

A model-based reinforcement learning framework for optimal liquidation of foreign currencies

Linwei Li



2022

A model-based reinforcement learning framework for optimal liquidation of foreign currencies

Linwei Li

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität
München

vorgelegt von
Linwei Li
aus China

München, den 29.07.2022

Erstgutacher/in: Prof. Dr. Christian Heumann

Zweitgutachter/in: Prof. Dr. Ulrich Wellisch

Tag der mündlichen Prüfung: 11.11.2022

Summary

This thesis is about solving the optimal liquidation problem of foreign currencies. Multinational companies liquidate their foreign profits in the form of foreign currency exchange (FX) to reallocate their resources within the company. Foreign currency liquidation is exchanging a certain amount of foreign currency for domestic currency at the prevailing FX rate within a liquidation period. The optimal liquidation strategy maximizes the total revenue by determining the trading timing and volume of transactions during the liquidation period. Our goal is to use algorithms to determine the optimal liquidation strategy of foreign currencies.

Forecasting the exchange rate during the liquidation period is essential in estimating the optimal liquidation strategy. The liquidation period is usually one quarter, and existing forecasting models perform sub-optimally in predicting FX rates for such a long period. For this reason, we design a novel forecasting model called Regression Prediction Network (RegPred Net) that precisely forecasts long-term FX rates and the dynamic parameters that describe their trend. We assume that the logarithmic exchange rate follows a stochastic process known as the generalized Ornstein-Uhlenbeck (OU) process, where taking logarithm keeps the value strictly positive after exponentiation. The parameters of the generalized OU process describe the dynamics of the exchange rate during the liquidation period (mean level, mean-reversion rate, volatility). We refer to these parameters as dynamic parameters. RegPred Net contains an online regression and prediction parts. The regression part can be stacked into a multilayer network that iteratively calibrates the dynamic parameters given the input series. The input sequence of the network's first layer is the logarithmic exchange rate, and the inputs of the remaining layers are calibrated parameter sequences. After the calibration of the regression part, the forecasting part processes the calibrated parameter sequences. It assumes that they remain constant in the future steps of its highest layer. The forecasting part calculates and finally forecasts the FX rate from the highest to the lowest layer. The predicted FX rate can be generated in the form of expected values of the trajectories simulated by Monte Carlo simulation. Finally, we use Bayesian optimization to find the appropriate hyperparameters for the RegPred Net that minimize the difference between the predicted and actual FX rates.

With the forecast of the exchange rate dynamics during the liquidation period, we now consider how to estimate the optimal liquidation strategy. In recent years, Reinforcement Learning (RL) has become the most popular algorithm for solving sequential decision problems. These algorithms are also preferred for estimating the optimal liquidation strategy. However, we found experimentally that the state-of-the-art RL algorithms do not perform satisfactorily on our task. Some of these methods use heuristic search to find the optimal solution, making the convergence process slow and unstable. Stochastic Dynamic Programming (SDP) can compute

the optimal policy given a complete environment model (state transition probability), but it is computationally costly. Therefore, we propose a novel RL algorithm, called Estimated Optimal Liquidation Strategy (EOLS), to solve the above problems. EOLS analyzes the optimal strategy computed by the SDP algorithm as a parametric equation, approximating the original solution and simplifying it. With the dynamic parameters predicted by RegPred Net, we can model a sufficient number of exchange rate trajectories. By evaluating the expected cumulative return on these trajectories and applying a simple grid search, we can find the optimal parameters for EOLS to determine the optimal liquidation strategy. As a model-based algorithm, EOLS requires a complete model of the environment, which we assume is determined by the dynamic parameters of the generalized OU process.

So far, the RegPred Net and the EOLS algorithm form a framework for solving the optimal foreign currencies liquidation problem. We first tested the forecasting performance of RegPred Net on three historical exchange rate data sets (EUR/CNY, EUR/USD and EUR/GBP) for 19-years. The results show that RegPred Net outperforms traditional forecasting models such as Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA), as well as deep learning models like Long Short-Term Memory (LSTM) and Autoencoder LSTM (Auto-LSTM) on 100-step daily FX rate forecasting, which reduce the Root Mean Square Error (RMSE) by 25-30%, correlation index (Pearson's R) by a factor of 2-7, and Mean Directional Accuracy (MDA) by 10%. The R-square coefficient of RegPred Net is positive, while the other algorithms are negative.

We then use the predicted dynamic parameters from RegPred Net to evaluate the liquidation performance of EOLS. We simulated FX rate trajectories with these parameters to train EOLS, SDP, Deep-Q Network (DQN), and Proximal Policy Optimization (PPO) to estimate the optimal liquidation strategy. We tested these algorithms with the same historical FX rates data set on a liquidation optimality metric, which measures the gap between the average transaction rate captured by a strategy and the minimum rate over the liquidation period. The results show that EOLS outperforms DQN and PPO by 14-22%, SDP (at a discretization number of 100) by 7.3%, and Time Weighted Average Price (TWAP) by 18.8%. It runs 44 times faster than SDP and 20 times faster than DQN and PPO. To the best of our knowledge, RegPred Net is the first regression network that focuses on long-term (over 100 steps) forecasting of FX rates. Meanwhile, EOLS is the first algorithm that utilizes a closed-form solution of the SDP strategy to achieve quasi-optimal decisions in a liquidation task. The optimal liquidation strategy framework resulting from these two algorithms has a simpler structure and is more interpretable than other methods. It achieves better performance while significantly improving computational efficiency and is more feasible for effectiveness-oriented financial liquidation tasks.

Summary

In dieser Arbeit geht es um die Lösung des Problems der optimalen Liquidation von Fremdwährungen. Multinationale Unternehmen liquidieren ihre ausländischen Gewinne in Form von Devisentausch, um ihre Ressourcen innerhalb des Unternehmens umzuschichten. Bei der Liquidation von Fremdwährungen wird innerhalb eines Liquidationszeitraums ein bestimmter Betrag an Fremdwährungen in inländische Währungen zum aktuellen Wechselkurs umgetauscht. Die optimale Liquidationsstrategie maximiert die Gesamteinnahmen, indem sie den Zeitpunkt des Handels und das Volumen der Transaktionen während des Liquidationszeitraums bestimmt. Unser Ziel ist es, mithilfe von Algorithmen die optimale Liquidationsstrategie für Fremdwährungen zu bestimmen.

Die Vorhersage des Wechselkurses während des Liquidationszeitraums ist für die Einschätzung der optimalen Liquidationsstrategie unerlässlich. Der Liquidationszeitraum beträgt in der Regel ein Quartal, und die bestehenden Prognosemodelle schneiden bei der Vorhersage von Devisenkursen für einen so langen Zeitraum suboptimal ab. Aus diesem Grund haben wir einen neuartigen Prognosealgorithmus namens Regression Prediction Network (RegPred Net) entwickelt, der langfristige Wechselkurse und die dynamischen Parameter, die ihren Trend beschreiben, präzise vorhersagt. Wir gehen davon aus, dass der logarithmische Wechselkurs einem stochastischen Prozess folgt, der als verallgemeinerter Ornstein-Uhlenbeck (OU)-Prozess bekannt ist, bei dem die Logarithmierung den Wert nach der Potenzierung streng positiv hält. Die Parameter des verallgemeinerten OU-Prozesses beschreiben die Dynamik des Wechselkurses während der Liquidationsperiode (mittleres Niveau, mittlere Umkehrrate, Volatilität). Wir bezeichnen diese Parameter als dynamische Parameter. RegPred Net enthält einen Online-Regressionsteil und einen Vorhersageteil. Der Regressionsteil kann zu einer mehrschichtigen Struktur gestapelt werden, die iterativ die dynamischen Parameter anhand der Eingangsreihen kalibriert. Die Eingangssequenz der ersten Schicht des Netzes ist der logarithmische Wechselkurs, und die Eingänge der übrigen Schichten sind kalibrierte Parametersequenzen. Nachdem der Regressionsteil die Eingangsreihen kalibriert hat, verarbeitet der Prognoseteil die kalibrierten Parametersequenzen. Er geht davon aus, dass sie in den zukünftigen Schritten seiner obersten Schicht konstant bleiben. Der Prognoseteil berechnet und prognostiziert schließlich den Devisenkurs von der höchsten bis zur niedrigsten Schicht. Die vorhergesagte FX-Rate kann in Form von Erwartungswerten der Trajektorien generiert werden, die unter Verwendung der dynamischen Parameter durch Monte-Carlo-Simulation simuliert wurden. Schließlich verwenden wir die Bayes'sche Optimierung, um die geeigneten Hyperparameter für das RegPred-Netz zu finden, die die Differenz zwischen den vorhergesagten und den tatsächlichen Wechselkursen minimieren.

Mit einer Vorhersage des Wechselkurses während des Liquidationszeitraums überlegen wir nun, wie wir die optimale Liquidationsstrategie abschätzen können. In den letzten Jahren hat sich das Verstärkungslernen (Reinforcement Learning, RL) zum beliebtesten Algorithmus für die Lösung von sequentiellen Entscheidungsproblemen entwickelt. Diese Algorithmen werden auch für die Schätzung der optimalen Liquidationsstrategie bevorzugt. Wir haben jedoch experimentell festgestellt, dass die modernsten RL-Algorithmen bei unserer Aufgabe nicht zufriedenstellend funktionieren. Einige dieser Methoden verwenden eine heuristische Suche, um die optimale Lösung zu finden, was den Konvergenzprozess langsam und instabil macht. Die stochastische dynamische Programmierung (SDP) kann die beste Strategie für ein gegebenes vollständiges Umgebungsmodell berechnen, ist aber sehr rechenaufwändig. Daher schlagen wir einen neuen RL-Algorithmus vor, der als Estimated Optimal Liquidation Strategy (EOLS) bezeichnet wird, um das oben genannte Problem zu lösen. EOLS analysiert die durch den SDP-Algorithmus berechnete optimale Strategie als parametrische Gleichung, die die ursprüngliche Lösung approximiert und vereinfacht. Mit den dynamischen Parametern, die von RegPred Net vorhergesagt werden, können wir eine ausreichende Anzahl von Wechselkurstrajektorien modellieren. Durch Auswertung der erwarteten kumulativen Rendite auf diesen Trajektorien und Anwendung einer einfachen Gittersuche können wir die optimalen Parameter für EOLS finden, um die optimale Liquidationsstrategie zu bestimmen. Als modellbasierter Algorithmus erfordert EOLS ein vollständiges Modell der Umgebung, von dem wir annehmen, dass es durch die dynamischen Parameter des verallgemeinerten OU-Prozesses bestimmt wird.

Bisher bilden das RegPred Net und der EOLS-Algorithmus einen Rahmen für die Lösung des Problems der optimalen Liquidation von Fremdwährungen. Wir haben zunächst die Prognoseleistung von RegPred Net an drei historischen Wechselkursdatensätzen (EUR/CNY, EUR/USD und EUR/GBP) für 19 Jahre getestet. Die Ergebnisse zeigen, dass RegPred Net traditionelle Prognosemodelle wie Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA) übertrifft, sowie Deep-Learning-Modelle wie Long Short-Term Memory (LSTM) und Autoencoder-LSTM (Auto-LSTM) bei der täglichen Devisenkursprognose in 100-Schritten, die den Root Mean Square Error (RMSE) um 25-30%, den Korrelationsindex (Pearson's R) um einen Faktor von 2-7 und die Mean Directional Accuracy (MDA) um 10% reduzieren. Der R-Quadrat-Koeffizient von RegPred Net ist positiv, während die anderen Algorithmen negativ sind.

Wir verwenden dann die von RegPred Net vorhergesagten dynamischen Parameter, um die Liquidationsleistung von EOLS zu bewerten. Wir simulierten eine ausreichende Anzahl von Devisenkursstrajektorien mit diesen Parametern. Mit ihnen trainierten wir EOLS, SDP, Deep-Q Network (DQN) und Proximal Policy Optimization (PPO), um die optimale Liquidationsstrategie zu ermitteln. Wir testeten

diese Algorithmen mit demselben historischen Devisenkursdatensatz anhand einer Liquidationsoptimalitätsmetrik, die den Abstand zwischen dem von einer Strategie erfassten durchschnittlichen Transaktionskurs und dem Mindestkurs über den Liquidationszeitraum misst. Die Ergebnisse zeigen, dass EOLS DQN und PPO um 14-22%, SDP (bei einer Diskretisierungszahl von 100) um 7,3% und den zeitgewichteten Durchschnittspreis (TWAP) um 18,8% übertrifft. Es läuft 44 mal schneller als SDP und 20 mal schneller als DQN und PPO. Soweit wir wissen, ist RegPred Net das erste Regressionsnetzwerk, das sich auf die langfristige (über 100 Schritte) Vorhersage von Wechselkursen konzentriert. In der Zwischenzeit ist EOLS der erste Algorithmus, der eine geschlossene Lösung der SDP-Strategie verwendet, um quasi-optimale Entscheidungen bei einer Liquidationsaufgabe zu erreichen. Der aus diesen beiden Algorithmen resultierende Rahmen für eine optimale Liquidationsstrategie hat eine einfachere Struktur und ist besser interpretierbar als andere Methoden. Er erzielt eine bessere Leistung bei deutlich höherer Recheneffizienz und ist für effektivitätsorientierte Finanzliquidationsaufgaben besser geeignet.

Acknowledgement

I would like to express my sincere gratitude to the following people for their vital guidance and help throughout my PhD. I want to thank ...

- my supervisor Prof. Dr. Christian Heumann for his continuous support of my PhD studies and related research. His tolerance and positive encouragement helped me through the most difficult periods of my research.
- my mentor Dr. Paul-Amaury Matt (Mercedes-Benz Group AG) for his tireless help with my research. His creativity has inspired a lot of my research. We discussed academic problems together, tried different approaches to solve them, experienced many failures, and finally designed the algorithms in this thesis. He is both my teacher and my friend. His attitude towards work, academics and life is worthy of lifelong learning.
- my ex-leader Alyssa Berger and current leader Lukas Jan Stroemsdoerfer (Mercedes-Benz Group AG) for giving me great freedom to do research in the company. They have been helping me as much as possible outside of research.
- Mercedes-Benz Group AG for providing me with research funding and a free working environment during these years.
- my thesis committee: Prof. Dr. Ulrich Wellisch and Prof. Dr. Volker Schmid, for their insightful comments and inspections while reviewing my dissertation.
- my friends in china (in alphabetical order: Bin, Haoxian, Jincheng, Minzhao, Nan, Sike, Senyao, Tao, Xiao, Yaru, Ye, Zhecheng) for giving me encouragement and confidence during my long research years and making me feel the power of friendship.
- Genshin Impact and Clever Fit Vaihingen. Thank them for enriching the world outside my work.
- Last but not least, my wife Qi and my son Yixin. Although writing the thesis and taking care of my baby is overwhelming for me, whenever I see my wife holding my son and they both smile at me, I feel the meaning of my life, and this picture means the world to me. I want to thank my parents, Tongpeng and Shuping. Thank you for your unwavering support of my choice to pursue a PhD. You have made me feel that there are people in this world who are always there to support you, help you, and love you unconditionally.

Contents

List of Tables	viii
List of Figures	ix
Notation	xii

I Preamble

Introduction	2
1.1 Outline	2
1.2 FX rates forecasting	5
1.3 Optimal liquidation of foreign currencies	7
Methodological background	9
2.1 General background of optimal liquidation	9
2.1.1 Stochastic processes	9
2.1.2 Foreign currency liquidation	12
2.1.3 A reinforcement learning liquidation environment	12
2.2 Deep learning for forecasting	17
2.2.1 Feedforward neural network	17
2.2.2 Recurrent neural network	19
2.2.3 Bayesian optimization	22
2.3 Reinforcement learning for optimal liquidation strategy estimation .	23
2.3.1 Fundamental algorithms in reinforcement learning	23
2.3.2 Deep reinforcement learning	31
Summary of the publications	37
3.1 RegPred Net and EOLS algorithm	37
Bibliography	39

II Publications

Forecasting foreign exchange rates with regression networks tuned by Bayesian optimization	48
Optimal liquidation of foreign currencies when FX rates follow a generalised Ornstein-Uhlenbeck process	78

III Conclusions

Liquidation performance when tested with historical FX rates	94
3.1 Experimental setting	94
3.2 Experimental results	95
Conclusion and Outlook	98
Complete list of publications	101

List of Tables

2.1	An overview of RL algorithms suitable for solving optimal liquidation tasks (1).	35
2.2	An overview of RL algorithms suitable for solving optimal liquidation tasks (2).	36
3.1	Performance and computation time of different algorithms on 3 data sets of real historical FX rate series. The dynamics of FX rate are forecasted by RegPred Net. For SDP, $N_x = N_v = T = 100$. For EOLS, $N_{pgc} = 10$. The best results are shown in bold.	96

List of Figures

1.1	The daily FX rates of EUR/CNY, EUR/USD, and EUR/GBP over 5000 days.	5
2.1	Trajectories of a Wiener process, Brownian motion, mean-reverting process and generalized OU process.	11
2.2	An illustration of state transition in MDP.	14
2.3	The interaction of agent with the MDP environment.	16
2.4	Single-layer perceptron of one neuron.	18
2.5	Multi-layer perceptron.	19
2.6	An illustration of FRNN.	20
2.7	An illustration of LSTM cell.	21
3.1	An illustration of optimal liquidation framework of foreign currency.	40
3.1	Liquidation strategies obtained by testing SDP, EOLS, TWAP, DQN and PPO with historical FX rate trajectories. The dynamics of the FX series are predicted by RegPred Net. The purple area indicates the 95% confidence interval of the prediction.	97

Notation

The next list describes several symbols that will be later used within the body of the thesis. Scalars are denoted by lowercase letters (x), while boldface lowercase letters (\mathbf{x}) denote (column) vectors and boldface uppercase letters (\mathbf{X}) denote matrices.

