Reinforcement Learning in Foreign Exchange Trading

Anson Chan
24 November 2021

Abstract

Finding a profitable strategy in the foreign exchange, market has always been an extremely challenging task, due to the limited market information and the volatile market behaviours. Instead of using financial models or relying on any traders' experience, we propose to put a reinforcement learning algorithm into the market to learn its intrinsic mechanisms and find a profitable strategy. Our trained algorithm has learnt to predict the upcoming trends in the market and to decide an optimal equity size for each trade, for any given balance. When being tested periodically throughout the training process, it was able to consistently achieve final returns of over 200% in two years for most of the tests.

Introduction

Foreign exchange market, Forex, is an over-the-counter market for trading currencies. Many participants use it to hedge against international currency, interest rate risk, to diversity portfolios and such. Different participants have developed various financial models for price or trend predictions. However, these models often are unable to predict or react to the short-term market behaviours and capitalise on such fluctuations. Given the volatility and complexity of the Forex market, traders then become the final strategy executioner, in attempt to compensate for the short-term market behaviours. Still, it is inevitable that human errors are prone to occur, such as delayed reaction time or judgement errors. Quantifying that human error and carrying out the appropriate adjustments then becomes the challenge, which is oftentimes very complicated, costly, and inconvenient. Hence, the interests in taking a machine learning approach started to develop in recent years, since a machine is trainable, adjustable, and its errors are traceable for improvements.

However, to create a profitable machine for the Forex market remains a very open-ended task. More specifically, we answer the question of whether a reinforcement learning algorithm can be used in the market to generate profit from spread betting. In comparison to a supervised learning approach, reinforcement learning allows us to not manually define the optimal action given the observations at every timestamps. Instead, it learns to generate better estimates of the optimal actions and is trained to take actions closer to the optimal action estimates. This is particularly useful because to define an optimal betting size for a particular position given the noisy and incomplete market information is an extremely challenging task and involves more human guidance.

The followings are how we teach a reinforcement learning algorithm to trade in the forex market. First, we formulate the objectives and details of our problem. Then, we construct our dataset that can then be used by our algorithm to learn from and test on. Next, the learning algorithm and trading environment are explained in detail. Lastly, the training and testing results of the algorithm are presented.

Objective

Spread betting is a form of derivative product that allows traders, using leverage, to capitalise upon the movement of an asset's price offered by a broker, without owning the underlying asset they bet on. In the Forex context, traders specify an amount per percentage in point, pip, that they bet on the price movement of a currency pair and gain or lose the amount per pip movement depending on the realised price direction relative to the opened trade position. For example, opening a long position with £1 per pip in EUR/USD at the ask price 1.1215 and closing the trade at the bid price 1.1220 will result in approximately a £5 profit. Fundamentally, to spread bet on a currency pair, two main decisions must be made at each timestamp, determining the currently appreciating currency and the amount to bet, and these are the actions our reinforcement learning agent, must learn to take optimally.

For this reinforcement learning task, the agent will keep making trades in the Forex market environment, with each trade resulting in either a gain or a loss. After each interaction with the environment, it aims to learn the intrinsic workings to maximise its cumulated profits. Specifically, at each discrete timestamp t, the agent observes the available market information, a state $s \in S$ of the market and makes a continuous action $a \in A$, which represents the portion of its current equity it uses to open or continue a trade in a long or short position, according to its decision-making process, policy $\pi : S \to A$, that identifies the estimated optimal action given the observation. Then, it arrives at the next timestamp and receives a reward $r' \in R$, indicating a gain or a loss, and a new observation of the new state $s' \in S^{+}$. The goal of the agent is to find the optimal policy that maximises the expected return, which the return, G_t, is defined as $G_t = \sum_{k=0}^{\infty} \gamma^k r'_{t+k+1}$, where $\gamma \in [0,1)$ is a discount factor that determines the agent's farsightedness, how much the agent values future rewards.

One of the challenges to overcome is that unlike many RL problems, where the agent is being trained and tested in the same environment, our trained agent will be tested in a non-identical environment. For the former, agents can achieve satisfactory returns during testing even with overfitting behaviours, the condition of exploiting the specific details of the states instead of developing a generalised solution, and it could even be beneficial for some tasks. In the market, however, overly memorising the exact details of previous market patterns is unlikely to be rewarding. It is because no exact patterns or information will repeat itself and given that we cannot obtain the full information of the market at any given time, any agent's observations can only represent limited areas of the actual states. Therefore, the agent must prevent any severe overfitting issue and generalise its solutions in order to maintain profitability in the real world.

