Final Report - Applied Deep Learning

AlphaTrading, an agent for stock market prediction

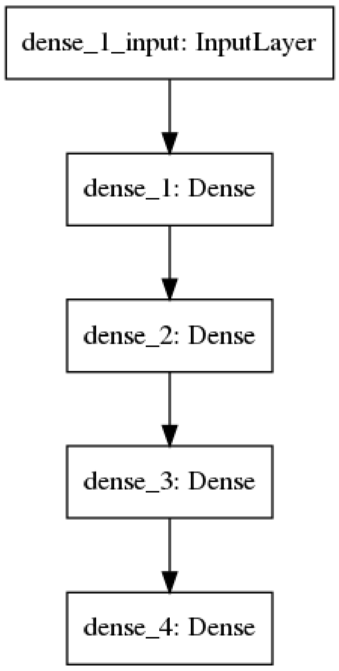
Problem Description

This project has the purpose to solve a problem, which has been haunting humanity since the beginning of time. Throughout the history many people had the dream of getting rich as fast as possible and thus came up with many different approaches. One idea which was very common by alchemists in the Middle Ages for example, was to create gold by mixing together different chemicals. But because humanity had a huge development since then, where we arrived in the age of computers and the Internet, this solution to wealth seems very outdated. The modern approach to this problem is not to create gold, but to beat the stock market. Even though this is not a real problem for humanity and its future developments, it would still be exciting to solve this problem since it would result in a very prosperous life.

Solution

Because of the hype of Deep Learning and its astounding results over the last couple of years, a similar approach for this problem was chosen. The focus for this project is especially on Deep Reinforcement Learning (DRL) with a **Deep Q Learning** agent. To be more precise, our solution to this problem is to create an agent, which trades on its own. The agent itself consists of a simple Deep Neural Network (different variations of dense layers), which allows it to predict the next tick depending on the **last 30**, or rather if it should buy or sell in the next time step. This agent will be trained and tested in a simulation environment of the stock market for **650 episodes**, where it will buy and sell over and over again and hopefully learn the needed patterns for correct prediction.

In the following, we illustrate the used simple Deep Neural Network which the agent uses to predict the next action.



Background Literature

As already mentioned, the idea is to create a Deep Q Network (DQN) which can trade financial products from tech-companies, such as Google or Apple. This topic seems to attract a great deal of attention, since there are dozens of scientific papers on sites like e.g. arXiv.org covering this problem. For this project we will use a simple DQN in combination with insights of the following four papers:

- Reinforcement Learning in Stock Trading
- Practical Deep Reinforcement Learning Approach forStock Trading
- Deep Reinforcement Learning in Financial Markets
- Deep Reinforcement Learning for Foreign Exchange Trading

These papers were mainly used to get an idea on how to preprocess financial data, design training- and testing datasets and define a benchmark to evaluate the performance of the implemented agent.

Implementation

To teach our agent the wished behavior, we had to create incentives in the form of rewards for the agent. Depending on its performance the reward will be positive or negative.

Reward is defined by the capability to correctly predict the direction of the stock price of the following day.
For example, if the price falls and the agent bet on falling prices (SHORT), it will receive a positive reward or if the price falls and the agent bet on rising prices (LONG), it will receive a negative reward, consisting of the price difference.

Profit is defined by the price difference between two time steps, where the agent changed its opinion about the trend, switching from LONG to SHORT or the other way around. This definition implies a trade, where the agent e.g. sells all its LONG-positions and buys as much SHORT-positions as possible, to not lose any money.

Error Metric
Every agent's structure, hyper parameters as well as the choice of scaling techniques, will be trained for 650 epochs on the trainings dataset (AAPL_train.csv, historical data from 31.12.2009 to 30.09.2019). Therefore, different approaches can be evaluated and compared using the average profit as well as the average reward of the last 50 epochs (600-650).

This metric is used to verify that the agent is actually making progress. Since this verification is only used on the trainings dataset, it does not give an estimation on the real-life performance on unseen data. Thus, a test suite was implemented to compare models on unseen data and compare them by earned profit and reward on a given test set (AAPL_test.csv, historical data from 15.08.2019 to 29.11.2019).

Error Metric Target
First benchmarks of the implemented agent were quite misleading, resulting in an average profit of **0.477** and an average reward of **3.568**.

Thus, we set our target to reach at least an average profit of **1**, which would mean
that the agent is at least profitable on the trainings set. After many iterations of adjusting hyper parameters and changing the model and still resulting in really bad and random performance.

