Plymouth University School of Computing, Electronics, and Mathematics

Deep-RL Advisor

BSc (Hons) Computer Science

PRCO304 Final Stage Computing Project

Aaron Wing 10556211
2018/2019

Abstract

This report documents the work taken in developing an automated trading solution, based on real pre-processed data. It starts with some background on the topic and how the solution was designed, followed by an overview of the implemented system. The document will also cover how the project was managed, the requirements and criteria for success.

Q-Learning is a popular tool used for 'Reinforcement Learning'. The software is able to decide what actions to take under different circumstances. This seems to be the perfect solution for navigating through different trading strategies. However, due to the complexity of the problem, a deep learning approach (Deep-Q) was adopted.

Despite multiple attempts at changing the strategy, the agent was unable to make substantial gains in profitability. The results are discussed in depth and potential problems are highlighted. Potential improvements to the project have been included, as well as alternative solutions that may prove to be more successful.

Contents

Introduction

Algorithmic trading is a highly popular branch of trading, where you follow an algorithm based on a set of indicators. A popular field in AI is neural networks and deep learning. In this project it is hoped to combine these fields to produce an agent that could advise a human in the best actions to take, to give an edge in the market, or trade entirely autonomously. A large part of an investors job is data analytics and that is a field that machine learning is very good at. The main objective of this project was to get a reinforcement agent to define its own algorithm, based on a neural network (NN) and a set of features passed to it.

For this project, the language chosen was Python as it is the fastest way to develop a neural network with the Keras API and TensorFlow as a backend. It also allows the network to be rapidly changed. Keras also allows the structure of the network to be customised in a short space of time.

Background research

The main source of research for the features is from an academic article (Kordos & Cwiok, 2011). In this paper they discuss the indicators used and the reasoning behind them. A subset of these were selected using different input parameters to allow the agent to run on longer timeframes (daily, weekly, and monthly). This was chosen to reduce noise and false signals and produce a more accurate output.

MATLAB was considered as an alternative technical solution. Data manipulation using MATLAB is simplified because it stores data in matrices. Python was chosen as the final solution due to its versatility and numerous libraries. There are dedicated libraries widely available to create neural networks. Other programming languages would be too cumbersome to use for this type of development. Keras, TensorFlow, Theano, and Microsoft cognitive toolkit are commonly used when developing machine learning solutions.

CCFp (complex common frames percent) is an indicator used in foreign exchange trading. This indicator takes in multiple currency prices and plots them in terms of strength against each other. The graph shows the strongest and weakest currencies at any given time period (semenych, 2007).

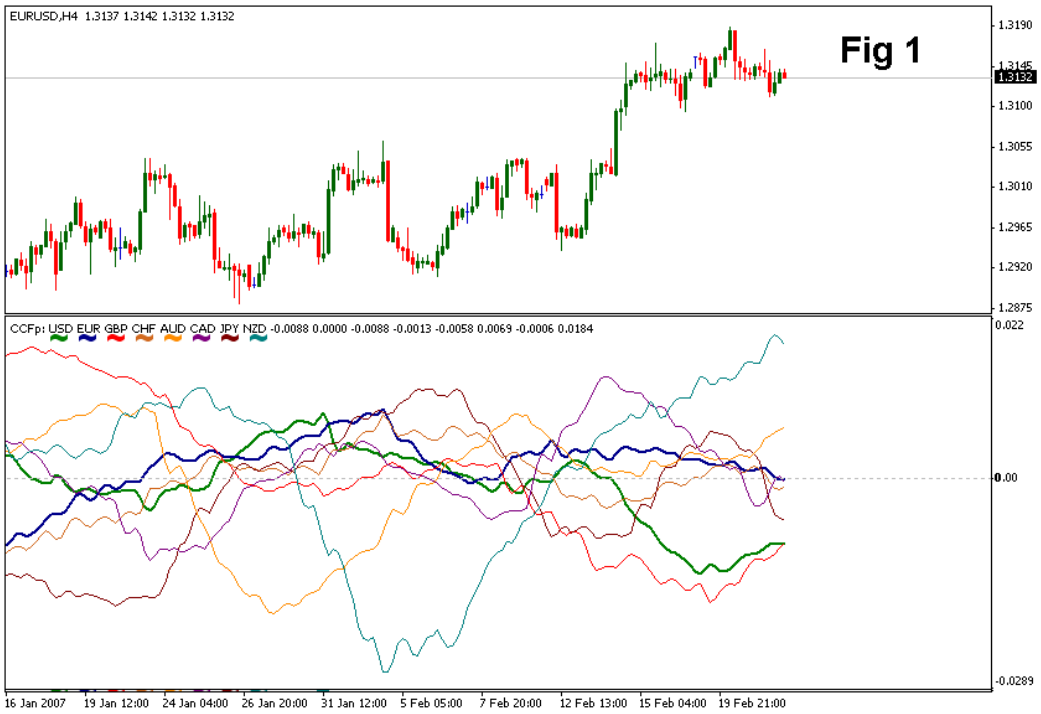


Figure 1 a graph showing candle sticks and the CCFp indicator - Taken from MetaTrader4 (semenych, 2007)

The graph above shows how the price of a currency differs from their scaled averages. In general, they always tend towards the zero line. This allows you to predict the direction of the price movement for each given currency. It also allows you to pick the strongest and weakest pairs to trade.

Traders will use a variety of different indicators to determine entry and exit points of a trade. This is called a trading strategy and is the basis for algorithmic trading. A combination of different indicators will filter out false buy and sell signals.

Requirements

The original requirements for this project was:

-Take in the current market price for a currency pair

-using a number of different price indicators, see how this correlates with the current price

-use machine learning to determine entry point of a buy or sell action

-Determine the exit point of when to close that buy or sell action

-Compare accuracy of the short, medium, and long-term timeframes.