$\boldsymbol{\theta}$	Parameters of a function
δ	TD residual
$\eta(s)$	Average time steps spent in state s in an episode
\hat{c}	Temporary cell state of LSTM
$\hat{v}_\pi(\cdot, \mathbf{w})/\hat{v}_\mathbf{w}$	Value function estimator of weights \mathbf{w} under policy π
\mathbb{E}	Expectation of a variable
\mathbb{R}	Real number
\mathbb{V}	Variance of a variable
\mathcal{T}	Parameter space
$\mu(\cdot)$	Mean function
Φ	Cumulative distribution function of a normal distribution
ϕ	Probability density function of a normal distribution
$\rho_{\pi_\theta}(s)$	Discounted visitation probability of s under policy π_θ
$\sigma^2(\cdot)$	Covariance function
$\widehat{\nabla}_\theta J(\boldsymbol{\theta})$	Estimate of the gradient of objective function $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$
ξ	A small positive value

$A_\pi(s, a)$	Advantage function of state s , action a
c	Cell state of a LSTM
D	Data pool
$d_\pi(s)$	The fraction of time spent in state s under policy π
f, i, o	Forget gate, input gate, output gate of LSTM
h	Hidden layer or hidden state of a neural network
$J(\cdot)$	Objective function
Q	Synonym for action value function q
$S, A(s), P, R$	State, action, transition probability and reward spaces of a Markov decision process
t	Time index
U	Weight of LSTM's hidden state
x_t	A random variable at time t , e.g. the logarithm of foreign exchange rate
$X_{d/f}(t)$	Foreign exchange rate of exchanging one unit of domestic currency d for foreign currency f at time t
$*$	Best result of a policy π , state value function v or action value function q
γ	Discount rate
\hat{y}	Model's output
Ω	Sample space
π	Reinforcement learning policy
A, N, Σ	Parameters of a generalized Ornstein-Uhlenbeck process
b	Bias of neural network
d	Domestic currency
$f(\cdot)$	Function
f	foreign currency

$g(\cdot)$	Activation function
G_t	Cumulative return obtained following time t
n_t	Decision variable at time t
n_{min}, n_{max}	Minimum and maximum transaction amount allowed
p_{in}, p_{out}	Input and output size of a neural network
Pr	Probability
$q_\pi(s, a)$	Action value function of state s and action a under policy π
R_t	Amount of exchanged domestic currency at time t
s, a, p, r	State, action, transition probability and reward of a Markov decision process
T	Length of time series
v_t	Remaining volume of foreign currency at time t
$v_\pi(s)$	State value function of state s under policy π
V	Total volume of liquidation
W_t	Cumulative revenue obtained until time t
w	Weight of neural network
y	Label of a neural network's output

Part I
Preamble

1. Introduction

1.1 Outline

Multinational companies realize their overseas profits by exchanging their incomes in the form of foreign currencies for the domestic currency at prevailing foreign currency exchange (FX) rates within a specific period. This is known as the foreign currencies liquidation task. The FX rate represents the volume of foreign currency that one unit of domestic currency can be exchanged for. It changes unceasingly over time due to various uncertainties and is quite volatile. For realizing the cash flow redistribution in each quarter, the companies transfer their profits from the foreign bank account to the domestic bank account by following a liquidation strategy. An optimal liquidation strategy determines the time and volume of currencies exchanged to maximize the expected cumulative revenue over the liquidation period. Because the FX rate during the liquidation period is unknown, it is necessary to forecast it. Therefore, the tasks concerned in this thesis are 1) forecasting FX rates during the liquidation period and 2) estimating the optimal liquidation strategy given the forecasted results.

The main challenge in forecasting FX rates is that it is quite difficult to forecast them over the liquidation period, which is usually a quarter. Existing studies have only focused on forecasting time series within one or a few steps ahead, or forecasting time series with a cyclical pattern. We find that the forecasts obtained using these methods deviate significantly from the true exchange rate and do not satisfy the key information needed to estimate the optimal liquidation strategy. Another problem arises from the sub-optimal performance of the state-of-the-art optimal liquidation strategy estimation algorithms. In recent years, reinforcement learning (RL) algorithms have been effective in solving optimal sequential decision problems. Some literature using RL algorithms in combination with Deep Learning (DL) techniques (DRL) has shown their applicability to financial trading tasks [11], [60], [1]. Unfortunately, we found through our experiments that the state-of-the-art DRL algorithms perform sub-optimally on our task. Moreover, its heuristic learning process requires too much computational time to converge to sub-optimal. In order to solve the above problems, we propose two models in contribution I and II. They complement each other and form the framework for addressing the issue

of optimal liquidation of foreign currencies.

Contribution I introduces a model called Regression Prediction Network (RegPred Net) that forecasts the daily FX rate over a quarter’s liquidation period as well as the dynamic parameters describing them, where the dynamic parameters are assumed as the coefficients of a generalized Ornstein-Uhlenbeck (OU) process [21], and we assume the logarithm of FX rate follows this process (the exponentiation of logarithm ensures that the forecasted FX rate is strictly positive). RegPred Net is a regression network that uses online regression to recurrently calibrate the dynamic parameters of logarithmic FX rate series. It can be stacked multiple layers like a Artificial Neural Network (ANN) [31], with the input to each layer being the time series calibrated from the previous layer. The dynamic parameters are generated top-down by assuming that the output series of the network’s top layer remains constant over the time that will be predicted. The forecasted FX rate is expressed as the expected value of the trajectories simulated by the dynamic parameters. RegPred Net uses Bayesian optimization [36] to tune its hyperparameters to predict the optimal results. In experimental validation, we use 3 historical FX rates of 19 years as the data set to test its performance. We experimentally show that RegPred Net outperforms the traditional forecasting models such as Autoregressive Moving Average (ARMA), Autoregressive Integrated Moving Average (ARIMA), and DL forecasting models like Long Short-Term Memory (LSTM), Autoencoder-LSTM (Auto-LSTM) in terms of several metrics that measure the correlation and absolute error between the forecast and historical FX rate on 100 steps FX rate forecasting. Compare to them, RegPred Net reduces the Root Mean Square Error (RMSE) by 25-30%, improves the *Pearson’s R* by 2-7 times and the Mean Directional Accuracy (MDA) by 10%. The R-square coefficient of RegPred Net is positive, while the other algorithms are negative. To the best of our knowledge, RegPred Net is the first model-based algorithm that can predict FX rates over 100 time steps. Compared to black box deep learning models such as LSTM, RegPred Net has better interpretability, simpler structure, much fewer parameters and lower time and space complexity required to train the model. In addition, it can predict dynamic parameters that reflect trends in exchange rates over time, including the mean value, mean reversion rate and volatility, which provides decision makers with important information when dealing with sequential decision-making tasks such as liquidation or financial trading.

Contribution II introduces a model called Estimated Optimal Liquidation Strategy (EOLS) to estimate the optimal strategy for foreign currencies liquidation. We derive the EOLS algorithm by analysing the closed-form optimal strategy computed by Stochastic Dynamic Programming (SDP) [39] and simplifying it to a general parametric equation, where SDP is a model-based RL algorithm that gives the optimal solution to a sequential decision problem when the complete model

of the environment (state transfer probabilities) is known. We assume that the logarithmic FX rate follows the generalized OU process in order to compute the complete environment model. However, the high time complexity of SDP makes it difficult to apply to practical tasks, in contrast the parameters in EOLS can be estimated by a simple grid search method, reducing the complexity considerably. In contribution II, we assume that the FX rates for the liquidation period are known. We only evaluate the performance of EOLS and do not involve any forecasting. Experiments involving the estimation of optimal policies using predictions from RegPred Net will be discussed at the end of the paper. In contribution II, a dataset containing dynamic parameters calibrated in an offline manner from historical FX rate series is then used to test the liquidation performance of EOLS compared to state-of-the-art DRL models such as SDP, Deep Q-Network (DQN)[35] and Proximal Policy Optimization (PPO) [44]. We evaluate the liquidation strategies estimated by each algorithm using a metric known as liquidation optimality, which measures the difference between the average transaction rate captured by a strategy and the minimum rate in the liquidation period. The experimental results show that the liquidation strategy estimated by EOLS outperforms the DQN and PPO on the test set by 15-27% in terms of liquidation optimality, requiring on average only 5% of their computation time. Meanwhile, SDP required 44 times as much computation time to achieve similar performance as EOLS. To the best of our knowledge, EOLS is the first algorithm to use the closed-form optimal solution computed by SDP to estimate a quasi-optimal strategy for a foreign currencies liquidation task. It can be seen as a generalised approximation to the solution of SDP, and its simple structure significantly reduces the computational cost. At the same time, EOLS performs significantly better in liquidation task compared to other RL algorithms that heuristically explore optimal solutions, making it more practical in real-world liquidation applications.

At the end of this thesis, we combine RegPred Net and EOLS to form a complete framework responsible for predicting the dynamics of the FX rate during the liquidation period and estimating the corresponding optimal strategy based on the prediction results. We evaluate the performance of the framework using the dataset in contribution I. The results obtained are generally consistent with those in contribution II, which indicates that our framework has solved the optimal liquidation problem of foreign currencies.

The rest of the thesis consists of three parts, namely part I Preamble, part II Publications and part III Conclusion. Part I formally defines the optimal liquidation problem of foreign currency and introduces the background of RegPred Net and EOLS. Sec. 1.2 and 1.3 list some literature related to FX rate forecasting and optimal liquidation of foreign currency. Chap. 2 present the methodological background of the proposed algorithms. Sec. 2.1 introduces the stochastic

processes used in our methods, mathematically defines the foreign currency liquidation and an RL environment of it. Sec. 2.2 briefly introduces the state-of-the-art DL algorithms related to FX rate prediction, some of them inspired us to design RegPred Net. The Bayesian optimization algorithm is also introduced in Sec. 2.2. We discuss the RL techniques used for optimal liquidation in Sec. 2.3, including the reference algorithm SDP when designing EOLS and the state-of-the-art RL algorithm DQN and PPO, which are compared with EOLS. We summarize contributions I and II in Chap. 3 and list the full articles in part II. We give the experimental results of RegPred Net and EOLS as an overall framework and conclude this thesis in part III.

1.2 FX rates forecasting

FX rate The FX rate $X_{d/f}(t)$ is formally defined as the amount of foreign currency f that can be exchanged for one unit of domestic currency d at time t . For example, suppose the daily FX rate for the EUR/USD (Euro/US dollar) at time t is $X_{EUR/USD}(t) = 1.12$, which indicates that at that time 1 Euro can be exchanged for 1.12 dollar. The FX rate is determined on the foreign exchange market, the world's largest and most liquid financial trading market. It is open to various types of traders around the world, and the trading takes place around the clock on working days. Fluctuations in FX rates are influenced by a variety of factors at the political and economic level, such as interest rates, inflation, monetary policy, risky investments, government intervention, etc., making the exchange rates one of the most complex and difficult to predict time series. We show some examples of FX rates in Fig. 1.1. For the purposes of this thesis, we will only consider spot foreign exchange rates, that is, the exchange rate at the current moment in time.

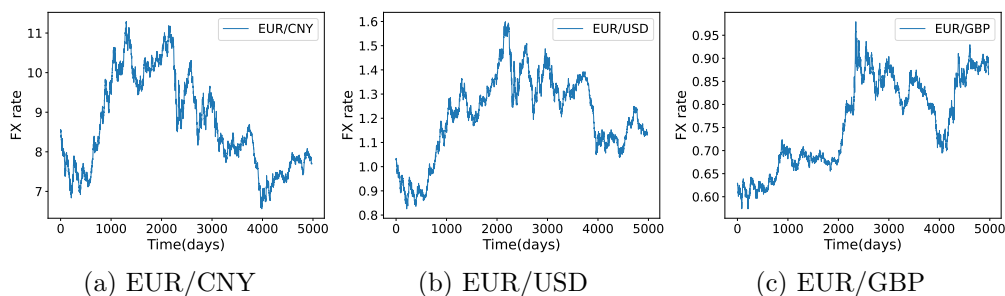


Figure 1.1: The daily FX rates of EUR/CNY, EUR/USD, and EUR/GBP over 5000 days.

With the success of DL algorithms in processing time series after 2000, more and more research has focused on testing the performance of different DL models in financial time series forecasting. ANN as the most basic DL algorithm was shown in [37] to outperform linear autoregressive and random walk models in both in-sample and out-of-sample one-step-ahead FX rates forecasting in terms of accuracy and correlation metrics.

[47] introduces the ensemble learning algorithm Bagging [7] to FX rate forecasting. They show that the Bagging algorithm with the LSTM as a base learner is more accurate in forecasting one-step foreign exchange rates (USD against EUR, GBP, CNY and JPY) than ANN, Autoregressive Moving Average (ARMA) [32] and other benchmark models. They also show that the predicted FX rates are more profitable than others when simulating FX trading. LSTM is one of the most representative DL models for dealing with sequence data. It maintains the short-term and long-term memory to preserve features extracted from sequences. A detailed explanation of the LSTM can be found in Sec. 2.2.2.

[9] introduced a coupled LSTM structure to forecast one-step-ahead FX rates. They use two LSTMs to process market and macroeconomic variables separately. The output is then fed to a new LSTM to generate FX rates. The experimental results show that the proposed deep coupled LSTM outperforms the Autoregressive Integrated Moving Average (ARIMA) [4], Convolutional Neural Network (CNN) [26] and single LSTM in several statistical metrics, e.g. Diebold-Mariano (DM) test, Pesaran-Timmermann (PT) test, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). More literature on forecasting foreign exchange rates with different DL models can be found at [16] [37], [14], [45].

Some other studies have worked on multi-step time series forecasting [29], [61]. They focus on forecasting time series that are intrinsically periodic (e.g. wind speed, river flow) and have a prediction horizon of less than five steps. Existing research on long-term forecasting (100-steps or more) of complex financial time series (e.g. foreign exchange rates) is still scarce. Some of the most advanced forecasting models (ARIMA, LSTM) perform unsatisfactorily and LSTMNet [25], which has been proposed in recent years, performs even worse than Autoregression (AR) in this task. For our liquidation task, the short-term FX rates predicted by these models contain limited information about future movements. Such forecasting results do not help to solve the optimal liquidation problem. It becomes necessary to design an algorithm that is specifically for long-term FX rate forecasting.

The RegPred Net we propose in contribution I aims to address the above shortcomings. It can predict the daily FX rate over 100 steps and the dynamics of FX rate during the liquidation period in the form of OU process parameters. These dynamic parameters are critical for the EOLS algorithm to estimate the optimal liquidation strategy.

1.3 Optimal liquidation of foreign currencies

Optimal liquidation Optimal trade execution is the general term for acquiring or liquidating a certain amount of assets within a limited period while minimizing execution costs or maximising total returns under price uncertainty. This thesis focuses on solving one of the cases of optimal trade execution where the trading method is liquidation (sale), and the assets are in foreign currencies. This is known as optimal liquidation of foreign currencies. In order to reduce the risk to cumulative returns from the uncertainty of the FX rate, decision-makers usually prefer to sell foreign currency in tranches during the liquidation period.

[3] is an early literature that systematically defines and solves the optimal liquidation problem. They assume that prices follow a simple linear model with temporary and permanent market impacts. They aim to minimize the combination of the expected value and variance of the implementation shortfall $\mathbb{E} + \lambda\mathbb{V}$, where the implementation shortfall measures the difference between the initial trading volume and the volume captured at the end of the liquidation period, and λ is a risk aversion parameter that controls the risk preference of the decision-maker. They provide an analytical, closed-form solution to the optimal liquidation problem. Different settings of λ will give the corresponding optimal liquidation strategy with different risk preferences.

