Recovering Lost Information from Exponential Moving Averages on HFT Market Data

by

Abdelmajid Essofi

Thesis submitted to the
Deanship of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the M.Sc degree in
ML

Deprtement of Machine Learning
Mohamed bin Zayed University of Artificial Intelligence (MBZUAI)

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Dr. Martin Takáč Associate Professor, Dept. of ML, MBZUAI
Supervisor(s):	Dr. Le Song Professor, Dept. of ML, MBZUAI Dr. Bin Gu Assistant Professor , Dept. of ML, MBZUAI

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

When it comes to algorithmic trading, traders and quantitative researchers work to develop models that use certain signals that study the market in a way that best predicts the times in which to open and close trades, with an aim to maximize the returns by the end of the trading period. Because of all the noise that comes with price changes, and the lack of interpretability that follows, algorithmic traders develop their signals by using a smoother version of the price changes through the application of a moving average. Moving averages are a technical tool that creates a smoother version of price data by bucketing noisy data in batches of size W, then mapping every data point to the bucketed average. This aims to simulate the noiseless trend of pricing data by taking the average shift of values over W, thereby removing any noise within the data itself. A specific type of moving averages is called the Exponential Moving Average (EMA), and it is heavily used when dealing with unevenly space time series data. EMA formulas include a specific ratio of every data point into the curve depending on the time it took since the last data point entered the stream. Subsequent iterations shrink the data point by a user-specified decay factor until its contribution is rendered negligible. A main disadvantage of this approach occurs in the high frequency trading world, when time differences occur in fractions of a second, and are even identical at times. This causes data points to enter the moving average stream at a ratio of 0.0%, causing significant information to be lost in the process. In this paper, we explore this phenomenon, and succeed in proposing an approach to recover the consequently lost information. Our solution introduces an additional hyperparameter r, representing the desired ratio at which every data point is to enter the smoothing curve, without compromising the logic of the EMA process, or allowing noisy data to contribute to the curve. Our proposed approach significantly improves on the currently used approach both on the quantitative metrics, and further shows the importance of generating real-time trends for the benefits of traders, where a reinforcement learning agent trained on each signal separately generates more profit on unseen data when using our proposed method, than using the current approach or the noisy market value data as a signal.

Acknowledgements

I would like to thank Professors Le Song and Bin Gu for supervising me throughout my Masters program and keeping track of my progress in various potential topics for my thesis project. I would also like to direct special thanks and unparalleled gratitude to Doctor Martin Takáč and Doctor Kun Zhang, for their continuous help and support through meetings and quick Q&A sessions. Finally, none of this would have been achieved if it wasn't for the incredible atmosphere created by my fellow colleagues and closest friends, many of which were going through this enriching, and equally challenging, final project experience.

Table of Contents

List of Tables

List of Figures

Listings

List of Abbreviations

AI	Artificial Intelligence
RL	Reinforcement Learning
MA	Moving Average
MAE	Mean Absolute Error
EMA	Exponential Moving Average
SMA	Simple Moving Average
HFT	High Frequency Trading
LFT	Low Frequency Trading
P&L	Profit and Loss
BBO	Best Bid & Offer

Chapter 1

Introduction

When studying the changes of pricing data through time, there is a very high likelihood for the pricing variable to include a lot of noise and outliers. More specifically, on market data, the value of a stock, currency, or exchange, is directly affected by certain parameters such as: number of requested lots, number of available lots, number of sellers, number of buyers, and the spread between the best buying and best selling prices (also referred to as bid/ask spread). Because of the variety of such directly affecting parameters, the value fluctuates at such high frequencies, that the price changes through time become so noisy and unstable, leading to an uninterpretable price trend.

More and more trading firms are opting for an increasingly, if not fully, automated trading process. The benefits of algorithmic trading are clear, they signify less menial tasks, as well as a much faster order entry, leading to equally higher returns with the right strategy [31]. Such incredible benefits render manual human trading almost illogical of an approach. Because of that, more and more trading firms are focusing most of their time into increasing the automation of their equity trading by order and volume. In fact, in the next one to three years, 66% of equity professional are expected to focus most of their time into automation processes [31].

When it comes to algorithmic trading, traders and quantitative researchers work to develop models that use certain signals that study the market in a way that best predicts the times in which to open and close trades, with an aim to maximize the returns by the end of the trading period. Every trading firm develops their own signals, which follow certain strategies in a way that they believe studies the market best at the concerned time period. As unique as these signals are to every algorithmic trading-based firm, most, all of them have to take price changes with respect to time into consideration. Because of all the noise that comes with price changes, and the lack of interpretability that follows, algorithmic traders develop their signals by using a smoother version of the price changes through the application of a moving average. Moving averages are a technical tool that creates a smoother version of price data by constantly updating the average price [35]. Although not exclusively based on moving averages, many algorithmic trading signals heavily rely on the output of this smoothing operator. A typical trend-smoothing moving average is represented in Figure 1.1.

Figure 1.1: Example of a Simple Moving Average. Image source [5].

The moving average represented in Figure 1.1 aims to study the overall price trend by not only avoiding high-frequency fluctuations, but also avoiding occasional dips in the market that are quickly recovered by the overall positive trend. Other moving averages prefer to avoid such a heavy loss of information by decreasing the duration that each sample contributes to the moving average. Moving averages with a low sample lifetime preserve the trend a bit better, and look as shown in Figure 1.2.

Figure 1.2: Simple Moving Average with Small Sample Lifetime. Image source [5].

Users can customize the amount of information represented in the smooth curve by controlling certain parameters such as: Type of the used moving average, including simple moving average, exponential moving average..., and the lifetime of a sample in the smoothing operation through the use of:

- **Look-back length** when dealing with simple moving averages, which defines the look-back period on the number of samples to include in the average value. The longer the look-back length, the longer the time period in which a sample contributes

to the moving average, leading to a more generalized version of the smoothing curve, and a higher likelihood to ignore noise and occasional outliers in form of dips and/or spikes.

- **Half-life** of a sample when dealing with exponential moving averages, which is the time period that a sample should last in the smoothing curve before it contributes half of its original value. The smoothing logic works similarly to the look-back method, but the half-life is most ideally used on unevenly-spaced time series because, using some exponential operator properties, the user can control the time period it takes a sample to contribute exactly half of its original value to the moving average, after which its contribution becomes negligible, and does not affect the smoothing operator anymore. We will discuss more details of the half-life in Section 3.4.2.

The type of moving average used, as well as the length of the half-life or look-back window size, all depend on the type of information sought from such a smoothing curve. In some instances, traders may be interested in the overall trend of pricing data, discarding any information that may be gained from the recurring fluctuations of the price changes, such instances occur when dealing with low-frequency trading (LFT), and performing trades that typically last a long period of time before being closed, aiming for a low trading volume, but high reward per trade. This type of trading is used when studying the value of a stock over-time, and predicting that its value will either increase, or decrease, in a certain number of months. This branch of trading, however, is typically performed by humans and has little to no use of powerful computers to be maintained [3]. High-frequency trading (HFT), on the other hand, is a branch that relies on performing trades that generates small profits by taking advantage of the occasional fluctuations in pricing data, this type of trading is typically used to perform a high volume of trades in a matter of fractions of a second, with small reward as well as risk per trade. [16]. Recently, with the development of powerful and equally complex strategies, it is becoming increasingly vital to develop complex models and algorithms to maintain a standing in the HFT market, unlike LFT [9]. This also means that, in order to develop a competent strategy for high-frequency trading, the study of price changes must represent the overall price trend, as well as the occasional dips and spikes, despite applying a smoothing operator such as the aforementioned moving averages.
Because market data follows a unevenly spaced time series model, the smoothing operator used to display price trends are typically chosen to portray such temporal properties. While simple moving averages control the number of iterations through which every data point contributes to the moving average, this number of iterations represents an ambiguous time period in an unevenly-spaced setting that can range from nanoseconds to a several minutes, which could lead to the inclusion of unimportant data in our smoothing curve, since enough time may go by that the value of such an ancient data point is rendered negligible [11]. In order to fix that, most high-frequency traders adopt an exponentially-weighted moving average (EMA) as a smoothing operator. In Section 3.4.1, we discuss how EMAs use an exponential operator with respect to a decay factor λ and the time difference between the current and past samples $dt = t_i - t_{i-1}$ in order to decay the contribution of every sample to the moving average through time. In other words, the EMA equation is written as:

$$EMA_i = (1 - \alpha_i)\, X_i + \alpha_i\, EMA_{i-1}.$$
$$\alpha = e^{-\lambda(t_i - t_{i-1})}. \tag{1.1}$$

Because of the additive property of the exponential operator, where $e^a * e^b = e^{a+b}$, we later discuss how equation (1.1), combined with the additive property of the exponential operator, are used to control the duration after which each data point contributes to the smoothing factor through the use of half-life, which is the time period it takes for a data-point to contribute half of its original value to the smoothing operator. One notable trait of HFT market data is that it is affected by the bid/ask orders sent by individual users to the concerned exchange. As mentioned above, most high-frequency traders perform their trades by using complex algorithms and powerful machinery. This results in an incredibly high volume of orders within a matter of nanoseconds. Sometimes, certain individuals simultaneously send several orders at the same time, either to manipulate the market in their favor, or to edit some previously sent order (editing for some exchanges works by cancelling and adding at the same time in form of two orders instead of a single edit function). Such market updates appear on datasets under the exact same timestamp, causing the smoothing operator to set t_i and t_{i-1} to either be equal, or very small in difference. Consequently, the initial multiplier that includes the later sample into the moving average is set to:

$$1 - e^{-\lambda(t_i - t_{i-1})} = 1 - e^{-0\lambda} = 1 - 1 = 0. \tag{1.2}$$

Similarly, when $t_i - t_{i-1}$ is set to a very small value, the initial multiplier approaches zero. As a result, most, sometimes all, of the information carried by the sample is lost from the smoothing curve, simply because the time difference between adjacent samples approaches or equals zero.

Such events occur on many different instances throughout the day. The drawback of these events is that the later market updates that arrive have more impact on the new valuation of the final price and has more control of the market state than earlier samples, which only initiate the order. With the loss of such valuable samples from the smoothing curve, some valuable instances are skipped and never included in the computation, causing the curve to diverge from the price trend, and consequently show trends that do not align with the actual current trend we want to study.

Figure 1.3 shows an example of such loss of information causing the smooth curve to diverge away from the price trend following a sample half-life of 30 seconds.

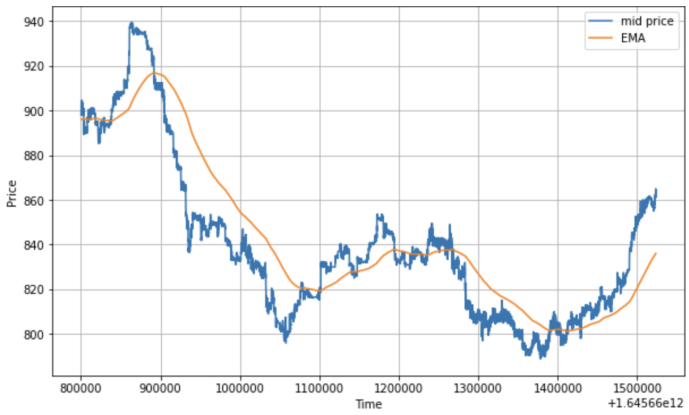


Figure 1.3: Currently Used EMA Curve with Halflife of 30 Seconds.

As shown in Figure 1.3, the initial smooth concave curve at the 900,000 x-point reaches a slope of zero at a significantly delayed time from the actual period at which the true price sets and prepares to drop. Hence, when relying on the EMA curve to study the price changes, a model will miss out on the true time at which the price trend changes and begins to drop, causing a significant potential loss of returns when relying on such a signal to perform trades. When analyzing the dataset, we found that the loss of information is caused by very close, if not identical, samples in time. In this paper, we tackle this issue by implementing a method that maintains the temporal aspect of the exponential smoothing technique, assigning increasingly low importance to samples that happened much further in the past, without compromising on adjacent samples that occur at close proximity or identical timestamps. To evaluate our method against the currently used approach, we go through a 2-step verification:

- We compute the average MSE & MAE between every experiment we run and the corresponding market value, then set the score as the distance to the ground truth error, which we define in Section 5.2, taking the smallest positive distance as the best possible approximation to the ground truth.
- To make sure our method is as beneficial to traders as it is quantitatively superior, we built a **Policy Gradient Reinforcement Learning** model that uses the EMA movement over the last M steps as a signal to judge the movement of the market, then performs trades accordingly with an aim to maximize the end-of-day returns as a reward.

This two-way verification method guarantees that, without loss of information, the proposed method benefits traders in terms of trading returns better than using the mid-price itself as a signal, or even the currently used EMA method. Our approach achieves a drop of mean absolute error from 0.45 using the currently used method, to 0.18 using our approach. Also, the implemented policy gradient method was tested on a day's worth of Japan Stock Exchange (JSE) data, and succeeded in generating more returns than using the mid-price and the currently used method.

Chapter 2

Motivation

In this chapter, we explore the process that motivated this project, and the practical uses that this project could have on many signals currently adopted by trading firms for their trading strategies.

Trading companies are constantly in pursuit of new and improved signals that best explain the market, and form a direct dependence to the future shifts in market value within a specified temporal window Δt or a set number of trades in the future. Intuitively, it is easy to understand why most of these signals are extracted from, or related to, past values of the market value being studied (prices of stock, futures, options, cryptocurrencies...). However, market data, especially in high-frequency, includes a lot of noise and uninterpretable fluctuations which may change the market value, but have no impact on the overall trend the traders seek when creating signals. For that reason, when developing such signals, traders use smoothing methods to map out the value trends over time and eliminate noise. Because trading is all about unevenly spaced time series data, the best smoothing operator used by trading firms is the exponential moving average (EMA).

During our time working with such signals, we noticed that in some occurrences, the smoothing curve was not mimicking the market value changes as accurately as sought from it. We later found that many of the occurrences where a lot of information is lost and the trend shifts is caused by a single trader who manages to send a heavy batch of orders to the books, all of which get mapped to the same timestamp, since they come from the same user and are sent at once. Although all orders are sent together, there is a sequential logic to the order in which they were sent, each of which holds an information about a shift that such an order caused in the market. Therefore, it is vital to find a way to avoid any loss of information that may occur from such occurrences. However, using the current EMA formula, as defined in Equation (1.1), adjacent data points with identical timestamps, or even negligible differences, get lost from the smoothing curve when the initial multiplier $(1 - e^{-\lambda \Delta t})$ is applied.

In this paper, we propose an improved version of the EMA that recovers lost information for marginally different and identical timestamps. Our method improves on the currently used approach by scoring the smallest distance from the ground truth comparison, and further redeems higher rewards when used as a state for our reinforcement learning agent.