The final requirement was deemed to be irrelevant as a daily data was the most stable and complete. The data that was retrieved for the minute timeframe often had gaps and was subject to high levels of noise.

Limitations

The main limitations of this project were

- The time that was given to complete the project.
- The computing power available. Due to the increasing complexity of the project, the time taken for training was growing exponentially. This meant that training the network on only a few hundred examples took half an hour.

Possible solutions to the project

Types of machine learning

Supervised

This type is good when you have a set of inputs and corresponding desired outputs. This would be suitable to predict the next n items in the sequence based on some features. It was not decided to pursue this as the level of complexity is too low. The solution would only require training a network based on input features.

Unsupervised

This learning is good for finding the underlying structure in the data. Unsupervised learning is more suited to be used in combination with supervised and reinforcement learning. I.e. identifying the best features to be inputted into a network.

Reinforcement

Best used when you have clearly defined states and goals and an environment from which you can generate a reward. This was the solution chosen as it was the most appropriate. There is a new branch called deep Q-learning that sounded ideal for this type of project.

Using reinforcement learning

This would be appropriate as you have a clear goal to make money, whilst minimising the losses and a set of states and actions from which to construct the environment. For example buy, sell, and hold. The idea of using deep reinforcement learning came from a paper published by google deep mind. In the paper they used deep reinforcement learning to play Atari games (Mnih, et al., 2013).

Using a multilayer perceptron

From previous studies, using a MLP has shown limited success due to the large amount of data required for training. The accuracy gets exponentially worse the further in time that you try to predict.

The idea is that you can use it to predict the next N opening prices in the sequence based on the previous closing price in the dataset. It is the most versatile network and can be used to approximate any function.

Using a LSTM

LSTM stands for a long short-term memory network, it is a combination of a long and short term memory network. It has the advantage of remembering events in the past without losing the reactivity to shorter term events. This is useful if you are trying to predict sequences.

Using a CNN

Convolutional neural networks are good for image recognition and should in theory, be good for recognising patterns in the forex market. When implemented they did not perform better than a LSTM and were more complex to pre-process.

Decision

The described problem is a POMDP partially observable Markov decision process, and thus Q-Learning can be used. This solution does have problems such as non-discrete input data which would need to be discretised if Q-learning is to be used. Alternatively, Deep Q-Learning can be used to avoid discretising the inputs.

To do this the most appropriate choice will be to use Python and Keras with TensorFlow as a back-end. A MLP was the best choice for the network as it was the simplest to implement and the most versatile. Although it will require more work to set up an artificial agent and the environment to be simulated, it should be able to create a more efficient algorithm than many current solutions.

As multiple currency pairs are planned to be used, the output will need to be a 4 by x, where x is the currency pair and the action of buy sell or hold. For input the agent will be supplied with features or indicators as they are called. Some of the most common indicators are MACD, EMA, SMA, ROC and RSI. If time allows, the project can be extended to calculate the CCFp of the pairs and treat them as a cluster.

Legal, social, ethical, and professional issues.

This problem has no legal issues as there is no real money that will be traded. A trading licence would have to be obtained to trade on a live market.

All types of trading come with financial risks. People can lose more money than they originally put in. If this system is used for live trading, any financial losses incurred not the responsibility of the programmer. Professionally there is no confidential or personal information pertained in the project. Any strategies developed that map to professionals are purely coincidental. It was decided it would be easier to not use a real client as there are a whole host of issues that go with using a real professional trader as all information pertaining to trades made would be their intellectual property or the property of the company they worked for.

Project Management

For this project a modified agile development was used. As there is not a real client the tasks and objectives were independently decided. Tasks to be worked on were set at the beginning of the sprint. Each sprint consisted of a week, this is shorter than the recommended time of two weeks. It suited the method of development as it was coded by only one person. To compensate for the short sprints it was focused on only getting one main feature per sprint to avoid being overloaded. This allowed the project to be shaped and adapt to the problems as new information came in. At the end of the sprint, performance and problems that occurred were evaluated. One of the largest problems was that while programming a wall was hit that required a redesign. This was manageable however, it did delay the time taken to complete the project and meant that the extra time allocated was required to be used. This method worked well for the duration of the project.

Data analysis

As stated in the background, the paper read had specific indicators. These were used on top of some of the most popular indicators (Bourne, 2018). Unfortunately there isn't a good way to test how well my data is analysed other than to build the system that will ultimately be using it.

The data was initially coming from an online data source that gave minute timeframe data. Unfortunately it did not end up being used as it contained large gaps. This was replaced by using investing.com which has a downloadable source with data from every day of the year. 2000-2018 was chosen as this was the most recent complete data and included over 4300 items. If more data was required, the source could go back to 1970 which would double the dataset.

Outcome

To produce a simulated trading environment based on real data and an agent that learns to trade in this environment. It was not feasible to implement the full set of currency pairs. The time taken to train the network on a single pair was over a day which would have extended the project beyond the time available. The program was not able to take advantage of parallelisation as the process was entirely sequential. Even if a GPU was available significant speedup would not be seen.

Data

The historical data was taken from investing.com, a reputable site hosting financial news and the latest trends and strategies. It was chosen to go this route as they had data back to 1970, farther than most data sources. This meant that daily data had to be used as it was the only time frame available.

The data contained:

- The open price, the ratio of one currency to another at the start of the interval.
- The close price, the ratio of one currency to another at the end of the interval.
- High, the highest ratio over the interval.
- Low the lowest ratio over the interval.

GBP-USD is a stable currency pair as it is backed by two large countries, which provided a large dataset for training.