Data

The dataset required for our trading environment should be composed of two separate parts. One is the unprocessed price dataset containing Bid and Ask prices, used for calculating accurate reward after each of the agent's action; and the other one is what the agent observes in the market, an agent-observable dataset, which can be any current or past information regarding the market, assuming the information can be, in some way, represented by numbers.

The historical data of EUR/USD currency pair from 2012 to 2020[1] is used for our training and testing processes. The agent-observable dataset is constructed using several technical indicators[2] extracted from the historical price dataset. Since no one can obtain the full information of the market at any time, variable selection for the optimal state-representation in an agent-observable dataset becomes a separate task that depends on the users' knowledge of the security. The task then is to find the factors that affect a particular security, as different securities are likely to require different variables to represent their states. Our agent-observable dataset includes the Simple Moving Average $(n = [10, 21, 50])$, Moving Average Convergence Divergence $(short = 12, long = 26, signal = 9)$, Full Stochastic Indicators $(k_f = 14, d_f = 3, d_s = 3)$, Relative Strength Index $(n = 14)$, Bollinger Bands $(n = 20, m = 2)$, and 5-lags of each variable for historical contexts of the current market. This is a useful starting point as they signal the price trend, price momentum, overbought and oversold behaviours, and price volatility. Although these price indicators are only an incomplete representation of a limited state space related to price, they are suitable for our agent because of their simplicity and interpretability.

The purpose of data pre-processing in this problem is to generate, to a certain degree, a more manageable and consistent dataset for the agent to stably learn from, then generalise some implicit rules of the market and develop a strategy to maximise the profit. The pre-processing process includes (I) splitting both the historical price dataset and the agent-observable dataset into training, and testing categories, with the first 80% data used for training and the remaining for testing, (II) clipping the values of each variable to $< 1\%$ of the minimum and maximum values in our training period, (III) rescaling all variables using the min-max scaler, The process aims to accelerate generalisation by removing outliers, and making variables share a similar scale while preserving the shape of the dataset.

[1] The unprocessed Forex price dataset used in this implementation is gathered from the brokage firm FXCM.
[2] Refer to the Appendix 1 for calculations.

Agent

The Twin Delayed Deep Deterministic Policy Gradients algorithm (Fujimoto et al, 2018), TD3, is used to maximise the expected return through interactions with the Forex spread betting environment. It offers several improvements from its predecessor Deep Deterministic Policy Gradients, DDPG, while retaining the desirable characteristics. It is an actor-critic, model-free, online, off-policy algorithm that learns through sampling mini-batches of previous experiences from an experience replay buffer, with each experience, denoted by (s, a, r', s'), recorded as the agent takes a training step. It utilises an actor-critic architecture, with the actor and critic both containing target and current networks for approximations. The two kinds of network are respectively responsible for producing outputs regarding the state in the next timestamp and the current state.

For the actor-critic architecture, the learning is done through updating the current and target networks of the actors (π, π') and the critics (Q, Q'). The actors are responsible for deterministically choosing the actions given the states, and the critics are used for estimating the values of the state-action pairs, known as Q-values, which signify the qualities of the actions given the states in quantitative terms. The target-critic, Q', estimates the Q-values of the next-state-action pair generated by the target-actor, π'. In order to learn, the current-critic, Q, improves on generating more accurate Q-value estimates by minimising the mean squared error between its estimates and the targets, y, which contain the bootstrapped next-state Q-value estimates from the target-critics and the target-actor, (Q', π'). Secondly, the current-actor, π, learns to take more rewarding actions by taking steps of gradient ascent to choose the actions with the maximum Q-value estimates from a critic, Q. Lastly, the target networks (Q', π') are updated using a portion of the weights of the current networks (Q, π).

TD3 employs three techniques to help the critics produce more accurate estimates and the actors take more rewarding actions, with the goal of achieving a more stable learning process and desirable performance. It introduces *Target Policy Smoothing*, *Clipped Double-Q Learning*, and *Delayed Policy Updates*, in order to (I) discourage overfitting behaviour in the action space, (II) reduce exploitation of the dramatically overestimated Q-values, and (III) update more stably.

