114
DQN:        19.10190412998199
DQN_LSTM:   302.2284741163254
```

Average training time of a single episode in seconds (avg. over 10 episodes)

```
random:     0.0797230935096740
PG:         3.5847126026153564
AC:         20.327430575847625
DQN:        23.080175398349763
DQN_LSTM:   [±42h / 500]
```

Average training time of a single episode in seconds (avg. over 500 episodes with new dataloader class)

With the last one being a DQN implementation using an LSTM network the duration per episode was problematic, but the results were a good deal better than the others.

Therefore, most time was allocated to finetune the DQN LSTM agent and try to improve results. Since training took this long on the available compute it resulted in some problems to keep the calculation running, for example, when a windows computer is not used for a certain time it pauses all processes in an attempt at saving power, which the python kernel does not appreciate. This made for multiple failed attempts at overnight training. If and when this research were to be continued with a greater budget and use of servers, this can be avoided.

3.7 Data-Collection

Data collection is extremely important for the scientific method so in every agent a data collector script was built-in to collect the important parameters and variables to observe performance of the agents.

At first this data was all appended to a list to be written into a CSV at the end of the computation run, but when encountering failed runs of the more advanced models and losing this data multiple times a modification had to be made where the data was written per-episode.

4 Reflection

With hindsight always being 20/20, I can say that this research could have been performed in a better and more efficient way. The start was slow and could have used more preparing research in advance. Also, the proper research methods should have been used more thoroughly from the start to make sure no data / results were lost while developing the models and algorithms to better show why certain decisions were made.

The subject of automated trading with Reinforcement Learning is certainly an interesting one and was a good choice for me, keeping my interest and motivation high during the whole process.

The code written for this project is well structured and easy to understand, a lot of attention was spent to make sure this was the case, as code efficiency and simplicity are one of my main foci while programming.

To reflect on this research, I can honestly say that it could have been performed better, but since this is my first thesis, I personally think I did a pretty good job. I have learnt a lot and will be able to use this experience in my further pursuits in education, with hopefully many more papers and theses to follow.

What are the stronger and weaker parts of the results in your research?

Research documentation during the research period could have helped when writing this thesis. As well as the collection of data from the first tests and experiments could have added some more context to certain decisions while developing the code.

Unfortunately, the maximal performance is far below what's needed for practical applications so none of this research is usable except for further study and research, which might be performed by myself in the near-future since this thesis has helped me form better conclusions and ideas than during the research period itself.

Is the resulting method usable in practice?

No, as mentioned above, the performance is marginally better than random choices during trading, even with the 36% regain of capital this leaves +70% loss to any funds invested. It's better to donate to charity if you're willing to throw away your money.

What are the hardships of implementation for a company?

The access to data and working algorithms to train the agents might be tough if you have no direct access to the market history. Also, the compute power required to train, retrain and update the networks would be substantial for most companies, although any company with the funds to invest at this capacity should be able to afford the compute.

What would be the added value for a company?

A bank, for example, might use this technology to utilise the vast amounts of funds in their possession to trade and receive interests from their investments. As well as a hedge fund or even a charity could use this to increase their available funds.

What are some alternatives / suggestions to this research given by the community?

Using multiple resolutions of data at the same time, dividing them onto different LSTM networks and merging these networks at the end so the agent may have a better understanding of the environment and state in which it finds itself. This could help with the differing strategies that can be used depending on the time resolutions you plan to trade on (scalping, intra-day, day, long-term).

Is there an added societal / economic / socioeconomic value?

Debatable, there certainly is *a* socioeconomic value but whether it's a positive influence can not yet be said. The ability to collect and grow capital is already problematic in today's economy. This technology could put even more power in the hands of those who already have too much.

If and when an advanced enough AI system that can make the perfect trades would be available, a large problem would arise. Since the driving force of the economy in capitalism is the accumulation of more wealth, this AI would enable its 'user' to increase their portfolio vastly. This increase would be exponential and unmaintainable, practically collecting all wealth of the economy.

A similar problem is already in motion today with the problematic, unethical imbalance of wealth in the world. It is obvious the current economical system cannot maintain in the age of AI, thus requiring a drastic change before such systems could, or even should, be implemented.

What are your suggestions for a possible continuation of this research?

If this research were to be recreated I suggest making sure no data could be lost during compute runs that fail, implement more fail safes earlier on in development to prevent the compute to be interrupted, and double check hyper parameters and final code before starting a long run.