Nowadays, as DL and RL techniques have matured, research on solving optimal liquidation problems has gradually shifted to these methods. These non-model-based approaches use deep ANNs to learn features from large amounts of financial data, which have the potential to yield better liquidation strategies compared to traditional models. [19] proposed a RL extension of [3]. They trained a RL algorithm called Q-learning (see Sec. 2.3.1) to learn the closed-form optimal solution computed by [3]. The algorithm improves the trading performance by an average of 10% when tested on three different stock datasets compared to the solution in [3]. [11] presents a DRL model for solving optimal financial trading tasks. The model uses the DL method to extract features from market data and train RL agents to estimate optimal strategies based on these features. They test the model's performance on a dataset including stocks and commodity futures using the Sharpe ratio and Profit & Loss as metrics. The results show that the proposed DRL model outperforms the RL model that does not use DL for feature learning. Similar works have been done in some other literatures to solve the problem with various DRL models [60], [1], [5]. We will introduce some DRL algorithms in Sec. 2.3.2.

As stated in the previous section, the existing state-of-the-art DRL algorithms perform sub-optimally in our liquidation task. The uncertainty of the FX rate leads to the unstable convergence of these algorithms during the training process.

There is still a gap between their estimated strategies and the theoretical optimal solutions computed by SDP, and all these algorithms are computationally inefficient. The application of DRL models in the area of optimal trade execution is still in the exploratory stage, and there are obvious drawbacks that need to be addressed: 1) poor interpretability of black-box algorithms like DL, 2) high computational time complexity, and 3) convergence is not guaranteed. The above reasons motivated us to design the EOLS algorithm.

2. Methodological background

2.1 General background of optimal liquidation

In this section, we present the general background of the framework for solving optimal liquidation. We first describe the generalised OU process and other related stochastic processes in section Sec. 2.1.1. We formally define the task of foreign currency liquidation and some constraints on it in Sec. 2.1.2. In Sec. 2.1.3, we briefly explain the background of a *Markov Decision Process* (MDP) [20] and RL. We also define an RL environment for the foreign currency liquidation task.

2.1.1 Stochastic processes

Since the FX rate $X_{d/f}(t)$ is always positive, we denote the logarithm of $X_{d/f}(t)$ by x_t and assume x_t follows a generalized OU process. x_t can be any real value and the FX rate $X_{d/f}(t)$ is computed by exponentiating x_t .

Stochastic process A stochastic process describes phenomena or experiments where the observed results are close to random, such as molecular motion, the spatial distribution of radiation, and changes in financial time series. It is also known as *random function*, indicating that the result x of some experiment is a function of the time variable t , denoted by $\{x(t), t \in T\}$. A formal definition of a stochastic process is given in Definition 1,

Definition 1 (stochastic process [28], p. 11). *A stochastic process with parameter space \mathcal{T} is a family*

$$\{x(t), t \in \mathcal{T}\}$$

of random variables, defined on a sample space Ω . If \mathcal{T} is a real line, the process is said to have continuous time. If \mathcal{T} is a sequence of integers, the process is said to have discrete time, and it is called a random sequence or time series.

The *sample space* Ω refers to all possible outcomes of a statistical experiment. A discrete random process is often denoted as $\{x(t), t = 0, 1, \dots\}$ and single $x(t)$ without the brackets represents a random variable. In this thesis, we simply write $x(t)$ as x_t .

Discrete Wiener process The discrete Wiener process [30] is a real-valued stochastic process with stationary and independent increments. It is often used in applied mathematics and finance field. The one-dimensional discrete Wiener process w_t has the following properties: i) $w_0 = 0$, ii) the increments $\{\Delta w_t, t = 1, 2, \dots\}$ are independent, where $\Delta w_t = w_t - w_{t-1}$, iii) the increment Δw_t follows the normal distribution $\mathcal{N}(0, 1)$. The Wiener process is also known as Brownian motion because it simulates Brownian movement in liquids.

Discrete Brownian motion with drift We say that the Brownian motion x_t has a drift μ when its increment has a constant mean $\mu \in \mathbb{R}$. The properties of x_t are defined by i) $x_0 \in \mathbb{R}$, ii) the increment $\Delta x_t = x_t - x_{t-1} = \mu + \sigma \Delta w_t$ for any $t = 1, 2, \dots$. $\sigma > 0$ is a parameter known as volatility, which controls the magnitude of noise in the process, and iii) Δx_t follows the normal distribution $\mathcal{N}(\mu, \sigma^2)$. Drift μ is the tendency of the stochastic process to change over time. For example, $\mu > 0$ means that x_t increases with t and vice versa. If the drift $\mu = 0$ and the volatility $\sigma = 1$, then Brownian motion with drift is a Wiener process.

Ornstein–Uhlenbeck process An OU process or *mean-reversion process* is a stochastic process commonly used in financial mathematics in which the random variable x_t fluctuates around a mean level over time. It is also a stationary Gauss-Markov process that satisfies the properties of both Gaussian and Markov processes. OU process is defined by the following properties: i) $x_0 \in \mathbb{R}$, ii) the increment $\Delta x_t = \alpha(n - x_{t-1}) + \sigma \Delta w_t$ for any $t = 1, 2, \dots$, where $\alpha > 0$ represents the mean reversion rate and n is the long-term mean level. The OU process describes the dynamics of the variable x_t , which returns to its mean level n over time, with an amplitude controlled by σ and a rate of α .

Generalized Ornstein–Uhlenbeck process The OU process can be generalized as i) $x_0 \in \mathbb{R}$, ii) the increment $\Delta x_t = Ax_{t-1} + N + \Sigma \Delta w_t$ for any $t = 1, 2, \dots$, where A, N, Σ are real-valued parameters. The generalized OU process includes both the Brownian motion of drift and the mean-reversion process. It is equivalent to Wiener process when $A = N = 0$ and $\Sigma = 1$. When $A = 0$, the generalized OU process is a discrete Brownian motion with drift $\mu = N$ and volatility $\sigma = \Sigma$. When $A = -\alpha$, $N = \alpha \cdot n$ and $\Sigma = \sigma$, it is a mean-reversion process with mean-reversion rate α , mean-reversion level n and volatility σ . The generalised OU process can describe the short- and long-term characteristics of FX rate time series. The FX rate can be regarded as a discrete Brownian motion with drift in the short term and mean-reversion process in the long term. We give the examples of Wiener process, Brownian motion, mean-reverting process and generalised OU process in Fig. 2.1, respectively. The parameters A, N, Σ of the generalized OU process can

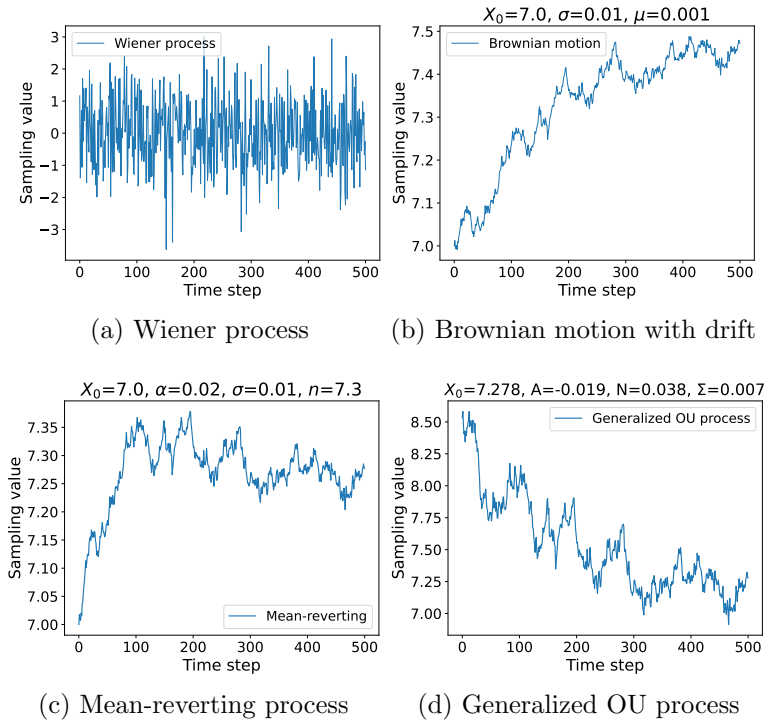


Figure 2.1: Trajectories of a Wiener process, Brownian motion, mean-reverting process and generalized OU process.

be calibrated from an FX rate time series offline and online. The offline calibration method that uses *ordinary least squares*, the online calibration approach that uses *gradient descent* [40] and *exponential moving average* are discussed in contribution I.

Gaussian process We refer to the definition of a *Gaussian process* as follows,

Definition 2 (Gaussian process [38], p. 13). *A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

Specifically, assume a stochastic process consists of a set of random variables $\{x_{t_1}, x_{t_2}, \dots\}$ indexed by time $t \in \mathcal{T}$. It is Gaussian if and only if any set of finite combination of variables $\{x_{t_1}, \dots, x_{t_n}\}$ follows a multivariate joint Gaussian distribution, where $n \in \mathbb{R}$.

2.1.2 Foreign currency liquidation

Assume the liquidation period is $t = 0, \dots, T$, where T is the total length. The foreign currency liquidation task needs to trade foreign currency f with a total volume of V into domestic currency d in T steps. The decision variable n_t denotes the amount of foreign currency f to be exchanged between time interval $(t, t + 1)$, where $()$ denotes the exclusion of time t and $t + 1$. Thus, the amount of domestic currency R_t to be exchanged is calculated by,

$$R_t = \frac{n_t}{\exp(x_t)} \quad (2.1)$$

the remaining volume v_t at time t is defined as,

$$v_t = V - \sum_{j=0}^{t-1} n_j, \quad t = 1, \dots, T \quad (2.2)$$

where $v_0 = V$. The list $[v_0, \dots, v_T]$ is called liquidation trajectory. The liquidation task requires that all volumes V should be traded before the last time step T of the liquidation period, thus $n_{T-1} = v_{T-1}$ and $v_T = 0$. Usually when liquidating foreign currencies, multinational companies agree with their partner banks on the minimum and maximum transaction amounts (n_{min} and n_{max}). Thus, for time steps $t = 0, \dots, T - 1$, the volume of transactions follows the constraint that either $n_t = 0$ or $n_{min} \leq n_t \leq n_{max}$. Based on the FX rate at time t , the liquidation strategy determines the amount of foreign currency n_t that needs to be exchanged in the time interval $(t, t + 1)$. The model for estimating the optimal liquidation strategy aims to maximize the expected cumulative revenue $\mathbb{E}(W_T)$ over the liquidation period, where the cumulative revenue W_t at arbitrary time t is denoted by,

$$W_t = \sum_{j=0}^{t-1} R_j \quad (2.3)$$

and

$$\mathbb{E}(W_T) = \mathbb{E}\left(\sum_{j=0}^{T-1} R_j\right) \quad (2.4)$$

Ideally, an optimal liquidation strategy would exchange all volumes V at the smallest FX rate during the liquidation period, and similarly, a worst case strategy would trade at the largest FX rate.

2.1.3 A reinforcement learning liquidation environment

In this section we describe how to convert the liquidation problem from Sec. 2.1.2 into an suitable environment for applying RL algorithms. First we introduce MDP

and RL.

Markov process A Markov process [15] is a stochastic process that describes the state transitions (changes) of a task. The process has the Markov property of being "memoryless": the probability distribution of the next state is determined only by the current state. Assume a discrete Markov process is a set of random variables $\{x_1, x_2, \dots\}$, the Markov property can be expressed as,

$$\Pr(x_{t+1} = s | x_1 = s_1, x_2 = s_2, \dots, x_t = s_t) = \Pr(x_{t+1} = s | x_t = s_t)$$

where s is a state at $t + 1$ and $\{s_1, s_2, \dots\}$ are a set of states.

Markov decision process MDP is a mathematical framework that is considered to be an extension of Markov processes for modelling discrete-time decision processes. An MDP is defined by the 4-tuple $(S, A(s), P, R)$, where $S = \{s, s', \dots\}$ denotes the *state* space including a set of states s, s' . A state s contains a set of variables that describe the environment at a given moment. $A(s) = \{a, a', \dots\}$ is the set of actions available to the decision maker in a given state s . a, a' denote any different actions in $A(s)$. $P = \{p(s'|s, a), \dots\}$ represents a set of *state transition* probabilities. $p(s'|s, a) = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the conditional probability that state s transitions to s' after choosing action a at time t . $R = \{r(s'|s, a), \dots\}$ is a set of *rewards*, $r(s'|s, a)$ represents the reward obtained from the environment at time t after performing action a in state s and then the state transitions to s' .

At each time t , the decision maker observes state s and selects action a from the action space $A(s)$ for execution. The state s then transitions to the next state s' with probability $p(s'|s, a)$ and the decision maker receives a reward $r(s'|s, a)$ in return. Fig. 2.2 shows the state transition of MDP. The example state s has three transitions after taking action a : 1) s transitions to s' with probability $p(s'|s, a)$ and receives reward $r(s'|s, a)$, 2) s transitions to s'' with probability $p(s''|s, a)$ and receives reward $r(s''|s, a)$, 3) s stays unchanged with probability $p(s|s, a)$ and obtains reward $r(s|s, a)$. Throughout the rest of the thesis, $r(s'|s, a)$ will be abbreviated to r . Only some special cases will be explained accordingly.

Reinforcement learning [49] RL refers to a set of computational methods that learn by interacting with the environment of a given task. They maximize the cumulative rewards gained from the environment by improving the strategies used for action selection. The *environment* fully describes the task and is usually defined by an MDP. The decision maker is often referred to as the *agent*. Fig. 2.3 shows how the agent interacts with an MDP environment. The interaction over time generates a sequence:

$$\{s_0, a_0, r_0, s_1, \dots, s_t, a_t, r_t, s_{t+1}, \dots\}$$

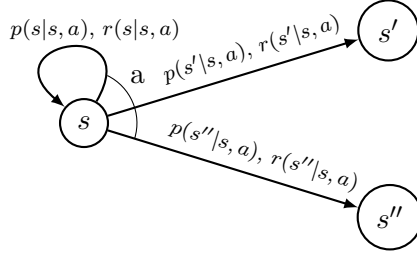


Figure 2.2: An illustration of state transition in MDP.

For tasks with an end condition, this sequence has a finite length and ends at the terminal time step T . Such tasks are referred to as episodic tasks. Tasks where the agent can interact with the environment indefinitely are referred to as non-episodic tasks. For episodic tasks, the cumulative reward or return G_t received by the agent after time t is defined as,

$$G_t := r_t + r_{t+1} + \dots + r_{T-1} \quad (2.5)$$

for non-episodic tasks, a discount rate γ is used to prevent the cumulative rewards from becoming infinite,

$$G_t := r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k} = r_t + \gamma G_{t+1} \quad (2.6)$$

The mapping that determines the probability of choosing action a in a given state s is called *policy*, denoted as $\pi(a|s)$. The expected cumulative reward that an agent collects from state s by following policy π is referred to as the state-value function $v_\pi(s)$,

$$\begin{aligned} v_\pi(s) &:= \mathbb{E}_\pi[G_t | s_t = s] \\ &= \mathbb{E}_\pi[r_t + \gamma G_{t+1} | s_t = s] \end{aligned} \quad (2.7)$$

where $s \in S$. We assume that all other mentions of s, s' in this thesis belong to state space S and that all actions a given state s belong to action space $A(s)$.

Similarly, the action-value function $q_\pi(s, a)$ is the expected return to the agent for taking action a in state s and following policy π ,

$$\begin{aligned} q_\pi(s, a) &:= \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_t + \gamma G_{t+1} | s_t = s, a_t = a] \end{aligned} \quad (2.8)$$

$v_\pi(s)$ in Eq. 2.7 can be expanded in Eq. 2.9 in terms of the value function $v_\pi(s')$

of its possible successor state s' ,

$$\begin{aligned}
v_\pi(s) &:= \mathbb{E}_\pi[r_t + \gamma G_{t+1} | s_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r | s, a) \left[r + \gamma \mathbb{E}_\pi[G_{t+1} | s_{t+1} = s'] \right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_\pi(s') \right]
\end{aligned} \tag{2.9}$$

where $p(s', r | s, a)$ is a probability that completely describes the dynamics of the environment. It is related to the state transition probability $p(s'|s, a)$ by,

$$p(s'|s, a) = \sum_{r \in R} p(s', r | s, a) \tag{2.10}$$

Eq. 2.9 is known as the *Bellman equation* and it defines the relationship between value functions. It decomposes the dynamic optimization problem into a collection of sub-problems and is the basis for the effectiveness of SDP and other RL algorithms.