Chapter 3

Background & Related Works

3.1 Trading

Trading is the process of making profits by taking advantage of fluctuations in the price changes of financial instruments, and buy at a low price and sell when the value is high. There are several financial instruments available for trading such as stocks, shares, funds, securities, futures, options... [7]. Trading has always been done by humans in the past, where professional traders study the changes in the market and how it is affected by recent political and economic events, in order to predict the direction it is expected to take in the near future. By developing knowledge of the market, traders are able to predict short-term fluctuations, and perform short and long trades in a way that generates profits. **Going Long** is a term used when traders buy at a low price and sell at a high price to generate profit. **Going Short** is another term used when traders make profit by selling at a high price, then buying at a low price [7]. Figure 3.1 shows how profits are made depending on how prices change after performing a certain type of trade. Unlike investors, who hold ownership of a stock for long periods of time until its value increases enough to grant them the profits they seek, traders aim to make relatively profits over short periods of time by exchanging long and short trades for the duration in which the exchange they are trading in remains open.

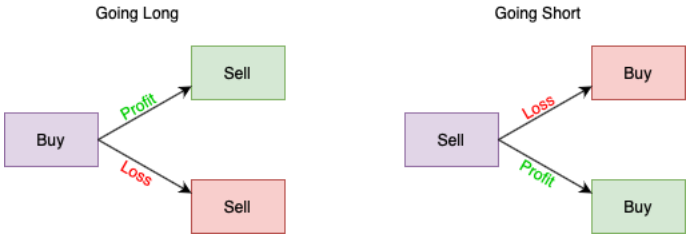


Figure 3.1: How Profit Works for Long/Short Trades.

There are different types of trading that differ a lot from each other, despite holding the core logic of what trading is. For some traders, news are the only aspect they need

to perform their trades. It is vital for this type of traditional traders to keep up with recent news to predict how the market is expected to perform at any given time, then trade accordingly to generate profits. Another type of trading is Quantitative Trading. For these traders, regardless of what is going on in the political and/or the economic world, statistics are all that counts. By performing statistical analyses on historic data, traders come up with signals that predict the best moves for these traders to make that will maximize their profits in the near future [32].
Just like the different types of traders, there are also different types of trading. The first type, which we primarily focus on in this paper, is called High Frequency Trading (HFT). These traders value the highly recurrent fluctuations in the market much more than the overall trend of the market value over long periods of time [20]. Over a lapse of a few seconds, high frequency traders perform several trades that make profit for long trades when the fluctuation was positive, and make profit for short trades when the fluctuation was negative. By the end of day, the negligible profits made by every trade accumulates to high returns, which are more likely than not to be significantly higher than any returns made by the next type of trading. Despite the main difference between traders and investors being the duration of the trade (among other reasons), there are still traders that prefer to bet on the value of the financial instrument changing over much longer periods of time, instead of a matter of seconds. These traders perform what is called Low Frequency Trading (LFT). By studying historic data showing price changes over the past several months, these traders notice that the value of a financial instrument often very significantly changes to different scales, which leaves room to consider holding the asset for longer periods of time until the value rises or drops to the desired level [20].

3.2 High Frequency Trading

For several decades in the past, trading was always conducted by humans who spend significant amounts of time studying the market and the factors that directly affect its trend. By doing that, traders were able to predict the direction of the market price over the next few hours or so, and gather enough knowledge to perform several trades that accumulate to generate major profits by the end of the trading period [26]. Recently, however, trading firms gained a lot of interest in increasing the volume of their trades, and instead of studying the trend of a certain index over the next few hours, it became clear that more benefits can be redeemed from studying the minor fluctuations of the price over much shorter periods of time. So, instead of simply relying on human abilities, trading firms began investing more and more in competent machinery to perform trades. With the inclusion of AI, traders are now able to teach a computer what to look for using quantified signals that translate the market trends, and thanks to the incredibly fast processing powers of machines nowadays, traders are now able to make profits using fluctuations down to fractions of a second [34]. Figure 3.2 shows that by zooming down to a few milliseconds, we can see some trends that are clear to be profitable when properly taken advantage of, but would otherwise be missed by simply relying on human minds.

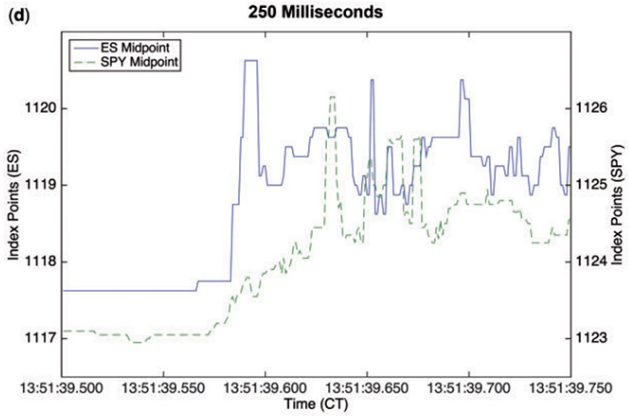


Figure 3.2: Millisecond Level Index Value Change. Image source [4].

3.3 Quantitative Trading

Traditional trading is an approach where traders rely on global political and economic news, as well as recent trends and historic data, to predict trend movement and perform trades that generate sought profits. However, a recently popularized approach, called quantitative trading, makes no use of such metrics. With access to historic data, quantitative traders develop statistical models, often linear, that carry a dependence with whatever target they are interested in. For example, if the trader is interested in the value change of a certain financial instrument over the next second, they develop a group of features (also called signals), that form a linear combination that directly correlates with the price change [21]. Because high frequency trading aims to carry results in fractions of a second, the difficulty lies in decreasing the complexity of such features, and limiting the relationships between features and the target variable to a linear one, where the value change can be predicted by fitting a simple linear regression model to make such a prediction. Figure 3.3 below shows the steps taken by quantitative traders in their approach.

Figure 3.3: Quantitative Trading Approach. Image source [1].

As shown in Figure 3.3, quantitative traders begin by developing a strategy, which is the ensemble of human logic of what symbols correlate best with the sought target, followed

by the process of developing signals in a form of features that can be linearly combined to predict the target. Then, back testing is the process of fitting a linear model using the developed signals on an ensemble of historic data, with the aim of testing the proposed strategy for generating profits. Following that is the execution systems and risk management phases that take care of testing the newly developed strategy on current data in real-time, with the aim of protecting the firm from any large scale losses, while maintaining a robust testing approach that would translate into real data in real time on the long run as well as it did on the back testing phase [15].

3.4 Moving Averages

When dealing with a dataset studying the changes of a certain variable with respect to time, it is more likely than not for that variable to include a distribution of noisy data and outliers which are generally useless to the overall trend of the variable. Spotting these outliers and noisy data points can be very tricky even when dealing with fixed data, on which we have access to future trend, but it is even trickier in real time, when one can not spot whether a spike/drop is a mere outlier, or the start of a new trend.
To handle such scenarios, a simple but powerful moving average technique was developed, which uses past and current data points as a way to model the current trend of the studied variable, without including the noisy data in the mix [39]. Different techniques of moving averages make use of past data in different ways, but the core logic is to use a chosen aggregation method on a window of past data, compute an average of the specified window, and use the average as the current point, rather than the noisy data point itself. There exist several types of moving averages, the most straight-forward of which is called a Simple Moving Average (SMA). For a given dataset $\mathbf{X}$ and a window size $\mathbf{N}$, the SMA at every point $\mathbf{i}$ is defined as:

$$\begin{aligned} X'_i &= \frac{1}{N}\Big(X_{i-N+1} + X_{i-N+2} + ... + X_i\Big). \\ \Rightarrow X'_i &= \frac{\sum_{j=0}^{N-1} X_{i-j}}{N}. \end{aligned} \tag{3.1}$$

This method is also called a smoothing method, which is intuitively understood to be due to the discarding process of noisy data, and maintaining a smooth curve on which the slope signals the overall trend of the variable being studied. Depending on the window size being specified, the trend on the smoothing curve gets more and more generalized. In other words, the patience window for fluctuations being considered noise gets larger with larger window sizes. Figure 3.4 portrays different smoothing curves according to different window sizes for the simple moving average being applied to the variable.

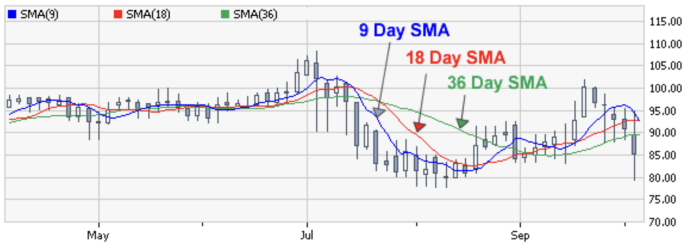


Figure 3.4: SMA Smoothing Curves With Different Window Sizes. Image source [6].

SMA is a powerful technique for smoothing temporal variables, but it has its weaknesses. One main drawback of using the SMA smoothing method is that it assigns an equal importance weight to every single data point being included in the smoothing curve, as long as the window size includes a point in time, then the corresponding point will carry as much importance as every other point within the window. This may not be seen a drawback when dealing with equally spaced time-series data, where sampling time from one data point to the other is equal throughout the dataset. However, when working with unevenly spaced time-series datasets, adjacent data points must logically carry more importance to the smoothing process when the time difference between them is larger, and a data point's importance must get smaller as the time difference to the next sample gets smaller [8]. In other words, if the time exhausted since the last sampled point is large enough, then the current data point is likely to define the standard at a much larger scale than whatever information passed through the previous samples, as opposed to adjacent samples at a very close proximity to each other, which should be seen as carrying minor information to the already defined trend by previous samples. This characteristic is not handled by the SMA technique because there is no temporal component to its smoothing equation. For that reason, a new approach to moving averages was proposed, called the **Exponential Moving Average (EMA)**, which we explore in Section 3.4.1.

3.4.1 Exponentially Weighted Moving Averages

To handle relative sample importance when dealing with unevenly spaced time-series data, the EMA method proposes the following formula for updating the smoothing curve at every point in time:

$$\begin{aligned} EMA(i) &= (1-\alpha_i)\, X_i + \alpha_i\, EMA(i-1). \\ EMA(0) &= X_0. \\ \alpha_i &= e^{-\lambda(t_i - t_{i-1})}. \end{aligned} \tag{3.2}$$

The way this smoothing operator works is by initializing the curve to the variable value associated with the first timestamp in the dataset ($t = 0$). Starting $t = 1$ on, a ratio of α_i is taken away from the current moving average, and compromised by a ratio of $(1 - \alpha_i)$ taken from the current data point [24].

α_i takes a different value for every data point as a way to measure the importance of each data point in the moving average being computed. For the remainder of this paper, α_i will be defined by an exponential decay operator such that: $\alpha_i = e^{-\lambda \Delta t}$, where $\Delta t = t_i - t_{i-1}$. Using this exponential operator, the larger the value of Δt, the larger the value of the initial multiplier $(1 - \alpha_i)$ as it approaches 1, whereas the smaller the time difference, the less information is contributed by the point being included in the moving average [25]. The advantage of the exponentially weighted moving average method is its ability to control the amount of information being introduced by every data point once it enters the smoothing process. However, the recursive nature of the EMA formula means that every sample entering the smoothing process will remain contributing in perpetuity, until the decay exponent eventually drops the contribution factor down to zero [25].
Let X_i be a data point enters the smoothing process. Initially, X_i contributes to the smoothing average with an initial ratio of $(1 - \alpha_i)$. With every iteration **j** that follows, X_i is then decayed by a factor $\alpha_j = e^{-\lambda(t_j - t_{j-1})}$. More specifically, for **T** iterations that follow the initial contribution of X_i, the product series of all decaying multipliers that X_i undergoes can be written in the following definition of $M(i, T)$, signaling the multipliers applied to X_i for the next **T** iterations:

$$
\begin{aligned}
M(i,T) &= \prod_{j=1}^{T} e^{-\lambda(t_{i+j} - t_{i+j-1})}. \\
\Rightarrow M(i,T) &= e^{-\lambda(t_{i+1} - t_i)} \times e^{-\lambda(t_{i+2} - t_{i+1})} \times \ldots \times e^{-\lambda(t_{i+T} - t_{i+T-1})}. \\
\Rightarrow M(i,T) &= e^{-\lambda(t_{i+1} - t_i) + (-\lambda(t_{i+2} - t_{i-1})) + \ldots + (-\lambda(t_{i+T} - t_{i+T-1}))}. \\
\Rightarrow M(i,T) &= e^{-\lambda\left[\cancelto{0}{t_{i+1}} \quad -t_i \quad \cancelto{0}{+t_{i+2} - t_{i+1} + \ldots} \quad +t_{i+T} \quad \cancelto{0}{-t_{i+T-1}}\right]}. \\
\Rightarrow M(i,T) &= e^{-\lambda(t_{i+T} - t_i)}. \\
\Rightarrow M(i,T) &= e^{-\lambda\tau}.
\end{aligned}
\tag{3.3}
$$

where τ is the time it took to go through **T** iterations. This multiplier formula gives rise to a new property of the exponential moving average that can give more control over every data point's contribution to the smoothing curve, which we will discuss in Section 3.4.2.

3.4.2 Half-Life

The half-life concept is the main reason trading firms are reliant on the exponential moving average rather than a simple, or any other moving average approach. The exponential operator's ability to convert products to addition brings a lot of power to the amount of control we have over sample contribution to the smoothing curve.
As mentioned in Section 3.4.1, as soon as a data point enters the smoothing operator, its contribution remains in perpetuity, until the factors, λ and the time differences, decay it

down to a negligible value. After every time period $\mathbf{T}$, the data point's contribution to the smooth curve decays, dropping the initial value by a factor of $e^{-\lambda T}$. Therefore, we as long as we know the duration we want to control, as well as the ratio by which we want the data point's initial contribution to drop, the only remaining decisive element is the decay factor λ, and that is where the half-life concept comes into play.

Half-life is a common concept that originated from the medical field. Half-life in the context of medical science typically refers to the elimination half-life. The definition of elimination half-life is the length of time required for the concentration of a particular substance to decrease to half of its starting dose in the body [22]. Once the half-life time period is passed, the remaining portion of the substance becomes negligible and no longer impactful on the study. Similarly, the half-life concept can be applied to the exponential moving average formula as follows. Let X_i be a data point that enters the EMA process at time t_i with an initial contribution of $(1 - \alpha_i)\ X_i$. Following the definition of half-life, we want to track the time it takes for X_i to contribute half of its original contribution value to the smoothing operation, after which its contribution becomes negligible, and no longer impactful on the smooth curve.

In this paper, we discuss the use of exponential moving averages on high frequency trading (HFT) market data. One notable thing about HFT is that data streams come in incredibly high frequency, and are measured in several streams per second, or per millisecond even [18]. Therefore, when strategies are developed for algorithmic trading, they seek to perform as many trades are possible in second-length intervals [33]. So, in that context, data-point contribution to the smoothing average should not last more than a few seconds, since the data stream is so high, that the market value is more likely to have moved past samples that entered the smoothing curve more than a few seconds before.