Also, perhaps another approach of learning could be tested since the results show that this method may not be optimal for continuing without drastic changes in the inner workings and processing of data. As mentioned before a more guided approach could be beneficial instead of making the agents figure out the workings of the stock market on their own.

Other approaches / ideas:

Pre-training the neural net itself with labelled data made by a manually made trading algorithm could have a huge impact on starting performance by guiding the network into the rough shape that should be achieved. Although this could be risky and deny the network from making its own interpretation of the data, it would probably be an improvement on the current performance achieved in this research.

5 Advice

The main advice to be added to this thesis is; do not use this project for trading.

A whole lot of extra deep diving research is yet to be done to adjust and finetune the methods used in this research to make it a profitable application.

However, if it had been a success on the first attempt it could easily be automated on a server with automated data retrieval and retraining to make the best possible trades. Just a case of setting it all up and letting it run in the background.

An alternative to this project / approach could be to develop an own trading algorithm by hand, this would ensure used techniques by professionals and allow for fail safes in the script to minimise losses. This is, however, a complete different approach to automated trading and should be further researched in comparison to the use of AI.[33]

The extra tools developed during this project are of good quality and still usable in further research or practical use. These tools include the data collector that gathers all stock data from an API access point, and a graphing script to visualise training data and performance to make clear comparisons between the different agents. Also, most of the structuring code is still applicable in any attempted continuing research, the base has been laid, just some classes and agents need adjusting / finetuning or re-writing.

6 Conclusions

Resulting this research, a couple of conclusions can be made; a better way of using the available data should be found to show the neural net certain trends or patterns more optimally. Perhaps data, hand labelled with certain trends to recognise, could help, but this would greatly increase the required time and effort to train the model.[1]

Another possible method we can derive from this is using multiple networks to recognise certain patterns in different resolution of data[28], for example a double top-reverse pattern over a month vs over a few hours to better create an instinct of which action to take depending on other network's values.[2]

The importance of uniformization in the research method and data collection/ usage has also been greatly emphasized by some conclusions made in the beginning of the research where a few mistakes in this aspect were made and corrected. This is something that should be well thought out and planned from the beginning of the research so no useless computations are made where data is not collected, or where training data is not comparable with concurrent agents.

Furthermore, compute efficiency can also play a large factor in the algorithms, by minimalizing useless computations and optimising multi-threading and GPU usage a great deal of time can be saved that can be used to re-iterate parameters and spend more attention on hyperparameter optimisation.

All in all, it can be concluded that the conducted research did not achieve nearly enough performance to be actually applied in any meaningful way, but it is a great step in the right direction and shows quite some potential in the concept of financial trading with Artificial Intelligence.

[1] *Paraphrased from advice by Yarne Coppens*

[2] *Suggestion from Reddit user u/Abhisheked85*

7 Literature

[1] R. Bellman, “A Markovian Decision Process,” *Indiana Univ Math J*, vol. 6, no. 4, pp. 679–684, 1957.

[2] J. Chen, “Stock Market Definition,” Mar. 2021. https://www.investopedia.com/terms/s/stockmarket.asp

[3] T. Parker, “5 Essentials You Need to Know About Every Stock You Buy,” May 2021. https://www.investopedia.com/financial-edge/0411/5-essential-things-you-need-to-know-about-every-stock-you-buy.aspx

[4] J. Fernando, “Relative Strength Index (RSI),” Apr. 2021. https://www.investopedia.com/terms/r/rsi.asp

[5] J. Fernando, “Moving Average Convergence Divergence (MACD),” Apr. 2021. https://www.investopedia.com/terms/m/macd.asp

[6] J. Fernando, “Moving Average (MA),” Apr. 2021. https://www.investopedia.com/terms/m/movingaverage.asp

[7] A. Hayes, “Bollinger Band® Definition,” Apr. 2021. https://www.investopedia.com/terms/b/bollingerbands.asp

[8] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement Learning: A Survey,” *arXiv:cs/9605103*, Apr. 1996, Accessed: May 29, 2021. [Online]. Available: http://arxiv.org/abs/cs/9605103

[9] OpenAI, “Gym Breakout-v0.” https://gym.openai.com (accessed May 29, 2021).