If the state-value function $v_\pi(s)$ of policy π is greater than that of policy π' for all s , we say that policy π is better than π' . Thus, the optimal policy π_* is defined as the policy with the optimal state value function $v_{\pi_*}(s)$ for all states:

$$\begin{aligned}
v_{\pi_*}(s) &:= \max_{\pi} v_\pi(s) \\
\pi_* &:= \arg \max_{\pi} v_\pi(s)
\end{aligned} \tag{2.11}$$

Similarly, the optimal action-value function is calculated by

$$q_{\pi_*}(s, a) := \max_{\pi} q_\pi(s, a) \tag{2.12}$$

The relationship between optimal state-value function $v_{\pi_*}(s)$ and optimal action-value function $q_{\pi_*}(s, a)$ is,

$$\begin{aligned}
v_{\pi_*}(s) &= \max_{a \in A(s)} q_{\pi_*}(s, a) \\
&= \max_a \mathbb{E}[r_t + \gamma v_{\pi_*}(s_{t+1}) | s_t = s, a_t = a] \\
&= \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\pi_*}(s')]
\end{aligned} \tag{2.13}$$

$v_{\pi_*}(s)$ is equal to the expected return for the best action from the state s . The last line of Eq. 2.13 is also called the *Bellman optimality equation* for state-value function. The Bellman optimality equation for action-value function is,

$$\begin{aligned}
q_{\pi_*}(s, a) &= \mathbb{E}[r_t + \gamma \max_{a'} q_{\pi_*}(s_{t+1}, a') | s_t = s, a_t = a] \\
&= \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_{\pi_*}(s', a')]
\end{aligned} \tag{2.14}$$

RL algorithms seek to find the optimal policy π_* by i) calculating it through Bellman optimality equation if the complete knowledge of the environmental dynamics (state transition probabilities) is known. Otherwise, ii) directly estimating π_* or the optimal value function v_{π_*} or q_{π_*} in a variety of ways.

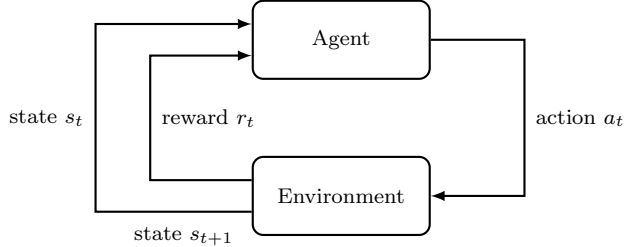


Figure 2.3: The interaction of agent with the MDP environment.

RL liquidation environment Foreign currencies liquidation can be modelled as an RL environment. However, the exchange rate in a real liquidation task is nonstationary. The real environment of a liquidation task may be a mixture of two scenarios [10]: 1) the environmental changes are limited to a finite number of stationary environments, and 2) the environment changes with a predictable trend. The time-varying dynamics of FX rates are unknown. Even if they are known, nonstationary environments are challenging for state-of-the-art RL algorithms to learn. Therefore, we assume that the liquidation environment is stationary, i.e. the transition probability between any states does not change over time. We set the liquidation task as an episodic task, with each episode terminating at time T and the discount rate γ set to 1. Finally, we model the environment as an MDP consisting of a finite set of states, actions, and rewards. The state s_t , action a_t , and reward r_t at time t are defined as follows,

- state $s_t = [x_t, v_t, t]$.
- action $a_t = n_t$.
- reward $r_t = R_t, \quad t = 0, 1, \dots, T - 1 \quad \text{and} \quad r_T = 0$

where x_t is the logarithm of FX rate at time t , v_t is the amount of foreign currency remaining at time t . a_t is defined as the decision variable n_t , which satisfies the maximum and minimum transaction volume constraints discussed in Sec. 2.1.2. For a given state s , the state space S and the action space $A(s)$ are discrete. r_t

can be derived from Eq. 2.15,

$$\begin{aligned}
r_t &= W_{t+1} - W_t \\
&= \sum_{j=0}^t R_j - \sum_{j=0}^{t-1} R_j \\
&= R_t = \frac{n_t}{\exp(x_t)}
\end{aligned} \tag{2.15}$$

Maximizing the expected cumulative rewards $\mathbb{E}(\sum_{t=0}^{T-1} r_t)$ is equivalent to maximizing the expected cumulative revenue $\mathbb{E}(\sum_{j=0}^{T-1} R_j)$. Thus the foreign currency liquidation task can be solved by RL. The detailed derivation of state transition probabilities will be discussed in Sec. 5.1, contribution II.

2.2 Deep learning for forecasting

In this section, we provide background on DL techniques for forecasting foreign exchange rates. These techniques provide a reference for the design of our RegPred Net. DL techniques are essentially parametric function approximations, which typically use large amounts of data containing inputs and corresponding labels (the correct outputs) to train numerous parameters in an ANN to learn and approximate complex functional relationships between inputs and outputs for a given task. For example, in a foreign exchange rate forecasting task, a DL algorithm can learn the mapping relationship between historical FX rate and FX rate over a future period to complete the exchange rate forecasting task. Mainstream ANNs that deal with time series include Feedforward Neural Network (FNN) [6], also known as Multi-layer Perceptron (MLP) [18], and Recurrent Neural Network (RNN) [41].

2.2.1 Feedforward neural network

Single-layer perceptron A single-layer perceptron is the simplest FNN, which is a non-linear mapping between the input and output of some function $f(\cdot)$. A single-layer perceptron with one neuron is shown in Fig. 2.4. It can be formulated as Eq. 2.16,

$$y = g(\mathbf{w}^T \mathbf{x} + b) \tag{2.16}$$

where $y \in \mathbb{R}$ is the output, $\mathbf{x} \in \mathbb{R}^{p_{in}}$ is the input vector of length p , $\mathbf{w} \in \mathbb{R}^{p_{in}}$ denotes the weight vector, $b \in \mathbb{R}$ is the bias term, and $g(\cdot)$ represents a nonlinear function, called an activation function. Common activation functions include Sigmoid, Tanh, ReLu [2], Softmax and their variants. A more general single-layer

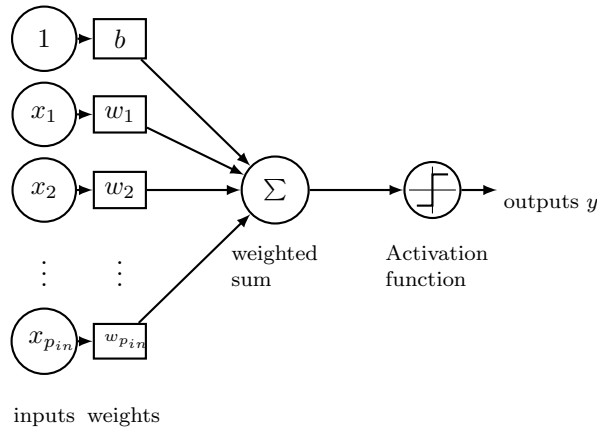


Figure 2.4: Single-layer perceptron of one neuron.

perceptron consisting of multiple neurons is formulated as Eq. 2.17,

$$\mathbf{y} = g(\mathbf{W}^T \mathbf{x} + \mathbf{b}) \quad (2.17)$$

where the output vector is of length p_{out} , W is a weight matrix of size (p_{in}, p_{out}) , and \mathbf{b} is a bias vector of length p_{out} .

Multi-layer perceptron Multiple perceptrons are stacked together to form a MLP, where the neurons between any two layers are fully connected to each other. Layers other than input and output are called hidden layers. Fig. 2.5 shows an MLP with one hidden layer, denoted as $\mathbf{h}^{(1)}$, where the superscript (1) denotes the index of the hidden layer. The dimension of input \mathbf{x} and output \mathbf{y} are denoted as p_{in} and p_{out} , respectively. This network can be expressed by Eq. 2.18,

$$\begin{aligned} \mathbf{h}^{(1)} &= g^{(1)}(\mathbf{W}^{(1)T} \mathbf{x} + \mathbf{b}^{(1)}) \\ \mathbf{y} &= g^{(2)}(\mathbf{W}^{(2)T} \mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \end{aligned} \quad (2.18)$$

There can be any number of hidden layers in a neural network, each as a function of the previous layer, and the number of neurons in each layer can be set arbitrarily. The number of layers and neurons are usually chosen experimentally for a specific task. "Deep" in "Deep Learning" refers to networks with many layers. The more layers and neurons a network has, the more complex tasks it can handle. However, due to the mechanism for updating network parameters and the characteristics of the activation functions, too many layers can cause the network to stop learning. Many parameters can also introduce excessive-high time and space complexity in the training process.

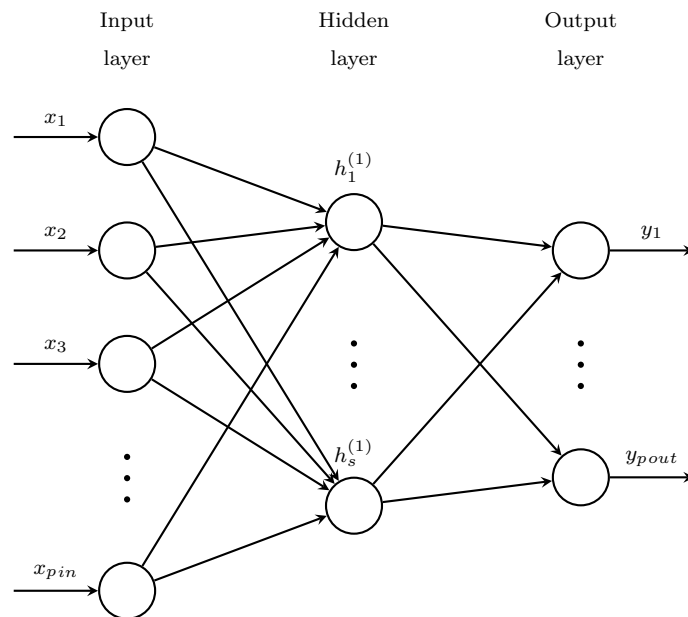


Figure 2.5: Multi-layer perceptron.

Training of feedforward neural networks The weights \mathbf{W} and biases \mathbf{b} in the FNN are trainable parameters and they can be trained to fit an objective function $f(\cdot)$. Assume the parameters of FNN are $\boldsymbol{\theta}$ and the true value of given input \mathbf{x} is \mathbf{y} , the output of the FNN is denoted as $f(\mathbf{x}, \boldsymbol{\theta}) = \hat{\mathbf{y}}$. We define a loss function $J(\hat{\mathbf{y}}, \mathbf{y})$ to calculate the error between $\hat{\mathbf{y}}$ and \mathbf{y} . The average loss evaluated on a given dataset reflects the predictive performance of the network, and the lower the loss, the better the prediction. We can reduce the average loss by adjusting the parameters in the network to make its predictions closer to the true values. In DL, the loss function is usually a differentiable function, such as the cross-entropy used in classification tasks and the mean-squared error used in regression tasks. Each parameter in the network is iteratively adjusted by gradient descent and Back-Propagation (BP) [24] algorithms, where gradient descent calculates the gradient of the loss function with respect to the weights, BP specifies rules for passing the gradient to the weights layer by layer. This process is known as FNN training. After repeated training over a large number of epochs, the losses converge to some local minimum. A trained NN can correctly predict unseen inputs \mathbf{x} .

2.2.2 Recurrent neural network

Fully recurrent neural network RNN is a neural network for processing sequential inputs. In addition to processing the inputs $\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_t$ between layers

like a normal FNN, the RNN also computes the weighted non-linear transformation of the sequence, saves the result to a hidden state and propagates it recurrently in the time direction. Fig. 2.6 shows the structure of a fully RNN (FRNN), where the FRNN on the left of the equal sign can be expanded into the structure on the right. For each time step t , the output \mathbf{y}_t and the hidden state \mathbf{h}_t are expressed

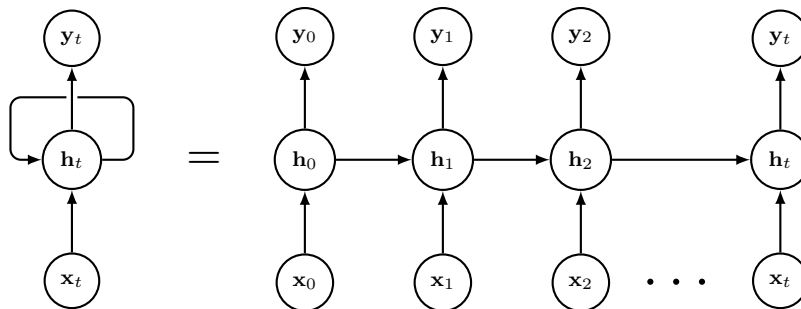


Figure 2.6: An illustration of FRNN.

as follows,

$$\begin{aligned} \mathbf{h}_t^{(1)} &= g^{(1)}(\mathbf{W}_{hh}^{(1)T} \mathbf{h}_{t-1}^{(1)} + \mathbf{W}_{hx}^{(1)T} \mathbf{x}_t + \mathbf{b}_h^{(1)}) \\ \mathbf{y}_t &= g^{(2)}(\mathbf{W}_{yh}^{(2)T} \mathbf{h}_t^{(1)} + \mathbf{b}_y^{(2)}) \end{aligned} \quad (2.19)$$

where \mathbf{W}_{hh} , \mathbf{W}_{hx} , \mathbf{W}_{yh} are weight matrices whose subscripts denote the layers they are connected. \mathbf{b}_h , \mathbf{b}_y are bias vectors whose subscripts indicate the layer to which they belong. Like FNN, FRNN can also vertically stack multiple hidden layers to form a deep neural network.

Maintaining hidden state h_t in the time direction allows RNN to mine the temporal dynamics in the input sequence. Theoretically, FRNN can handle input sequences of arbitrary length. However, the gradients of the loss function with respect to \mathbf{W}_{hh} and \mathbf{W}_{hx} used for weights updating can easily be 0 (gradient vanishing) or tend to infinity (gradient explosion). This is because calculating the gradients requires computing the partial derivative $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ of hidden states over $t = 0, 1, \dots, T$ and multiplying them together. Each $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ at t is a value either always above 1 or always in the range of $[0, 1]$. As a result, their successive multiplication can cause the gradient to disappear or explode, causing the network to stop learning or collapse. To alleviate this problem, LSTM has been proposed as a more widely used RNN.

Long short-term memory The hierarchy of the LSTM is the same as that of the FRNN, the major difference between them being that the LSTM replaces the hidden state h_t in the FRNN with a more complex state with gating structure.

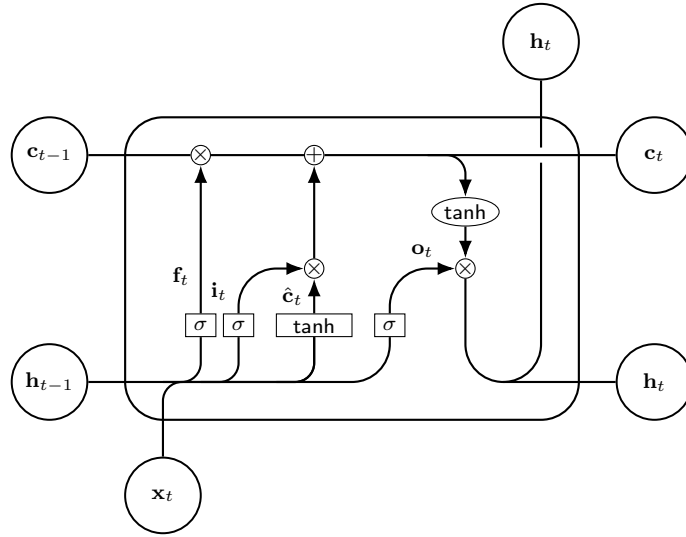


Figure 2.7: An illustration of LSTM cell.