Firstly, target policy smoothing aims to reduce the target variance caused by the policies overfitting to inaccurate narrow peaks for certain actions, sometimes due to function approximation errors, by adding a small amount of random noise to a target action and generalising the area around that target action. The new target, y, used for updates becomes

$$y = r + \gamma Q'(s', \pi'(s') + \epsilon), \epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$$

where the clips define the small area around the target action that the policy attempts to generalise, or smooth out. In return, a smaller variance is achieved with fewer occurrence of inaccurately overfitting behaviours.

Secondly, clipped double-Q learning draws its inspiration from the original Double Q-learning (Van Hasselt, 2010). It splits the greedy action selection and Q-value evaluation by employing a pair of target-critics to generate separate targets, each of which is used to make an unbiased update of the adjacent target-critic if both targets are independent. In the actor-critic setting implementation, the author uses two separated critics, while using the two target-critics $(Q_1' \& Q_2')$ to generate two targets (y_1, y_2) used in the Bellman error loss functions.

$$y_1 = r + \gamma Q_1'\big(s', \pi'(s')\big)$$
$$y_2 = r + \gamma Q_2'\big(s', \pi'(s')\big)$$

Unlike the original, they use a single target-actor, π', to reduce computational costs. Also, due to the shared replay buffer in the actor-critic setting, the targets are not entirely independent, leading to a largely reduced, but not eliminated, overestimation bias. To further the reduction, the algorithm chooses the minimum of the two targets for the critics' updates.

$$y_{target} = r + \gamma \min_{i=1,2} Q_i'\big(s', \pi'(s')\big)$$

Each of the critics learns by regressing to their respective losses, which of each is the mean squared error between their respective current Q-value estimates and the shared target, y_{target}. Then, each of the target-critic is updated with $(1 - \tau)$ of its own weights plus τ of the current-critic's weights. Although this method may induce underestimation bias, it is acceptable as it will not be propagated through the policy update. In conjunction with the reduction in overestimation bias, the clipped doubled-Q learning largely improves the stability of critics' updates. Combined with target policy smoothing, the target, y, becomes

$$y_{target} = r + \gamma \min_{i=1,2} Q_i'(s', \pi'(s') + \epsilon), \epsilon \sim clip(\mathcal{N}(0, \sigma), -c, c)$$

Lastly, delayed policy update intends to reduce the chances of the policy maximising over any substantially inaccurate value estimates during training by having the actor updates less frequent than the critics, every d step $(d \geq 2)$. The actor is updated by maximising the target from the critic Q_1, then the target-actor is updated with $(1 - \tau)$ of its own weights plus τ of the actor's weights. This technique attempts to prevent the actor from magnifying any errors in the value approximation and to achieve less volatile performance.

Implementation

In the Forex market environment that the agent interacts with, at each timestamp t, the agent receives, (I) an observation of its own balance, denoted by b_t, (II) any previously opened position, denoted by a_{t-1} for $t \geq 1$, or 0 for $t = 0$, and (III) knowledge of the current market, which is given by our agent-observable dataset that contains information such as prices, price changes, and the price indicators. Then, it takes an action, denoted by a, where $a \in [-1,1]$. It is either sampled from a continuous uniform distribution, $a \sim U[-1,1]$, for pure exploration at the early stages, or given by the actor, with added clipped noise for continuous explorations,

$$a = clip(\pi(s) + \epsilon_a, -1,1), \epsilon_a \sim \mathcal{N}(0, \sigma_a)$$

where σ_a decides the rate that it explores. The action is defined to be the portion of the agent's balance that is used to take either a short or long position. However, given different brokers offer different leverages to different currency pairs and their various respective entry margins, the actual fraction of the balance that the agent can trade with per day varies. For implementation purposes, we set a portion $p\%$, such that the maximum balance fraction the agent can use to open a trade is $p\%$ per timestamp, either long or short, with p unchanged throughout the whole training process. For example, if $p = 0.1$, and $a = -0.1$, then the agent has decided to short 0.01% of its entire balance per point or long 0.01% of its balance per point if $a = 0.1$. After taking an action, a reward r is received, defined as

$$r = a \times b \times (0.01)p \times (ExitPrice - EntryPrice) \times 10000$$

$$r_{a,t} = \begin{cases} a_t \times b_t \times (0.01)p \times (BidClose_{t+1} - AskClose_t) \times 10000, if\ a_t > 0 \\ a_t \times b_t \times (0.01)p \times (AskClose_{t+1} - BidClose_t) \times 10000, if\ a_t < 0 \end{cases}$$