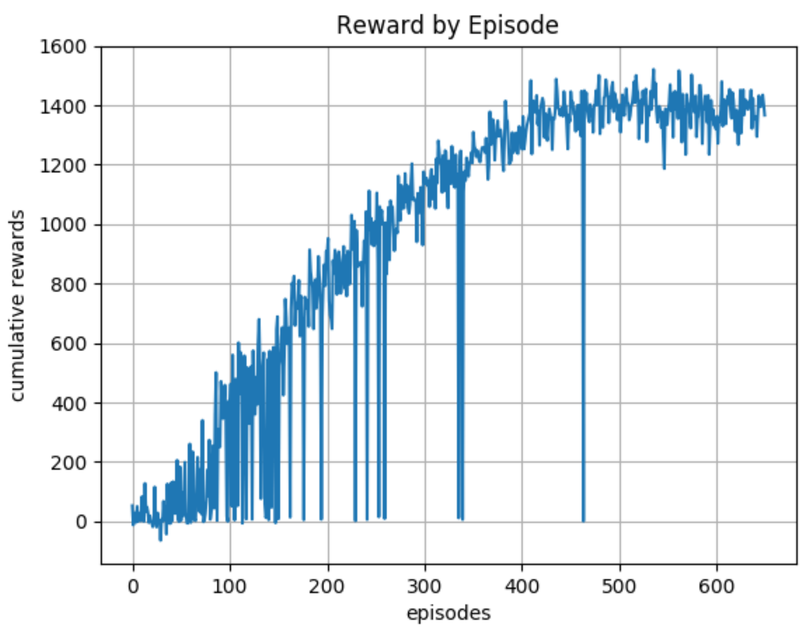
We took a closer look on the implementation of the used third-party environment, called AnyTrading. After a short observation, we defined my own calculations of reward and profit. This change finally helped the agent to make progress and actually learn patterns. Thus, earlier saved models and plots are not comparable to newer ones. After the change the target goal of 1 was quite simple to archive and is therefore not really representative.

Error Metric Achievement
The following table displays the performance results of the last 7 agent variations, which all performed better than the target of 1.

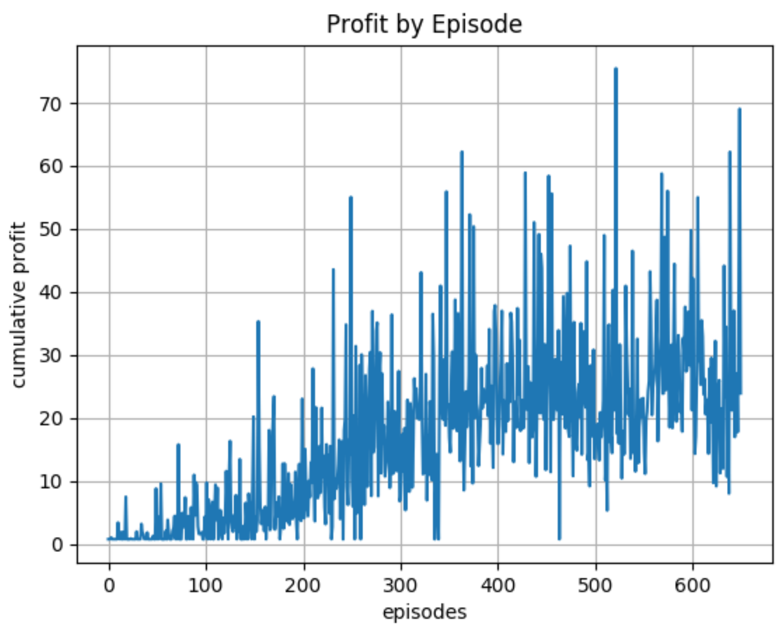
Average Profit	Average Reward
19.794	984.336
2.763	507.834
6.313	207.225
22.684	992.019
8.445	730.180
15.148	474.520
5.843	349.651

Results

The following plot shows the average reward by episode. One can clearly see, that the implemented agent has a huge learning curve, which fluctuates at the beginning, but gets more and more stable, till it finally converges at around 500 episodes.