A single LSTM cell at time step t is shown in Fig. 2.7, where \mathbf{c}_{t-1} , \mathbf{c}_t are cell states at step $t-1$ and t , respectively. The cell state preserves the long-term memory extracted from the input sequence. \mathbf{h}_{t-1} , \mathbf{h}_t are hidden states that preserve the short-term memory. σ is the sigmoid activation. \mathbf{f}_t , \mathbf{i}_t , \mathbf{o}_t are forget, input and output gates, respectively. $\hat{\mathbf{c}}_t$ is the temporary cell state. These gates are formulated as follows,

$$\begin{aligned}
 \mathbf{f}_t &= \sigma(\mathbf{W}_f^T \mathbf{x}_t + \mathbf{U}_f^T \mathbf{h}_{t-1} + \mathbf{b}_f) \\
 \mathbf{i}_t &= \sigma(\mathbf{W}_i^T \mathbf{x}_t + \mathbf{U}_i^T \mathbf{h}_{t-1} + \mathbf{b}_i) \\
 \mathbf{o}_t &= \sigma(\mathbf{W}_o^T \mathbf{x}_t + \mathbf{U}_o^T \mathbf{h}_{t-1} + \mathbf{b}_o)
 \end{aligned} \tag{2.20}$$

where the forget gate \mathbf{f}_t calculates the proportion of long-term memory stored in \mathbf{c}_{t-1} that should be deleted. The input gate \mathbf{i}_t calculates the proportion of part of the input \mathbf{x}_t that will be updated into the current cell state \mathbf{c}_t , and the output gate \mathbf{o}_t calculates the proportion of \mathbf{c}_t that is retained in the hidden state \mathbf{h}_t . The current cell state \mathbf{c}_t and the hidden state \mathbf{h}_t can be calculated as follows,

$$\begin{aligned}
 \hat{\mathbf{c}}_t &= \tanh(\mathbf{W}_c^T \mathbf{x}_t + \mathbf{U}_c^T \mathbf{h}_{t-1} + \mathbf{b}_c) \\
 \mathbf{c}_t &= \mathbf{f}_t \cdot \mathbf{c}_{t-1} + \mathbf{i}_t \cdot \hat{\mathbf{c}}_t \\
 \mathbf{h}_t &= \mathbf{o}_t \cdot \tanh(\mathbf{c}_t)
 \end{aligned} \tag{2.21}$$

The temporary cell state $\hat{\mathbf{c}}_t$ is the weighted sum of the input \mathbf{x}_t and the previous hidden state \mathbf{h}_{t-1} , which retains some of the information in the input. Current cell state \mathbf{c}_t is updated by summing a part of the previous cell state \mathbf{c}_{t-1} filtered by the

forget gate \mathbf{f}_{t-1} and the temporary cell state \mathbf{c}_{t-1} filtered by the input gate \mathbf{i}_t . The current hidden state \mathbf{h}_t is the non-linearly transformed \mathbf{c}_t filtered by the output gate \mathbf{o}_t , which retains some of the information in \mathbf{c}_t as short-term memory. If the information in the new input \mathbf{x}_t is not important, the value of the input gate \mathbf{i}_t is close to $\mathbf{0}$ and the value of the forget gate \mathbf{f}_t is close to $\mathbf{1}$, and vice versa. In this way the LSTM can filter the important information from the input at each time step to retain it. Note that the partial derivatives between the cell states $\frac{\partial \mathbf{c}_t}{\partial \mathbf{c}_{t-1}}$ and the hidden states $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}$ in the gradient calculations are no longer restricted to a range of values that is above 1 or just in between 0 and 1. This can prevent the gradient from exploding or vanishing to some extent when the input sequence is long.

2.2.3 Bayesian optimization

Bayesian optimization is a global optimization method that searches sequentially for the optimal value $\mathbf{x}_* \in \mathbb{R}^{p_{in}}$ that maximizes the function $f(\mathbf{x})$,

$$\mathbf{x}_* = \arg \max_{\mathbf{x} \in \mathbb{R}^{p_{in}}} f(\mathbf{x}) \quad (2.22)$$

$f(\mathbf{x})$ is usually a real-valued objective function that either has an unknown range of inputs, an unknown structure, or an expensive evaluation cost. Due to these characteristics, it is challenging to derive inputs that maximize $f(\mathbf{x})$. Bayesian optimization is usually able to estimate the optimal inputs using the smallest number of evaluations. It is often used to optimize the hyperparameters or structure of some models in machine learning [13], [46], [23].

Bayesian optimization heuristically searches for the potentially optimal input. It assumes that $f(\mathbf{x})$ has a Gaussian prior, i.e., $\{f(\mathbf{x}_1), f(\mathbf{x}_2), \dots\}$ is a Gaussian process and any combination of them follows a multivariate normal distribution, where $\{\mathbf{x}_1, \mathbf{x}_2, \dots\}$ are the different inputs in the iterative evaluation of the algorithm. At each step of evaluation, Bayesian optimization computes $f(\mathbf{x})$ given suggested input \mathbf{x} and collects the $(\mathbf{x}, f(\mathbf{x}))$ pairs into a data pool D . The mean $\mu(\mathbf{x})$ of the prior distribution and the covariance function $\sigma^2(\mathbf{x})$ are then updated to form the posterior distribution $p(f(\mathbf{x})|D_{1:n}, \mathbf{x})$, where $D_{1:n}$ denotes the observations $\{(\mathbf{x}_1, f(\mathbf{x}_1)), \dots, (\mathbf{x}_n, f(\mathbf{x}_n))\}$ saved in D . With the knowledge of observations, the next possibly optimal input \mathbf{x}' can be determined by constructing an acquisition function $utility(\mathbf{x}|D_{1:n})$ and solving the following sub-optimization problem,

$$\mathbf{x}' = \arg \max_{\mathbf{x}} utility(\mathbf{x}|D_{1:n}) \quad (2.23)$$

The sub-optimization problem is easier to solve compared to the original optimization problem in Eq. 2.22. There are many choices of the acquisition function

discussed in [8], the most common one is the Expected Improvement (EI). EI is the expected value of an improvement function I :

$$I(\mathbf{x}) = \max\{0, f(\mathbf{x}) - f(\mathbf{x}_*)\} \quad (2.24)$$

where $f(\mathbf{x}_*)$ denotes the maximal estimate of the objective function so far. The expectation of the improvement $\mathbb{E}(I)$ can be analytically evaluated as:

$$\mathbb{E}(I) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}_*))\Phi(Z) + \sigma(\mathbf{x})\phi(Z), & \text{if } \sigma(\mathbf{x}) > 0 \\ 0, & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \quad (2.25)$$

$$Z = \frac{\mu(\mathbf{x}) - f(\mathbf{x}_*)}{\sigma(\mathbf{x})}$$

where Φ and ϕ indicate the CDF and PDF of the standard normal distribution, respectively.

After several iterations, Bayesian optimization can find the quasi-optimal input $\mathbf{x} \approx \mathbf{x}_*$. In contribution I, we use Bayesian optimization to search the optimal hyperparameters for RegPred Net. We define a negative loss function during RegPred Net training as the objective function $f(\cdot)$ to be maximized, and the hyperparameters as the inputs \mathbf{x} to be optimized. We will explain Bayesian optimization in more detail in Sec. 5.1 of contribution I, and give the specific steps for applying it to RegPred Net in Algorithm 6 of contribution I.

2.3 Reinforcement learning for optimal liquidation strategy estimation

In this section, we present the background of the RL algorithm that can be applied to solve the optimal liquidation of foreign currencies. First, the fundamental RL methods, including SDP, Monte Carlo (MC) methods, Temporal-Difference (TD) methods, policy gradient and Actor-Critic (AC) are discussed in Sec. 2.3.1. These algorithms are referenced from [50]. The combination and improvement of these methods is the basis for state-of-the-art RL algorithms. In Sec. 2.3.2, we briefly introduce the state-of-the-art RL models used to compare with our EOLS algorithm in contribution II.

2.3.1 Fundamental algorithms in reinforcement learning

Stochastic dynamic programming In RL, SDP [51] is a collection of algorithms that computes optimal strategies for sequential decision problems using the complete dynamics of the MDP environment. It divides the original optimization problem into multiple subproblems through the Bellman equation (Eq. 2.9).

The primal problem is then solved by recursively solving each subproblem. The strategies computed by SDP can be considered theoretically optimal solutions that existing RL algorithms can achieve. However, SDP is difficult to use in practice because 1) the complete dynamics of the environment in most tasks are unknown, and 2) the time complexity of the algorithm is very high. Because SDP needs to evaluate the expected return for each possible state, the size of the state space grows exponentially as the state variables increase.

Value iteration [51] is a representative SDP algorithm, which we list in Algorithm 1. The algorithm first iteratively computes the state value function (rows

Algorithm 1 Value iteration

Input: Value function array \mathbf{v} of size $|S|$, difference array $\boldsymbol{\delta}$ of size $|S|$, a threshold coefficient $\xi \in \mathbb{R}_{>0}$

Output: Optimal policy π_*

- 1: For each $s \in S$, initialize $\mathbf{v}(s) = 0$ and $\boldsymbol{\delta}(s) = \infty$
 - 2: **repeat**
 - 3: **for** $s \in S$ **do**
 - 4: $v_{old} \leftarrow \mathbf{v}(s)$
 - 5: $\mathbf{v}(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma \mathbf{v}(s')]$
 - 6: $\boldsymbol{\delta}(s) \leftarrow \min(\boldsymbol{\delta}(s), |v_{old} - \mathbf{v}(s)|)$
 - 7: **end for**
 - 8: **until** $\boldsymbol{\delta}(s) < \xi$ for all $s \in S$
 - 9: **for** $s \in S$ **do**
 - 10: $\pi_*(s) = \arg \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma \mathbf{v}(s')]$
 - 11: **end for**
 - 12: Return π_*
-

1-7) for each state s and estimates the difference $\boldsymbol{\delta}(s)$ between the newly computed value function and the previously computed one. The iteration stops if for each s , $\boldsymbol{\delta}(s)$ is less than a small positive threshold factor ξ . The best action $\pi_*(s)$ is then chosen for each s according to the Bellman equation, and the optimal strategy π_* is output.

Dynamic Programming (DP) can be seen as SDP under special conditions: 1) its state transition is deterministic,

$$p(s'|s, a) = 1 \quad \text{for all } s, s' \in S, a \in A(s)$$

2) its value function is estimated by,

$$v_\pi(s) \leftarrow \max_a [r(s, a) + v_\pi(s')]$$

where $r(s, a)$ denotes the reward received by the agent after taking action a at state s . Then s will transition to s' with a probability of 1.

Monte Carlo methods MC methods [52] refer to a collection of algorithms that estimate optimal value functions v_{π_*} , q_{π_*} and optimal policy π_* from the *experiences* sampled in the agent’s interaction with the environment. These experiences are sequences of states, actions and rewards $\{s_0, a_0, r_0, s_1, \dots, s_t, a_t, r_t, s_{t+1}, \dots\}$. MC methods use the average return of state s_t in the sampled episodes to estimate the value function $v_{\pi}(s_t)$, which is unbiased but has a high variance. $v_{\pi}(s_t)$ is updated by,

$$v_{\pi}(s_t) \leftarrow v_{\pi}(s_t) + \alpha[G_t - v_{\pi}(s_t)] \quad (2.26)$$

where $\alpha \in \mathbb{R}_{>0}$ is a step-size coefficient that controls the update rate. Eq. 2.26 shows that the MC methods require a complete episode when updating $v_{\pi}(s_t)$, because calculating the return G_t needs all rewards following t .

MC methods still require an environmental model that can generate transitions (s, a, r, s') but does not need the complete knowledge of the model required by SDP. Estimating the optimal solution utilizing sampling is also known as a heuristic method.

A representative MC method called on-policy MC control [52] is listed in Algorithm 2, where on-policy means that the policy used for sampling is the same as the policy to be improved. On the contrary, off-policy refers to sampling and improving using two different policies. $G(s, a)$ in line 7 indicates the return collected after the first encounter of (s, a) pair. The algorithm evaluates action-value functions by averaging the sampled returns (lines 6-9) and estimating π_* by ϵ -greedy policy (lines 10-15). The ϵ -greedy policy is an action selection method (Eq. 2.27, 2.28), where ϵ is a small positive value. ϵ -greedy policy selects non-optimal actions with probability $\frac{\epsilon}{|A(s)|}$ during action selection, encouraging the algorithm to explore potentially more optimal solutions.

Temporal-Difference learning TD learning [53] refers to a collection of algorithms that learn the optimal value function v_{π_*} , q_{π_*} and the optimal strategy π_* from sampled experiences in a *bootstrapping* way. As shown in Eq. 2.29, bootstrapping denotes estimating the value function from the estimated return $r_t + \gamma v_{\pi}(s_{t+1})$,

$$v_{\pi}(s_t) \leftarrow v_{\pi}(s_t) + \alpha[r_t + \gamma v_{\pi}(s_{t+1}) - v_{\pi}(s_t)] \quad (2.29)$$

In contrast to the MC method in Eq. 2.26, which uses a complete return G_t of the experience episode to update the value function, the TD method updates it while sampling the experience in an online manner, which improves the sampling efficiency. Due to bootstrapping, TD learning has a bias greater than 0, but its variance is lower than that of the MC methods and converges faster than them.

Algorithm 2 On-policy MC control

Input: Arbitrary ϵ -greedy policy π

Output: Estimated optimal policy π

1: **for** all $s \in S, a \in A(s)$ **do**
2: Initialize arbitrary action-value function $q(s, a)$ and empty return list $R(s, a)$
3: **end for**
4: **repeat**
5: Generate an episode $E = \{s_0, a_0, r_0, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T\}$ using policy π
6: **for** each pair of $s, a \in E$ **do**
7: Append return $G(s, a)$ to list $R(s, a)$
8: $q_\pi(s, a) \leftarrow \text{mean}(R(s, a))$
9: **end for**
10: **for** each $s \in E$ **do**

11:
$$a^* \leftarrow \arg \max_a q_\pi(s, a) \tag{2.27}$$

12: **for** $a \in A(s)$ **do**

13:
$$\pi(a|s) \leftarrow \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A(s)|}, & \text{if } a = a^* \\ \frac{\epsilon}{|A(s)|}, & \text{otherwise} \end{cases} \tag{2.28}$$

14: **end for**

15: **end for**

16: **until** $\pi(a|s)$ does not change for all $s \in S, a \in A(s)$

17: **Return** π

Similar to MC methods, TD learning does not require a complete model of the environment.

A representative off-policy TD method known as Q-learning [59] is shown in Algorithm 3, where Q is a synonym of the action-value function q_π . At each step,

Algorithm 3 Q-learning (off-policy TD control)

Input: Arbitrary action-value function $q_\pi(s, a)$ for all $s \in S$, $a \in A(s)$, number of evaluations N , learning rate α , positive value ϵ

Output: Estimated optimal policy π

- 1: **repeat**
- 2: Initialize state s
- 3: **repeat**
- 4: Choose a from $A(s)$ by ϵ -greedy policy, take a and get r, s'
- 5: Update $q_\pi(s, a)$ by

$$q_\pi(s, a) \leftarrow q_\pi(s, a) + \alpha[r + \gamma \max_{a'} q_\pi(s', a') - q_\pi(s, a)]$$

- 6: $s \leftarrow s'$
 - 7: **until** s is the terminal state
 - 8: **until** N times
 - 9: Generate π from q_π
 - 10: Return π
-

Algorithm 3 samples the experience (s, a, r, s') by ϵ -greedy policy (line 4) and updates the action-value function $q_\pi(s, a)$ in bootstrapping way (line 5). After a predetermined number of epochs or if the change in q value for each (s, a) pair is less than a predetermined value, the algorithm stops and returns the optimal policy π .

TD learning has become one of the most widely used RL algorithms due to its combination of SDP's bootstrapping and MC's learning by interacting with the environment. We will continue to introduce an important TD method called DQN in Sec. 2.3.2.

All of these methods described above are classified as value-based methods, i.e. finding the optimal strategy by estimating a value function. The estimated value function can be recorded in a table of size $|S| \times |A|$. For problems where the state space $|S|$ is large or where the action values are continuous, it is clearly not feasible to record the value function in a table. To address this limitation, it is more common to use the function $\hat{v}_\pi(s, \mathbf{w})$ (e.g. NN) to approximate the value function, where \mathbf{w} denotes the weights of the function. This is known as the *function approximation* [54].

Policy gradient methods Policy gradient methods [55] refer to a collection of algorithms that use parameterized policy $\pi(a|s, \boldsymbol{\theta}) = \Pr(a_t = a | s_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta})$ to select the optimal action a_* by maximizing a performance measure $J(\boldsymbol{\theta}) := v_\pi(s_0)$, where $\boldsymbol{\theta}$ are policy parameters and s_0 is some particular initial state of an experience episode $\{s_0, a_0, r_0, s_1, a_1, r_1, \dots\}$.

The gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ is calculated by,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) := \nabla_{\boldsymbol{\theta}} v_\pi(s_0) \quad (2.30)$$

$$= \sum_s \left(\sum_{k=0}^{\infty} \Pr(s_0 \rightarrow s, k, \pi) \right) \sum_a \nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a) \quad (2.31)$$

$$= \sum_s \eta_\pi(s) \sum_a \nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a) \quad (2.32)$$

$$= \left(\sum_s \eta_\pi(s) \right) \sum_s \frac{\eta_\pi(s)}{\sum_s \eta_\pi(s)} \sum_a \nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a) \quad (2.33)$$

$$\propto \sum_s \frac{\eta_\pi(s)}{\sum_s \eta_\pi(s)} \sum_a \nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a) \quad (2.34)$$

$$= \sum_s d_\pi(s) \sum_a \nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a) \quad (2.35)$$

where in Eq. 2.31, $\Pr(s_0 \rightarrow s, k, \pi)$ is the probability of transition from state s_0 to state s in k steps under policy π , $\Pr(s_0 \rightarrow s, k, \pi)$ can be recursively computed by,

$$\Pr(s_0 \rightarrow s, k, \pi) = \sum_{\bar{s}} \Pr(s_0 \rightarrow \bar{s}, k-1, \pi) \Pr(\bar{s} \rightarrow s, 1, \pi) \quad (2.36)$$

and

$$\Pr(\bar{s} \rightarrow s, 1, \pi) = \sum_a \pi(a|\bar{s}, \boldsymbol{\theta}) \Pr(s|\bar{s}, a) \quad (2.37)$$

$\Pr(s|\bar{s}, a)$ is the state transition probability. For any state s , its state value function $v_\pi(s)$ is the expectation of action-value functions $\sum_a \pi(a|s, \boldsymbol{\theta}) q_\pi(s, a)$ over all $a \in A(s)$. In Eq. 2.32, $\eta_\pi(s) = \sum_{k=0}^{\infty} \Pr(s_0 \rightarrow s, k, \pi)$ denotes the average number of time steps spent in state s in a single episode under policy π , and the time spent in s if an episode starts from s ($\Pr(s_0 \rightarrow s, 0, \pi)$) or if a preceding state \bar{s} transitions to s ($\Pr(\bar{s} \rightarrow s, k, \pi)$). $\eta_\pi(s)$ is also called the visitation probability of s . In Eq. 2.33, $\eta_\pi(s)$ is normalized to be a probability distribution. In Eq. 2.34, the constant $\sum_s \eta_\pi(s)$ of Eq. 2.33 is omitted and the rest is proportional to Eq. 2.33. In Eq. 2.35, $\frac{\eta_\pi(s)}{\sum_s \eta_\pi(s)}$ is written as $d_\pi(s)$ for simplicity. $d_\pi(s)$ is a distribution under policy π that reflects the fraction of time spent on a state s .