[10] W. H. Fleming, “Chapter IV. Dynamic Programming,” in *Deterministic and Stochastic Optimal Control*, 3rd ed., vol. 1, New York: Springer, 1975, pp. 81–83. [Online]. Available: https://www.google.com/books/edition/_/qJDbBwAAQBAJ?hl=en&gbpv=1&pg=PA81

[11] S. Hochreiter and J. Schmidhuber, “Long Short-term Memory,” *Neural Comput.*, vol. 9, pp. 1735–80, 1997, doi: 10.1162/neco.1997.9.8.1735.

[12] B. Yang, “Deep Reinforcement Learning for Automated Stock Trading,” *Medium*, Mar. 08, 2021. https://towardsdatascience.com/deep-reinforcement-learning-for-automated-stock-trading-f1dad0126a02 (accessed May 29, 2021).

[13] “Deep Reinforcement Learning For Trading Applications,” *Alpha Architect*, Feb. 26, 2020. https://alphaarchitect.com/2020/02/26/reinforcement-learning-for-trading/ (accessed May 29, 2021).

[14] X. Gao, “Deep reinforcement learning for time series: playing idealized trading games,” p. 8.

[15] J. Hanlon, “Why is so much memory needed for deep neural networks?” https://www.graphcore.ai/posts/why-is-so-much-memory-needed-for-deep-neural-networks (accessed May 31, 2021).

[16] U. India, “Why Python is the most popular language used for Machine Learning,” *Medium*, Mar. 14, 2018. https://medium.com/@UdacityINDIA/why-use-python-for-machine-learning-e4b0b4457a77 (accessed May 31, 2021).

[17] "Tensorflow API." https://www.tensorflow.org/api_docs/python/tf

[18] "Why are Python Programs often slower than the Equivalent Program Written in C or C++?," *Stack Overflow*. https://stackoverflow.com/questions/3033329/why-are-python-programs-often-slower-than-the-equivalent-program-written-in-c-or (accessed May 29, 2021).

[19] B. Stroustrup, "Standard C++." https://isocpp.org/

[20] G. Hoare, "Rust." https://www.rust-lang.org/

[21] "Is Rust good for deep learning and artificial intelligence?," *The Rust Programming Language Forum*, Dec. 04, 2018. https://users.rust-lang.org/t/is-rust-good-for-deep-learning-and-artificial-intelligence/22866 (accessed May 29, 2021).

[22] N. Tamimi, "How Fast Is C++ Compared to Python?," Dec. 2020. https://towardsdatascience.com/how-fast-is-c-compared-to-python-978f18f474c7

[23] H. K. & D. Kukieła., *Neural Networks from Scratch (NNFS)*. https://nnfs.io, 2020.

[24] A. D, *Trust me bro*. 2021.

[25] K. Healy, "Fuck Nuance," *Sociol. Theory*, vol. 35, no. 2, pp. 118–127, Jun. 2017, doi: 10.1177/0735275117709046.

[26] J. Peters, "Policy gradient methods," *Scholarpedia*, vol. 5, no. 11, p. 3698, Nov. 2010, doi: 10.4249/scholarpedia.3698.

[27] "Policy Gradient Algorithms," *Lil'Log*, Apr. 08, 2018. https://lilianweng.github.io/2018/04/08/policy-gradient-algorithms.html (accessed May 29, 2021).

[28] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy," Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 3690996, Sep. 2020. doi: 10.2139/ssrn.3690996.

[29] C. Yoon, "Understanding Actor Critic Methods," *Medium*, Jul. 17, 2019. https://towardsdatascience.com/understanding-actor-critic-methods-931b97b6df3f (accessed May 29, 2021).

[30] F. S. Melo, "Convergence of Q-learning: a simple proof," p. 4.

[31] S. Karagiannakos, "Deep Q Learning and Deep Q Networks," *AI Summer*, Oct. 01, 2018. https://theaisummer.com/Deep_Q_Learning/ (accessed May 29, 2021).

[32] "Monte Carlo Method - an overview | ScienceDirect Topics." https://www.sciencedirect.com/topics/neuroscience/monte-carlo-method (accessed May 29, 2021).

[33] Y. De Wael, "Algoritmisch of reïnforcement learning ,welke optie is het beste voor het automatiseren van financial trading?," HoWest University of Applied Sciences, Kortrijk, 2021.

[34] A. Pešterac, "The Impact of Automated Trading Systems on Financial Market Stability," *Econ. Organ.*, vol. 16, no. 3, pp. 255–268, 2019.