Fig 2 (a graph of price per day for GBP-USD)

As can be seen in the figure above, the data is reversed when converted into a csv file. In addition the first 200 bars (days) had to be cut off as a 200-day average was used. The first 200 bars would not have an accurate representation of a 200 day average. It made it easier to work from front to back and thus the excel graph is reversed.

Features/Indicators.

SMA200

The simple moving average of the close price of the past 200 days. This gives a system a lookback as it is a summary of information of the past N days.

EMA15, 12, 26

Exponential moving average of the close price of the past 200 days. Exponential moving averages are similar to the simple moving average however they react much faster than the SMAs.

The formula for this is: the (open price - the ema of the previous day) * K+EMA of previous day | where k= 2/(days+1)

MACD

MACD stands for the moving average convergence divergence it is a measure of one moving average compared to another. as an indicator this is a graph of two lines. You can turn this into a numerical value of the distance of one average to the other buy taking one away from the other (hayes, 2019).

MACD = EMA12 - EMA15

Bollinger bands 200 day

They show when a unit has been bought too much (overbought) or oversold. The currency tends to bounce off of the upper band and lower band and this can be used to perform trades (investopedia, 2018).

The middle line in a Bollinger band is the simple moving average of a period. Because a slow 200-day band is used the 200-day SMA was already calculated.

The other two bands are the 200 day exponential moving average -+ the standard deviation for those 200 days. (stockschool, unknown) and supported by (investopedia, 2018).

Stochastic

This indicator displays two lines which vary between values of 1 to 100. It tells you when a stock is over sold or overbought. When the blue reaches the bottom threshold (20) it is oversold and when the red reaches the top threshold (80) it is overbought (babypips, unknown). This can be used in conjunction with the Bollinger bands to confirm a signal. If they are both showing the same thing then it is more likely for the signal to be true. To calculate it the formula is.

```
%K = (Current Close - Lowest Low)/(Highest High - Lowest Low) * 100
%D = 3-day SMA of %K

Lowest Low = lowest low for the look-back period
Highest High = highest high for the look-back period
%K is multiplied by 100 to move the decimal point two places
```

Taken from (stockcharts, n.d.)

All the data was pre-processed in an excel document. This was chosen as the most appropriate solution to this problem as I would only have to process this data once and it would always result in the same data. By doing the pre-processing in excel, the performance was increased as it does not need to be calculated every time you load the program. Doing so would be computationally expensive and inefficient. Once the features had been extracted the earliest 200 bars were cut off and it was flipped in time order and exported as a csv file for the program.

When imported into the program it separated the data into training and testing sets of 8:2 this was done to prevent overfitting. The data, once normalised was

When imported into the program the data was normalised and split into training and testing sets. This was done to prevent overfitting. The normalisation used was minmax per column. Other types of normalisation are L1 which is the sum of the absolute values. Or L2 which is the Euclidian length. Euclidian length prevents overfitting while the L1 norm tends to shrink less important features values. As I know the data's features are important, 'max' was chosen.

Environment

The first part of the project was to create a simulated trading environment to avoid to placing real trades. To build the environment we need to import the data and step through it depending on the actions that the agent can take.

In the system there were 4 actions available:

- Buy, to trade one currency for another reducing account value in hopes that it will increase in value in the future.
- Sell, to trade account balance for simulated currency in hopes it will decrease in value.
- Hold to do nothing for the day.
- Close, to close the current trade and increase the account balance.

The formula for calculating the profit of a trade is the difference in pips * amount you bought or sold. you also get your initial amount back leading to:

$$balance = balance + Amount\ traded * (1 + difference\ of\ pips).$$

A large balance of 1,000,000 was chosen so that it could learn without having to worry about running out of money. If this were to occur, then it would not learn anything beneficial and may learn action-pairs incorrectly. Only inputting the current information would give the agent feedback if a trade were active or the direction they were placed. This information is all stored in the environment however, this is the first major problem of the system. Having this information gives better oversight to the network, it does mean each state has to be calculated during run-time. This means that the whole system cant be parallelised to any significant speed up. As a result, each training run on around 4,000 items with 100 episodes takes around 4 hours.

Seeing as this is something that can't be overcome, to make it faster some of the repeated features are forgotten. The items that don't get sent to the network are the highest price, lowest price, EMA12, EMA15, and EMA26. The EMAs are all used in the MACD, while highs and lows are used in the stochastic. Once dropped the data table is normalized between 0 and 1 and an extra feature is calculated during the run. This feature is whether a trade is open and the direction.

It is represented by:

- -1 no trade open
- 0 sell trade placed
- 1 buy trade placed

There is a lot of difficulty in designing a reward algorithm that works well. Ultimately the agent will only learn to optimise its actions depending on the reward it gets.

The reward was changed and tested; it was found to work best with:

When closing a trade, the reward is the change in pips If there was a trade open and a small negative reward if there was no trade open. This makes the reward invariant of the amount of money made no matter the initial investment when making a trade.

When selling and buying, a small positive reward is given for opening a trade, and a negative one if there was already a open . This encourages the agent to make trades but only if there is no trade open.

When holding, the reward was the unrealised profit/loss. This is the money you would get if you closed the trade at this instance. The actual formula was tanh(Difference in pips) *0.001. This would make the value always very small and to a maximum of ±0.001. In general, this worked well when it was removed the agent performed poorly as is discussed in the results section.

The whole "state" in our case, is the normalised data of:

'Close', 'Open', 'SMA200', 'MACD', 'Bollinger_band_upper_3sd_200', 'bollinger_band_lower_3sd_200', 'StochK', 'StockD', 'Direction'

The direction is added in place when you go to retrieve the column as each action changes this value.