The result in Eq. 2.35 shows that $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is proportional to the expectation of gradient $\nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta}) q_{\pi}(s, a)$ with respect to distribution $d_{\pi}(s)$. This conclusion is known as the *policy gradient theorem* [55].

Eq. 2.35 is an estimate $\widehat{\nabla_{\boldsymbol{\theta}} J}(\boldsymbol{\theta})$ of the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, $\widehat{\nabla_{\boldsymbol{\theta}} J}(\boldsymbol{\theta})$ can be further formulated as,

$$\widehat{\nabla_{\boldsymbol{\theta}} J}(\boldsymbol{\theta}) = \sum_s d_{\pi}(s) \sum_a \nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta}) q_{\pi}(s, a) \quad (2.38)$$

$$= \sum_s d_{\pi}(s) \sum_a \pi(a|s, \boldsymbol{\theta}) q_{\pi}(s, a) \frac{\nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta})}{\pi(a|s, \boldsymbol{\theta})} \quad (2.39)$$

$$= \mathbb{E}_{\pi}[q_{\pi}(s, a) \nabla_{\boldsymbol{\theta}} \ln \pi(a|s, \boldsymbol{\theta})] \quad (2.40)$$

where $\nabla_{\boldsymbol{\theta}} \ln \pi(a|s, \boldsymbol{\theta})$ in Eq. 2.40 is equal to $\frac{\nabla_{\boldsymbol{\theta}} \pi(a|s, \boldsymbol{\theta})}{\pi(a|s, \boldsymbol{\theta})}$ in Eq. 2.39. Eq. 2.40 shows that the estimate $\widehat{\nabla_{\boldsymbol{\theta}} J}(\boldsymbol{\theta})$ of gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the expected value of $q_{\pi}(s, a) \nabla_{\boldsymbol{\theta}} \ln \pi(a|s, \boldsymbol{\theta})$ under policy π . In practice, based on the fact that the expectation of the sample gradient is equal to the actual gradient, $\widehat{\nabla_{\boldsymbol{\theta}} J}(\boldsymbol{\theta})$ is approximated by sampling. $J(\boldsymbol{\theta})$ is maximized by updating $\boldsymbol{\theta}$ over time t through an approximate gradient ascent algorithm,

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \widehat{\nabla_{\boldsymbol{\theta}_t} J}(\boldsymbol{\theta}_t) \quad (2.41)$$

where α is the step-size parameter.

The expression in Eq. 2.40 is the general form of $\widehat{\nabla_{\boldsymbol{\theta}} J}(\boldsymbol{\theta})$ in policy gradient methods. Different policy gradient methods has different estimation of the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, it can be formed by replacing $q_{\pi}(s, a)$ in Eq. 2.40 with different terms [43], such as the return G_t , the advantage function $A_{\pi}(s, a) := q_{\pi}(s, a) - v_{\pi}(s)$, or the TD residual $r + v_{\pi}(s') - v_{\pi}(s)$, etc.

Algorithm 4 shows an MC policy gradient method known as REINFORCE [48]. It uses episodic samples to update the policy parameter $\boldsymbol{\theta}$ in lines 1 - 6. Since $q_{\pi}(s_t, a_t) = \mathbb{E}[G_t | s_t, a_t]$, the action-value function $q_{\pi}(s_t, a_t)$ is replaced with return G_t in line 4. $\pi_{\boldsymbol{\theta}}$ is the abbreviation of $\pi(a|s, \boldsymbol{\theta})$ for any $s \in S, a \in A(s)$.

Actor-critic methods AC methods [56] refer to algorithms that use a parameterized policy $\pi_{\boldsymbol{\theta}}$ as the "actor" to select an action and a parameterized value function $\hat{v}_{\mathbf{w}}$ as the "critic" to rate the selection. The AC methods combine the advantages of value function-based and policy-based methods, reduce the variance of policy gradient and accelerate the convergence. Algorithm 5 exhibits a basic one-step AC method [56], the algorithm uses the one-step TD residual $\delta = r + \gamma \hat{v}(s', \mathbf{w}) - \hat{v}(s, \mathbf{w})$ (see Eq. 2.29) to update the policy parameter $\boldsymbol{\theta}$ and value function parameter \mathbf{w} (lines 6 - 7).

Algorithm 4 REINFORCE (MC policy gradient)

Input: Parameterized policy π_{θ} with arbitrary parameter vector $\theta \in \mathbb{R}^{p_{in}}$, number of iterations N

Output: Estimated optimal policy π_{θ}

- 1: **repeat**
 - 2: Generate episodic experiences $s_0, a_0, r_0, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T$ by following π_{θ}
 - 3: **for** $t = 0, \dots, T - 1$ **do**
 - 4: $\theta_{t+1} \leftarrow \theta_t + \alpha \gamma^t G_t \nabla_{\theta_t} \ln \pi(a_t | s_t, \theta_t)$
 - 5: **end for**
 - 6: **until** N times
 - 7: Return π_{θ}
-

Algorithm 5 One-step AC

Input: Parameterized policy π_{θ} with arbitrary parameter vector $\theta \in \mathbb{R}^{p_{in}}$, parameterized state-value $\hat{v}_{\mathbf{w}}$ with arbitrary parameter vector $\mathbf{w} \in \mathbb{R}^{p_{in}}$, step-size parameters α_{θ} and $\alpha_{\mathbf{w}}$, number of iterations N

Output: Estimated optimal policy π_{θ}

- 1: **repeat**
 - 2: Initialize the first state s and vector $\mathbf{I} = \mathbf{1}$
 - 3: **repeat**
 - 4: At state s , sample a from $\pi(\cdot | s, \theta)$, take action a and get r, s'
 - 5: $\delta \leftarrow r + \gamma \hat{v}(s', \mathbf{w}) - \hat{v}(s, \mathbf{w})$
 - 6: $\mathbf{w} \leftarrow \mathbf{w} + \alpha_{\mathbf{w}} \mathbf{I} \delta \nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$
 - 7: $\theta \leftarrow \theta + \alpha_{\theta} \mathbf{I} \delta \nabla_{\theta} \ln \pi(a | s, \theta)$
 - 8: $\mathbf{I} \leftarrow \gamma \mathbf{I}$
 - 9: $s \leftarrow s'$
 - 10: **until** s is the terminal state
 - 11: **until** N times
 - 12: Return π_{θ}
-

2.3.2 Deep reinforcement learning

DRL refers to the collection of RL algorithms that use DL techniques to model a policy or value function. Compared to typical RL algorithms, DRL methods have the following advantages: 1) DL accepts any form of input of any size, e.g. CNN can process image input, and fully connected-NN or LSTM can process sequence input. 2) DL is more robust and powerful in learning feature information from input data. This section presents two state-of-the-art DRL methods used for comparison with the EOLS model, namely the value-based method DQN and the actor-critic method PPO.

Deep Q-network DQN [35] is a DL version of Q-learning, initially used in video games to train RL agents to play at a level that exceeds or is comparable to human players. The action-value function Q or q is represented by a nonlinear approximator (ANN), which can lead to algorithmic diverging. The reason for the divergence is that there is a strong correlation between the sampled experiences, and a slight change in Q -value will significantly change the policy π . DQN uses a trick called *experience replay* to solve this problem. The sampled experiences are stored in a data pool. When the parameters are updated, a random batch of experiences is selected to compute the gradient and update the parameters in Q network. The randomly selected experiences come from different trajectories and thus have less correlation. We give the DQN in Algorithm 6, DQN assumes a target neural network \hat{q}_{θ^-} or $\hat{q}(\cdot|\cdot, \theta^-)$ is the true value for the estimated action-value function q_{θ} or $q(\cdot|\cdot, \theta)$. θ^- is initialized as θ at the beginning (line 3). The algorithm samples the experiences in MC way at each step t and saves them into a data pool D (line 8). Line 10 calculates the target y of $q(a|s, \theta)$ by bootstrapping. Line 11 computes the loss $(y - q(a|s, \theta))^2$ and performs gradient descent on parameters θ . The gradient of $(y - q(a|s, \theta))^2$ with respect to θ is computed from randomly selected experiences from D . For every constant steps C , q_{θ} is assigned to q_{θ^-} . This can keep the target of q_{θ} unchanged for C steps. In the video game task, the CNN is used as a nonlinear approximator, while in our foreign currency liquidation task, we use the fully connected-NN instead.

Trust region policy optimization Trust Region Policy Optimization (TRPO) [42] is a policy gradient method that solves the instability problem during the training of policy approximated by large-scale nonlinear ANNs. The stability is realized by two points: 1)TRPO optimizes a surrogate objective function $J(\theta)$ that measures the estimated advantage $\hat{A}(\cdot)$ over the state visitation distribution and actions in Eq. 2.42. At each step of training, TRPO samples trajectories under the old policy $\pi_{\theta_{\text{old}}}$ learned at the current step to update the new policy π_{θ} . It uses importance sampling [22] to balance the distribution difference between the

Algorithm 6 DQN

- Input:** Number of iterations N , length of each episode T , number of steps C to reset the target network, replay buffer size D_N
- Output:** Estimated optimal policy π
- 1: Initialize replay buffer D with size D_N
 - 2: Initialize action-value function q_{θ} with arbitrary parameter θ
 - 3: Initialize target action-value function \hat{q}_{θ^-} with weights $\theta^- = \theta$
 - 4: **repeat**
 - 5: Initialize s_0
 - 6: **for** $t = 0, \dots, T - 1$ **do**
 - 7: For all $a \in A(s_t)$, select action a_t with ϵ -greedy, take action a_t and obtain r_t and s_{t+1}
 - 8: Store transition (s_t, a_t, r_t, s_{t+1}) in D
 - 9: Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from D
 - 10: Set $y_j = \begin{cases} r_j, & \text{if episode terminates at step } j + 1, \\ r_j + \max_{a'} \hat{q}(a'|s_{j+1}, \theta^-), & \text{otherwise.} \end{cases}$
 - 11: Perform a gradient descent step on $(y_j - q(a_j|s_j, \theta))^2$ with respect to θ
 - 12: Every C steps reset $\theta^- \leftarrow \theta$
 - 13: **end for**
 - 14: **until** N times
 - 15: Estimate and return π based on q_{θ}
-

new policy and the policy in Eq. 2.43. The surrogate objective function $J(\boldsymbol{\theta})$ is defined as follows,

$$J(\boldsymbol{\theta}) := \sum_s \rho_{\pi_{\boldsymbol{\theta}_{\text{old}}}}(s) \sum_a \pi(a|s, \boldsymbol{\theta}) \hat{A}_\pi(s, a, \boldsymbol{\theta}_{\text{old}}) \quad (2.42)$$

$$= \sum_s \rho_{\pi_{\boldsymbol{\theta}_{\text{old}}}}(s) \sum_a \pi(a|s, \boldsymbol{\theta}_{\text{old}}) \frac{\pi(a|s, \boldsymbol{\theta})}{\pi(a|s, \boldsymbol{\theta}_{\text{old}})} \hat{A}_\pi(s, a, \boldsymbol{\theta}_{\text{old}}) \quad (2.43)$$

$$= \mathbb{E}_{s \sim \rho_{\pi_{\boldsymbol{\theta}_{\text{old}}}}, a \sim \pi_{\boldsymbol{\theta}_{\text{old}}}} \left[\frac{\pi(a|s, \boldsymbol{\theta})}{\pi(a|s, \boldsymbol{\theta}_{\text{old}})} \hat{A}_\pi(s, a, \boldsymbol{\theta}_{\text{old}}) \right] \quad (2.44)$$

where $\boldsymbol{\theta}_{\text{old}}, \boldsymbol{\theta}$ are the parameters of current learned policy and new policy, respectively. $\rho_{\pi_{\boldsymbol{\theta}_{\text{old}}}}(s) = \sum_{\bar{s}} \sum_{k=0}^{\infty} \gamma^{k-1} \Pr(s_0 = \bar{s}) \Pr(\bar{s} \rightarrow s, k, \pi_{\boldsymbol{\theta}_{\text{old}}})$ is the discounted visitation probability of s under $\pi_{\boldsymbol{\theta}_{\text{old}}}$ (discounted version of $\eta_\pi(s)$ in Eq. 2.32). \bar{s} represents arbitrary state in an episode. γ is the discount rate. $\Pr(s_0 = \bar{s})$ is the initial probability over \bar{s} . $s \sim \rho_{\pi_{\boldsymbol{\theta}_{\text{old}}}}, a \sim \pi_{\boldsymbol{\theta}_{\text{old}}}$ means s follows the visitation distribution $\rho_{\pi_{\boldsymbol{\theta}_{\text{old}}}}$ and a is sampled from policy $\pi_{\boldsymbol{\theta}_{\text{old}}}$. $\hat{A}_\pi(s, a, \boldsymbol{\theta}_{\text{old}})$ is the estimate of an advantage function. A fundamental advantage function is a difference between the action-value function and the state-value function. It measures the improvement in the expected cumulative returns obtained by the agent after selecting a particular action a at state s compared to selecting other actions. There are many choices for advantage functions discussed in [33] and [44]. 2) TRPO constrains the size of policy update with KL divergence $D_{KL}(\pi(\cdot|s, \boldsymbol{\theta}_{\text{old}}) || \pi(\cdot|s, \boldsymbol{\theta}))$ between the old and new policies by,

$$\mathbb{E}_{s \sim \rho_{\pi_{\boldsymbol{\theta}_{\text{old}}}}(s)} [D_{KL}(\pi(\cdot|s, \boldsymbol{\theta}_{\text{old}}) || \pi(\cdot|s, \boldsymbol{\theta}))] \leq \xi \quad (2.45)$$

where $\xi \in \mathbb{R}_{>0}$ is a constraint coefficient. Maximizing $J(\boldsymbol{\theta})$ under the trust region constraint of Eq. 2.45 can be solved by sample-based estimation. This way prevents the new policy from changing too much compared with the current policy and is proved to have monotonically improvement during training.

Proximal policy optimization PPO [44] is an actor-critic algorithm that simplifies the complex trust-region constraints in TRPO by using a clipped surrogate objective function while retaining similar performance. The importance sampling rate of TRPO is denoted in Eq. 2.43 as $r(\boldsymbol{\theta})$,

$$r(\boldsymbol{\theta}) = \frac{\pi(a|s, \boldsymbol{\theta})}{\pi(a|s, \boldsymbol{\theta}_{\text{old}})} \quad (2.46)$$

PPO uses a clipped surrogate function to constrain $r(\boldsymbol{\theta})$ to an interval around 1 as a way to control the magnitude of new policy updates:

$$J^{\text{CLIP}}(\boldsymbol{\theta}) = \mathbb{E}[\min(r(\boldsymbol{\theta}) \hat{A}_\pi(s, a, \boldsymbol{\theta}_{\text{old}}), \text{clip}(r(\boldsymbol{\theta}), 1 - \epsilon, 1 + \epsilon) \hat{A}_\pi(s, a, \boldsymbol{\theta}_{\text{old}}))] \quad (2.47)$$

where $s \sim \rho_{\pi_{\theta_{\text{old}}}}$ and $a \sim \pi_{\theta_{\text{old}}}$. $\epsilon \in \mathbb{R}_{>0}$ is a small value parameter that controls the update size. The function $\text{clip}(r(\boldsymbol{\theta}), 1 - \epsilon, 1 + \epsilon)$ clips $r(\boldsymbol{\theta})$ into the interval $[1 - \epsilon, 1 + \epsilon]$.