8 Appendix

8.1 Installation manual

This document will be instructions on how to prepare this project for use.

Step 1: installation

Clone git repository:

```
git clone https://github.com/Arno989/Bachelor-Thesis.git
```

Install dependencies:

```
pip install tensorflow pandas gym gym-anytrading matplotlib jupyter jupyterlab
```

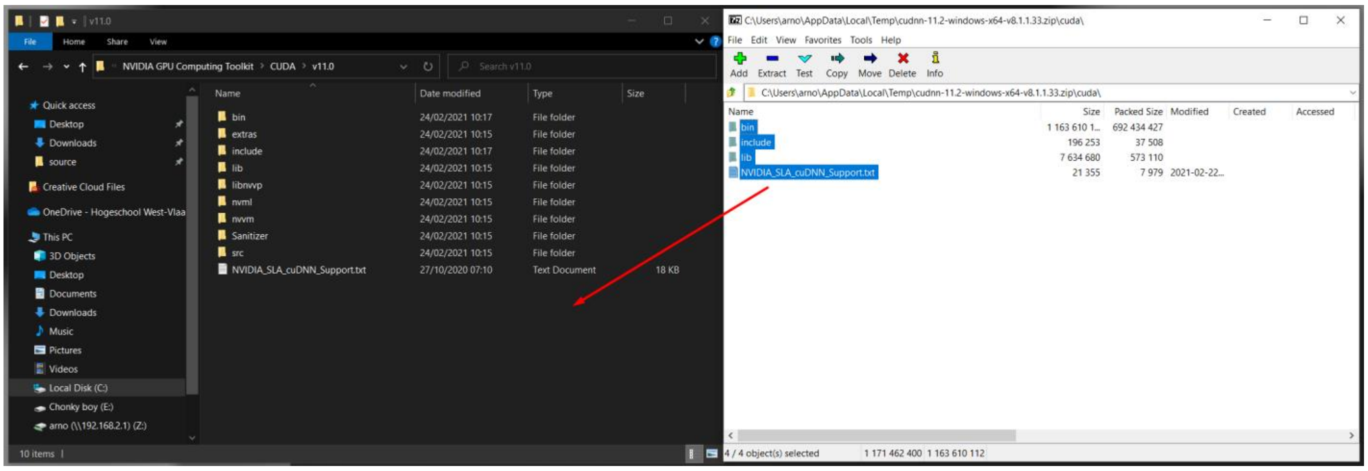
You will have to make sure the NVIDIA CUDA compute library is installed and functional

Instructions for installing NVIDIA CUDA libraries for TensorFlow:

[source guides: TensorFlow, NVIDIA]

This guide is for Nvidia GPU's with the Ampère architecture (compute capability 8.6) and TF v2.4.0+, different architectures might need other versions of the libraries.

1. Download & install the most recent Nvidia drivers at: Nvidia drivers
2. Download & install CUDA 11.2.2 at: Nvidia CUDA Toolkit
3. Download cuDNN 8.0.4
   a. Create account at: Nvidia
   b. Download v8.1.1 for CUDA 11.2.2 at: cuDNN Archive (Direct link for win_x86)
4. Drop contents of the downloaded folder '`cuda\`' directly into the install folder of CUDNN, default location is '`C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.2`', illustration:



Name	Date modified	Type	Size
bin	24/02/2021 10:17	File folder	
extras	24/02/2021 10:15	File folder	
include	24/02/2021 10:17	File folder	
lib	24/02/2021 10:15	File folder	
libnvvp	24/02/2021 10:15	File folder	
nvml	24/02/2021 10:15	File folder	
nvvm	24/02/2021 10:15	File folder	
Sanitizer	24/02/2021 10:15	File folder	
src	24/02/2021 10:15	File folder	
NVIDIA_SLA_cuDNN_Support.txt	27/10/2020 07:10	Text Document	18 KB

C:\Users\arno\AppData\Local\Temp\cudnn-11.2-windows-x64-v8.1.1.33.zip\cuda\

Name	Size	Packed Size	Modified	Created	Accessed
bin	1 163 610 1...	692 434 427			
include	196 253	37 508			
lib	7 634 680	573 110			
NVIDIA_SLA_cuDNN_Support.txt	21 355	7 979	2021-02-22...		

4 / 4 object(s) selected 1 171 462 400 1 163 610 112

Figure 9: [Screenshot] Illustration of CuDNN installation

5. Reboot required

Step 2: data prep

To prepare the data required to train the agents yourself it is just as simple to run a script.