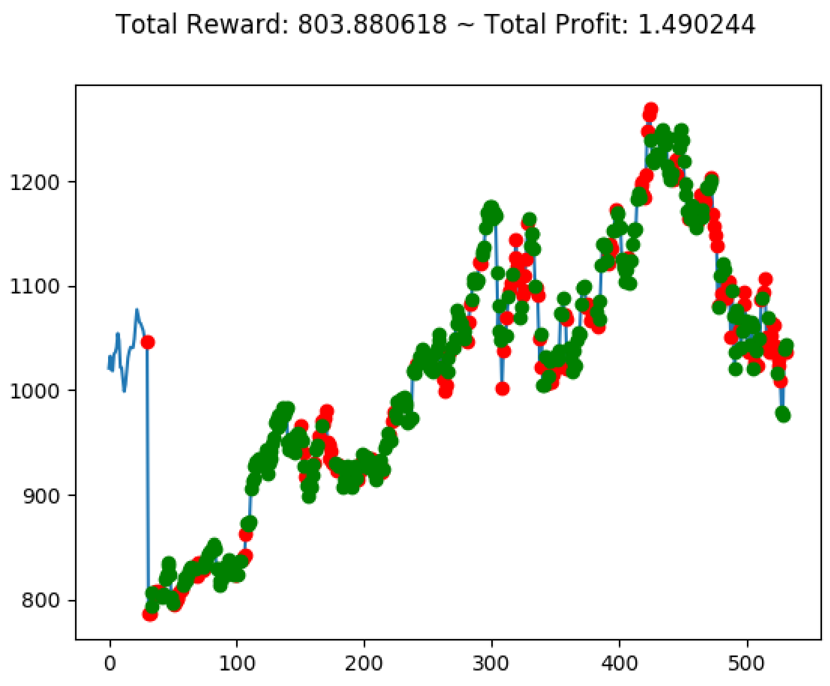
The progress is not as clear, when only focusing on the profit by episode. As one can see, there is still a steady rise noticeable, which grows less and less after 300 episodes. After that the average profit seems to fluctuate around 25, which means that the agent had grown its starting capital by the factor of 25 on average.



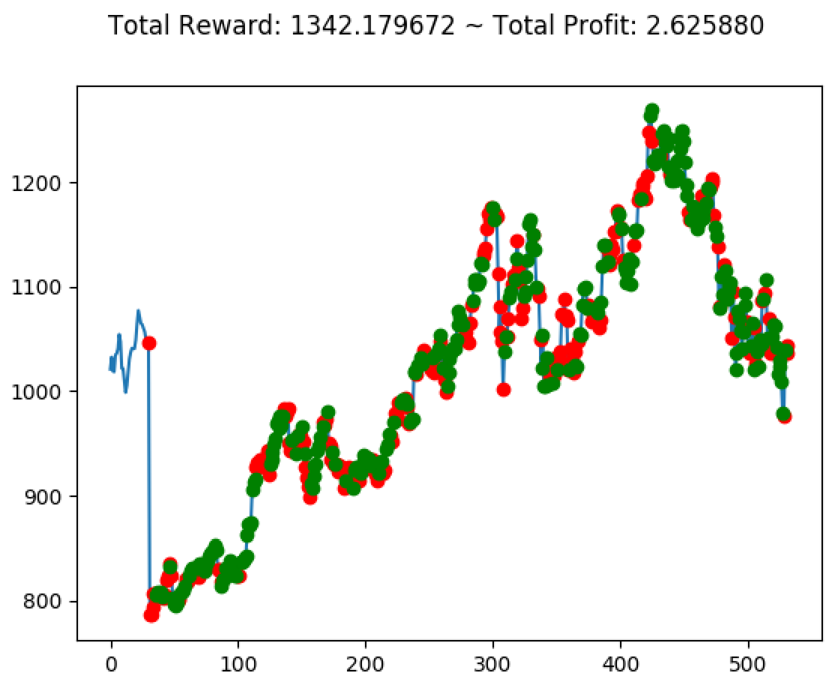
Since the evaluation of the agent on the trainings set is not as interesting and is only used to verify that the agent is actually learning something, the following plots will show the actual performance of the model on unseen data.

Green dots are time steps, where the agent decided to go **LONG** and therefore buy. **Red** dots are time steps, where the agent decided to go **SHORT** and therefore sell. For the market prediction the agent uses the previous 30 days as input and starts predicting from the 31st. Before prediction, the data was scaled with a standard scaler between -1 and 1 to remove the trend from the time series and therefore allow better generalization.

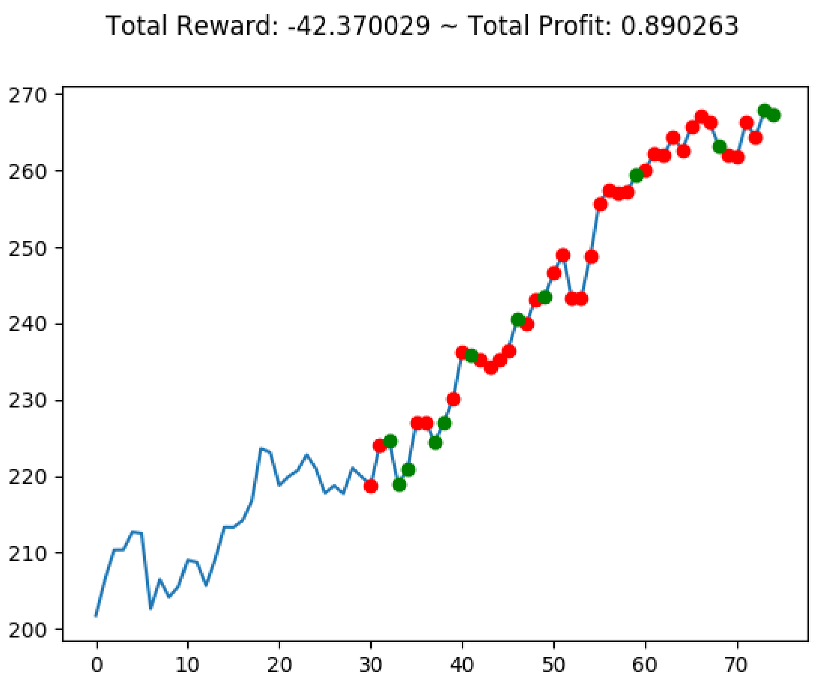
Plot of a model trained on Apple-Stocks and tested on Google-Stocks