For the case of sharing the same parameters $\boldsymbol{\theta}$ between policy network $\pi_{\boldsymbol{\theta}}$ and value function network $\hat{v}_{\boldsymbol{\theta}}$, $J^{\text{CLIP}}(\boldsymbol{\theta})$ can be revised to a more general version in Eq. 2.48 by adding an squared-error loss $(\hat{v}_{\pi}(s, \boldsymbol{\theta}) - v_{\pi}^{\text{target}})^2$ for including value function rating and an entropy term $H(\pi(\cdot|s, \boldsymbol{\theta}))$ for encouraging exploration,

$$J^{\text{CLIP}'}(\boldsymbol{\theta}) = \mathbb{E}[J^{\text{CLIP}}(\boldsymbol{\theta}) - c_1(\hat{v}_{\pi}(s, \boldsymbol{\theta}) - v_{\pi}^{\text{target}}(s))^2 + c_2 H(\pi(\cdot|s, \boldsymbol{\theta}))] \quad (2.48)$$

where c_1, c_2 are coefficients. $v_{\pi}^{\text{target}}(s)$ can be estimated by averaging a batch of discounted cumulative rewards over a period. $H(\pi(\cdot|s, \boldsymbol{\theta}))$ is an entropy bonus that encourages sufficient exploration in action space, i.e. the more the policy $\pi_{\boldsymbol{\theta}}$ tends to be random, the higher the entropy. PPO with clipped surrogate objective function in Eq. 2.48 is given in Algorithm 7.

At the end of this chapter, we provide an overview of RL algorithms suitable for solving the optimal liquidation tasks in Table 2.1 and 2.2. We categorize the EOLS and other state-of-the-art RL methods, listing their advantages and disadvantages for a clear comparison.

Algorithm 7 PPO with clipped objective

Input: Shared parameters $\boldsymbol{\theta}_{\text{old}}$ for policy and value function, clipping parameter ϵ , number of iterations N , updating epochs K , minibatch size M , trajectory length T

Output: Estimated optimal policy $\pi_{\boldsymbol{\theta}_{\text{old}}}$

- 1: **repeat**
- 2: Collect set of experiences D using policy $\pi_{\boldsymbol{\theta}_{\text{old}}}$, each experience has length T
- 3: Estimate $\hat{A}_{\pi}(s_0, a_0, \boldsymbol{\theta}_{\text{old}}), \dots, \hat{A}_{\pi}(s_{T-1}, a_{T-1}, \boldsymbol{\theta}_{\text{old}})$ using an advantage estimation algorithm
- 4: Update parameters $\boldsymbol{\theta}$

$$\boldsymbol{\theta} = \arg \max_{\boldsymbol{\theta}} \frac{1}{|D|T} \sum_{\tau \in D} \sum_{t=0}^T J^{\text{CLIP}'}(\boldsymbol{\theta})$$

by taking K epochs gradient ascent with minibatch of size M

- 5: $\boldsymbol{\theta}_{\text{old}} \leftarrow \boldsymbol{\theta}$
 - 6: **until** N times
 - 7: Return $\pi_{\boldsymbol{\theta}_{\text{old}}}$
-

Reinforcement Learning (1)				
Category	Method	Description	Advantage	Disadvantage
Value-based RL with model	DP, SDP	These methods compute the optimal policy given a perfect model of a MDP environment through Bellman equation.	1) Provide the theoretical optimal strategy for all of the other RL algorithms.	1) Need an MDP environment that describes the complete probability distributions of state transitions, 2) are extremely computationally intensive and usually not applicable in practice.
	EOLS	This method estimates the quasi-optimal strategy based on the analytical solution computed by SDP and greatly simplifies it.	1) Estimate the quasi-optimal strategy while greatly improving computational efficiency, 2) outperforms other heuristic RL algorithms on optimal liquidation task.	Need an assumption of the environment model.
Value-based RL	MC	MC methods estimate the optimal strategy based on the averaged return of the sampled (s, a, r, s') trajectories.	1) Are Model-free methods, 2) estimate based on averaged returns of trajectories, thus have no bias.	1) Are more computational intensive than TD methods, 2) need to estimate the value function for each possible state, 3) have high variance.
	TD	TD methods combine the ideas of MC and DP to estimate the optimal strategy directly from a partial learning experience (bootstrapping) without the need for a complete environment model and the collection of the complete trajectories.	1) Are Model-free, 2) use bootstrapping. Do not need to wait until the end of the trajectory, 3) have low variance.	1) Have Bias > 0 , 2) need to estimate the value function for each possible state.

Table 2.1: An overview of RL algorithms suitable for solving optimal liquidation tasks (1).

Reinforcement Learning (2)				
Category	Method	Description	Advantage	Disadvantage
Value- & policy-based RL	Function approximation, e.g. DQN, Double-DQN[34], policy gradient, AC, etc.	These methods estimate the optimal strategy using the parameterized linear or nonlinear function to model the value function or policy.	1) Do not need a table to record the value of each state or state-action pair, can handle tasks of continuous action and state spaces, 2) provide the probability for action selection if the policy is approximated by a function.	1) Nonlinear function approximation normally finds the local optimum of value function or policy. 2) have oscillation during learning, a small change in value functions will lead to a significant change in policy estimation.
Policy-based methods	Policy gradient, e.g. REINFORCE	These methods use the gradient ascent to directly optimize a parameterized policy. They intend to increase the probability of occurrence of high-return trajectories and decrease the probability of low-return trajectories.	1) Same as 1) in function approximation, 2) can learn stochastic policy, 3) more effective in high dimensional action spaces, 4) have better convergence properties than value-based methods.	1) Only guaranteed to converge on a local maximum, 2) have high variance and slowly convergence.
AC	TRPO, PPO, A3C[12], DDPG [27], SAC [17], etc.	AC methods combine the value-based and policy-based methods. They use a parameterized policy model (actor) to estimate the optimal action for a given state, and a parameterized value function model (critic) to help to improve the action selection.	1) Are off-policy methods, have more data efficiency and explorations, 2) converge faster than policy gradient, 3) converge to local optimum is guaranteed, 4) some methods reduce the variance while keeping the bias unchanged, 5) some methods stabilize the policy update by constraining its size.	1) For cases with huge state space, the estimation of the policy needs a lot of samples, 2) in practice, stable exploration is still a challenge, 3) policy can still change dramatically between training updates, making it challenging to converge on a good policy, 4) the policy is only as good as the value function is accurate.

Table 2.2: An overview of RL algorithms suitable for solving optimal liquidation tasks (2).

3. Summary of the publications

In this chapter, we briefly conclude the proposed RegPred Net and EOLS algorithm in terms of methodology and results in Sec. 3.1. Then we illustrate how these two algorithms are combined to form an end-to-end framework for solving the optimal liquidation problem of foreign currency.

3.1 RegPred Net and EOLS algorithm

RegPred Net For solving the foreign currency optimal liquidation problem, the RegPred Net is used to forecast the unknown dynamics of the FX rate series over the liquidation period. The design of RegPred Net is inspired by RNN (Sec. 2.2.2). One of the difference between them is that RegPred Net does not have a large number of learnable parameters and is a model-based algorithm. We assume the logarithm of FX rate $x_t, t \in \{0, \dots, T\}$ follows a generalized OU process (Sec. 2.1.1). The parameters A, N, Σ of this process describe the dynamics of the logarithmic FX rate over some period. Based on this we use an online regression method to calibrate these parameters of a given series $\{x_0, \dots, x_T\}$, where online indicates gradually updating the parameters A_t, N_t, Σ_t with new input x_t at each step t . The online updating algorithm is encapsulated in a computing unit called the Regression Cell (RegCell). RegCell recurrently processes the input series over time direction to calibrate a sequence of parameters $\{[A_0, N_0, \Sigma_0], [A_1, N_1, \Sigma_1], \dots, [A_T, N_T, \Sigma_T]\}$. Multiple RegCells can be stacked vertically to form a Regression Network (RegNet). Similar to the recurrent neural network, the calibrated parameter sequence from the previous layer is the input for the next layer. The calibrated parameters at layer k and time t are denoted as $[A_t^{(k)}, N_t^{(k)}, \Sigma_t^{(k)}]$.

Suppose the number of total layers is K . We obtain $[A_T^{(K)}, N_T^{(K)}, \Sigma_T^{(K)}]$ at layer K and time T after calibrating the parameters from time 0- T and layer 1- K . We assume that they remain unchanged over future N steps $t \in \{T + 1, \dots, T + N\}$, where N is the length FX rate to be forecasted. We use a unit called Prediction Cell (PredCell) to recover the parameters $[A_t^{(K)}, N_t^{(K)}, \Sigma_t^{(K)}]$ to $[A_t^{(1)}, N_t^{(1)}, \Sigma_t^{(1)}]$ layer by layer for future N steps. The stacked PredCells over time and vertical directions form the Prediction Network (PredNet). In the end, PredNet outputs the forecasted dynamic parameters over $t \in \{T + 1, \dots, T + N\}$.

With these parameters, we can simulate the logarithm of the FX rate x_t in the next N steps and calculate its exponential to recover to the FX rate series. RegNet and PredNet together is called the Regression-Prediction Network (RegPred Net). The hyperparameters of RegPred Net include the initial states of dynamic parameters and the learning rate. They determine the forecasted results of the network and need to be chosen carefully. We use Bayesian optimization to search for suitable hyperparameters.

We split the complete FX rate series into sub-series to form a data set. For each sample in the data set, we divide it into training, validation and testing sections in order. The training section is used as input for training the network, and the validation section is used as the label to calculate the square loss. Then, we use Bayesian optimization to heuristically tune the hyperparameters for RegPred Net and find the set of parameters that minimizes the loss. For forecasting, the concatenation of training and validation sections is used as input, and the testing section is used as the label. We evaluate the difference between the forecast and testing section with several metrics, including RMSE, *Pearson's R*, R-squared and MDA and take the average values as the final results. The results show that RegPred Net can forecast the long-term FX rate series and its dynamic parameters. Long-term, in our case, indicates the 100 steps' liquidation period. It outperforms other state-of-the-art forecasting methods, including LSTM, Auto-LSTM, ARMA and ARIMA. Compared to DL algorithms, RegPred Net is an explainable method designed explicitly for FX rate forecasting. The forecasted dynamic parameters help the EOLS algorithm estimate the optimal liquidation strategy.

EOLS EOLS algorithm estimates the quasi-optimal strategy for the foreign currency liquidation task. The design of EOLS is inspired by SDP (see Sec. 2.3.1). The solution computed by the SDP is considered to be the optimal solution under a hypothetical environmental model in which the logarithm of the foreign exchange rate x_t follows a generalized OU process. The state transition probability in the environmental model is $p(x_{t+1}|x_t) = \Pr(x_{t+1}|x_t), t \in [0, T - 1]$. If the dynamic parameters A, N, Σ are known, we can use them to simulate a large number of random trajectories of x_t . Each has a length T . Then estimate the maximum and minimum values x_{max}, x_{min} over the trajectories and assume they are the upper and lower bound of x_t . We can discretize x_t with an interval of $\frac{x_{max}-x_{min}}{N_x}$, where N_x is the discretization number of x_t and it divides the range between x_{min} and x_{max} into N_x parts. The probability $p(x_{t+1} = x' | x_t = x)$ of x at t transition to x' at $t + 1$ can be calculated from a cumulative normal distribution. Other state variables such as time and remaining volume can be discretized similarly. At this point, we obtained complete knowledge about the state transitions in the environment. SDP estimates the optimal policy by traversing all possible states $s \in S$ at

each step t . It calculates the expected cumulative reward or return obtained after taking some possible actions $a \in A(s)$ for state s and selects the one that generates the largest return as the optimal action at s . SDP algorithm has extremely high time complexity. The estimated strategy’s performance varies with the granularity of variable discretization (e.g. the size of N_x). In general, the more dense the granularity of the discretization, the more accurate the optimal policy estimated by SDP is and the more computation time required. For these reasons, we found that SDP is difficult to apply to solve our problem. This is also the case for other RL algorithms.

EOLS is derived by analyzing the solution calculated by SDP and greatly simplifying it. Suppose the state of liquidation consists of 3 variables: the logarithmic FX rate x , the remaining volume v and the time t . We use the SDP algorithm to compute the optimal strategy a^* . a^* can be visualized by a video sequence, where the image of each frame $t = 0, \dots, T$ is a 3D plot of the function $f_t : (x, v) \mapsto a^*(x, v, t)$. We observe these images and analytically formulize a^* in form of a parameterized function $a^*(x, v, t, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the parameters that determine a threshold value x_{thres} for x above which $a^*(x, v, t, \boldsymbol{\theta}) = 0$. It implies that the agent will not sell at steps where x is larger than x_{thres} due to a low reward of $a/\exp(x)$. We use a grid search method to find $\boldsymbol{\theta}$ that determines $a^*(x, v, t, \boldsymbol{\theta})$. We choose the $\boldsymbol{\theta}$ that maximizes the average return by following $a^*(x, v, t, \boldsymbol{\theta})$ on sufficient number of FX rate trajectories simulated by dynamic parameters. We experimentally show that the EOLS algorithm greatly improves the computational efficiency compared to SDP, DQN and PPO. The liquidation performance of the optimal strategy estimated by EOLS outperforms DQN and PPO and is similar to SDP when the discretization number $N_x = 100$.

The RegPred Net and EOLS algorithms form the complete framework for estimating the optimal liquidation strategy of foreign currency. We illustrate it in Fig. 3.1.

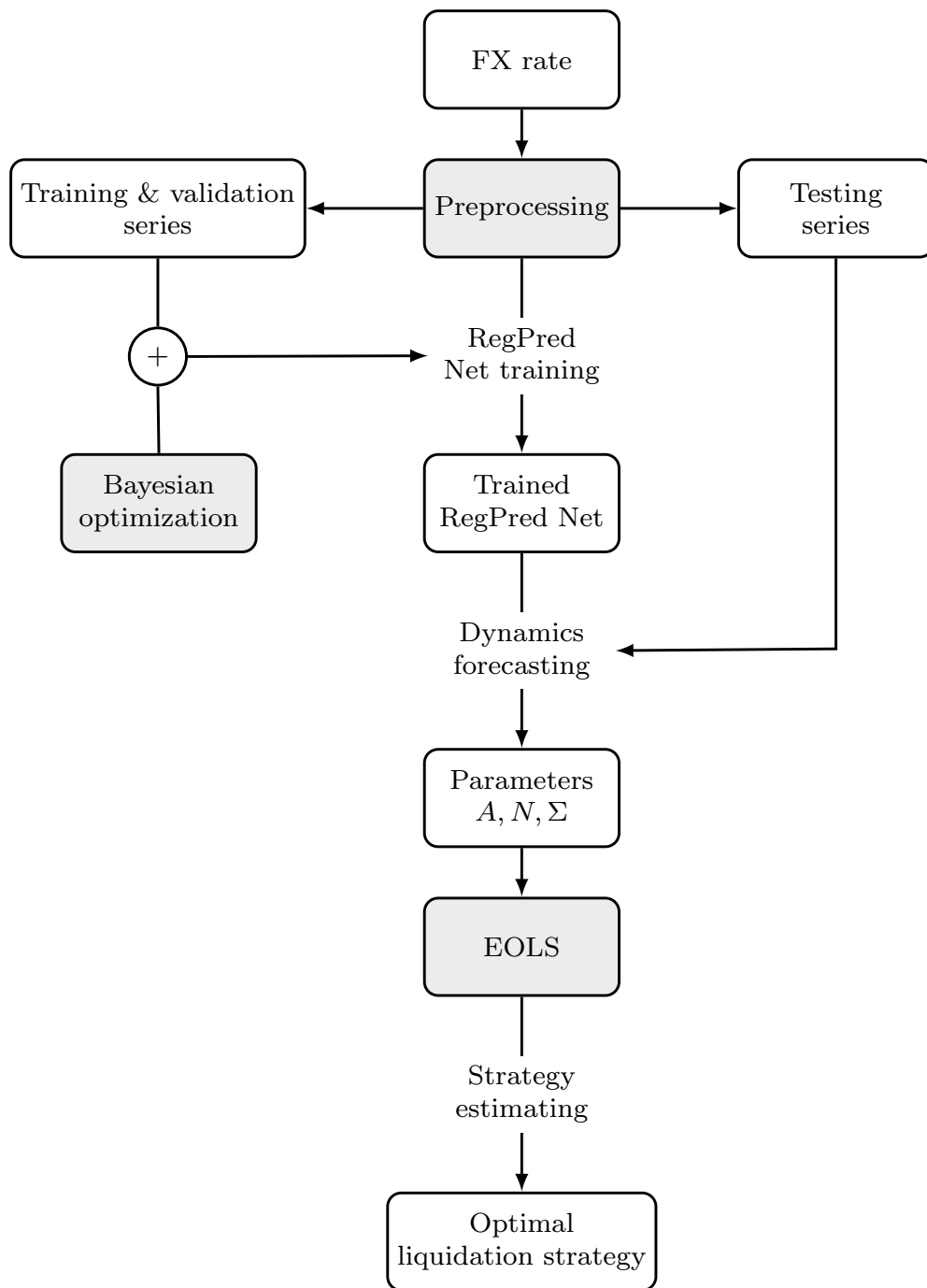


Figure 3.1: An illustration of optimal liquidation framework of foreign currency.