Deep Q-learning

Reinforcement learning was used for the main theme of the project, and was started by coding a single network to represent the Q-network. This network was trained using online learning as it ran through the data. The Q-Network used was an MLP made up of 2 hidden layers of width of 500 (Qnet = (12/500/500/4)). The activation function used for the hidden layers is Re-LU with a final linear output. The network itself doesn't have to be very deep as it only needs to store the Q function, not extract features. Re-LU was chosen as it is the most common type of activation function used today over others like tanh and sigmoid, as they are used for classification.

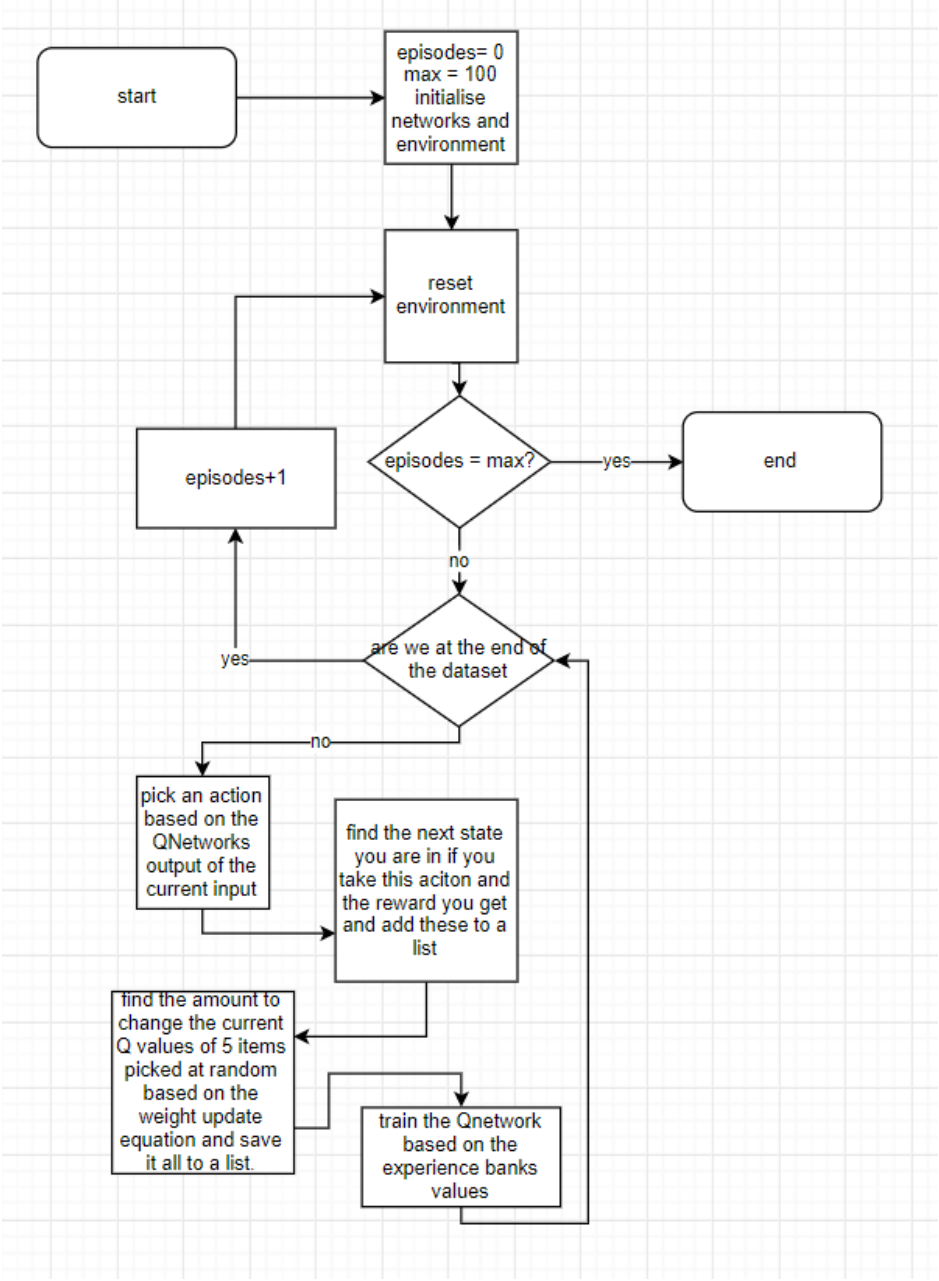


Fig 3 a high-level flow chart showing the structure of the project not including the environment.

There also needs to be a second network that is a replica of the first. This network is to be fixed in time and named the TD network. At the end of the episode the TD-network is updated with the Q-networks weights to synchronise them. The TD network is used by the Q-network to predict the next reward. Using the second network stabilises the learning as one change to the Q-network early in the dataset would snowball down and cause large changes later if you were to only use the Q-network. This was discovered first-hand as it made the graph of account balance look like random static pre implementation.

Once this was fixed, to improve performance, as per the Atari paper experience replay was implemented. Experience replay the technique of training the network on a bank of past "experiences". To do this the state, action, reward, and next state need to be saved to a list and trained on a set of N random items from this list. The system deviated from this, the first item that was trained on was always the current state action to prevent missing items from training. The advantage of using this is that If you

happened to do a bad action it would trickle backwards and make you less likely to do subsequent actions that were still beneficial. By essentially randomising the order in which you train it prevents this as actions can still be represented as good. When the network learns via experience replay it's also able to save these good and bad actions and re-use them to learn in the future, so it doesn't just forget them.