The reward specifies a loss or a gain in terms of the fraction of the balance used to open the trade. For simplicity, it is assumed that the agent only takes actions using close prices. Next, the new balance becomes

$$b_{t+1} = b_t + r_{a,t}$$

For any $t > 0$, the balance describes the size of the current balance in terms of some multiple m of the initial balance at $t = 0$.

$$b_t = m \times b_0$$

If we set $b_0 = 1$, representing 100% of the original equity, then for any $t > 0$, m becomes the balance. This is mathematically convenient, and it can accommodate different users' initial balance adequately without re-training the model. We can then calculate the percentage gain or loss at any timestamp, defined as

$$\%\ gain\ or\ loss = (m - 1) \times 100\%$$

During training, a random starting timestamp is uniformly selected, with the selected timestamp set to $t = 1$, along with $b_0 = 1$, and $a_0 = 0$. Afterwards, the agent starts trading until it reaches the terminal state where it either arrives at the end of the training dataset or has a balance below 0. We treat the interactions from the starting timestamp to termination as an episode and run g episodes for the training process.

Result

We have created a TD3 agent[3] using 4 layers of 1024-dimension densely connected layer for each of the actors and critics. The performance of the learning algorithm can be represented by its returns in the training environment and the testing environment. Testing environment is where the agent has never been trained on and can never store any experience from, and each testing episode starts from the 21/3/2019 to 31/12/2020 with a balance of 1. To monitor the agent's progress, after the first 200 episodes of training, tests are conducted every 10 episodes using the latest weights of the actor, the most recent iteration of the actor.

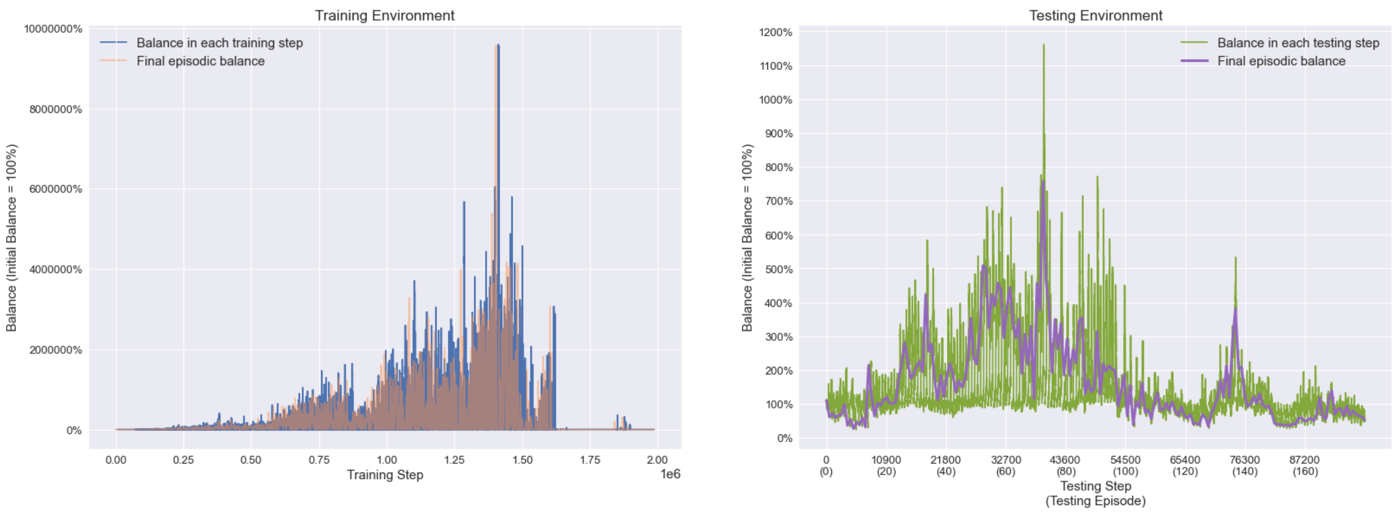

Figure 1. The balances at each training (blue) and testing (green) step, proportional to the original balance.