Let τ be the time we seek for a data point to contribute half of its original contribution to the EMA, to which we shall henceforth refer as half-life. τ is typically chosen to be anywhere between 1 to 5 seconds. Equation (3.3) shows that a data-point X_i's subsequent contribution to the EMA after the initial multiplier is written as:

$$M(i, T) = e^{-\lambda \tau}. \tag{3.4}$$

where $\mathbf{T}$ is the number of iterations until the half-life τ passes. Since we seek for half of the sample's original contribution to drop down to half, then we may write the sought contribution of X_i as:

$$M(i, T) = \frac{1}{2}. \tag{3.5}$$

By combining equations (3.4) and (3.5), we may solve for λ as follows:

$$
\begin{aligned}
e^{-\lambda\tau} &= \frac{1}{2}. \\
\Rightarrow -\lambda\tau &= \ln\frac{1}{2}. \\
\Rightarrow \lambda\tau &= \ln 2. \\
\Rightarrow \lambda &= \frac{\ln 2}{\tau}.
\end{aligned}
\tag{3.6}
$$

Therefore, by setting the half-life value and solving for λ, it is entirely possible to control the time it takes for the data point to contribute half of its original value by simply setting λ to the natural log of 2 divided by the chosen half-life duration. One main advantage of using this method is that, as long as the half-life is set, the sample contribution only depends on the time differences, and is completely independent of the number of data points that come into the stream during the set time period.
Using this method, the logic of how the smoothing works with respect to the chosen half-life value works similarly to the example shown in Figure 3.4, where the larger the half-life value, the more general the trend, and the more fluctuations are ignored in the smoothing process. Figure 3.5 shows an example, where half-lives were chosen to be $10, 20, 30, 40$, and 50 seconds, showing that the larger the half-life value, the further the curve gets from the true mid-price value.

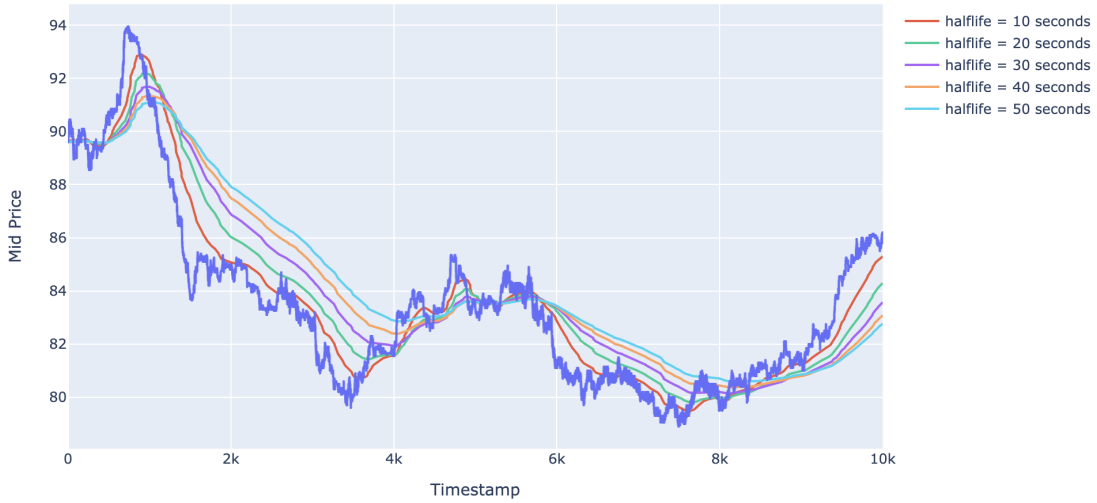


Figure 3.5: EMA Smoothing Curves With Respect To Different Half-life Values.

3.5 Reinforcement Learning

In this section, we provide a brief overview of reinforcement learning, the motivation that inspired the theory behind it, a few relevant algorithms, followed by a dive into Policy

Gradient, the optimization process we adopted in this paper.
Machine learning model-based techniques, such as supervised, unsupervised, and self-supervised techniques, are methods that rely on learning linear combinations, non-linear combinations, and representations in the input they were fed. When the model is trained on a dataset and all patterns are understood and retrieved, all outputs from the aforementioned techniques are independent of each other, and all answers are solely based on the input given at the beginning.
Reinforcement Learning (RL) is a technique where, unlike supervised learning, no label is associated with the input or starting position, and unlike unsupervised learning, the answer is not a simple combination of inputs that define a pattern within the output. Given a starting position in the training environment, reinforcement learning aims to find a pattern from the starting position that maximizes the reward associated with the moves that the agent redeems from walking around the provided environment. Every move taken by the agent within the environment depends on his current position, as well as the transitive action it took from the last position to get where it is [41].
More technically, a reinforcement learning agent interacts with an environment over time. At each time step t, the agent receives a state s_t in a state space S and selects an action at from an action space A, following a policy $\pi(a_t|s_t)$, which is the agent's behavior, i.e., a mapping from state s_t to actions a_t, receives a scalar reward r_t, and transitions to the next state s_{t+1}, according to the environment dynamics, or model, for reward function $R(S, A)$ and state transition probability $P(s_{t+1}|s_t, a_t)$ respectively. In an episodic problem, this process continues until the agent reaches a terminal state and then it restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the discounted, accumulated reward with the discount factor $\gamma \in (0, 1]$. The agent aims to maximize the expectation of such long term return from each state. The problem is set up in discrete state and action spaces. It is not hard to extend it to continuous spaces [28].

3.5.1 Deep Reinforcement Learning

In reinforcement learning, the aim is to generate a policy $\pi(a_t|s_t)$ which, given the current state of the agent, outputs the action that will best maximize the final reward. There are many different approaches developed to generate such policies, but most fall within two possible categories: **Shallow Reinforcement Learning (SRL)** and **Deep Reinforcement Learning (DRL)** [41]. Shallow reinforcement learning fits a linear function, decision tree, tile coding, or other methods to extract the best policy for reward maximization. In Shallow reinforcement learning, the policy is defined by the weight parameters output by the models. Deep Reinforcement Learning is a different category where the policy $\pi(a_t|s_t)$ is defined a deep neural network [28]. In this paper, we built a deep neural network to approximate the policy, which uses a stochastic gradient ascent approach for the optimization process. We explore the policy gradient method in more detail in the next section.

3.5.2 Stochastic Policy Gradient

Reinforcement learning techniques differ in many ways. For the purpose of this paper, the only criterion was to pick an approach that requires no knowledge of future states, or the overall environment, to create a state-to-action mapping policy. For that reason, we decided to develop our model using a stochastic policy gradient method, which, given the current agent state, generates a probability distribution over the possible actions to take. If we look at the purpose of reinforcement learning as a walk through a certain trajectory that maximizes the rewards redeemed from every step in the said trajectory, then the optimization process can be written as follows:

$$J(\theta) = \mathbb{E}_{\pi}\Big[r(\tau)\Big]. \tag{3.7}$$

where $r(\tau)$ is the final reward accumulated through the chosen path [40].
If the trajectory can be defined as a Markov Decision Process (MDP), then such a problem is guaranteed to have an optimal policy that is associated with maximum reward. In other words, being able to define the problem as a MDP means that the agent is headed for a final set of parameters θ that maximize J, and will not roam in a never-ending path. More technically, this allows us to perform a stochastic gradient ascent optimization technique to accelerate the process, which acts as an unbiased estimator for the deterministic process, and be guaranteed to reach a deterministic policy that maps actions to deterministic actions, rather than mere probability distributions [40]. Using stochastic gradient ascent, we can perform steps through the parameters using the following standard formula [43]:

$$\theta_{t+1} = \theta_t + \lambda \nabla J(\theta_t). \tag{3.8}$$

To be able to perform the gradient ascent, we need to retrieve the objective function gradient $\nabla J(\theta_t)$. To do so, we need to find the derivative of the expected value defined in Equation (3.7). The probability distribution associated with the reward redeemed from the chosen trajectory is defined by the policy generated by the trained deep neural network [29]. Therefore, the derivative of the continuous expectation is defined as:

$$\mathbb{E}_{\pi}\Big[r(\tau)\Big] = \int \pi(\tau) r(\tau). \tag{3.9}$$

To be able to find the derivative of this expectation, we must find a way around the derivative of the policy $\pi(\tau)$. For that, we use the fact that the log derivative is defined as $\nabla \log(f) = \frac{\nabla f}{f} \Rightarrow \nabla f = f \nabla \log(f)$. This defines the derivative of the reward expectation as:

$$\begin{aligned}\nabla \mathbb{E}_{\pi}\Big[r(\tau)\Big] &= \int \pi(\tau)\nabla \log\Big(\pi(\tau)\Big) r(\tau). \\ &= \mathbb{E}_{\pi}\Big[r(\tau)\nabla \log\Big(\pi(\tau)\Big)\Big].\end{aligned} \tag{3.10}$$

This is a very important extraction, as it significantly facilitates the process of computing the policy gradient from the unsolvable problem it was [40]. To elaborate, we begin by defining the policy with respect to the initial state, state probabilities throughout the trajectory, and the neural network outputs, as:

$$\pi(\tau) = p(s_1)\prod_{t}^{\tau}\pi(a_t|s_t)p(s_{t+1}|s_t,a_t). \tag{3.11}$$

The derivative of Equation (3.11) is incredibly complicated to find. However, by applying a log function, which converts multiplication into additive functions, the equation and its derivative become:

$$\begin{aligned}\log\Big[\pi(\tau)\Big] &= \log\Big[p(s_1)\prod_{t}^{\tau}\pi(a_t|s_t)p(s_{t+1}|s_t,a_t)\Big]. \\ &= \log\Big[p(s_1)\Big] + \log\Big[\sum_{t}^{\tau}\pi(a_t|s_t)\Big] + \log\Big[\sum_{t}^{\tau}p(s_{t+1}|s_t,a_t)\Big]. \\ \Rightarrow \nabla\log\Big[\pi(\tau)\Big] &= \nabla\left(\cancelto{0}{\log\Big[p(s_1)\Big]} + \log\Big[\sum_{t}^{\tau}\pi(a_t|s_t)\Big] + \cancelto{0}{\log\Big[\sum_{t}^{\tau}p(s_{t+1}|s_t,a_t)\Big]}\right). \\ &= \nabla\log\Big[\sum_{t}^{\tau}\pi(a_t|s_t)\Big].\end{aligned} \tag{3.12}$$

As the equation shows, the log derivative of the trajectory policy is reduced to the mere derivative of the sum of logs of individual policy outputs at every step, which can easily be computed in practice.
Finally, since the derivative of the expectation defined in Equation (3.10) is reduced to the expectation of the product of the trajectory reward times the log derivative of the policy of individual steps, all that remains is to estimate the expectation [38]. As stochastic processes go, the way to build an unbiased estimator of the sought policy, every epoch must consist of a significantly large sample of trajectories, which are then averaged out to represent the expectation that defines our objective function.
In practice, policy gradient is a reinforcement learning technique where we initialize an agent in a starting position within the environment, called a state. Steps are given by a

deep neural network that generates a probability distribution over the possible actions to take, from which we sample the action by following the distribution, i.e. policy. Every step the agent takes in the environment is redeemed with a reward or a penalty. When the agent reaches the end of its trajectory, we compute the log of the sampled probabilities, and apply a dot product between the log probability, and the reward vectors. The resulting value is then used for the gradient ascent step defined in Equation (3.8). After several epochs, the aim of this technique is to convert the stochastic process into a deterministic one, which is achieved by the deep neural network's convergence that outputs a deterministic probability distribution favoring one of the actions with every given state, rather than the quasi-uniform one it starts with.

3.5.3 Chosen Pipeline

In this project, a policy gradient agent is trained to evaluate our methodology. The algorithm retrieves lost information from exponential moving averages on unevenly spaced time series data against the currently used EMA method. The aim of reinforcement learning is to only use the EMA as a state, and compare the rewards redeemed from the currently used method to those redeemed from the several experiments we run.
The method used for evaluation follows an actor-environment interactive architecture, which is portrayed in Figure 3.6 below.

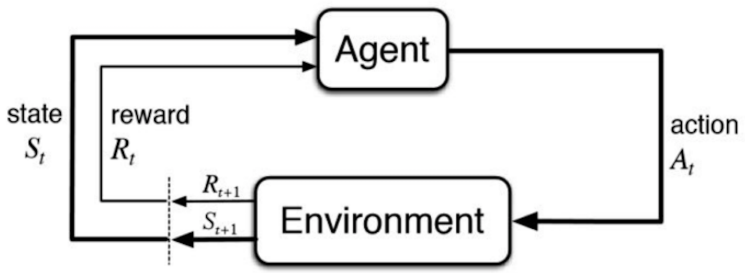


Figure 3.6: Our Strategy Evaluation Architecture. Image source [2].

The logic behind this architecture is fairly intuitive, and can be interpreted from the Figure 3.6. An agent performs one action at a time inside an environment, then a critic that holds knowledge about every state within the environment redeems the action with a reward/penalty, depending on the action taken, as well as an updated state that defines the new position of the agent. This cycle is repeated in an iterative way for the duration of a full rollout, which signals the end of the ongoing epoch [27].
For our project, the actor is defined as a deep neural network that takes a tuple of three components as its state, and outputs a probability distribution deciding whether the agent should go long, short, or hold its position in the trading platform. The critic is then given the sampled direction, performs the step, and returns the new position as well as the gain or loss that the agent redeemed from the performed trade.