For training an adaptive epsilon greedy approach for action selection was used. Epsilon greedy is where you pick either a random action or the highest action value in your Q-table. Usually you would want the agent to pick the best action it can however, if it did this every time it would miss out on potentially better actions. This is balanced by picking the best action most of the time but occasionally exploring. For the greedy chance, the chance to pick the best action. The formula for this was:

$$N \in \mathbb{Z}^+ \qquad \boldsymbol{max}(N) + \mathbf{1} = \boldsymbol{m}$$

$$P(N) = \begin{cases} N \leq 91 & 0.7 + 0.305 * \log_m(N) \\ else & 1 \end{cases}$$

This gave me a curve that looks like the following:

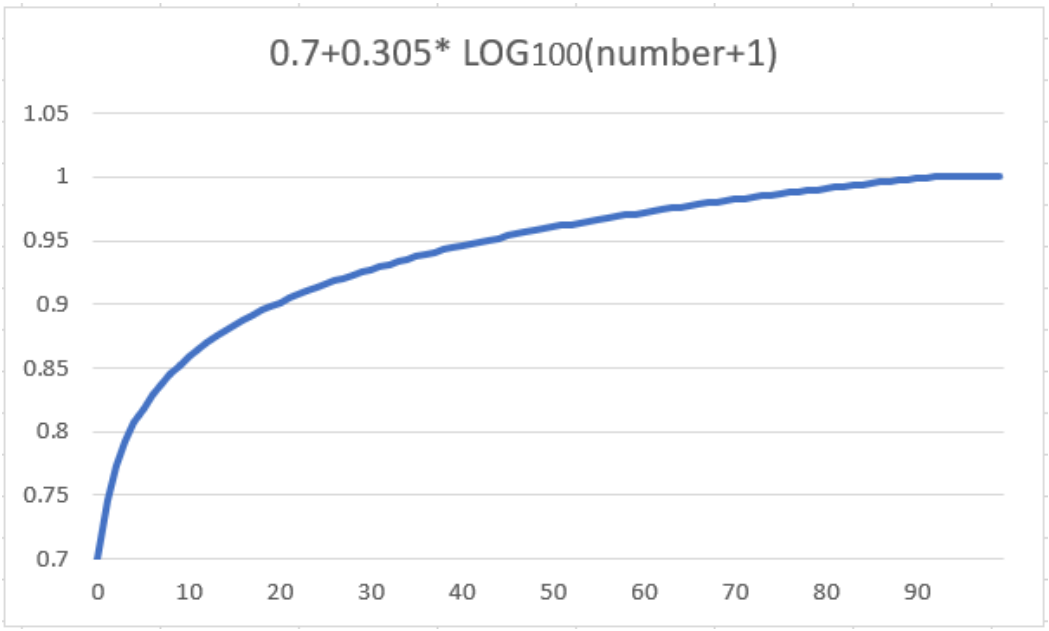


Fig 4 a graph of the chance of picking a greedy action.

This gives a good balance as it will take more random actions in the beginning and less towards the end. This was chosen over an exponential function as it gives more time for the agents' actions to even out and stabilize. It should be noted that this function caps at 1 as there can't be a greater chance than 1.

Learning rates are tricky for this as there are 2 different weights. The first is the weight of the Q-learning who's formula is:

$$Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma\, maxQ(s',a) - Q(s,a)]$$

Q(s, a) is the current Q-value for a given state, s and action, a

Alpha is the learning rate. The learning rate has to be small not to overshoot its optimal value. however it has to be large enough as the other learning rate is multiplied causing the overall learning rate to be small. Smaller values take a longer time to train to a optimal result.

R(s, a) is the reward you get for moving into the next state

γ Epsilon is the discount factor. This is used to weight how salient the future rewards are higher being more important. This is in itself a balancing act as if you don't weight future rewards too low otherwise the agent would sell as soon as it could. If this value is too high then it would only hold on to trades.

maxQ(s', a) is the value in the Q-column of the next sates best (highest) action.

The environment also needs to give a reward for Q–Learning to work. It also needs to be able to find the next state without actually performing the action. To find the next state it has to find the action it is about to do. Once it has that action it can simulate the outcome of that to provide the reward for moving into the next state and the state itself.

The second weight is the weight adjustment of the network. This is controlled using the Adam optimizer. This optimiser uses the uncentered variance to adapt the changes in weights.

Performance metrics were simple as the system already had the rewards stored as well as the final balances. The only thing that had to be added was the ability to store the loss while training. This was done in the exact same way as the other metrics, in a list The losses for the episodes were summed and added to the list to be graphed. One problem encountered with the system was that it would not give the actual account balance at the end of the episode. This was found to be the result of there still being ongoing trades. This was solved by putting a final sell action that was not trained on at the end of the sequence.

Results

The graphs produced contained the amount of money the agent had , the total loss , and the total reward at the end of the episode. This visually gives a good idea about how well the system works. The reward itself Is not directly proportional to the amount of money the agent gets. The reward algorithm also tries to prevent actions that do nothing or that the agent can't take. An example of this would be selling when a trade is open or closing a trade when there is none open. As the graph shows it does optimise reward, and account balance does generally hover well above 0. Although these results are not quite what was hoped to be achieved, the results are not a total failure. The documented results act as proof of concept that this type of learning is not lost and further development could yield promising results.

