Application and evaluation of deep reinforcement learning algorithms in simulated trading environment

VERONIKA CUCOROVÁ
cucorova@kth.se

October 17, 2020

Abstract

This paper investigates the applicability of algorithms previously tested in theoretical environments with unrealistic conditions. Namely, the algorithms investigated are deep reinforcement learning with feed forward neural networks and recurrent neural networks as these seem to attain the highest results in the recent research. The testing is done in real time on simulated trading platform Oanda using a free public API with the updates frequency set to fifteen seconds. This allows monitoring how well the algorithms handle time latency in placed orders and price updates, the differences between real ask and bid prices and transactions costs. In addition, the performance is evaluated not only in terms of profit but also in terms of Sharpe ratio, thus taking into account the risk level.

Contents

List of Acronyms and Abbreviations

For clarity some of the frequently used terms are abbreviated and a short list of them is given before presenting them in next sections.

ML	Machine Learning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
RRL	Recurrent Reinforcement Learning
LSTM	Long short-term memory (a recurrent layer)

1 Introduction

In general the problem of trading comes down to deciding at any given moment, based on available information from the market, whether to buy or sell given amount of stocks, currency, commodity, derivative contracts etc. This problem can be modelled as Partially Observable Markov Decision Process as only limited part of the market conditions are known to the agent or trader making the decision. If we were to model all of the complex states by the conditions of the market environment considering how much the conditions can vary, we would easily reach a state explosion. Moreover, considering the vast amount of possible actions (not just taking into whether to buy or sell but also after what time delta to do it based on currently observed conditions), while very few of these many possible actions result in a future profit, it is called sparse-reward space problem. Using a trained agent to make the decision while maximizing the profit, minimizing the risk and all that in the shortest possible time is the field of algorithmic trading.
Algorithmic trading has generally been used in the last decade by the market makers. These are companies whose main purpose is to provide liquidity to the market by continuously posting bid and ask orders thus minimizing the bid-ask spread, enabling competition and giving the possibility to investors to always buy and sell a product at the best possible price.
In the recent years, applying Machine Learning(ML) algorithms became accessible to people with less computational and financial resources and level of expertise thanks to the increasing power of personal machines, but also publicly accessible online trading platforms. As further elaborated in 2 the public research(outside of the market-making companies) done is mostly realized in sandbox environments, with historical data with the aim to prove the applicability of certain algorithm to the problem.

1.1 Background and rationale

This paper aims to put deep reinforcement learning(DRL) and recurrent reinforcement learning(RRL) algorithms into a more realistic scenario, where the training an running shall be done in the real time to lead a way towards democratization of algorithmic trading and true confirmation of the algorithms in real conditions. The context of this research is set in between the heavy applied research done by the market making companies and the research done by the computer and data science researchers testing their algorithms in the domain. This paper aims for a practical applicability of its results, meaning an individual with an average machine is able to build on top of this paper and run the implementations that will be openly accessible. On the other hand, this paper seeks to prove the recent advancements in ML algorithms. The goal is to implement one or more realistic trading methods for improving the training process when utilizing RRL or DRL within trading domain, determine it is possible to apply them in a described scenario or alternatively determine their ineffectiveness or unsuitability. The implemented code shall be open source in order to make it possible. As a result, the code can thereafter be used to develop a system or application which makes algorithmic trading available for any user.
Moreover, there is naturally great interest in building a working automated trading system and to author's knowledge, there is lack of resources that would on one hand have a theoretical base, but on the other hand provide a working example of their code, out of which even less provide any example on how to connect the proposed trading algorithms to real-time simulated market, which is a gap

that this paper aims to fill.
While various versions of Reinforcement learning(RL) algorithms have been applied in algorithmic trading, they encounter similar issues in the training part, due to the unpredictability of the market, such as the fact that the training data from the randomly selected period in the past, might not reflect the trends in the future. The project shall contribute to and push research within training and testing of DRL and RRL models in financial trading by further adapting the training process to the complex environment with few rewarding states for the agent, while also comparing the two algorithms to one another in terms of suitable metrics. It shall combine the approaches applied in recent research papers tackling similar problem, however, avoiding the unrealistic assumptions about the data and using more realistic evaluation function. Such analysis would set a base for future research, provide a resource for people entering the field of algorithmic trading to start with and build upon. A side goals is to challenge future researchers in the field to focus on testing their algorithms in real environments in order to close the gap between the results in theoretical research and real trading environment.