3.6 Related Works

We review previous works that are related to the two components of this project: Moving averages used on unevenly spaced time series data, and uses of reinforcement learning for evaluating or creating trading strategies.
When it comes to the uses of reinforcement learning in high frequency trading, an interesting paper titled *Deep Reinforcement Learning for Active High Frequency Trading* [12], trains a reinforcement learning agent to trade one unit of Intel Corp. stock at a time, where at every step, it can either hold the position it is currently in, open a new trade if none are opened yet, or close the currently open trade. The state used in this paper uses multiple components including: the current position held by the agent, the available volumes from the top 10 levels of the bid/ask book, the last 9 tick changes from pricing data, the expected revenue to be expected from closing the currently opened trade (defaults to zero if no trades are open), and the spread between the best bidding price and the best asking price (also known as the bid/ask spread). Because the agent can not get information about any future state of the environment, and because the chosen states are continuous, this paper adopted a proximal policy method for the optimization process. The paper uses two months' worth of high frequency limit order book data, followed by one additional month for testing. To fine-tune the hyperparameters, this paper also uses a sequential model-based optimization technique, which is built to update hyperparameter values using a decay factor every time a checkpoint, specified as a condition by the user, is reached. Another paper, titled *Reinforcement Learning for High-Frequency Market Making* [30], develops a reinforcement learning agent for market making, instead of normal trading. Market making [36] is a technique used by some traders when the book spread (difference between the best bidding price and best asking price) is too high. The way these traders work is by creating a new threshold in the market by adding offers in-between the best bid & offer (BBO) without their contribution counting as a bid or an ask, which makes it possible to sell or buy such an asset, allowing the market maker to make profit using the original spread by having their asset bought, or making profit as a security option in case their asset is sold before being bought. This paper uses a discrete state for the agent they develop, which allows them to use simple dynamic programming-based Q-learning for the optimization process. Additionally, at the end of the training process, they use a *Constant Absolute Risk Aversion* (CARA) to cover the risk taken by their market-making agent. CARA is defined using the following formula:

$$R_T = \alpha - e^{-r(C_T - i_T S_T)}.$$

where α is a constant, r is a risk aversion score, C_T is the profit/loss made by the end of the trading period, and S_T is the price at which i_T stocks can be liquidated. *Reinforcement Learning for Stock Prediction and High-Frequency Trading with T+1 Rules* [44] is another paper that fixes a vital issue about current algorithmic training models, which is the lack of adaptation to current market data distributions, as well as an inability to automatically adjust to market conditions upon a shift. To handle that, this paper proposes a reverse reinforcement learning framework which regards the strategy as an action, and aims to estimate the reward associated with every action taken by the agent. For the terminal

reward enhancement used by this paper uses the *Upper Confidence Bound* [13], which is a multi-armed bandit technique used to automatically optimize the model parameters of trading logic. This technique is ideally used in real-time trading, which this paper primarily targets. Finally, *Deep Reinforcement Learning for Trading* [45] is a paper that differs from the rest by proposing a technique for using reinforcement learning to design strategies in discrete as well as continuous action spaces. This project trades in futures, and uses 50 very liquid futures contracts for their training and testing processes. The advantage of this technique is its ability to train agents using both Q-Learning, as well as policy gradient-based methods to compare performances on similar markets, and judging which is better adapted for such types of data.

When it comes to moving average techniques on unevenly spaced time series, a paper titled *Algorithms for Unevenly Spaced Time Series: Moving Averages and Other Rolling Operators* [19] handles the issue that a lot of importance has been given to smoothing operators on equally spaced time series data, with the availability of libraries and a countless number of automated processes facilitating the process, which marginalized the attention on unevenly spaced time series. This paper highlighted different smoothing techniques, including the Simple Moving Average (SMA), Exponential Moving Average (EMA), Non-Causal Rolling Operators, and other general techniques that are widely used, by not only proposing different approaches to adapt such operators for unevenly spaced data, but also proposing different versions of each averaging technique to handle different distributions within the provided data. Another paper titled *An irregularly spaced first-order moving average model* [37] suggests two rolling average techniques on unevenly spaced time series data, which are equivalent under Gaussianity. The first one relies on normally distributed data and the specification of second-order moments. The second definition provided is more flexible in the sense that it allows for considering other distributional assumptions.

Chapter 4

Methodology

In this chapter, we explore our proposed methodology, starting off with a brief description of the dataset used throughout this project, followed by a detailed breakdown of our smoothing approach to retrieve lost information from the exponential moving average, as well as a description of our two-part evaluation approach, those being the quantitative loss to evaluate the distance to the defined ground truth, in addition to the reinforcement learning agent that demonstrates the quality of each moving average as a trading signal for maximizing end-of-day profits.

4.1 Dataset

Due to intellectual property and privacy issues, the experiments of this project had to be conducted on a publicly available dataset that translates all the issues we are working on solving. In other words, the dataset was required to contain high frequency trading (HFT) data, broken down into fractions of a second per trade, where several instances had to include identical, or marginally different, adjacent timestamps.
The dataset adopted for such a purpose was extracted from the TrueFX public database [42]. This website provides historical HFT millisecond data of several currency pairs. Currency pairs are two currencies tied to one another, such as the U.S. Dollar and the Euro. Trading these pairs is possible when the value of one currency is being quoted against the other in a Forex market. By trading a currency pair unit, traders are essentially buying one currency (the base currency) while selling the other (the quote currency).[17]
For this project, we chose the historical data of the Euro-Japanese Yen (EUR/JPY) [10] currency pair for our experiments. Upon careful examination, the historical trading data of this currency pair had all the required instances, and showed a significant loss of information when an through the currently used EMA approach, which we aim to describe a detailed solution to in future sections of this paper.

4.2 Problems With The Current Method

To run a smoothing curve on noisy changes of unevenly spaced time series data, we use the EMA formula defined in Equation (1.1). In practice, the implementation for the currently used approach is detailed in Algorithm 1:

Algorithm 1 Currently Used EMA

Require: Dataset X, Half-Life τ, Size N

$\lambda \leftarrow \frac{\ln(2)}{\tau}$ ▷ λ is the Decay Factor to control the half-life
$t_{prev} \leftarrow 0$
$EMA \leftarrow 0$
for $i = 0 \rightarrow N$ **do**
    $\Delta t \leftarrow t_i - t_{prev}$
    $\alpha \leftarrow e^{-\lambda \Delta t}$
    $EMA \leftarrow (1 - \alpha)\, X_i + \alpha\, EMA$
    $t_{prev} \leftarrow t_i$
end for

As we iterate through the time series, every data point X_i enters the moving average by a fraction of $(1 - e^{-\lambda \Delta t_i})$, leaving the remaining $e^{-\lambda \Delta t_i}$ as a contribution from the running EMA. The problem with this approach, as defined in Chapter 1, rises when adjacent timestamps are identical, or differ by a epsilon-sized amount.
After in-depth analysis of the currently used method, we pinpointed the main issues with this approach, all of which are centered in the difference between adjacent timestamps. On one hand, differentiating between data point importance by using the time it takes for a new data point to join the smoothing process is a valuable aspect that should be kept. However, when dealing with HFT datasets, the frequency by which orders are sent has an incredibly high variance. In some instances, it is even possible for adjacent timestamps to be identical when several orders are sent by the same user as a batch. A scenario where that is possible is when a user simply wants to manipulate the market by sending a new order to tighten the bid/ask spread, then send a long series of order modification requests that make the order fluctuate from one book level to another, so that their order would not get bought or sold, and at the same time create an illusion of a tighter spread for new traders to add legitimate orders that tighten the spread to the new level desired by the market manipulator [23]. Other instances, where the time difference between adjacent timestamps is so small that it becomes negligible, are created by high volatility and a significantly liquid market. Traders with similar strategies would create algorithms that race for trades, therefore causing the exchange to receive orders at quasi-identical timestamps[14]. To clarify the issue behind this instance, let $\Delta t = t_i - t_{prev} = 5\ ms$, and the half-life of a data-point in the EMA curve being $\tau = 5\ s = 5,000\ ms$. The initial multiplier for the associated data point X_i enters the smoothing curve with a ratio of:

$$\begin{aligned} 1 - e^{\lambda \Delta t} &= 1 - e^{\frac{\ln 2}{5{,}000}5}. \\ &= 6.93 \times 10^{-4}. \\ &\approx 0. \end{aligned} \tag{4.1}$$

Because of this similarity of strategy, and the existence of market manipulators, the market value fluctuates through time due to several orders that directly affect the best bid and offer (BBO). However, when computing the smoothing curve, all this information is lost.
To come up with an approach to recover the information being lost, we went through several approaches, all of which were inspired by the following list of questions:

1. Figure out if the two multipliers α_i and $(1 - \alpha_i)$ must sum up to 1. If not, define a different initial multiplier $\beta \neq (1 - \alpha_i)$ that makes sure enough information is included from every data point reached.
2. For every different value of Δt_i, define a different α that makes sure enough information from every X_i is being contributed to the smoothing curve.
3. Define a different function $\alpha(i, \lambda)$ that handles the inclusion of all samples, regardless of the time difference between them.

The third option in the list above gave a rise to the approach of converting the data to an evenly spaced time series, then running a classic EMA smoothing process on the converted data by applying a chosen aggregation function on the list of data points that enter the pipeline within the fixed interval defining the time series. This method is guaranteed not to lose information of data points entering the smoothing curve through the possibility of choosing a fixed window size that sets the initial multiplier to include the desired ratio of every data point in the EMA curve (e.g. if half-life is $\tau = 5\ s$, fixed windows can be $\Delta t = 75\ ms$ to guarantee 1% of every sample contributing to the curve. While this approach is impractical in the real world, it is possible for it to serve as a valid success metric to compare any proposed methodology to. For this approach, we had to make a decision out of the list below, about the aggregate function to apply to the data that falls within the specified window:

1. For every window Δt of fixed length, take the last data point to enter the pipeline is chosen as the defining element of that window for contributing to the curve.
2. Apply an average to the ensemble of every data point entering the pipeline within a fixed window as a contribution to the curve.
3. Run an EMA process on the ensemble of data points, and pick the last data point in the EMA as a defining data point to avoid the possibility of including noisy data to define the window.

In the following section, we define the proposed method that best solves this issue. A further exploration of every item on the enumerated lists above is detailed in Chapter 5

4.3 Proposed Method

Our proposed method suggests an adjustment to the EMA algorithm being applied to HFT market data under the following requirements:

- The importance value of data points should remain judged by the time difference between adjacent timestamps.
- The higher the number of consecutive data points with identical or marginally different timestamps, the higher the importance value assigned to later data points as they define the new standard for the market value.
- At least 1% of the original value of every single data point must contribute to the EMA curve, unlike the example shown in Equation (4.1).
- When the stream of orders is high and the market is increasingly volatile, data points that entered the stream at earlier timestamps may need their importance lowered with respect to later data point in the stream.

Keeping in mind all the requirements mentioned above, the algorithm we came up with to handle this issue is detailed in the Algorithm 2:

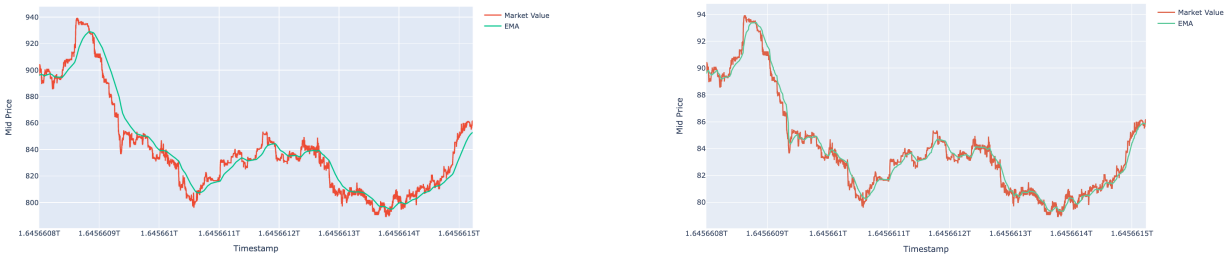
Algorithm 2 Proposed EMA

Require: Dataset X, Half-Life τ, Size N

$\lambda \leftarrow \frac{\ln(2)}{\tau}$ ▷ Decay Factor to control the half-life
$t_{prev} \leftarrow 0$
$t_{valid} \leftarrow 0$ ▷ Last valid timestamp such that $\Delta t \geq \epsilon$
$EMA \leftarrow 0$
$\epsilon \leftarrow -\frac{\ln 0.99}{\lambda}$ ▷ Minimum Δt for samples to contribute 1% value to the curve
for $i = 0 \rightarrow N$ **do**
    $\Delta t \leftarrow t_i - t_{prev}$
    if $\Delta t <= \epsilon$ **then**
        $\Delta t \leftarrow t_i - t_{valid}$
    else
        if t_{prev} equals 0 **then**
            $t_{valid} \leftarrow t_i$
        else
            $t_{valid} \leftarrow t_{prev}$
        end if
    end if
    $\alpha \leftarrow e^{-\lambda \Delta t}$
    $EMA \leftarrow (1 - \alpha)\, X_i + \alpha\, EMA$
    $t_{prev} \leftarrow t_i$
end for

The core logic of the original Algorithm 1 was maintained, with a few adjustments made to the formula for fixing the aforementioned issues. Similarly to the currently used approach, as shown in Equation (3.6), once a data point enters the smoothing curve with a ratio of $(1 - \alpha_i)$, then $\frac{\ln 2}{\tau}$ is the value assigned to the decay factor λ in order for every data point to contribute half of its value to the subsequent EMA value by the time the chosen half-life τ passes. Also, the vital importance of the two multipliers α_i and $(1 - \alpha_i)$ summing up to 1 was made clear in one of our experiments, as shown in Chapter 5. To satisfy the requirement of Equation (1.1) for the initial value $EMA(0) = X_0$, we indirectly enforce that by setting the initial timestamp $t_{prev} = 0$, which sets the first Δt to be equal to the first timestamp t_0. Because of the typically very high value of timestamps, the initial multiplier of X_0 would be $1 - e{-}\lambda t_0 = 1 - 0 = 1$. Finally, at the end of every iteration, t_{prev} is updated to the current timestamp.
As for the updates, we added a couple of new variables, which we explain in the next few paragraphs.
ϵ is a way to make sure all data points contribute the desired ratio of their original value to the smoothing curve. It defines the minimum difference between adjacent timestamps such that the initial multiplier $(1 - \alpha_i) \geq p$. ϵ is computed with respect to the decay factor λ, as well as the desired percentage of every sample that we desire to include in the EMA curve. After running several experiments and testing how smoothing curves change with respect to the minimum percentage being included in the curve, we found that, by including a minimum of 1% of every data point to the curve, we make sure that, while the curve shows no loss of information along the way, we still manage to avoid the noise within the data to achieve a smooth curve that ideally describes the overall trend of the market value data.
t_{valid} is a way to make sure adjacent timestamps are never different by less than ϵ. While t_{prev} is updated every iteration to the current timestamp, t_{valid} keeps track of the last timestamp t_{prev} associated with $\Delta t = t_i - t_{prev} > \epsilon$. That way, every time we get $\Delta t = t_i - t_{prev} \leq \epsilon$, we update Δt to take the value $t_i - t_{valid}$ instead of $t_i - t_{prev}$ to make sure no sample ever contributes less than 1% of its original value to the EMA curve.
These updates, while they bring changes to the EMA logic, preserve every sought aspect from the smoothing process. For example, let the timestamps of our dataset, showing the time taken for a data point to enter the pipeline from the market opening time, be $T = \{4\ ms, 20\ ms, 35\ ms, 36\ ms, 37\ ms, 38\ ms, 39\ ms, 40\ ms, 40\ ms, 40\ ms, 60\ ms\}$ and the halflife $\tau = 1\ s = 1,000\ ms$. Therefore, $\lambda = \frac{\ln 2}{\tau} \approx 7 \times 10^{-4}$, making $\epsilon = -\frac{ln0.99}{\lambda} \approx 14.5\ ms$.
Following the currently used approach, the values for Δt for every data point in the dataset will be $\Delta t = \{4\ ms, 16\ ms, 15\ ms, 1\ ms, 1\ ms, 1\ ms, 1\ ms, 1\ ms, 0\ ms, 0\ ms, 20\ ms\}$. These time differences have 0 data points falling outside of the minimum threshold set by ϵ, meaning their contribution to the smoothing curve will be completely negligible and therefore lost, causing a shift in the smoothing curve similar to the one shown in Figure 4.1a. Our approach, on the other hand, would verify the loss of information before every inclusion, making the set of time differences look as follows: $\{4\ ms$, $16\ ms$, $15\ ms$, $16\ ms$, $17\ ms$, $18\ ms$, $19\ ms$, $20\ ms$, $20\ ms$, $20\ ms$, $20\ ms\}$. Intuitively, this makes sense, because due the high volatility of the market, such actions or trades were likely sent as

orders because of the impact that the last valid data point has made on the market, as opposed to the recently added orders with a negligible time difference. There are lots of contributing factors to the time at which an order is received by the exchange, and the order may be due to a queue set by the exchange itself. This would mean that the trading algorithm was not aware of the orders in the queue before sending its order, giving great significance to the caching of the last valid timestamp according to a predefined epsilon, showing the minimum time it takes for a data point to have an impact on future orders entering the market. An example of a lossless curve is shown in Figure 4.1b.
When it comes to the EMA multipliers explained in Equation (3.3), no updates are being made to the definition of λ, as we want the half-life to remain a variable changing with respect to time, and for half of the initial contribution to vanish by the time τ passes. However, an adjustment has been made to the real time it takes for every sample to reach its half-life, as it now depends on the volatility of the market, and the stream of data points that enter the pipeline within every ϵ window.
As shown in Equation (3.3), once a data point enters the EMA, it runs through a series of multiplications in every subsequent iteration of value $\alpha_i = e^{-\lambda \Delta t_i}$. By the time it reaches its halflife, the data point will have contributed its value by being multiplied by a cumulative factor of $e^{-\lambda \tau}$.