As shown in figure1, the agent has learnt to improve in the training environment as expected and is achieving higher return as the training progresses, until 1.5 million training steps, the agent's return plummets. In the testing settings, we can see significant improvements in returns up to about 120 training episodes, the rewards started to fall substantially and returns became negative. It is likely that the agent then had started exploiting noises or rare events in our training dataset and overestimating those events' values. As a result, the actor deviated from the general strategy, causing significant drops in both the training and testing performance. However, it is apparent that in two-third of the testing episodes, the balances are consistently above 100%, and many of the final episodic balances had gone beyond 200%, prior to decline in the training performance. Although the learning algorithm was not able to stabilise, and it was likely due to overestimation bias, the algorithm was able to develop some form of general strategy during training and achieve considerable amounts of returns.

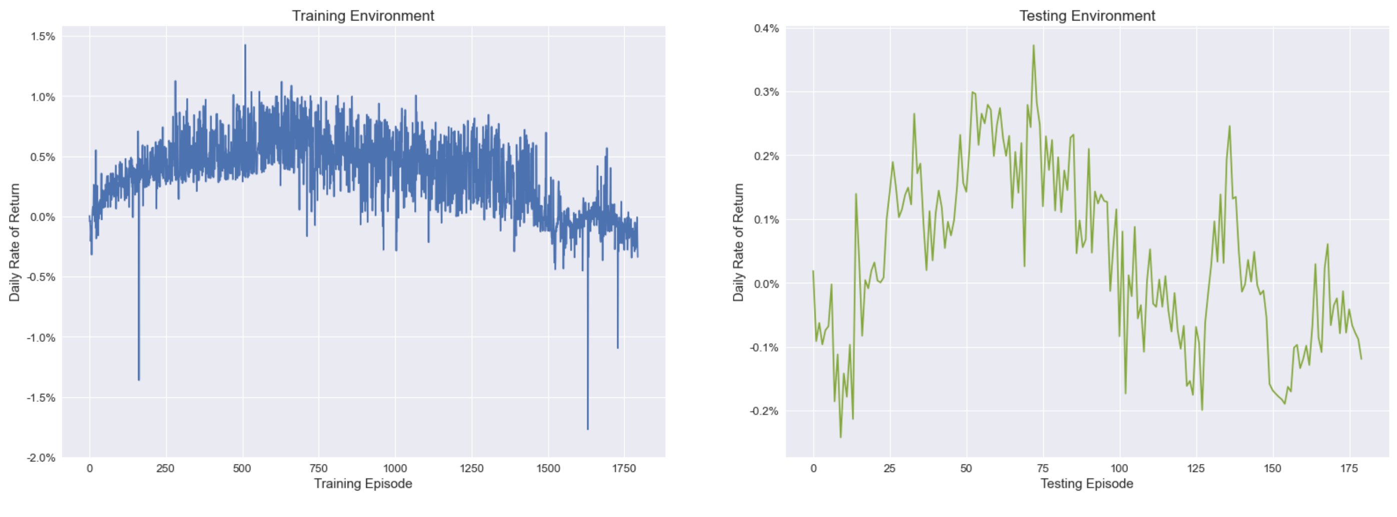

Figure 2. The average daily rate of return in each episode.

Since our agent reinvests daily, we can estimate the daily rate of return, daily ROR, by taking the geometric mean of the daily return within each episode. In the training environment, the averages were not able to maintain at the 0.5%-1.0% level

[3] Other hyperparameters include controllable balance p = 0.1, actor and critic's learning rate = 1e-5, policy update frequency d = 4, farsightedness γ = 0.995, exploration standard deviation σ_a = 0.3, policy noise standard deviation σ = 0.15, policy noise clip c = 0.3, target networks update coefficient τ = 2e-3.

and it started to decline as the training goes for longer. For the testing environment, the averages of daily ROR reassemble the pattern of the testing balances. It steadily climbs up for the first one-third of the testing episodes, peaked with a daily ROR of 0.37%, and then started to drop drastically.