2 Theoretical framework and literature

There is clear interest in using RL in various domains that present extreme uncertainty due to their complexity and dynamic changing, one of which is the domain of algorithmic trading. First time reinforcement learning was proposed in the trading domain was in 1998[1]. Searching on google scholar for ”deep reinforcement learning stock market” returns roughly 30 thousand results, of which around 17 thousand were papers published in the last three years. The ratio is even bigger for RRL: ”recurrent reinforcement learning stock market” gives around 22 thousand results, while 17 thousand of those were published in the last three years. This clearly indicates there is a trend in using these methods for algorithmic trading, however, as multiple sources suggest, there is still a lot of improvement needed.
Review paper published in 2019 summarized many advantages and disadvantages of recent research, such as: ”*The results showed that deep reinforcement learning was not as successful in capturing the dynamic changes in the stock market as originally thought*.” or ”*...whilst the disadvantage was that the algorithm made the extremely simplifying assumption of not considering transaction costs at all and still achieving a low level of profitability.*” [2]:p.3, which suggests that the success and profitability of DRL algorithms in the field is not a solved issue yet.
Analyzing papers with given keywords shows there are several papers that merely aim to prove using a certain variation DRL algorithms is a viable way of learning a stock market policy, but do not aim to create a automated bot. Several others try to incorporate news streams using NLP into the trading model as a secondary input to improve the models, which is out of the scope of this research. Most papers concern themselves either with stocks or forex trading, rarely they examine the model in both. Huang [3] focuses on improving deep recurrent neural networks for trading and provide elaborate descriptions of the steps taken. However, the success of the model is measured only by cumulative profit not reflecting the risk taken. Moreover, the author states certain assumptions need to be made to make the proposed improvement of the training phase of their algorithm, which might be unrealistic. These approaches shall be combined and verified in general conditions and the most realistic scenarios in order to confirm they applicability.
Deng et al.[4] concern themselves mainly with improving the input data representation using so-called ”fuzzy representation”. They briefly mention testing the algorithm on both historical prices

of stocks, commodities, futures on commodities and S&P index but do not provide a comparison of performance under various conditions or how does the algorithm differ in them. The algorithm proposed is an improved version of DRL and RRL using ”task-aware backpropagation” built in keras. The same predefined amount of stocks is being sold or bought at each time unit, which is however not preventing the agent from buying or selling multiple times in a row. Algorithms is tested on real historical data and transaction costs are taken into account, but it is done offline with a downloaded database of historical prices. They identify a pitfall of the research in the fact that the financial markets are not stationary - meaning they change in the real time and therefore using an offline database of prices does not fully represent trading in the real conditions.
Despite the fact the application domain is different, there are a lot of advancements that cannot be ignored made in the area of robotics and e-sports using similar algorithms as in the financial domain. Some of the big contributors are Deep Mind and Open AI aiming to solve the sparse-reward training in a recent paper [5], but also in general by helping to organize e-sports competitions and providing systems and APIs to competitors, students and researchers. Similar successful attempts are made in the field of robotics, such as handling the sparse reward environment by auxiliary tasks and learning off-policy [6].

3 Problem

The aim of this research is to confirm the success of recent papers introducing DRL and RRL into the financial domain using just personal machine and real-time data retrieval and testing keeping the conditions as close as possible to trading in real market. Their performance is going to be compared in terms of profit and risk measures in various common settings of the trading environment. Moreover, if the theoretical implementations of RL and DRL can be applied in described context, the research will aim to improve training phase of the algorithms, which can benefit from application of sparse-reward exploration already used in other domains.

4 Research methodology

4.1 Data collection

The source of the data for this research is a public freely-available API providing time-series financial data from a foreign exchange. This API provides the possibility to access historical data for back-testing the models as well as the real-time access to the current prices, which is essential to simulate more realistic scenario.

For this purpose, the Oanda trading platform[7] was chosen due to satisfying the given requirements. Oanda provides various options of APIs, even within their practice account. The two main categories are Streaming and REST APIs, of which the REST was chosen to access historical data and Streaming API is used for online testing. Oanda lists v1 and v20 of their REST API, but v1 is discontinued since 2018. Thus, v20 is used, despite the lower amount of examples and literature. This shall make this project more relevant in the future, especially, because for now not that much content using the new version is available. The intention was to train the algorithms in offline conditions allowing much faster speed of training of the neural network, therefore historical data from the period of 31.10.2019 - 13.11.2019 with a tick every 15 seconds was downloaded.