In your preferred terminal, cd to dir: '`./Data`' and run

```
python DataCollector.py
```

This can take a while but will collect all available data for the list of indices specified in the code. By default, this should be the top 50 large cap of the US market.

To specify which symbols, you would like to collect you can do this in the code.

If you wish to update your data you just run the script again, this will collect all missing new data.

8.2 User manual

This document will be instructions on how to use this project.

Training

If you wish to train any agent for a specific amount of episodes you can do this in the bottom cell of ./main.py where you can adjust the number of episodes and uncomment which agents you wish to train.

While training, after every episode, both the model and the episode results will be saved. The model in ./Models/.h5 , and the data in ./Data/Training Records .

Visualizing

When you wish to visualize the date with graphs it is best you move the data into ./Data/Records, then the script ./Data/Graphing.py will be able to visualize this.

8.3 [Deleted] Socioeconomic Impact of Automated Trading

Trade Depersonalisation in Financial Markets [34]

In the old days of trading the main problem of stock exchange was the fact that trade was carried out at trade venues through direct communication of traders. All trades were executed by manual labour and decisions were largely uninformed and speculative and needed to be quick, instinctive decisions or the arisen opportunity might be missed.

The rise of Information Technology gave way to a better, faster approach of trading. Where the traders could instantly see the real-time prices of the market and instantaneously execute and fill orders by manner of automation. This was a paradigm shift in the volatility of market prices and the volume of trades.

In the further process of depersonalisation of the trading process, the logical evolution took place to completely cut out the middleman and centralise all stock information in a location trusted by all participants of the market. This immobilisation and decentralisation of traders greatly reduced costs and time required to trade, and opened up the market for far more people all over the world.

The digitalisation of the transfer of ownership of stocks also diminished the risk and time required for completion of the process. This also prevented the risk of losing or stealing financial instruments as the stocks were immaterialised.

However, even with the evolutionary use of ICT all decisions regarding trades were still made by traders themselves. So, the same problem of quick decisions and mistakes was still present in trading, caused by human nature itself.

Automated Trading Systems [34]

Traders realized that, if properly programmed, computer systems, being deprived of deconcentration and hesitation, were able to eliminate the problem of slow reaction introduced by human nature.

Algorithmic trading involves computer execution of sales transactions, in which computers use the parameters of a predefined algorithm to make a decision on a trading instrument, moment and quantity. In algorithmic trading, computer algorithms not only distribute orders, but also make decisions about the moment and amount of their execution.

With this, the definition of High Frequency Trading was offered in 2012 by the Technical Advisory Committee for Automated and High Frequency Trading (CFTC)

- Decision-making algorithm (initiating, generating, directing, or executing orders) for each individual transaction, with no human intervention.
- Low latency technology designed to minimize response times and make trading closer.
- Technology that establishes a fast connection, i.e. communication with the market in order to enter orders and transfer a large number of messages about orders, quotes or cancellation of order execution.

Algorithmic trading can comprise trading in shares, bonds, currencies, and goods. With the help of advanced and complex mathematical models for making decisions in automated systems, strict rules determine the optimal time for an order execution.

The last 20 years recorded a remarkable growth achieved by automated trading systems, which is evidenced by the fact that a half of the overall trading in global stock markets is done through these systems.

Application of AI into Automated Trading

For automation of trading a strategy needs to be predefined that, in detail, describes what the prerequisites are to open and close trades. To do this the developer of this algorithm needs to be confident in his strategy and implement fail safes to make sure no unforeseen situations occur where the algorithm makes large losses.

Since most strategies are based largely on numerical data, an AI system might be able to make the same, or even better, interpretations of the market as a trader would. However, with today's technology, the use of AI in trading is still near impossible to achieve.

[Relocated] Consequences on Socioeconomics

If and when an advanced enough AI system that can make the perfect trades would be available, a large problem would arise. Since the driving force of the economy in capitalism is the accumulation of more wealth, this AI would enable its 'user' to increase their portfolio vastly. This increase would be exponential and unmaintainable, practically collecting all wealth of the economy.

A similar problem is already in motion today with the problematic, unethical imbalance of wealth in the world. It is obvious the current economical system cannot maintain in the age of AI, thus requiring a drastic change before such systems could, or even should, be implemented.