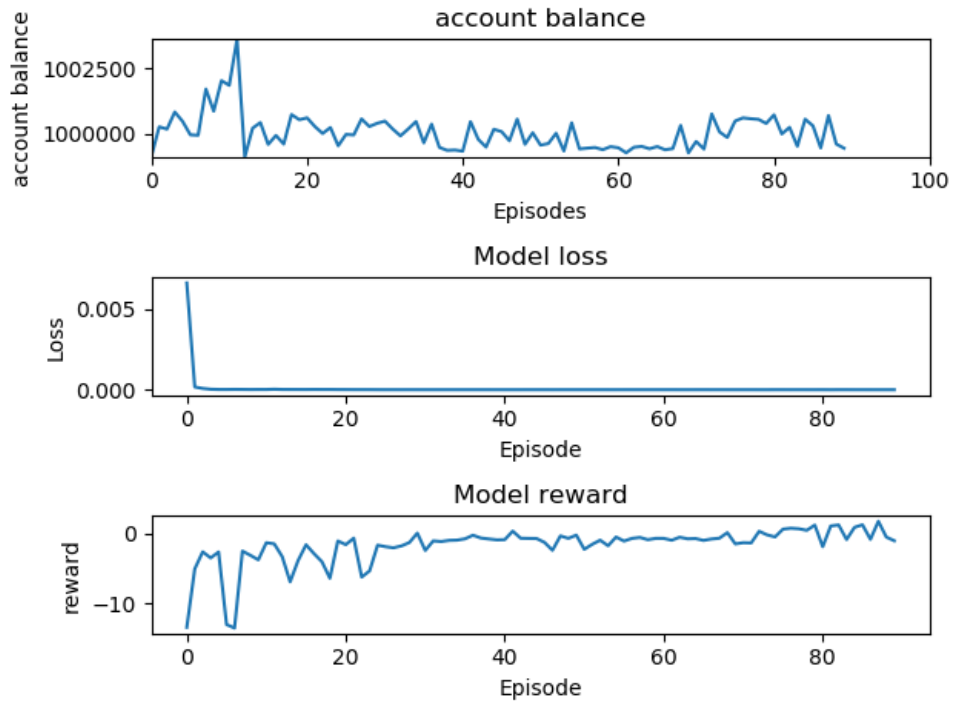


Fig 5. Graphs showing the result of a training run of the default reward algorithm documented above.

The network did generally not make money unless you chose it to specialise over a small dataset or have far less episodes. Over larger datasets the final account balance would always taper back to 0. However it would always bounce above and below the value. As the graph above shows the training worked well overall. the amount of reward gained increased, and and the amount of loss decreased. As it started, the agent was able to make money. Something happened at near episode 15 whereby it stopped making money. A tweak to the reward algorithm might help as it would more truly equate the reward to the amount of money made. The reason for the inequality of reward to balance would be it holding most of the time. Holding would give you a reward of 0.1% of the reward of selling maximum. This was done to increase the chance of it selling at a good time. When removed it produced a more erratic graph and it didn't work quite as well.

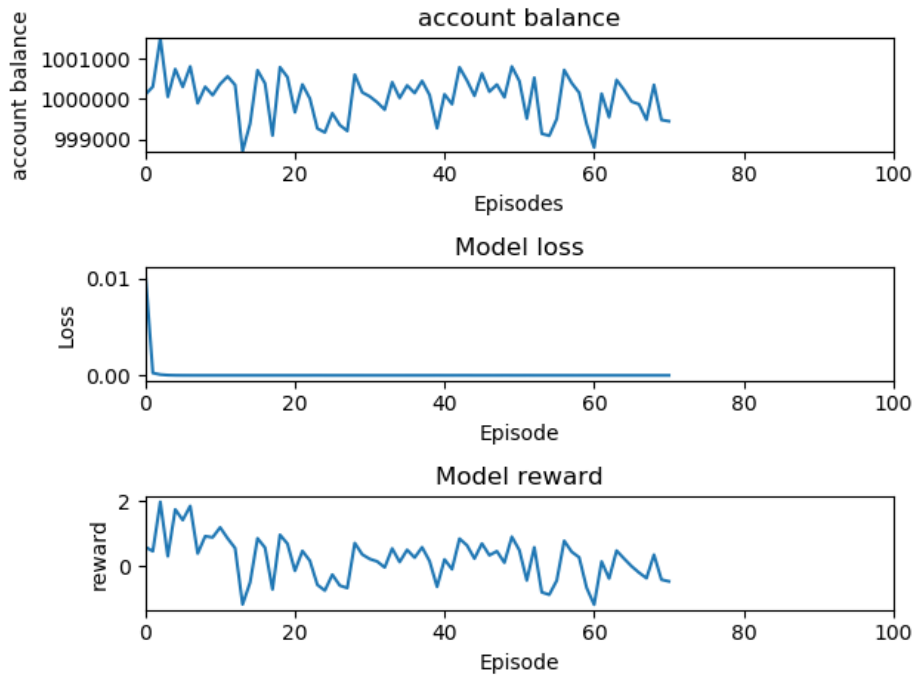


Fig 6. A graph showing rewards with the negative rewards removed other than that gained through trading.

In this test the reward algorithm was modified to only include positive rewards for the buy and sell actions. It also included no negative for closing without a trade. As the graph shows the agent still learns. However, the reward is harder to maximise as there is no way to represent which actions perform the best. The decision was made to end it early as it takes hours for a result that was evidently not working as intended. The graph in this instance is very erratic and doesn't manage to maximise reward. The agent in this case is not able to find a connection of placing trades when one is already open and closing when not open. As a result, it performs far worse than that of the original function.