Unfortunately, the data was not as clean as expected and sometimes contained much wider gaps between the ticks and did not contain any data over the weekends (which was expected because worldwide forex market is open roughly 24/5). In total this downloaded data contained 16126 rows (ticks) of which exactly 80% equaling to 12900 rows were used for training and the rest for quick offline backtesting. The downloaded file was in the form of invalid json which was first prepossessed using regular expression and then parsed using custom python function into csv format. Following features were extracted from the data, reflecting the fields of open-high-low-close chart: *complete,volume,time,b_o,b_h,b_l,b_c,a_o,a_h,a_l,a_c*. We have two typs of prices: *bid* representing the price for which we can sell and *ask* representing the price for which we can buy. The rule is that *ask* price is usually above *bid*. For each of them we have *open*, *high*, *low*, *close*, but we will mainly use *bid_close* and *ask_close* prices in the model.

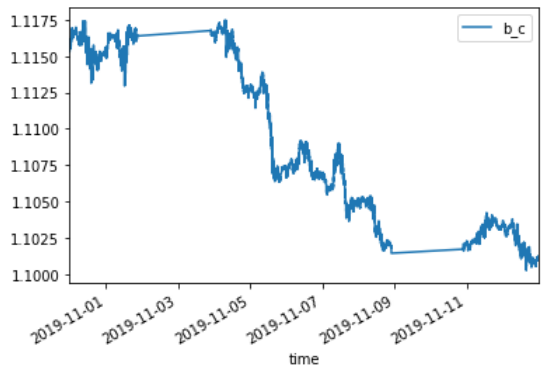

Figure 1: Development of bid close price in the training set. One can see the lacking data for a few days which correspond to the weekends.

Figure 2: Screenshot of open-high-low-close chart from oanda platform[7]. Each tick represents a time duration (5 seconds in this case) over which the price changed from *open* to *close* and reached its maximum at *high* and minimum at *low*. If the price went up, the tick color is green, otherwise red.

4.2 Software frameworks and hardware

The code written in the scope of this research shall be executed on a personal machine. Data was prepossessed using regular expressions in Notebook++ and Python using Jupyter notebook and Atom. Python libraries such as Numpy and Pandas were used for data manipulation and Keras for building models. Execution of the algorithms was partially done on a laptop with 8GB RAM and in remote Jupyter server environment both meeting the personal machine requirement set initially. A versioning tool shall be used to keep track over the progress in the code. After the end of the project the code will be publicly available at the repository `vcucu/forextrader_public` on github.

4.3 Tasks and methods

Systematic review papers[2] [8] and papers describing successful implementation of DRL and RRL algorithms[4] were analysed to reach a working implementation. This by itself is a challenging step, since statistically, it is non-trivial to develop a algorithmic trading engine and the requirement of profitability is not always realistic. This research aimed to go one step further and evaluate the models built in offline environment using a real-time online API. According to Meng and Khushi[2]

the two most common evaluation metrics are the rate of return and Sharpe ratio, out of which the latter was chosen in order to reflect the riskiness of investment in the algorithm performance.
There are many options regarding what is possible to trade on the market online. Besides foreign exchange currencies and stocks, more could have been taken into consideration such as trading derivatives on the top of the underlying stocks or trading cryptocurrencies. Derivatives value can be mathematically derived, but since this is the domain in which most market making companies compete, the entry requirements in order to be profitable are very high in terms of knowledge of the market and computing power. On the other hand, cryptocurrencies are not traded by many professional companies and therefore could provide interesting ground to establish a working system without that much competition, but do not have the risk lowering system. On top of these there are many more techniques that could be investigated in order to lower the risk, such as hedging, pair-trading the stocks or portfolio management. However this research focused on the forex trading. This was due to high interpretability of the prices, relative simplicity and also because many of the research papers examined in the theoretical framework were trading forex and therefore comparison could be made.
Arguably, good results in trading are achievable with less automized algoritms, where the human trader tunes the parameters in the real time just applying the knowledge of financial markets, following the news and developing trading strategies. With the author's lack of background in finance this is not a option and moreover the goal of this paper is to develop and improve the algorithms mentioned. To justify the choice of DRL and RRL models, it has to be mentioned the other viable option is using supervised learning to develop ideal strategy. However, that takes incomparably higher amount of effort and training iterations and might not even compare in terms of performance[3], hence the approach using recurrent learning algorithms will be taken. Out of the ML methods, DRL combines the standard RL with deep neural networks and is superior to the supervised learning and RL. Its self-learning process suits the ever-evolving market environment and sparse-reward space problems better, provides an automated way to create a trading policy and handles the extreme number of different states. Using recurrent neural networks also previous states in addition to the currently observed one can be taken into account in making decisions. On the other hand, DRL and RRL algorithms consume more computational resources compared to RL due to neural network training, which might make the usage of RL more realistic in environments with limited resources.