(a) Current Approach with $\tau = 10\ s$.

(b) Proposed Approach with $\tau = 10\ s$.

Figure 4.1: Comparison Between Currently Used and Proposed EMA.

For regular days with low volatility, the EMA curve will resemble the currently used approach without suffering any loss of information, as most, if not all, samples will reach the exchange at a relatively slow rate, leaving enough room for Δt to be consistently higher than ϵ. On high volatility days, however, there are times where the exchange receives an abnormal stream of orders at the same time or at an incredibly high frequency. This phenomenon is handled by our approach by setting the values for Δt to be the difference between every data point in the stream and the last order received by the exchange before the start of the stream. This method deals with the following:

- The data points entering the pipeline at an early stage of the overwhelming stream contribute to the curve, but they do not have a long-lasting effect on the market value by the end of the stream, higher importance should be assigned to later samples, as they set a new standard in the market (they are assigned the highest value by the current method).

- By assigning Δt as the difference between every timestamp and the last valid one, the earlier data points in the stream get multiplied by the regular $t_j - t_{j-1}$, as well as an additional $t_{j-1} - t_{valid}$ that further decays their contribution, making them reach their half-life even before the value τ passes in real time.

- The last sample in the stream is assigned the highest importance, and would regularly contribute to the curve until its half-life is reached in real-time, as long as no other irregular streams occur.

Visually speaking, Figure 4.1 shows that our method yields great results compared to the currently used approach with the same half-life value. However, there is no existing method to quantify the true improvement made by the proposed approach, so we had to come up with our own, under the following two assumptions:

1. Quantitatively speaking, a great approach shows the trend of the pricing data in real-time by removing all the noise, without any shift in the curve as shown by Figure 4.1b.

2. An improved approach can be used as a better signal for making trades in real-time than the currently used EMA.

Keeping these criteria in mind, we propose a two-step verification that best quantifies the quality of our experiments, and shows exactly how much better each method is to the target traders.

4.4 Evaluation

4.4.1 Quantitative Metrics

To evaluate the different methods we ran our experiments on, we had to keep in mind the issue with the current method that motivated the search for a fix. When shifts occur in the overall trend, as we notice from Figure 4.1a, as the shift begins, we notice how the smoothing curve drifts away from the true market value. In other words, the EMA points seem to be significantly far from the associated true market value.
For that reason, we decided that the first metric to evaluate the quality of each moving average, including the currently used approach, is by projecting the EMA averages onto the real price values using two different metrics: a Mean Absolute Error (MAE) metric, as well as a Mean Squared Error (MSE). These two metrics are marginally different depending on the type of error they are dealing with.

- MSE is used in this project to compare our own experiments against the currently used method. The use of MSE penalizes larger errors at a much higher magnitude than smaller errors. This approach is trusted to handle the shifting issue, and make

sure that the method with the smallest error is the one that can best keep track of the overall trend shown by the market value in real-time without a latency, all while omitting any noise or outliers in the data.

- After finding an approach that outperforms the currently used method, MAE is then used to compare the different experiments we run. Once an approach is agreed to outperform the current EMA method, different experiments are run to reach an agreement on the minimum percentage sought from a data point to best smooth out the noisy data, find the most optimal ϵ value, find the possible tweaks to bring on the used α multiplier, etc. For that purpose, we do not care about the magnitude of the error as much as we care about the small improvements brought to every experiments. Therefore, a Mean Absolute Error is adopted to quantify the improvements linearly rather than quadratically.

Using the two aforementioned metrics works for comparing improvements to the currently used approach. However, a notable weakness of this method is that, by not running any smoothing operator, and simply comparing market value changes to themselves as a smoothing approach, we would receive an ensemble of errors where $MSE = MAE = 0$, meaning that the best signal to use is the noisy price itself. Of course, that is not the case, and further checks must be conducted to judge the developed signals against using the noisy data. To handle that, we kept in mind that the main reason this project exists is for traders to get better trading signals, and that a better approach than the currently used one is an approach that generates more money. Therefore, we decided to run an additional check that we explore in further detail in the Section 4.4.2.

4.4.2 Reinforcement Learning Agent Reward Comparison

The main aim of this project is to provide traders with a better curve that smooths out the pricing data without loss of information. Therefore, the ideal evaluation test for our experiments is one where a trader uses the same signals to perform trades during a time period, with the difference being the smoothing technique used to develop such signals. Because the signals used by real traders are considered intellectual property and are strictly unavailable to the public, we propose a reinforcement learning-based approach where an agent is trained to trade one unit of the public EUR/JPY dataset [42] at a time, with an aim to generate the maximum profit possible.
To keep the comparison fair, the agent's state is limited to three elements:

- Agent's Position
  - -1: Performing a short trade
  - 0: Flat position
  - $+1$: Performing a long trade
- EMA positive and negative shifts over the last W trades.

This state is purely made to evaluate the smoothing operator's ability to track the trend of the instrument being traded without making use of any additional elements that could overshadow the quality of the smoothing curve.
To make sure we avoid the risk of overfit, we train the agent on a day's worth of data, evaluate the training performance by redeeming the collected profits from running the agent through the same data used for training, then choose a different day for out-of-sample testing, which will then be used for the final comparison of different methods we picked for final testing. We detail the process undergone by the agent in form of a pseudo-code showing the rollout and optimization process, followed by the interactive environment's logic within every rollout.

Algorithm 3 Rollout

Require: Agent π_θ, Batch Size **BS**, Cap Exceeding Penalty **CP**, Dataset **X**
  $S \leftarrow (0, up, down) \times BS$ ▷ Initializes the state to pos 0 and initial EMA shifts
  $buy \leftarrow 0$ ▷ Keeps track of buying price
  $sell \leftarrow 0$ ▷ Keeps track of selling price
  $Rewards \leftarrow [\,]$
  $Log\_probs \leftarrow [\,]$
  for $i = 0 \rightarrow N$ **do**
    $p(a_i) = \pi_\theta(a_i|S_i)$
    $d \leftarrow sample(p(a_i))$ ▷ Samples direction from DNN probability distribution output
    $pos \leftarrow S[0]$
    if $abs(pos + d) > 1$ **then**
      $R \leftarrow R - CP$ ▷ Penalized when attempts to trade more than one unit
      $d \leftarrow 0$
      Reject Action
    else
      if d equals -1 **then**
        $sell \leftarrow X_i$
      else if d equals $+1$ **then**
        $buy \leftarrow X_i$
      end if
      if $abs(pos + d) < abs(pos)$ **then** ▷ An open trade has been closed
        $R \leftarrow R + (sell - buy)$
      end if
    end if
    $S \leftarrow (pos + d, up, down)$ ▷ Updates the state after action is performed
    $Rewards.append(R)$
    $Log\_probs.append(\log d)$
  end for
  Return $Rewards$, Log_probs

The pseudo-code shown in Algorithm 3 shows the process that a batch of agent undergoes in one epoch. Let's break it down.

The rollout function is given as input:

- An agent in form of a deep neural network.
- A batch size signaling the number of generated agents undergoing the rollout as part of the stochastic nature of the process.
- A penalty assigned to the agent when it attempts to trade more than one unit at a time before closing the currently open trade. In other words, the penalty is assigned when the agent tries to go to direction +1 when its current position already is +1.
- The dataset X showing the prices associated with each timestamp of the trading period.

The dataset contains the pricing data, the EMA smoothing of the noisy values, and the positive and negative shifts of the EMA over a user-specified window W. That way, the agent can have immediate access to the EMA shift elements of the state, as they are associated with the timestamp reached in the rollout process.
After initializing all elements of the state, the rollout process begins by running the state through the agent **DNN** to get the probability distribution over possible directions to take. By following the generated policy, the agent samples a direction and performs the step. The agent's step is rejected, and is penalized if the action taken puts it above the one-unit limit for simultaneously open trades. Otherwise, the reward is redeemed after a trade is closed, and depends on the P&L (profit/loss) generated from that trade. Finally, the environment updates the state of the agent according to its new position, as well as the updated EMA shifts
By the end of the trading period, the rollout function returns the list of rewards, as well as probability logs, in order to compute the objective function and perform the update step in order to update the agent's knowledge of its environment. This optimization process is detailed in Algoritm 4.

Algorithm 4 Optimization Process

Require: Agent π_θ, Epochs **n_epochs**, Batch Size **BS**, Dataset **X**, LR α, Penalty **CP**
  for $e = 1 \rightarrow n\_epochs$ **do**
    $R, LP \leftarrow Rollout(\pi_\theta, BS, CP, X)$ ▷ Gets list of rewards and log probs from rollout
    $G = -\frac{1}{BS}\sum_{i=1}^{BS}(LP_i \cdot R_i)$ ▷ Gain Function
    $\theta \leftarrow \theta + \alpha\nabla_\theta G$ ▷ Stochastic Gradient Ascent Update Step
    if $e \,\%\, 200 == 0$ **then**
      $\alpha \leftarrow \alpha \cdot 0.8$ ▷ Scheduled learning rate decay
    end if
  end for
  Return π_θ ▷ Returns a trained agent

As explained in Algorithm 3, the rollout function returns a list of rewards and log probabilities which are received by the optimization function. Using the rollout return values, we compute the objective (gain) function as a mean on the batch of actors, which we then compute the gradient of to update the agent's parameters. As a way to help with convergence, the learning rate α is decayed by a factor of 0.8 every 200 epochs.
By the end of the training and optimization process, the function returns a fully trained agent, ready to be tested on in-sample, as well as out-of-sample data to evaluate the rewards redeemed from the specified trading period, and subsequently use that to compare different generated experiments.

Chapter 5

Experiments

In this chapter, we explore the different experiments that led us to the proposed method, as well as the verification process that our method underwent to prove its superiority over the currently used approach. We approach this by visiting the enumerated lists mentioned in Section 4.2 one by one, attempting to use each item to improve on the current method, or prove why the approach is unusable and not worth exploring.
The main issue with the current EMA method is centered in the fact that when timestamps are identical or marginally different the decaying exponent α sets the exponent power to 1, making the initial multiplier $1 - e^{-\lambda\Delta t} = 1 - 1 = 0$. Such a loss of information is what we are trying to avoid. To tackle that, we attempt the first experiment detailed in Section 5.1.

5.1 Changing the Initial Multiplier $1 - \alpha$

Since the issue is centered in the first multiplier, we began by exploring the importance of the first and second multipliers summing up to 1. To change the multiplier $1 - \alpha$ with an independent multiplier β, we conducted this experiments in two parts.

5.1.1 Setting a Fixed β

To test this experiment, we had to choose a multiplier β that made sense for the data we had. To balance the ratios out, we went through our dataset, ran a standard EMA smoothing operator, and recorded the list of initial multipliers $1 - \alpha$ that the data underwent. A summary of all multipliers gave us the following result:

Table 5.1: Summary of Initial Multipliers.

Statistic	**Value**
Mean	0.005
Std.	0.013
Min	0.000
25%	0.000
50%	0.004
75%	0.004
Max	1.000

For our first trial, we attempt to balance the ratios by keeping the same algorithm set by the currently used approach, and setting the initial multiplier to be the average value shown in the summary 5.1, i.e. $\beta = 5 \times 10^{-3}$, meaning that the EMA formula would be:

$$\begin{aligned} EMA(i) &= \beta\ X_i + \alpha_i EMA(i-1). \\ &= 0.005\ X_i + \alpha_i EMA(i-1). \end{aligned} \tag{5.1}$$

Doing that gave the result shown in Figure 5.1:

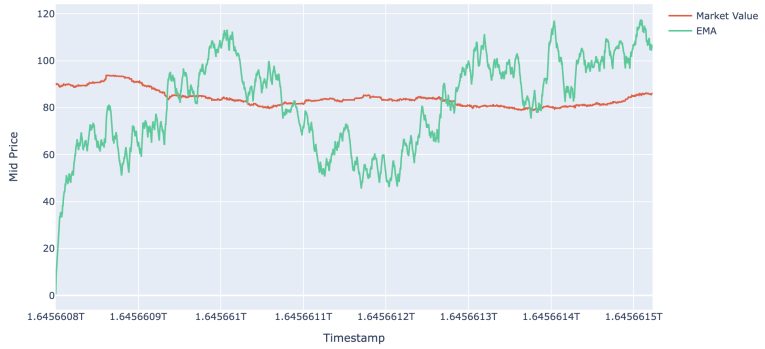


Figure 5.1: EMA with Fixed $\beta = 5 \times 10^{-3}$.

As can be seen from Figure 5.1, Equation (5.1) in no way serves to smooth out the noisy data, and rather fluctuate above and below the market value, possibly depending on the second α multiplier, and whether it goes above or below $1 - \beta = 0.995$.
From the result in Figure 5.1, this approach is clearly not the way to go, and beta can definitely not be set to a fixed value, and must be somewhat related to the second multiplier to balance the ratios out. Before submitting to summing the ratios to 1, we attempt to fix the value of β away from a fixed one. This approach is explored in the next section.

5.1.2 Setting a Dynamic β

Instead of setting a fixed β value, we had to find a way to fix the nullification of the initial multiplier in some instances. To do so, we propose the following approach.