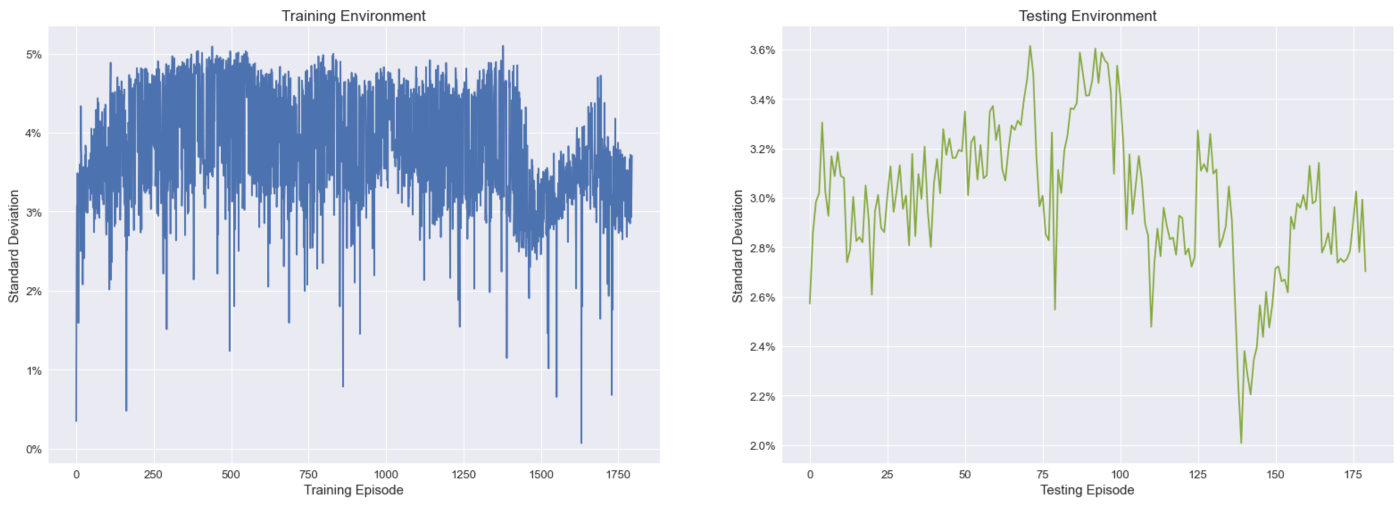

Figure 3. The standard deviations of the daily return in each episode

Normally, as the agent demands higher return, it is likely to take risks more frequently, and tends to generate a higher standard deviation. In Figure3, the standard deviations of the training data do not show any clear pattern, and it generally ranges from 3% to 5%. In the testing environment, the standard deviations display a similar shape to the daily rate of return; however, while the daily return declined after the 72-th episode, the peak episode for the daily RORs, the standard deviation remains as high in several of the following episodes.

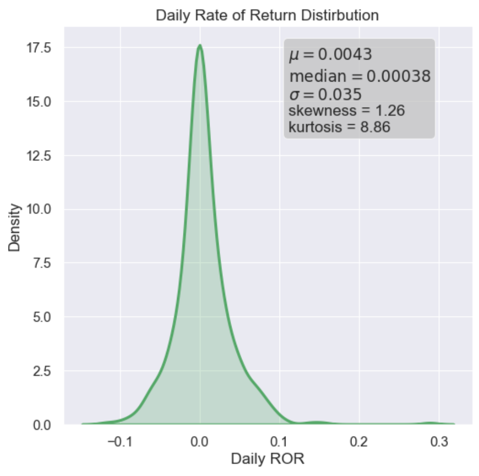

Figure 4. Daily rate of return distribution of the best-iteration agent

In a practical point of view, although the agent's results did not stabilise, we can select a specific iteration of the actor, in a model selection sense, and treat the current testing result as our validation result, for real-world implementation purposes. To select an iteration of the actor for actual usage, it is obvious to choose one that has the highest return with minimal risks. Since all the validation episodes suggest that the training done after the 72-th validation episode worsened the agent's performance, we will further investigate that iteration of the agent. The agent has achieved a final balance of 738% of the original balance. The descriptive statistics and a daily return distribution in figure4 show a leptokurtic distribution with positive skewness, mean, and median. The distribution's positive skewness suggests frequent small losses and a few large gains; the higher than 3 kurtosis indicates that we should expect higher return fluctuations than normally distributed returns. Consistently, it predicted the upcoming price trends correctly and took the appropriate positions. It has fulfilled our requirements of achieving a positive return in the validation settings.