5 Results and Analysis

Based on the reasoning given in 4, following algorithms were implemented. Despite the focus on the aforementioned ML algorithms, following section also illustrates the algorithm called Momentum strategy that merely provides a simple rule-based approach to trading and allowed to check whether the interaction with the Oanda API works correctly before the more complex algorithms were designed to interact with it.

5.1 Momentum strategy trader

In order to provide a certain baseline performance and a default algorithm to run, a non-ML strategy was implemented. The time-series momentum strategy is in contrast with the buy-low sell-high

strategy but it is relatively simple approach and is based on exponential moving averages(EMA). The underlying assumption is that if stock price is moving in certain way then it will continue to do so.
Without going into too much detail that can be found in the implementation, it can be described by fluctuating between three states: one where stocks are sold, one where stocks are bough and where no stocks are being bought or sold. Depending on the current momentum taking into consideration preset number of time units, we calculate whether the momentum is upwards or downwards on average. Combination of the current state and the calculated momentum determines whether anything needs to happen and if so, whether the stocks shall be bought or sold. For simplicity, the amount of stocks to buy or sell is determined by one of the parametres beforehand. In some cases we buy or sell double the amount in order to even out the current state and adjust to the new trend.
Results of this model will not be analysed as it is not the purpose of this paper, but the code will be be available in the repository for future reference on how to build a simple auto-trader interacting with the API.

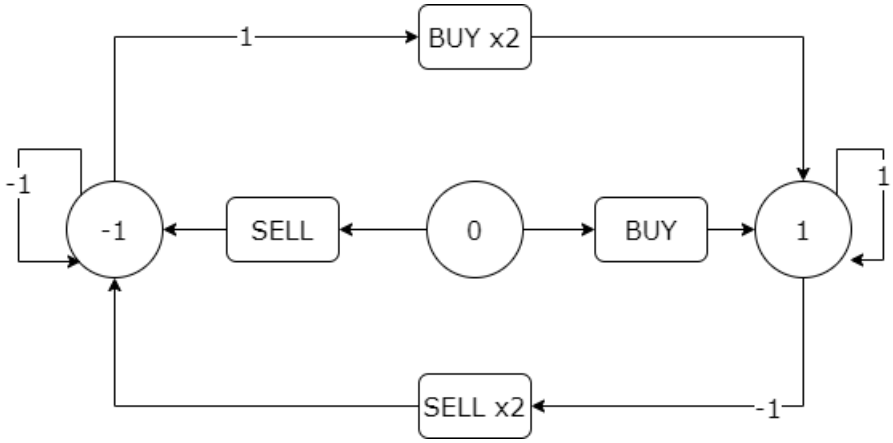

Figure 3: A simple state machine-like description of the momentum algorithm implementation. Initially, the state is 0 as the balance is 0. State -1 corresponds to currently selling(or having sold) and 1 corresponds to currently buying(or having bought) the stock. Transitions are marked either with -1 or 1, indicating whether the value of the stock is currently growing or declining. Transition that result into buying or selling are marked as such. When going from selling to buying and the other way around the order contains double the amount.

5.2 DRL and RRL autotrader

Based on the paper of Deng et al. [4] who built a successful model as described in 2, two reinforcement learning models are built, one using feed forward neural network and one having the model augmented with two long short-term memory layers(LSTM) maing the neural net recurrent and making it possible to take previous states into the account.