In our experiments, we set the half-life to be $\tau = 1\ s$. Therefore making the decay factor $\lambda = \frac{\ln 2}{\tau} \approx 7{\times}10^{-4}$. As discussed in Section 4.3, the ideal ratio of a data point to be included in the EMA curve to best simulate the trend in real-time without letting any noisy data points affect the curve is 1%. Therefore, the minimum time difference Δt required for 1% for every data point to be included in the curve with the predefined values of τ and λ is $\Delta t = 15ms$. So, the formula tweak suggested in this experiment is:

$$\begin{aligned} EMA(i) &= (1 - e^{-\lambda \min(\epsilon, \Delta t_i)})\ X_i + e^{-\lambda \Delta t_i} EMA(i-1). \\ EMA(0) &= X_0. \\ \epsilon &= 15. \end{aligned} \tag{5.2}$$

Following this tweaked approach, the smoothing operator produced the plot shown in Figure 5.2:

Figure 5.2: EMA With Fixed $\beta = (1 - e^{-\lambda \min(\epsilon, \Delta t)})$.

Obviously, the approach did not produce satisfying results. However, it motivated a further tweak that showed us the direction we must take in further experiments.
While the EMA curve is divergent from the market value data, we see that it is doing much better than the one shown in Figure 5.1. The observed difference is that the value for β is much closer to α, and only differs in occasional instances when $\Delta t < \epsilon$. This makes sense because the purpose of an EMA curve is to simulate the noisy data being smoothed out. To do that, we initialize the very first EMA to $EMA(0) = X_0$ as a way to set the scale. In subsequent iterations, the purpose behind the two multipliers α and β summing up to one is because it is a way to adjust the scale according to the current direction of the price. The current method does that by taking a ratio of $1 - \alpha$ away from the current trend, and replacing it with a ratio of $1 - \alpha$ from the current data point used to update the trend. We then understood that, regardless of the method we use, we must respect the set standard and require α and β to sum up to one.
As an initial test to this, we tested Equation (5.2), with a simple tweak such that:

$$
\begin{aligned}
EMA(i) &= (1 - e^{-\lambda \min(\epsilon, \Delta t_i)})\ X_i + e^{-\lambda \min(\epsilon, \Delta t_i)} EMA(i-1). \\
EMA(0) &= X_0. \\
\epsilon &= 15.
\end{aligned}
\tag{5.3}
$$

This update gave the following plot:

Figure 5.3: EMA With Fixed $\beta = (1 - e^{-\lambda \min(\epsilon, \Delta t)})$ and $\beta + \alpha = 1$.

The figure confirms the importance of the two factors summing up to 1, but still shows a significant loss of information and complete misinterpretation of the trend, starting from the first spike, which keeps showing a rise in the smooth curve well after the real market value data began to drop. Knowing where to move forward, we began the experiments that we detail in future sections.

5.2 Conversion to Evenly Spaced Time Series

In this section, we explore a batch of experiments that were targeted towards verifying the possibility of getting an ideal smoothing curve with no loss of information. In other words, we generated an experiment that is considered "*ground truth*" and a standard to compare future experiments against.
The way this experiment works is by bypassing the time difference issue causing the nullification of some data points. In evenly spaced time series, the stream of data points is regulated and expects a data point within a fixed temporal window Δt. This means that, given a fixed window $\Delta t = w$, the equation becomes:

$$
\begin{aligned}
EMA(i) &= (1 - e^{-\lambda w})\ X_i + e^{-\lambda w} EMA(i-1). \\
EMA(0) &= X_0.
\end{aligned}
\tag{5.4}
$$

By setting the window and half-life to values of our choosing, for example $w = 10\ ms$ and $\tau = 1\ s$, we could automatically understand that it will take 100 streams of data for a sample to reach its half-life, and absolutely no samples would be lost in the process. If our assumptions are right, then applying this approach would recover any loss of information in the data, and show a perfect trend of the pricing data in real-time.
The first step to this experiment was converting our dataset into an evenly spaced ensemble from the unevenly spaced data we started with. This was easily achievable by following the steps enumerated below:

1. Create equally spaced bins of size w using timestamps from the start of your trading period to the end.
2. Map every timestamp in your data to the corresponding bin.
3. Group your data by the generated bins, and use an aggregate function to select which data is kept as a signal for every generated window. Aggregate functions depend on which data is considered a representation of the associated window, and can be chosen from the sub-list below.
    - Mean of all data points within a window as a market value representation.
    - Last order from the previous window as a current market value representation.
    - First order in the current window as a current market value representation.
4. After aggregating all windows, we may get some windows that saw no observation in the unevenly spaced stream. To fill those gaps, we implement a forward fill, where we use the last observation from the previous window as a representation of the market value in the gap period.

By following the list of steps above, we converted the unevenly spaced data stream into an evenly spaced time series. Then, by setting a half-life value of our choosing, we got the plot shown in Figure 5.4.

Figure 5.4: EMA on Evenly Spaced Time Series.

Figure 5.4 shows that, with the right algorithm, EMA smoothing plots are supposed to recover real-time trends, with the only notable difference being the absence of any noise within the data. By incrementing or decrementing the half-life value, i.e. the number of contributing iterations of every data point, the curve either removes further noise and further generalizes the smoothing result, or allows more noise into the curve, causing the EMA to be less smooth, and more resembling to the market value plot. With the knowledge of the possibility of total recovery of lost information and the existence of the hereby mentioned possibility of a perfect smoothing curve, we kept our experiments going, and came up with approaches that we detail in the following section.

5.3 Readjusting for Negligible Time Differences

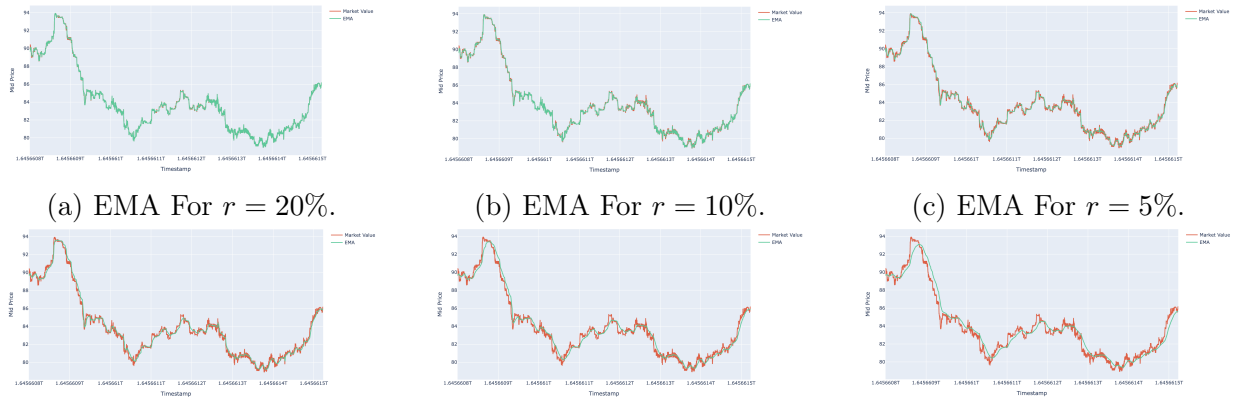
In this section, we explore the experiments that made us agree on the proposed method, as well as the minimum percentage that must be included by every data point in order to best simulate the pricing trend in real-time while discarding any noise within the data. To solve our research problem, we first have to look at the issue from a different perspective, which we chose to be the trading algorithm's perspective when performing the trade. What is meant by negligible time differences is the instances where adjacent timestamps are not affected by each other at the time of trade, in other words, it is entirely possible for a trade to be affected by a previous market value, but be associated to a completely different adjacent trade that was late to reach the exchange due to a processing time period. Therefore, this experiment aims to fix that issue by associating every trade with the latest market value that most likely affected the algorithm's judgement to make such a trade. To achieve that, we implement the following approach:

1. Define an ϵ signaling the minimum difference between adjacent timestamps for a direct impact to be considered between them.
2. Every time adjacent timestamps are different by more than ϵ, update the last valid timestamp to be the previous timestamp t_{prev} of that iteration, call it t_{valid}.
3. If $\Delta t < \epsilon$, set $\Delta t = t_i - t_{valid}$ instead of $\Delta t = t_i - t_{prev}$ in order to ensure that all timestamps used in the EMA formula are accurately associating timestamps that affect each other than mere adjacent ones.
4. The minimum ratio of every sample entering the EMA curve varies with respect to the chosen ϵ value, find the minimum ratio that ensures the best real-time tracking of the pricing trend.

Instead of setting ϵ to different constant values, we used the initial multiplier $1 - \alpha$ to define the following formula:

$$
\begin{aligned}
1 - \alpha &= r\%. \\
\Rightarrow 1 - e^{-\lambda\epsilon} &= r\%. \\
\Rightarrow e^{-\lambda\epsilon} &= 1 - r\%. \\
\Rightarrow -\lambda\epsilon &= \ln(1 - r\%). \\
\Rightarrow \epsilon &= -\frac{\ln(1 - r\%)}{\lambda}.
\end{aligned}
\tag{5.5}
$$

This formula allows us to choose a desired ratio r for every sample to contribute to the EMA curve, and automatically plug it to find the minimum value for Δt for a direct association to occur. After implementing the aforementioned algorithm, we ran a few experiments to test its viability, and choose the best value for ϵ to include the desired ratio for the sought smoothing curve. We ran several experiments with a similar half-life value of $\tau = 1\ s$ and different values of ϵ, which returned the results shown in the Figures 5.5.



(a) EMA For $r = 20\%$. (b) EMA For $r = 10\%$. (c) EMA For $r = 5\%$.

(d) EMA For $r = 2\%$. (e) EMA For $r = 1\%$. (f) EMA For $r = 0.5\%$.

Figure 5.5: Proposed EMA Method for Different Values of r ratio.

Visually speaking, the experiments shown above suggest that the EMA curve that best simulates real-time price trend while minimizing the included noise is associated with $r = 1\%$. Further examination of the retrieved results will be explored in the Chapter 6.

Chapter 6

Results & Discussion

Chapter 5 explores the different experiments we conducted to reach the proposed method, as well as different variations of our method that led to the ideal minimum ratio r to be included in the EMA for noise to be omitted while simulating a smooth trend of the studied data. In this chapter, we analyze the evaluation metrics defined in Section 4.4 that quantify our results as an error value, then compare our experiments against the currently used method using the implemented reinforcement learning agent.

6.1 Quantitative Results

As defined in Section 4.4.1, the quantitative evaluation metric is an ensemble of Mean Absolute Error (MAE) to verify the absolute distance between the EMA and the market value changes, as well as the Mean Squared Error (MSE) in order to verify the magnitude of such errors such that, the further the MSE is from the MAE, the further the EMA curve is from the market value, signaling a significant portion of large errors, as opposed to small ones.
This quantitative approach has its advantages, but it also has major drawbacks. If an experiment lets too much noise into the curve, both MSE and MAE will approach 0.0, but the curve will in no way be an improvement to the currently used approach, and will be no better than simply using the noisy data as it is. To tackle this issue, we refer back to the experiment we ran in Section 5.2 where we verified the legitimacy of our research by applying the EMA to our dataset after converting it to an evenly spaced time series. As shown on Figure 5.4, the trend is being simulated in real-time, and shows the exact result we seek from our work. Therefore, we decided to use that plot as a ground truth to compare different error metrics.
To elaborate, this section will retrieve results as follows:

1. Compute the MSE and MAE values between every EMA experiment with the market value data.

2. Convert the dataset to an evenly spaced time series

3. Run an EMA with a similar half-life (10 seconds) to the one being used by our experiments.
4. Compute the MSE and MAE values for the evenly spaced experiment.
5. Evaluate experiments by comparing the absolute distance between error values of an experiment against the error values generated by the evenly spaced method.

To begin, we list the results we got from the 6 experiments listed in Figure 5.5 as well as Figure 5.4 in Table 6.1.

Table 6.1: Ground Truth and Experiment-based MAE & MSE Values.

Method	MAE	MSE
Evenly Spaced	**0.078**	**0.011**
Currently Used Method	0.700	0.8700
Proposed Method $r = 20\%$	0.003	0.0007
Proposed Method $r = 10\%$	0.003	0.0013
Proposed Method $r = 5\%$	0.008	0.0060
Proposed Method $r = 2\%$	0.009	0.0080
Proposed Method r = 1%	**0.15**	**0.040**
Proposed Method $r = 0.5\%$	0.530	0.500

Table 6.2: Distance Between Average Experimental and Ground Truth Error Values.

Method	Ground Truth Diff.
Currently Used Method	0.740
Proposed Method $r = 20\%$	−0.042
Proposed Method $r = 10\%$	−0.042
Proposed Method $r = 5\%$	−0.037
Proposed Method $r = 2\%$	−0.036
Proposed Method r = 1%	**0.05**
Proposed Method $r = 0.5\%$	0.470

The methods that score below the MSE and MAE of the chosen ground truth are, visually speaking, letting much more noise in than needed, and defy the main purpose of the EMA smoothing technique. To choose the best ratios, we decided to pick the experiment with the smallest positive distance between the produced average error and the ground truth's average error ($\frac{MSE+MAE}{2}$). As shown in Table 6.2, the proposed method adds a hyperparameter r, producing a minimum time difference $\epsilon = -\frac{\ln(1-r)}{\lambda}$, which brings a significant improvement to the currently used approach, simulating the real-time trend of the price, removing the noise within the data, and approaching the ground truth shown by an evenly spaced time series. In order to prove that our proposed approach is not only quantitatively superior, but is also more beneficial to traders than the currently used approach, we move on to the final verification step in Section 6.2.

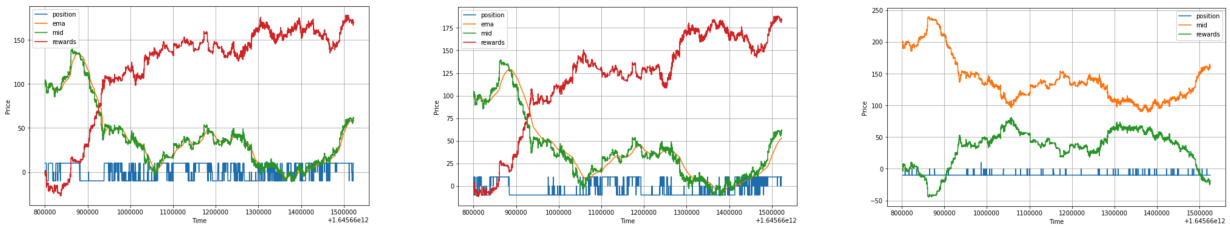
6.2 RL Agent-Redeemed Rewards

With the mean error-based evaluation approach, we understood the quantitative superiority of our proposed method over the currently used EMA approach. However, there is one additional aspect that motivated this research, and must not be overlooked. This research was made for traders to get a signal that better translates the market, and therefore providing a clearer trend of the market value in real-time and with no lags.
In order to evaluate that, we trained a reinforcement learning agent to perform trades based only on the EMA shifts over a fixed window W of past trades as a signal/state. However, we chose not to double-check all of our experiments, as that would render the quantitative approach rather useless. We learned from Section 6.1 that, out of all the experiments we ran on our proposed method, $r = 1\%$ is most ideal, and returns the smallest error when compared to the defined ground truth. Therefore, for our reinforcement learning-based evaluation, we decided to run a comparison over the following signals as states:

- Our proposed method with $r = 1\%$ and half-life $\tau = 10\ s$.
- Currently used approach with half-life $\tau = 10\ s$.
- Noisy market value data without smoothing.