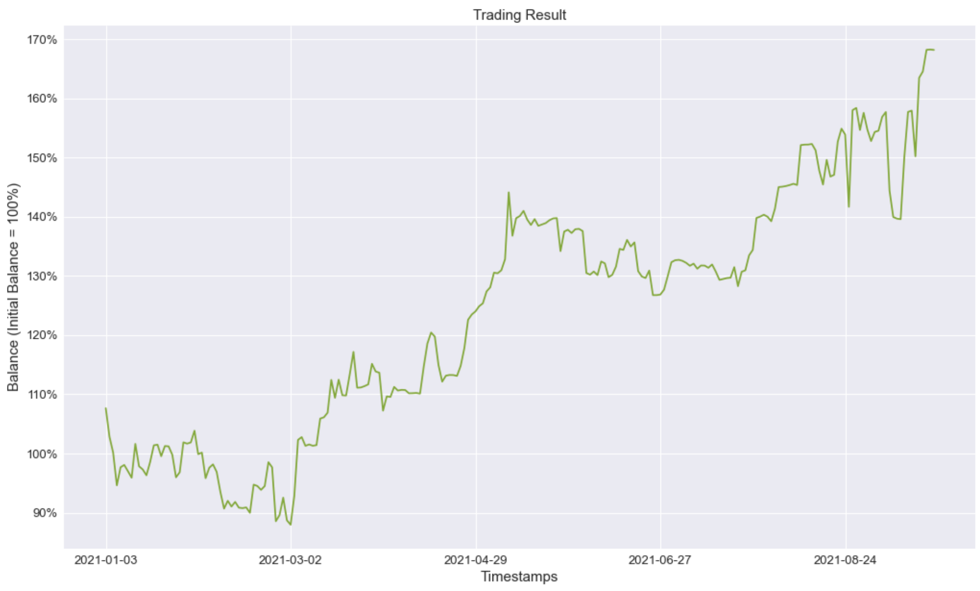

Figure 5. Trading Result in 2021

After Implementing that iteration of the agent in the new testing environment, figure5 displays its performance when being tested in the EUR/USD market, from 3/1/2021 to 24/11/2021[4]. It has achieved 168% return, or 68% profit, with an estimated daily ROR of 0.23% and a standard deviation of 3.3%. Although a particular iteration of the model had to be selected for actual usage because of the sudden decline in performance in longer training time, the result has shown the practicality and profitability of the agent throughout the validation and testing data. This strengthens our initial belief that a reinforcement learning algorithm is capable of finding a profitable strategy in the forex market given the appropriate training data. The challenge for further implementation in other securities remains in the variable selection task and performance stabilisation if the model selection process is to be removed.

Conclusion

Forex market is an intrinsically complex environment with very limited information regarding its dynamics. When a reinforcement learning agent is put into this noisy environment, it is a very challenging task to find a profitable strategy that is general enough to account for the constantly changing market behaviours. We have chosen the learning algorithm TD3 to learn spread betting in the EUR/USD market. With our defined learning objectives, the model needed to correctly select a long or short position and decide the betting amount, and it is suitable for any equity size in implementation.

Our EUR/USD trading result suggests that the agent was able to learn a profitable strategy before being affected by the overfitting issues and overestimation bias. For practical usage, we have selected an iteration of the agent by utilising the validation dataset. Our result has shown that through training, the agent was able to achieve over 200% returns during 2019 to 2020 consistently in many of the validation episodes, with the maximum final return of over 700% in one validation episode. Utilising model selection, the applied model was able to achieve 168% return in the testing environment during 2021. All in all, with a sufficient representation of the market that the TD3 agent can observe, it is able to generate a considerable amount of profit in the forex market. The challenge for implementations in the overall market becomes feeding the agent the adequate information regarding the specific market and possibly performing model selection tasks.

[4] The unprocessed Forex price dataset used in this implementation is gathered from the broker FXCM.

Appendix 1 – Indicators calculation

Simple Moving Average

The n-period Simple Moving Average (SMA) at each time t, is given by

$$SMA_{n,t} = \frac{1}{n}\sum_{i=0}^{n-1} C_{t-i}$$

$C = Closing\ price$

$n = Number\ of\ periods$

Moving Average Convergence Divergence

The n-period Moving Average Convergence Divergence (MACD) and its signal line at each time t, are given by

$$MACD_{short,long,t} = EMA_{short,t} - EMA_{long,t}$$

$$MACD_{signal,t} = \left[MACD_{short,long,t} \times \frac{s}{10}\right] + EMA_{signal,t-1} \times \left[1 - \frac{s}{10}\right]$$

$$EMA_{n,t} = \left[C_t \times \frac{s}{(1+n)}\right] + EMA_{n,t-1} \times \left[1 - \frac{s}{(1+n)}\right]$$