5.2.1 Neural Network architecture

The neural net had three output neurons being one of the three possibilities: 0 - sell, 1 - hold, 2 - buy and 6 inputs neurons on a continuous scale:

- *balance* - current amount of money on the account in euros. The initial investment was set to be 1000 after ranging the values between 1000 and 5000 which did not seem to have much influence on the model's performance.
- *stock owned* - current amount of dollars the model bought. This can be also a negative value if the model decides to sell dollars that it does not own resulting into so-called "short position", which might happen if the model expects the price to go down and expect to buy it back later for lower price. The model always buys or sells the same amount of dollars at once, in this case set to be 100 after experimentation with values between 100-500, which did not have a significant impact on the model.
- *bid price* - the price for which we can currently sell the dollars
- *ask price* - the price for which we can current buy the dollars
- *previous bid price* - the bid price from previous tick allowing the DRL model to determine whether the price is going down or up even though the neural net does not have the recurrent layer that RRL has
- *previous ask price* - the ask price from previous tick

For DRL model a simple network with two hidden dense layers, 120 neurons each, was created, after experimentation with various values between 24 and 256. Two layers are considered sufficient as such neural net can approximate any continuous function according the Universal Approximation Theorem. The hidden layers used ReLU activation function while the output layer used linear activation. The learning rate was set to 0.00015. The RRL model contained, again after experimentation with various values, two LSTM layers first one with 300 neurons and second one with 200, followed by a dense layer of 100 neurons. LSTM layer was preferred over standard recurrent layer thanks to its ability to preserve information about long-term dependencies in the network.p Same as before the dense layer used ReLu for activation function a output layer used linear. In both cases, MSE loss function was used with Adam optimizer. DRL algorithm was trained in 2000 epochs while RRL algorithm seemed to be balanced much sooner therefore it was terminated after 1000 epochs.

5.2.2 Replay memory

Experience replay is a technique used to store last n experiences and then training the network on a random subsample of those. Each experience contains information about initial state, action taken, reward and the following state. The reason for only training on this subsample is breaking down the correlation between the consecutive experiences which would result in inefficient training. The size of the memory was experimentally determined to be 3000 while training was done on a subsample of 100 experiences.

5.2.3 Training and testing

In order to perform the training efficiently and as fast as possible, it was performed on an offline dataset of historical data as described in 4.1. The testing was done in an environment simulating real trading environment using Oanda simulated trading API. Every tick was recorded and while

it contain many more feature fields compared to the historical data used for trading, these were not used in order to keep the data consistent with the training. The environment created allows to create two types of orders: market orders and limit orders. In contrast to trading offline, when the prices do not move in between the agent placing and order and the order being executed, in online environment these possible changes need to be dealt with. The market order deals with it by placing an order with undefined price, simply specifying the amount to buy or to sell. This order is executed as soon as it reaches the online market place regardless of whether the price moved in the meantime, possible buying it for more than the agent expects or selling it for less. The limit order contains not just the information about the amount to buy or sell but additionally also the minimal price to sell for or maximal price to buy for. The order is placed but it is not executed until the price reaches the specified range, possibly resulting in many unexecuted orders. Unfortunately, both options differ from the conditions in which the model was trained and there is not a straightforward way to simulate this possible latency with offline data. The models will be evaluated with the market order option, because using limit orders, the reward function would also need to take into account uncompleted orders.

5.3 Analysis

Both of the algorithms tested were tested in different time periods, in order to avoid them interfering with one another on the same trading account. This unfortunately forbid comparison of the algorithms on exactly the same data, but the nature of the data was still very similar so the results are comparable. The results very strictly negative, meaning the algorithms were not profitable regardless of the modifications in training and testing. The were tested in a range of a few hours each, multiple times. As stated in stated in methods section 4, Sharpe ratio was used for evaluation of the algorithms, where DRL algorithm achieved average of -52.088029 and RRL algorithm -18.835230. This would be a very bad result if the aim of this project was to create a profitable autotrading software, as the negative value means that the algorithm is less profitable than benchmark, which is equal to 5% in the formula for calculation. However, it can be concluded that that RRL performed consistently significantly better in terms of the Sharpe ratio.
More surprising than the non-profitability of the algorithms, however, is their behaviour. In fact, in a few hours of running, the agent rarely decided to perform the action of buying or selling. This might relate to the fact that during the training, the best performance was also only lowering the profit to 0, meaning the algorithm might have learned to simply not do anything (or very little). In the beginning of the training, all versions of both algorithms were losing hundreds of yours but towards the end they stopped improving and simply stagnated around losing a few cents. Therefore the non-profitability was expected in the testing environment as well, but it is still surprising the algorithm was not able to learn how to make profit and preferred to stagnate around not doing anything. There can be multiple hypothesis of what could possibly improve it as elaborated in section 6.