For each one of these signals, we compute the shifts in the signal over the last W trades, separate the signal into positive and negative shifts, then train the agent to perform trades using each signal independently. After training is done, we test the agent on the same data they were trained on, then a batch of out-of-sample data. The different signals are evaluated against each other based on the profit/loss (P&L) redeemed by each one. We ran every signal on different window sizes to evaluate how window sizes affect the agent's ability to predict market direction for maximizing profits.
Below are results returned from training the agent on different window sizes for computing the shift in the used signal, for every one of the three methods being compared.

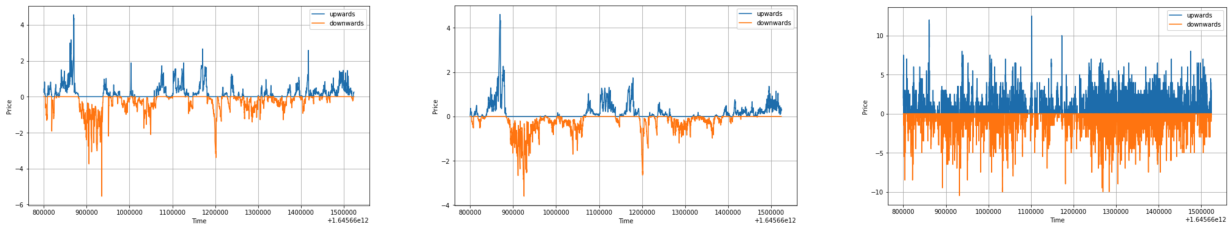
- **Figure 6.1 shows the results for an agent trained on in-sample data for $W = 10$.**



(a) Using Proposed Method. (b) Using Current Approach. (c) Using Noisy Price.

Figure 6.1: Rewards redeemed by every method on in sample data w.r.t. actions taken and price data.

- **Figure 6.2 shows the signal associated to each one of the agents of Figure 6.1:**



(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure 6.2: States associated to Figures 6.1 (a), (b), and (c), respectively on in sample data.

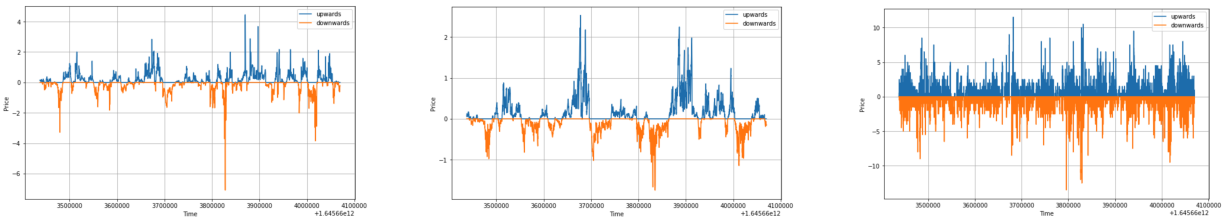
By the looks of it, the currently used approach portrayed in Figure 6.1b surpasses the proposed method in Figure 6.1a in making more profitable trades and even ending the trading period on a higher profit margin. The noisy data, however, as shown in Figure 6.2c, deals with a very uninformative signal, and ends up opening and closing short trades with an aim that the price will drop to make positive profits on the long run. So, on in-sample data, the current approach seems to be the way to go. However, when tested on unseen data, the out-of-sample rewards tell a different story.

- **Figure 6.1 shows the results for an agent trained on out-of-sample data for W = 10.**

(a) Using Proposed Method. (b) Using Current Approach. (c) Using Noisy Price.

Figure 6.3: Rewards redeemed by every method on out of sample data w.r.t. actions taken and price data.

- **Figure 6.4 shows the signal associated to each one of the agents of Figure 6.3:**



(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure 6.4: States associated to Figures 6.3 (a), (b), and (c), respectively on out of sample data.

As predicted, when using a signal that perfectly simulated the trend of the pricing data in real-time through our proposed method, the agent succeeds in redeeming significantly more rewards than when using the currently followed approach, even though the signals, as shown in Figure 6.4, seems as informative as the one we propose. This shows that the agent is misled by the shifts as they are translated by the current approach, unlike our method which keeps a quasi-perfect track of it.
We ran similar tests, on in-sample as well as out of sample data, for window sizes $5, 10$, and 50, and, just as shown in Figures 6.4, the pattern remains consistent. The figures detailing the ensemble of experiments we conducted is detailed in Appendix A, but for ease of visualization, we summarized the results we got in Table 6.3 for tests conducted on in-sample data, and Table 6.4 for tests conducted on out-of-sample data.

Table 6.3: Rewards Redeemed on In Sample Data by Different Methods Using Different Window Sizes for State.

Window Size	Current EMA Approach	Proposed Method	Noisy Price Data
$W = 5$	100	130	−10
$W = 10$	190	175	−80
$W = 50$	50	75	−20

Table 6.4: Rewards Redeemed on Out of Sample Data by Different Methods Using Different Window Sizes for State.

Window Size	Current EMA Approach	Proposed Method	Noisy Price Data
$W = 5$	−50	70	−60
$W = 10$	−20	150	−25
$W = 50$	−75	80	−60

The tables show the most outperforming method in green, and the most underperforming one in red. Despite the currently used approach either outperforming our proposed method, or at least coming close to it, when tested on in-sample data, our method proved robustness and much more real-time advantage when evaluated on out-of-sample data. Experiments run on noisy price data, however, simply show how uninformative and affected by fluctuations the state can be, which is thereby also translated on redeemed rewards when the agent is trained on such a signal as a state. Therefore, our experiments show not only the importance of smoothing methods, but also the vital advantage of having real-time trend portrayal, rather than a delayed one caused by loss of information, as is the case with the currently used EMA approach.

Chapter 7

Conclusion

This research showed us that, when signals are perceived from the exchange market's perspective, and orders are related to the latest order that most likely affected the algorithm's decision to perform a trade, signals are therefore much more significant and make just as much quantitative sense as they do in practice. As deduced from a logical perspective, our method showed significant improvements on the currently used exponential moving average algorithm on all evaluation metrics. Quantitatively speaking, comparing the current approach to the ground truth gave an error value of 0.74, whereas our method improved on that by scoring 0.05 average distance from the ground truth. Furthermore, the reinforcement learning agent evaluation step completely failed to make any profits from using noisy market value as a signal, proving the extent of importance of the smoothing process. When trained on noisy data, the agent learns from its training process to ignore the signal it is given, and simply perform long trades that therefore simulate the movement of the price itself, hoping that that would make it end on a positive note to make accidental profits. This makes sense because, as shown in Figures 6.2c, 6.4c, and Appendix A, the signal computed on noisy data with all chosen window sizes $5, 10$, and 50 simply generate noise, and are completely uninformative when used as an agent's state both on in-sample, as well as out-of-sample data. When trained on the current approach, the agent learns good patterns from the given signal, and redeems significant rewards on in-sample data; as shown in Figure 6.2, the current approach even redeems more profits than our proposed method. However, this was proven to be more of an overfit sign than it is about pattern learning when out-of-sample testing was conducted. Figure 6.4 shows that our proposed method redeems significantly more profits than the currently used approach on unseen data. This is explained by the agent simply learning patterns hinted out by the delayed trend that the current approach shows. Once trained on a completely unobserved dataset, the agent loses prospect on the trend it must follow, and thereby fails to redeem better rewards than those redeemed by using our method as a signal, which teaches the agent much more about the real-time movements of the market value, than merely teaching it patterns that show more overfit than much sought robustness.

The results shown by the quantitative evaluation approach as well as the reinforcement learning agent not only prove the importance of smoothing and omitting noise from the data being studied, but also the significance of generating a signal that simulated real-time

trends and gives every single data point the legitimate importance it holds in the smoothing process, rather than allowing needless information loss that make trends show in an EMA curve much later than it occurs in real-time.
This research was conducted for the benefit of traders because of the high-frequency data they deal with in their work, and because of the rare occurrence of identical timestamps showing up for different orders in a dataset. Beyond the trading world, it is quite obvious that this research could have significant uses outside of the trading world. Given a time series dataset with noisy data and unevenly spaced timestamps, our research helps solve a problem that is much more theoretically beneficial than it is economically. Therefore, it would be a nice direction in this project to explore the importance of real-time trend simulation, recovery of lost information, and other benefits of this approach in fields other than trading.

References

[1] Components of quantitative trading. https://www.wallstreetmojo.com/quantitative-trading/. Accessed: 2022-11-04.

[2] Continuous control with deep reinforcement learning. https://neptune.ai/blog/continuous-control-with-deep-reinforcement-learning. Accessed: 2022-11-04.

[3] Cryptocurrency and blockchain glossary. https://etorox.com/about-etorox/glossary/low-frequency-trading-lft/. Accessed: 2022-11-04.

[4] High frequency trading. https://insightincmiami.org/low-frequency-trading/. Accessed: 2022-11-04.

[5] How to use a moving average to buy stocks. https://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp. Accessed: 2022-11-04.

[6] Simple moving average. http://www.thehotpennystocks.com/learn/simple-moving-average/. Accessed: 2022-11-04.

[7] What is trading? https://www.ig.com/uk/trading-need-to-knows/what-is-trading.

[8] Harliyus Agustian, Asih Pujiastuti, and Muhammad Varian Sayoga. Comparison of simple moving average and exponential smoothing methods to predict seaweed prices. *CCIT Journal*, 13:175–184, 2020.

[9] Alpari. Alpari. https://alpari.com/en/beginner/articles/high-low-frequency-trading/, Jul 1970.

[10] Avatrade. Eur-jpy trading: Great conditions. https://www.avatrade.com/trading-info/financial-instruments-index/forex/eur-jpy#:~:text=EURJPY%20is%20the%20ticker%20symbol,pair%20of%20many%20forex%20traders., Mar 2022.

[11] Avercast. Exponential smoothing. https://www.avercast.eu/post/exponential-smoothing, Oct 2021.

[12] Antonio Briola, Jeremy D. Turiel, Riccardo Marcaccioli, and Tomaso Aste. Deep reinforcement learning for active high frequency trading. *CoRR*, abs/2101.07107, 2021.

[13] Alexandra Carpentier, Alessandro Lazaric, Mohammad Ghavamzadeh, Rémi Munos, and Peter Auer. Upper-confidence-bound algorithms for active learning in multi-armed bandits. In *International Conference on Algorithmic Learning Theory*, pages 189–203. Springer, 2011.

[14] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. *Algorithmic and high-frequency trading*. Cambridge University Press, 2015.

[15] ERNEST P. CHAN. *Quantitative trading: How to build your own algorithmic trading business*. JOHN WILEY & SONS, 2021.

[16] James Chen. High-frequency trading (hft) definition. `https://www.investopedia.com/terms/h/high-frequency-trading.asp`, Jun 2022.

[17] James Chen. What is a currency pair? major, minor, and exotic examples. `https://www.investopedia.com/terms/c/currencypair.asp#:~:text=When%20you%20buy%20a%20currency,and%20ask%20prices%20(sell).`, Sep 2022.

[18] Chi-Young Choi, Nelson Mark, and Donggyu Sul. Unbiased estimation of the half-life to ppp convergence in panel data. Working Paper 10614, National Bureau of Economic Research, July 2004.

[19] Andreas Eckner. Algorithms for unevenly-spaced time series: Moving averages and other rolling operators. In *Working Paper*, 2012.

[20] Nia Fuller. Low-frequency vs high-frequency forex trading " learn to trade the market. `https://www.learntotradethemarket.com/forex-articles/low-frequency-vs-high-frequency-forex-trading`, Jan 2021.

[21] Xin Guo. Quantitative trading: Algorithms, analytics, data, models, optimization. `https://books.google.com/books/about/Quantitative_Trading.html?id=xbTZDQAAQBAJ`, Jan 2017.

[22] Jericho Hallare and Valerie Gerriets. Half life, 2022.

[23] Larry Harris. What to do about high-frequency trading, 2013.

[24] J. Stuart Hunter. The exponentially weighted moving average. *Journal of Quality Technology*, 18(4):203–210, 1986.

[25] J Stuart Hunter. The exponentially weighted moving average. *Journal of quality technology*, 18(4):203–210, 1986.

[26] Charles M. Jones. What we know about high frequency trading. `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2236201`, Mar 2013.

[27] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[28] Yuxi Li. Deep reinforcement learning: An overview, 2017.

[29] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[30] Ye-Sheen Lim and Denise Gorse. Reinforcement learning for high-frequency market making. In *ESANN 2018-Proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 521–526. ESANN, 2018.

[31] Quentin Limouzi. Why automation is the future of equities trading. `https://www.refinitiv.com/perspectives/future-of-investing-trading/why-automation-is-the-future-of-equities-trading/`, Mar 2022.

[32] Hamlin Lovell and Andy Webb. Quant trading vs traditional trading. `https://thehedgefundjournal.com/quant-trading-vs-traditional-trading/`, Jun 2007.

[33] Albert J Menkveld. The economics of high-frequency trading. *Annual Review of Financial Economics*, 8:1–24, 2016.

[34] Albert J. Menkveld. Taking stock. `https://www.jstor.org/stable/26774066`, Oct 2022.

[35] Cory Mitchell. How to use a moving average to buy stocks. `https://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp`, Jul 2022.

[36] Maureen O'hara and George S Oldfield. The microeconomics of market making. *Journal of Financial and Quantitative analysis*, 21(4):361–376, 1986.

[37] Cesar Ojeda, Wilfredo Palma, Susana Eyheramendy, and Felipe Elorrieta. An irregularly spaced first-order moving average model, 2021.

[38] Santiago Paternain, Juan Andrés Bazerque, Austin Small, and Alejandro Ribeiro. Stochastic policy gradient ascent in reproducing kernel hilbert spaces. *IEEE Transactions on Automatic Control*, 66(8):3429–3444, 2020.

[39] Alex Pierrefeu. A new adaptive moving average (vama) technical indicator for financial data smoothing. 2019.

[40] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. PMLR, 2014.

[41] Richard S Sutton, Andrew G Barto, et al. Introduction to reinforcement learning. 1998.

[42] TrueFX. Streaming fx data. historical tick-by-tick data. `https://www.truefx.com/streaming-market-data-truefx/`.