Plot of a model trained on Google-Stocks, tested on Google-Stocks



Plot of a model trained on Google-Stocks, tested on Apple-Stocks



Overall one can conclude that the model's performance is not completely off since in two of the three cases above it was profitable. Only in the last example had the model made a loss of about ~11%. Despite this fact, this model is far away form being used in production. Many more observations have to be made, since false choices by the agent will lead to major losses.

Besides that, the prospects of a profitable agent seem realistic and achievable in the future, which shows that a Deep Learning approach works quite well for this type of problems. Since the model is quite simple and Deep Q Learning is also one of the less sophisticated DRL approaches, simple changes alone will offer huge potential for improvements in performance.

Another important aspect is that for those evaluation, the agent was only trained on one specific data set for around ten years. More evaluation has to be made with more general trained models of different markets and branches. One general model has already been trained and deployed in the web-application, but has not yet been evaluated.

Lastly, we have to bear in mind that the market has been steadily growing over the last decade. Therefore, it is not that difficult for such a solution to be profitable and thus we must also evaluate the model in a bearish market instead of a bullish one.

Take-Aways

First take-away for me was to not trust third-party tools and to spend more time researching. Finding errors in the implementation took me much time and was very frustrating. If I had found them earlier, I might have chosen my project differently. Time in general was a huge aspect as in hindsight it seems I mostly underestimated the difficulty of each task.

Another important lesson I learned is not to focus on using the best hardware. I spent many hours setting up Keras/Tensorflow and Cuda/Cudnn, to take fully advantage of my GPU but after I installed it, the GPU was barley used. This is due to the different training mechanism used in DRL, where the agent will only learn a certain amount of steps each episodes. Surly is it possible to create a multi-threaded environment, but this would go beyond this lecture.

Concerning the optimizers, stochastically gradient decent worked the best. Major improvements had been established by stopping early if an agent lost a certain amount of its cash balance as well as training the model four or more times per episode. Expanding the mini-batch-size as well as replay memory buffer enabled the agent to also develop a feeling for patterns which happen over a long time period. Dropout was also used but did not seem to make a major difference in performance.

Choice

In hindsight I would probably choose a different project, which uses more of the theory, we learned in the lecture, so I can apply my gathered knowledge also practical. It would have been really interesting for me to also try out CNN's as well as RNN's. Even tough there are many papers on this subject for financial market prediction as well, I would rather choose a project where the results are better comprehensible and do not seem as random as in stock market predictions. If I have to choose another DRL project I think I would choose a game, so when I play against the agent I would get a feeling for its performance. Nonetheless this project gave me a create insight in DRL and I learned a lot about software engineering and the process that comes with it, deciding on a project, implementing it and preparing a demo application.

In general, I would change my approach of selecting a suiting topic and spend more time deciding for an pre-implemented environment, since rewriting one and loosing all your data on the progression was really frustrating.

Work-Breakdown Structure

Individual Task	Time estimate	Time used
research topic and first draft	5h	13h
setting up the environment and acquiring the datasets	3h	7h
designing and building an appropriate network	15h	22h
fine-tuning and varying that network	15h	15h
building an application to present the results	6h	13h
writing the final report	5h	5h
preparing the presentation of the project	3h	2h

The overall estimation results in 52 hours of work, whereas I spent 77 hours resulting in 25 hours more workload. Most of my underestimations are due to the usage of many new technologies, which made it hard for me to correctly predict the workload.

Project Implementation References

Finally I will finish this report by mentioning my two implemented application for Applied Deep Learning.

The first one is a web application, which is solely used to view stock data and as well as for market prediction. The application starts with € 1000,- as an initial account balance. After an prediction has been issued, the balance as well as the taken actions of the trained model will be updated and shown accordingly.

The second application has multiple functions. Firstly, it is used to train new models on a single CSV file or multiple ones. Secondly, one can predict financial data directly over a CLI, which offers the same functionality as the first application. Lastly, it provides the backend for web application.