6 Discussion

Considering the author's background in computer science and the lack of background in finance, the experience can provide insight for the people coming from the same direction.
Firstly, there has been a underestimation of the knowledge necessary to start understanding the

domain. The research of definitions of financial terms and understanding API documentation is by far not enough to fully understand the underlying complex processes and to develop intuition about trading.

Another difficulty that was not sufficiently considered in the planning is the effort needed to familiarize oneself with the trading platform of choice, which would be needed even with the initial knowledge of stock markets in general was present. This includes both understanding their definitions and functions in the API as well as simply ”making it work” in real time, rather than using certain static dataset. This was however one of the major requirements set in the research and therefore could not be circumvented. This was eventually a successful effort and a functional implementation of autotrader that can run in real time was achieved.

At the same time, there was goal of building a system that would step in the direction of democratization of algorithmic trading systems, hence making it accessible to anyone with an average machine and access to internet making it possible for anyone to learn how they work and build one by themselves.

There are several reasons for which the algorithms might not have performed that well on the online trading platform. One, which is probably the most likely, is that the offline trading was simply not a good representation when it comes to trading online. Ideally we could have the model trained with the same environment using the real-time API, although that would require extensively longer amount of time to train the models. This approach could also make it possible to use many more features, in comparison to training on the historical data. Secondly, a solution might be also letting the model run longer in the online platform, as the training data also corresponded to a few days or weeks of trading and few hours of testing might not be enough to see whether the algorithms perform well. Thirdly, the fact that this implementation is not profitable, even though it is inspired by the successful versions from recent research papers, could be because their fine-tuned improvements to the algorithm were not implemented. Regardless of this, question remains, why is it that the algorithms were not able to make profit on the offline testing data, which should not suffer from the difference in testing and trading platform.

Because of this, it cannot really be concluded, whether the DRL and RRL algorithms are applicable in online simulated trading, since this implementation did not satisfy the offline profitability either. We can conclude the expected result that RRL handles the complex trading environment conditions slightly better, thanks to its ability to work with long term dependencies as well.

References

[1] J. Moody, L. Wu, Y. Liao, and M. Saffell, "Performance functions and reinforcement learning for trading systems and portfolios," *Journal of Forecasting*, vol. 17, no. 5-6, pp. 441–470, Dec 1998. doi: 10.1002/(SICI)1099-131X(1998090)17:5/6.441::AID-FOR707 3.0.CO;2-# Accessed: Nov 2019, Submitted: February 1997. Accepted and revised: July 1997. Updated: December 1997. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.8437&rep=rep1&type=pdf

[2] Meng; Lingze, "Reinforcement learning in financial markets," Date published:, Jul 2019, accessed: Nov 2019, publisher: Multidisciplinary Digital Publishing Institute. [Online]. Available: https://www.mdpi.com/2306-5729/4/3/110/htm

[3] C. Huang, "Financial trading as a game: A deep reinforcement learning approach," Date published:, July 2018, accessed: Nov 2019, publisher: arXiv, q-fin.TR, 1807.02787. [Online]. Available: https://arxiv.org/pdf/1807.02787.pdf

[4] D. Yue, B. Feng, K. Youyong, R. Zhiquan, and D. Qionghai, "Deep direct reinforcement learning for financial signal representation and trading," Date published:, Feb 2016, accessed: Nov 2019, publisher: Deep Direct Reinforcement Learning for Financial Signal Representation and Trading - IEEE Journals Magazine, doi: 10.1109/TNNLS.2016.2522401. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7407387/authors#authors

[5] M. Plappert, "Ingredients for robotics research," Date published:, Mar 2019, accessed: Nov 2019, publisher: OpenAI. [Online]. Available: https://openai.com/blog/ingredients-for-robotics-research/

[6] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Van de Wiele, V. Mnih, J. T. Springenberg, and N. Heess, "Learning by playing - solving sparse reward tasks from scratch," Date published:, Feb 2018, accessed: Nov 2019, publisher: arXiv. [Online]. Available: https://arxiv.org/abs/1802.10567

[7] Oanda trading platform. [Online]. Available: https://www1.oanda.com/

[8] "Key papers in deep rl," Date published:, 2018, accessed: Nov 2019, publisher: OpenAI, Revision 2ce0ee91. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/keypapers.html