[43] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

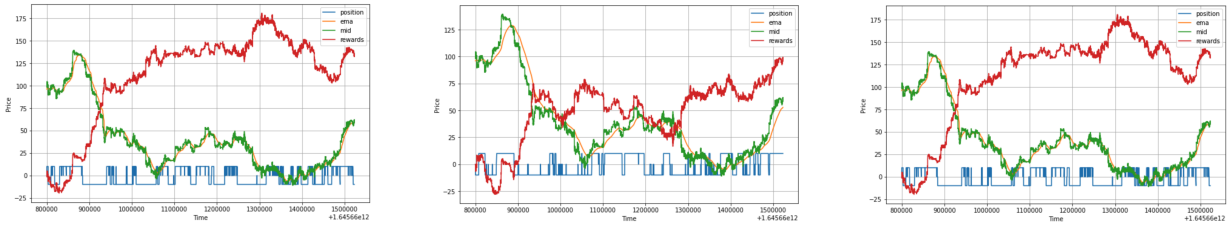
[44] Weipeng Zhang, Tao Yin, Yunan Zhao, Bing Han, and Huanxi Liu. Reinforcement learning for stock prediction and high-frequency trading with t+ 1 rules. *IEEE Access*, 2022.

[45] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2(2):25–40, 2020.

APPENDICES

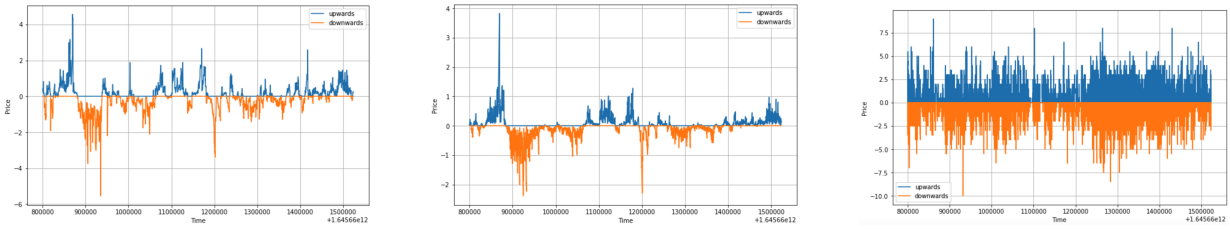
Appendix A

Further Results on Different Window Sizes



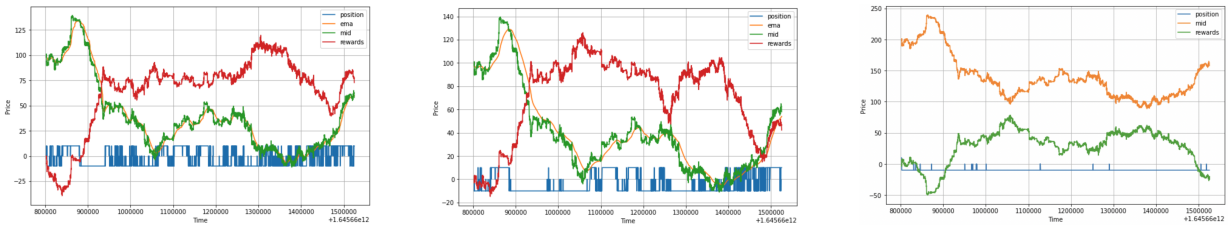
(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure A.1: Rewards redeemed by every method on in sample data w.r.t. actions taken and price data using window size 5.



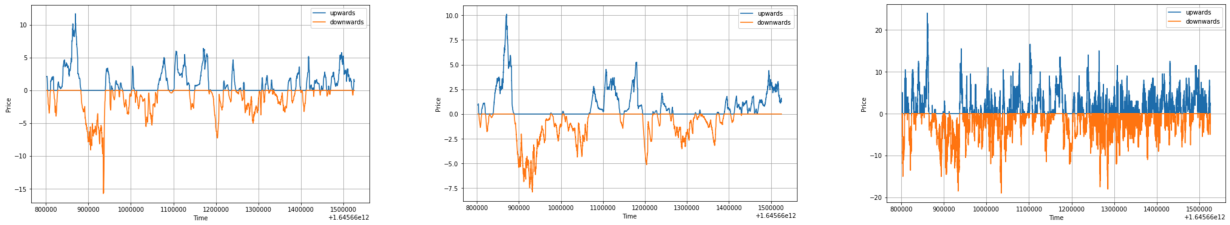
(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure A.2: States associated to Figures A.7 (a), (b), and (c), respectively on in sample data and window size 5.



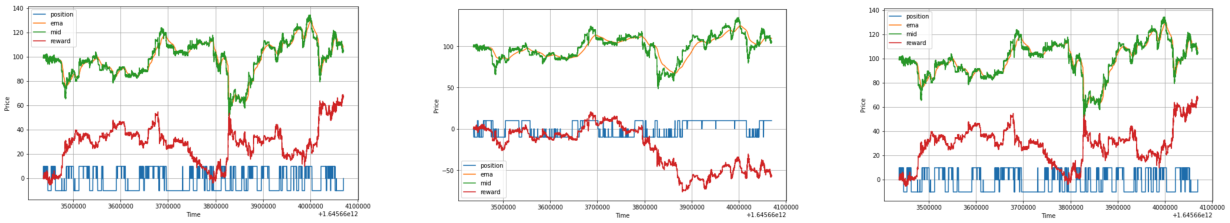
(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure A.3: Rewards redeemed by every method on in sample data w.r.t. actions taken and price data using window size 50.



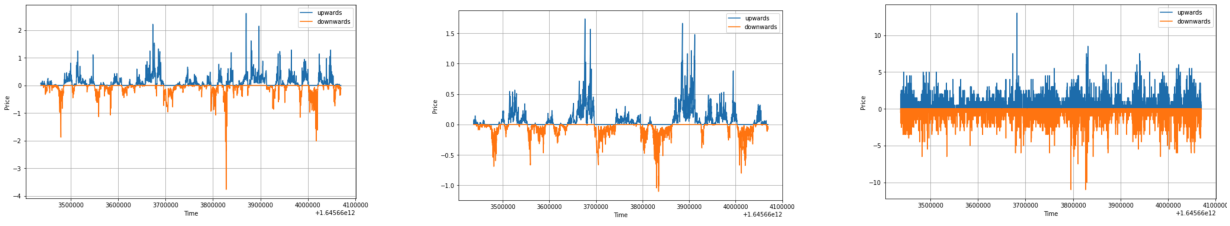
(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure A.4: States associated to Figures A.7 (a), (b), and (c), respectively on in sample data and window size 50.



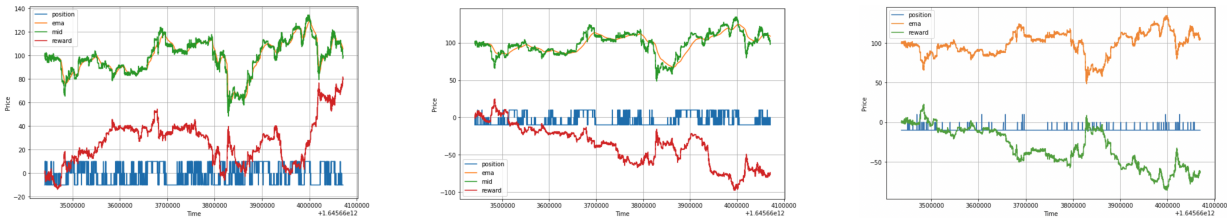
(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure A.5: Rewards redeemed by every method on out of sample data w.r.t. actions taken and price data using window size 5.



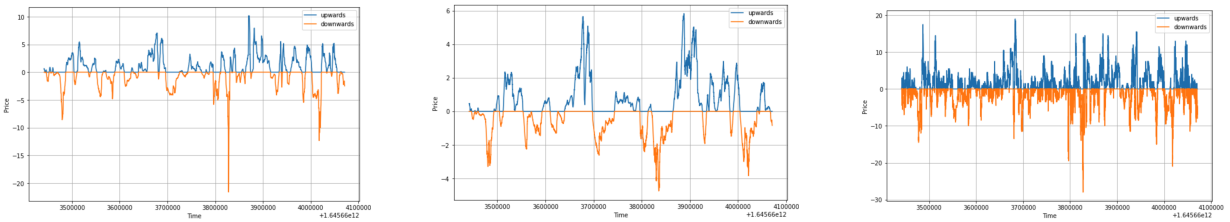
(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure A.6: States associated to Figures A.7 (a), (b), and (c), respectively on out of sample data and window size 5.



(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure A.7: Rewards redeemed by every method on out of sample data w.r.t. actions taken and price data using window size 50.



(a) Proposed Method State. (b) Current Approach State. (c) Noisy Price State.

Figure A.8: States associated to Figures A.7 (a), (b), and (c), respectively on out of sample data and window size 50.

Appendix B

Python Implementation

B.1 Libraries

```
import numpy as np
import pandas as pd
import torch
import datetime
import matplotlib
import plotly.express as px
from random import randrange
```

Listing B.1: All Libraries Imported for the Completion of This Project.

B.2 Code

```
class Old_EMA:
    def __init__(self, halflife=30):
        self.prev_timestamp = 0
        self.dt = 0
        self.eps = 0
        self.halflife = halflife
        self.lmd = np.log(2.0)/(halflife * 1e3)
        self.ema = 0
        self.alpha = 1

    def get_lmd(self):
        return self.lmd

    def get_alpha(self):
        return self.alpha

    def step(self, timestamp, price):
        self.dt = timestamp - self.prev_timestamp
        self.alpha = np.exp(- self.lmd * self.dt)
        self.ema = (1 - self.alpha) * price + self.alpha * self.ema
        self.prev_timestamp = timestamp
        return self.ema
```

Listing B.2: Currently Used EMA Algorithm.

```
class EMA:
    def __init__(self, halflife, r):
        self.prev_timestamp = 0
        self.base_timestamp = 0
        self.dt = 0
        self.halflife = halflife * 1e3
        self.lmd = np.log(2.0)/self.halflife
        self.eps = -np.log(1.0 - r)/self.lmd
        self.ema = 0
        self.alpha = 1

    def get_lmd(self):
        return self.lmd

    def get_first(self):
        return self.first_mult

    def get_alpha(self):
        return self.alpha

    def step(self, timestamp, price):

        if (timestamp - self.prev_timestamp) > self.eps:
            self.dt = timestamp - self.prev_timestamp
            if self.prev_timestamp == 0:
```

```
                self.base_timestamp = timestamp
            else:
                self.base_timestamp = self.prev_timestamp
        else:
            self.dt = timestamp - self.base_timestamp

        self.alpha = np.exp(- self.lmd * self.dt)

        self.ema = (1-self.alpha) * price + self.alpha * self.ema
        self.prev_timestamp = timestamp

        return self.ema
```

Listing B.3: Proposed EMA Algorithm.

```
class Actor(nn.Module):
    def __init__(self):
        super(Actor, self).__init__()
        self.Activation = F.leaky_relu
        self.fc1 = nn.Linear(3, 16)
        self.fc2 = nn.Linear(16, 16)
        self.fc3 = nn.Linear(16,3)

    def forward(self, state):
        state = self.Activation(self.fc1(state))
        state = self.Activation(self.fc2(state))
        action = self.fc3(state)

        action_probabilities = F.softmax(action, dim=1)

        return action_probabilities
```

Listing B.4: Neural Net Returning Action Probability Distribution Given State.

```
class MarketDecision:

    def __init__(self, batch_size, position_cap, cap_penalty, model_df):
        self.model_df = model_df
        self.batch_size = batch_size
        self.trade_period = 9990 # time allocated for trading
        self.pos_cap = position_cap
        self.cap_penalty = cap_penalty
        self.reset()
        np.random.seed(60)

    # Called every rollout to reinitialize certain variables
    def reset(self):
        self.n_open_longs, self.n_open_shorts = np.zeros(self.batch_size)
    , np.zeros(self.batch_size)

        self.closed_trade = np.zeros(self.batch_size, dtype=np.int32)
        self.short_unr, self.long_unr = np.zeros(self.batch_size), np.
    zeros(self.batch_size)
```

```
18
19          self.idx = 1
20
21          self.state = np.zeros([self.batch_size, 3])
22          self.update_state(0)
23
24      # Function to perform the step given the action.
25      def step(self, action):
26          # action: 1 = long, -1 = short, 0 = hold
27          exceeded_cap = np.where(np.abs(self.state[:,0] + action) > self.
    pos_cap, 1, 0)
28          action = np.where(np.abs(self.state[:,0] + action) > self.pos_cap
    , 0, action)
29
30          self.closed_trade = np.where(np.abs(self.state[:,0]) > np.abs(
    self.state[:,0] + action), 1, 0)
31          self.closed_trade = np.where(np.abs(self.state[:,0]) < np.abs(
    self.state[:,0] + action), -1,
32                                       self.closed_trade)
33
34          opening_penalty = self.claim_reward(action)
35
36          done = (self.idx > self.trade_period)
37
38          unrealized_pl = self.short_unr + self.long_unr
39          self.update_state(action)
40
41          pl = np.where(exceeded_cap == 1, -self.cap_penalty, 0)
42          pl = np.where(self.closed_trade == -1, -opening_penalty, pl)
43
44          return self.state, pl, unrealized_pl, done
45
46      # Updates the state
47      def update_state(self, action):
48          df_state = self.model_df.loc[self.idx]
49          self.state[:,0] += action
50          self.state[:,1] = df_state['micro_up']
51          self.state[:,2] = df_state['micro_down']
52
53      # Computes the rewards and penalties and updates open trades
54      def claim_reward(self, action):
55          trades = self.model_df.loc[self.idx]
56          mid = trades['mid']
57          prev_mid = self.model_df.loc[self.idx - 1]['mid']
58
59          closing_short = np.where((self.closed_trade == 1) & (action > 0))
    [0]
60          closing_long = np.where((self.closed_trade == 1) & (action < 0))
    [0]
61
62          self.n_open_shorts[closing_short] = self.n_open_shorts[
    closing_short] - 1
63          self.n_open_longs[closing_long] = self.n_open_longs[closing_long]
     - 1
```

```
64
65          self.short_unr = self.n_open_shorts*(prev_mid - mid)
66          self.long_unr = self.n_open_longs*(mid - prev_mid)
67
68          opening_short = np.where((self.closed_trade == -1) & (action < 0)
    )[0]
69          opening_long = np.where((self.closed_trade == -1) & (action > 0))
    [0]
70
71          self.n_open_shorts[opening_short] = self.n_open_shorts[
    opening_short] + 1
72          self.n_open_longs[opening_long] = self.n_open_longs[opening_long]
     + 1
73
74          trade_opening_penalty = (0.5*mid)/10000
75          self.idx += 1
76          return trade_opening_penalty
```

Listing B.5: Rollout Function Running an Agent Through a Full Trading Period.