$n = Selected\ time\ period$

$EMA_n = Exponential\ Moving\ Average\ for\ a\ n\ period$

$C = Close\ price$

$s = Smoothing, usually\ equals\ to\ 2$

Full Stochastic Indicator

The Stochastic Oscillator at each time t, is given by

$$\%K_{f,t} = \frac{100 \times (C_t - L_{kf,t})}{H_{kf,t} - L_{kf,t}}$$

$$\%D_{f,t} = \frac{\sum_{i=0}^{2} \%K_{f,t-i}}{d_f}$$

$$\%D_{s,t} = \frac{\sum_{i=0}^{2} \%D_{f,t-i}}{d_s}$$

$k_f = Fast\ Stochastic\ period$

$d_f = Period\ of\ Moving\ Average\ of\ \%K_f$

$d_s = Period\ of\ Moving\ Average\ of\ \%D_f$

$C = Closing\ price$

$L_{kf} = Lowest\ price\ of\ the\ k_f\ most\ recent\ period$

$H_{kf} = Highest\ price\ of\ the\ k_f\ most\ recent\ period$

Relative Strength Index (RSI)

The n-period Relative Strength Index (RSI) using exponential moving average at each time t, is given by

$$RSI_{n,t} = 100 - \frac{100}{(1 + RS_{n,t})}$$

$$RS_{n,t} = \frac{AvgU_n}{AvgD_n}$$

$$AvgU_{n,t} = \left[UC_t \times \frac{2}{(1+n)}\right] + AvgU_{n,t-1} \times \left[1 - \frac{2}{(1+n)}\right]$$

$$AvgD_{n,t} = \left[DC_t \times \frac{2}{(1+n)}\right] + AvgD_{n,t-1} \times \left[1 - \frac{2}{(1+n)}\right]$$

$n = RSI\ period$

$UC = Close\ price\ of\ upward\ moving\ price$

$DC = Close\ price\ of\ downward\ moving\ price$

Bollinger Bands (including %B & Bandwidth)

The n-period Bollinger bands, including the %B and the bandwidth, at each time t, are given by

$$TypicalPrice_t = (High_t + Low_t + Close_t) \div 3$$

$$UpperBollinger_{n,t} = MA_{TP,n,t} + m \times \sigma_{TP,n,t}$$

$$LowerBollinger_{n,t} = MA_{TP,n,t} - m \times \sigma_{TP,n,t}$$

$$Bandwidth_{n,t} = \frac{UpperBollinger_{n,t} - LowerBollinger_{n,t}}{TypicalPrice_t} \times 100$$

$$\%B_{n,t} = \frac{C_t - LowerBollinger_{n,t}}{UpperBollinger_{n,t} - LowerBollinger_{n,t}}$$

$n = Bollinger\ bands\ period$

$MA_{TP,n} = Moving\ Average\ of\ typical\ price\ in\ the\ most\ recent\ n\ period$

$m = Number\ of\ standard\ deviations$

$\sigma_{TP,n} = Standard\ Deviation\ over\ the\ most\ recent\ n\ periods\ of\ typical\ price$

$C = Closing\ price$

Reference

Addressing Function Approximation Error in Actor-Critic Methods, Fujimoto et al, 2018
https://arxiv.org/abs/1802.09477

Bollinger Bands, Investopedia
https://www.investopedia.com/terms/b/bollingerbands.asp

Bollinger Bandwidth, Stockcharts
https://school.stockcharts.com/doku.php?id=technical_indicators:bollinger_band_width

Bollinger Band %B, Stockcharts
https://school.stockcharts.com/doku.php?id=technical_indicators:bollinger_band_perce

Double Q-Learning, Van Hasselt, 2010
https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf

EUR/USD Historical Price Dataset used, FXCM
https://www.fxcm.com/uk/algorithmic-trading/market-data/

Moving Average, Investopedia
https://www.investopedia.com/terms/s/sma.asp

Moving Average Convergence Divergence, Investopedia
https://www.investopedia.com/terms/m/macd.asp

Relative Strength Index, Macroption
https://www.macroption.com/rsi-calculation/

Stochastic Oscillator, Stockcharts
https://school.stockcharts.com/doku.php?id=technical_indicators:stochastic_oscillator_fast_slow_and_full