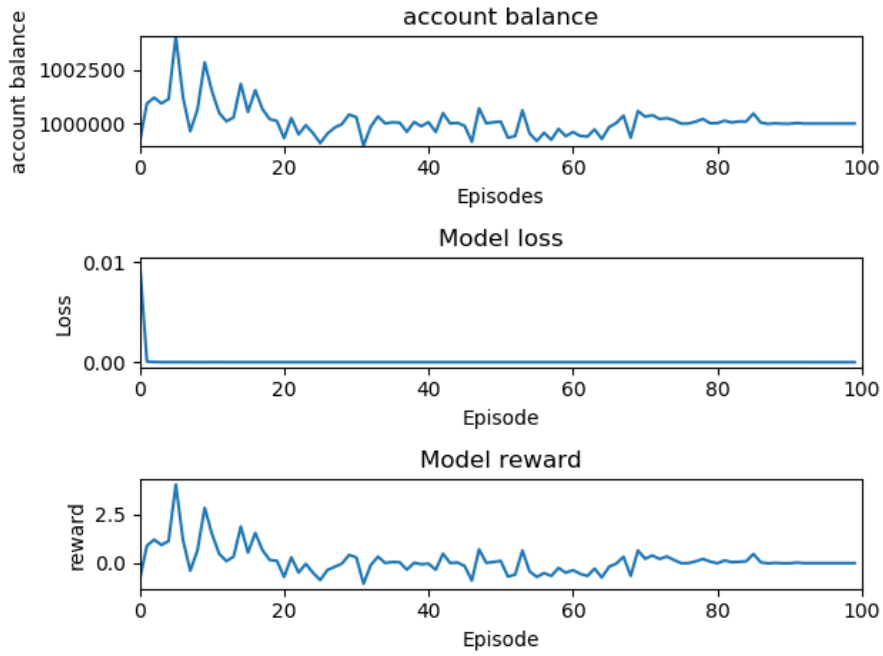


Fig 7 a graph showing the performance of reward only when the agent closes a trade.

The next test performed was to change the reward algorithm, so the only reward was that of closing a trade. Unfortunately, this method caused the agent to only hold and never place trades. This is understandable as buying and selling would lead to being able to close and that could potentially lead to a negative result. This would backpropagate through the system as the discount factor is 0.7.

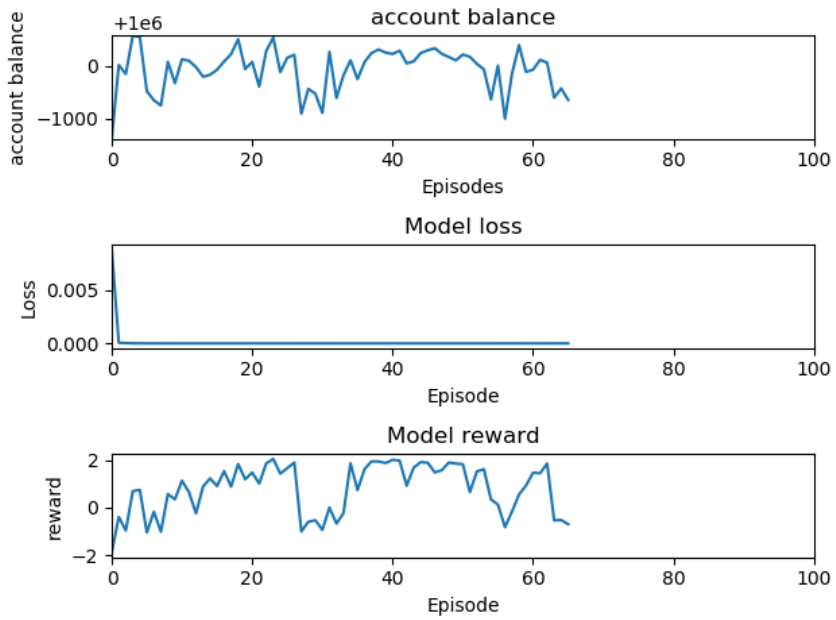


Fig 8 a graph showing the results of training on a modified reward algorithm function.

Finally, the reward algorithm was modified to switch the position of a trade when the opposite signal was received. To do this, if there was a trade open, the environment would close the current trade, take the profit and open the opposing position.

This resulted is a more erratic graph, although it was able to optimise the reward it received. The changes are likely caused due to the random actions now being far more sensitive. It would be able to perform actions it usually could not.

Overall the best results were attained when the agent recieves negative and positive rewards and was not allowed to switch the position of a trade without first closing.

Post-mortem

All requirements have been satisfied.

-Take in the current market price for a currency pair

This has been completed and expanded upon as its not just the market price multiple other features are taken into consideration.

-using a number of different price indicators, see how this correlates with the current price

This has been fulfilled. While these indicators may not give the agent great insight, it is still able to make both profit and loss.

-use machine learning to determine entry point of a buy or sell action

-Determine the exit point of when to close that buy or sell action

These two have been bundled together. It did not find the best actions to take with regards to the data. This however was a partial success as it learnt how to optimise the reward gained. A better fitting reward algorithm would help it greatly in this respect. As would more testing with different inputs.

Q-Learning worked flawlessly and is still a viable option for this problem. It still would be a good solution if more time were to be allocated to the project as there only needs the reward and data to be tested with. Further research into this topic of machine learning should be done. Unfortunately, CCFp wasn't able to be implemented the as there wasn't enough time to add multiple currencies. Adding this would potentially make the whole system perform far better as it would be able to compare actual value against many other currencies. It would also further make it invariant of the market price as the output of the system is simply a percentage. The technologies used were perfect as they functioned as intended although they may have taken a lot of getting used to as well as some difficulty installing. If this was to be done again Keras and TensorFlow would be the best option for it. Python once learnt worked flawlessly and worked well with my programming style. It allowed me to change most aspects of the program when needed. The program has been designed to be changed so most of the parameters such as the data, the networks depth and all of the nodes can be changed with ease. Personally, the only thing that would need changing would be the data and the reward function. These areas have proven difficult and have seriously limited the learning of the agent. In terms of my own performance it is felt I lost interest in the project when it was almost finished and was not working as expected. I also got locked on single problems. This was fixed by taking a break and coming back to it with a fresh pair of eyes. This was especially helpful when the problem was something simple like using the wrong letter in a loop.

Conclusions

Overall the method of using deep Q-Learning worked well and is a promising start for further study. A good result was achieved when giving the agent rewards both positive and negative on all actions. Epsilon greedy adaptive worked as a good balance when starting at a lower chance to begin with, and raising it as the episodes progress. Although the agent may not have functioned as well as intended, the data and reward can be changed to produce a better result if more time were to be allocated. If the project were to be done again, an LSTM would be suggested as this network could provide valuable lookback and a memory of events in the past. Potentially this could improve the performance of the agent. It was expected that the features extracted would have contained enough of the past information however they only provided an average performance. To improve this, one could filter the current set of indicators to provide stronger signals or add more indicators. Generally more experimentation would be required.

Alternatively, this could have been solved by firstly coding the algorithm for trading. Then getting the network to only optimise the parameters rather than choose actions. Q-learning was a good choice for action selection in sequential data as it was able to learn patterns to maximise reward. If more time were to be allocated to this project a separate testing section would be added. This would simply be re-running the program without training on the data as this would give a better idea on how it would perform in a real-life situation. Another addition if more time was allocated would be to make the program a command line application. This would make set up information such as training, testing, learning rates, and network sizes easily fed application. This would also make using a separate training and testing sections trivial.

References

babypips, unknown. *babypips.* [Online]
Available at: https://www.babypips.com/learn/forex/stochastic
[Accessed 2019].

Bourne, A., 2018. *Investopedia.* [Online]
Available at: https://www.investopedia.com/articles/active-trading/011815/top-technical-indicators-rookie-traders.asp
[Accessed 02 2019].

hayes, a., 2019. *investopedia MACD.* [Online]
Available at: https://www.investopedia.com/terms/m/macd.asp
[Accessed 2019].

investopedia, 2018. *investopedia bollinger bands.* [Online]
Available at: https://www.investopedia.com/articles/technical/102201.asp
[Accessed 2019].

investopedia, 2018. *understanding bollinger bands.* [Online]
Available at: https://www.investopedia.com/video/play/understanding-bollinger-bands/
[Accessed 2019].

Kordos, M. & Cwiok, A., 2011. *A New Approach to Neural Network based.* s.l., ResearchGate.

Mnih, V. et al., 2013. *Playing Atari with Deep Reinforcement Learning,* s.l.: Deepmind Technologies.

semenych, s., 2007. *MQL community.* [Online]
Available at: https://www.mql5.com/en/articles/1464
[Accessed 2019].

stockcharts, n.d. *stockcharts.* [Online]
Available at:
https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:stochastic_oscillator_fast_slow_and_full
[Accessed 2019].

stockschool, unknown. *stock school bollinger bands.* [Online]
Available at: https://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:bollinger_bands
[Accessed 2019].

Word count 5900

User manual

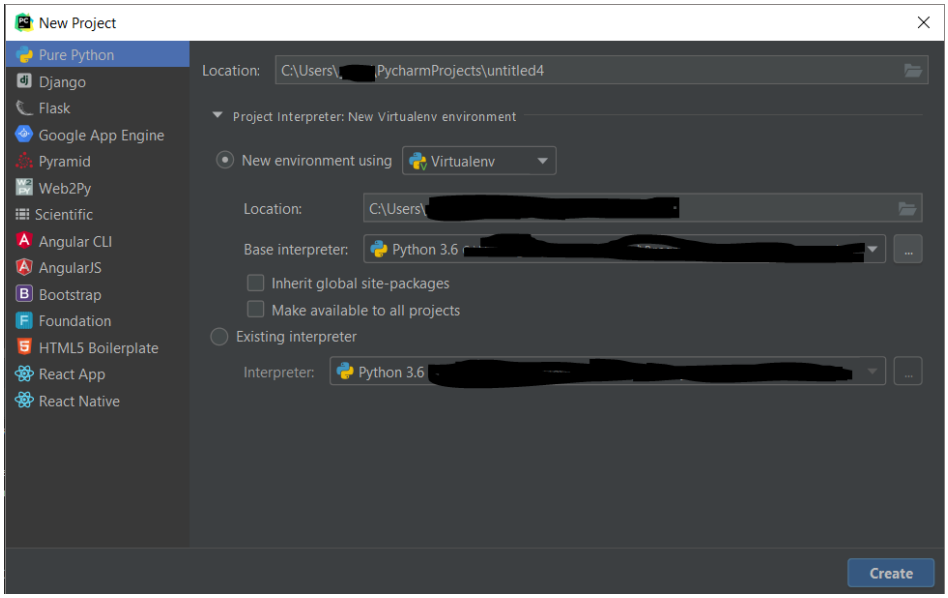
To install the project, you need to firstly install GitHub and python 3.6.7. This can be any client that you wish as a recommendation use PyCharm by JetBrains.

To set up the GitHub repository make the directory that you wish to store it in and clone the repository.

To install Keras and TensorFlow you need to use the command [sudo pip install keras] or if you wish you can install it in an environment without the sudo.

To install TensorFlow you can use [pip install tensorflow] if you choose not to use a virtual environment then follow the instructions on https://www.tensorflow.org/install/pip

PyCharm allows you to make a virtual environment by opening the app and clicking File > new Project and should display:



If using a virtual environment, you can then drag and drop the project files across. If wanted, you can generate your own data. This data by default should take the format of a csv file in the order of:

'Close', 'Open', 'High', 'Low', 'SMA200', 'EMA15', 'EMA12', 'EMA26', 'MACD', 'Bollinger_band_upper_3sd_200', 'bollinger_band_lower_3sd_200', 'StochK', 'StockD'

If you want to change the data's format, then you must change the columns that are dropped on line 15 in environment.py. if you want to add more data add in columns from the line on 10-12.