Bibliography

- [1] Amine Mohamed Aboussalah and Chi-Guhn Lee. Continuous control with stacked deep dynamic recurrent reinforcement learning for portfolio optimization. *Expert Systems with Applications*, 140:112891, 2020.
- [2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.
- [3] Robert Almgren and Neil Chriss. Optimal execution of portfolio transactions. *Journal of Risk*, pages 5–39.
- [4] Dimitros Asteriou and Stephen G Hall. Arima models and the box–jenkins methodology. *Applied Econometrics*, 2(2):265–286, 2011.
- [5] Wenhong Bao and Xiao-yang Liu. Multi-agent deep reinforcement learning for liquidation strategy analysis. *arXiv preprint arXiv:1906.11046*, 2019.
- [6] G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994.
- [7] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- [8] E. Brochu, M. V. Cora, and N. de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modelling and Hierarchical Reinforcement Learning. *CoRR*, abs/1012.2, 2010.
- [9] Wei Cao, Weidong Zhu, Wenjun Wang, Yves Demazeau, and Chen Zhang. A deep coupled lstm approach for usd/cny exchange rate forecasting. *IEEE Intelligent Systems*, 35(2):43–53, 2020.
- [10] Ping-Man Choi. *Reinforcement Learning in Nonstationary Environments*, chapter 6, pages 89–102. 2000.
- [11] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading.

- IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.
- [12] Volodymyr Mnih et al. Asynchronous methods for deep reinforcement learning. <https://arxiv.org/pdf/1602.01783.pdf>, 20–22 Jun 2016.
- [13] P. I. Frazier. A Tutorial on Bayesian Optimization. *ArXiv*, abs/1807.0, 2018.
- [14] Shen Furao, Jing Chao, and Jinxi Zhao. Forecasting exchange rate using deep belief networks and conjugate gradient method. *Neurocomputing*, 167:243–253, 2015.
- [15] Paul Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*. 05 2017.
- [16] Svitlana Galeshchuk. Neural networks performance in exchange rate prediction. *Neurocomputing*, 172:446–452, 2016.
- [17] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause, editors, *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018.
- [18] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.
- [19] Dieter Hendricks and Diane Wilcox. A reinforcement learning extension to the almgren-chriss framework for optimal trade execution. In *2014 IEEE Conference on Computational Intelligence for Financial Engineering Economics (CIFEr)*, pages 457–464, 2014.
- [20] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.
- [21] Martin Jacobsen. Laplace and the origin of the Ornstein-Uhlenbeck process. *Bernoulli*, 2(3):271 – 286, 1996.
- [22] Herman Kahn and Theodore E Harris. Estimation of particle transmission by random sampling. *National Bureau of Standards applied mathematics series*, 12:27–30, 1951.
- [23] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. Neural architecture search with bayesian optimisation and optimal transport.

- In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2016–2025. Curran Associates, Inc., 2018.
- [24] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.
- [25] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. *CoRR*, abs/1703.07015, 2017.
- [26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [27] 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. <https://arxiv.org/pdf/1509.02971.pdf>, 2015.
- [28] G. Lindgren, H. Rootzen, and M. Sandsten. *Stationary Stochastic Processes for Scientists and Engineers*. CRC Press, 2013.
- [29] Hui Liu, Xiwei Mi, and Yanfei Li. Smart multi-step deep learning model for wind speed forecasting based on variational mode decomposition, singular spectrum analysis, lstm network and elm. *Energy Conversion and Management*, 159:54–64, 2018.
- [30] A. G. Malliaris. *Wiener Process*, pages 276–278. Palgrave Macmillan UK, London, 1990.
- [31] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [32] Terence C Mills and Terence C Mills. *Time series techniques for economists*. Cambridge University Press, 1991.
- [33] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

- [34] Volodymyr Mnih and Kavukcuoglu et al. Human-level control through deep reinforcement learning. <https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf>, 02 2015.
- [35] Volodymyr Mnih, Koray Kavukcuoglu, and David Silver et al. Playing atari with deep reinforcement learning. <https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf>, 2013.
- [36] J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization 2*, 2:117–129, 2014.
- [37] Chakradhara Panda and V. Narasimhan. Forecasting exchange rate better with artificial neural network. *Journal of Policy Modeling*, 29(2):227–236, 2007.
- [38] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.
- [39] Sheldon M. Ross. *Introduction to Stochastic Dynamic Programming: Probability and Mathematical*. Academic Press, Inc., USA, 1983.
- [40] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [41] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. 1986.
- [42] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.
- [43] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.
- [44] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

- [45] Georgios Sermpinis, Christian Dunis, Jason Laws, and Charalampos Stasinakis. Forecasting and trading the eur/usd exchange rate with stochastic neural network combination and time-varying leverage. *Decision Support Systems*, 54(1):316–329, 2012.
- [46] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012.
- [47] Shaolong Sun, Shouyang Wang, and Yunjie Wei. A new ensemble deep learning approach for exchange rates forecasting and trading. *Advanced Engineering Informatics*, 46:101160, 2020.
- [48] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, chapter 13, pages 270–273. MIT Press, Cambridge, MA, USA, 2st edition, 2017.
- [49] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, chapter 3, pages 42–55. A Bradford Book, Cambridge, MA, USA, 2018.
- [50] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.
- [51] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, chapter 4, pages 59–73. A Bradford Book, Cambridge, MA, USA, 2018.
- [52] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, chapter 5, pages 75–95. A Bradford Book, Cambridge, MA, USA, 2018.
- [53] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, chapter 6, pages 97–113. A Bradford Book, Cambridge, MA, USA, 2018.
- [54] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, chapter 9-11, pages 161–232. A Bradford Book, Cambridge, MA, USA, 2018.
- [55] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, chapter 13, pages 265–279. A Bradford Book, Cambridge, MA, USA, 2018.

- [56] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, chapter 13, pages 274–275. A Bradford Book, Cambridge, MA, USA, 2018.
- [57] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando De Freitas. Bayesian optimization in a billion dimensions via random embeddings. *J. Artif. Int. Res.*, 55(1):361–387, jan 2016.
- [58] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, Nando De Freitas, et al. Bayesian optimization in high dimensions via random embeddings. In *IJCAI*, pages 1778–1784. Citeseer, 2013.
- [59] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.
- [60] Xing Wu, Haolei Chen, Jianjia Wang, Luigi Troiano, Vincenzo Loia, and Hamido Fujita. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*, 538:142–158, 2020.
- [61] Liu Yunpeng, Hou Di, Bao Junpeng, and Qi Yong. Multi-step ahead time series forecasting for different data patterns based on lstm recurrent neural network. In *2017 14th Web Information Systems and Applications Conference (WISA)*, pages 305–310, 2017.

Part II

Publications

Forecasting foreign exchange rates with regression networks tuned by Bayesian optimization	48
Optimal liquidation of foreign currencies when FX rates follow a generalised Ornstein-Uhlenbeck process	78

I. Forecasting foreign exchange rates with regression networks tuned by Bayesian optimization

Bibliographic Information

Li, L., Matt, PA., Heumann, C. (2022). Forecasting foreign exchange rates with regression networks tuned by Bayesian optimization. arXiv:2204.12914.

Author's contribution

Linwei Li and Paul-Amaury Matt jointly design the models and the experiments for performance comparison on this task. The implementation of the model and experiments were done by Linwei Li and jointly revised with Paul-Amaury Matt. The paper was reviewed by Paul-Amaury Matt and revised by Linwei Li. Christian Heumann provided valuable suggestions for the paper and further reviewed it.

Copyright Notice

©2023 arXiv. This is an accepted version of this article published in arXiv:2204.12914. Copyright: arXiv.org perpetual, non-exclusive license.

Forecasting foreign exchange rates with regression networks tuned by Bayesian optimization

Linwei Li^{a,b,*,1}, Paul-Amaury Matt^{b,2} and Christian Heumann^{a,3}

^aDepartment of Statistics, University of Munich (LMU), Ludwigstr. 33, 80539, Munich, Germany

^bDaimler AG, Breitwiesenstr. 5, Stuttgart, Germany

ARTICLE INFO

Keywords:

Time series forecasting
Foreign Exchange rate
Regression network
Bayesian optimization
Ornstein-Uhlenbeck process

ABSTRACT


The article is concerned with the problem of multi-step financial time series forecasting of Foreign Exchange (FX) rates. To address this problem, we introduce a regression network termed RegPred Net. The exchange rate to forecast is treated as a stochastic process. It is assumed to follow a generalization of Brownian motion and the mean-reverting process referred to as generalized *Ornstein-Uhlenbeck* (OU) process, with time-dependent coefficients. Using past observed values of the input time series, these coefficients can be regressed online by the cells of the first half of the network (Reg). The regressed coefficients depend only on t but are very sensitive to t - a small number of hyperparameters required to be set by a global optimization procedure for which, Bayesian optimization is an adequate heuristic. Thanks to its multi-layered architecture, the second half of the regression network (Pred) can project time-dependent values for the OU process coefficients and generate realistic trajectories of the time series. Predictions can be easily derived in the form of expected values estimated by averaging values obtained by Monte Carlo simulation. The forecasting accuracy on a 100 days horizon is evaluated for several of the most important FX rates such as EUR/USD, EUR/CNY and EUR/GBP. Our experimental results show that the RegPred Net significantly outperforms ARMA, ARIMA, LSTMs, and Autoencoder-LSTM models in terms of metrics measuring the absolute error (RMSE) and correlation between predicted and actual values (*Pearson's R*, *R-squared*, *MDA*). Compared to black-box deep learning models such as LSTM, RegPred Net has better interpretability, simpler structure, and fewer parameters. In addition, it can predict dynamic parameters that reflect trends in exchange rates over time, which provides decision-makers with important information when dealing with sequential decision-making tasks.

1. Introduction

Foreign Exchange (FX) rate time series are considered in Academia and the Finance Industry to be some of the most challenging time series to forecast, due to their fast-changing trends, high volatilities and complex dependencies on a large number of macro-economic factors. Nevertheless, several important applications in Market Finance (FX trading (Donnelly, 2019), pricing and hedging of FX derivatives (Hull, 2018)) and International Corporate Finance (Currency Risk Management) (Jacque, 2014) rely on accurate long-term forecasts or realistic simulations of FX rates.

With the rise of Deep Learning (LeCun et al., 2015), (Hochreiter and Schmidhuber, 1997) and its successes in Computer vision and Natural Language Processing in the last years, deep neural networks have started to be introduced in the area of Financial Time Series Forecasting, as with the works of (Guo et al., 2014), (Bao et al., 2017) and (Dingli and Fournier, 2017). Nevertheless, we observe experimentally that when applied to FX rates famous Deep Learning models for time series such as LSTMs (Hochreiter and Schmidhuber, 1997) do not offer particularly good performance for multi-step forecasting. Additionally, these types of neural networks operate like “black boxes” and do not offer any insight regarding the dynamics of the time series considered.

To address the issues of performance and explainability, we propose a novel regression network called RegPred Net. The explainability of the model is achieved by design of the network’s architecture, whereby it is assumed that the time series follows a stochastic (random) process whose parameters can be directly interpreted in terms of drift, mean-reversion level, mean-reversion rate and volatility. The network’s forecasting performance is essentially the result of i) using a sufficiently general stochastic process, referred to as generalized *Ornstein-Uhlenbeck* (OU) process which encompasses Brownian motion with drift and the mean-reverting process as special cases, ii) the online regression of the parameters of the stochastic model by Regression Cells, iii) carefully choosing the network’s hyperparameters

 Linwei.Li@campus.lmu.de (L. Li); paul-amaury.matt@daimler.com (P. Matt); chris@stat.uni-muenchen.de (C. Heumann)

by Bayesian optimization (Jonas, 1989), and iv) using a multi-layer network architecture to also capture the time-dependency of the parameters of the stochastic process.

The RegPred Net is described by a number of hyperparameters playing the role of learning rates or initial regression cell states (RegCells). The choice of the hyperparameters' values has a huge impact on the network's predictions and accuracy, so instead of setting hyperparameters arbitrarily, we adopt Bayesian optimization as an efficient procedure to set their value optimally. Bayesian optimization is a global optimization heuristic that is gaining popularity in Machine Learning for hyperparameter-tuning.

The rest of this article is organized as follows. Section 2 is dedicated to a discussion of related work. In Section 3, we introduce the stochastic process used for modelling FX rates, viz. the generalized OU process, explain how its parameters can be regressed in an offline-fashion and then derive an online regression procedure. Section 4 provides a detailed description of the regression cells and architecture of the RegPred Net, as well as algorithms for simulation, forecasting and calculation of the network's loss function. Section 5 offers some background on Bayesian optimization and presents the method and algorithms used for training RegPred Net. Section 6 offers a detailed experimental validation and we finally conclude in Section 7.

2. Related work

Time series forecasting has been since decades an important area of research and development in Academia and Industry. A particular difficulty with time series is multi-step predictions of time series that do not exhibit a clear and stable trend or seasonality, which is typically the case with financial time series such as stock prices or currency rates.

In (Siarni-Namini et al., 2018), the authors compared the performance of the ARIMA model with LSTM and concluded that LSTM outperforms ARIMA, with more than 80% reduction of error on one-step-ahead time series forecasting. The authors of (Gensler et al., 2016) compared Auto-LSTM with the standard LSTM on solar power data forecasting tasks and showed that in that case, Auto-LSTM performed better than LSTM when predicting two-steps-ahead, although the performance of both models was similar. (Bao et al., 2017) proposed a novel LSTM-based structure that stacks wavelet transformation, autoencoders, and LSTM together and this new model outperformed LSTM on a one-step-ahead financial time series forecasting problem.

All the research works mentioned in the previous paragraph are limited to only one or two-step ahead predictions. Unfortunately, such approaches do not benefit a whole range of real-world activities that typically relate to risk analysis or sequential decision making. Regarding multi-steps forecasting models, (Yunpeng et al., 2017) used the LSTM to predict different types of periodic time series and achieved better performance than the ARIMA model. (Liu et al., 2018) also proposed a LSTM-based model that combines variational mode decomposition, singular spectrum analysis, and extreme learning machine to make one to five-steps ahead wind speed prediction. However, five-steps ahead is still considered a short period, and time series like wind speed have intrinsic cycles, unlike currency rates.

Instead of predicting the values of a times series, a simpler approach often followed consists in assessing either the probability of an increase or decrease of the time series at some future point in time compared to the present, or the probability of being higher or lower than a reference value. In (Gyamerah, 2019), the authors use an LSTM-based model to predict whether a *S&P500* stock price will increase or decrease in the next time step and conclude that the LSTM performs better than other machine learning models such as random forest and logistic regression. In (Fischer and Krauss, 2017), the authors seek to predict the probability that a stock outperforms its cross-sectional median at the next time step. Their results also indicate that LSTMs outperforms other traditional machine learning models. (Rangapuram et al., 2018) forecasts the posterior distribution of future trajectories of time series given the past. The experiments were made on periodic electricity and traffic time series and showed that the proposed method performed well (especially on limited data) by modelling the seasonal structure of the dataset.

The above-proposed forecasting approaches all are either limited to short term forecasts, to categorical forecasting, or applied to periodic data. To the best of our knowledge, frameworks suitable for numerically forecasting complicated non-stationary time series in the medium to long term (100 steps or more) are quite rare. The RegPred Net is a novel type of Recurrent Network that was developed to meet these requirements and that unlike some other RNNs extracts interpretable features of the predicted time series in the form of the parameters of an OU process, thereby providing accurate information about the trend, mean-reversion level or rate and volatility of the process. Unlike neural networks in Deep Learning that often have millions of weights to learn and store in memory, the RegPred Net is a completely weight-free network.

Finally, (Brochu et al., 2010) provided a detailed tutorial on Bayesian optimization and discussed the pros and

cons of this method in practice. (Lizotte, 2008) explained in his Ph.D. thesis that Bayes-optimal acquisition criteria although being rarely studied can improve the efficiency of Bayesian optimization and indicates that using $\xi = 0.01$ as exploration parameter of acquisition function performs well in most cases. Bayesian optimization is also widely used in Machine Learning. The authors in (Snoek et al., 2012) show that Bayesian optimization outperforms human expert-level on parameter tuning of machine learning algorithms like SVMs, Convolutional Neural Networks, and Latent Dirichlet Allocations. (Kandasamy et al., 2018) proposed a framework based on Bayesian optimization to automatically select the architecture for deep neural networks and their results show that their framework outperforms other baseline methods on several data sets.

3. Stochastic process for FX rates

3.1. Foreign exchange rates

A currency is a system of money in general use in a particular country. We refer to a given country's currency as its domestic currency and refer to the currencies of other countries as foreign currencies. In Finance, a foreign exchange (FX) rate is the rate at which one currency is exchanged for another. It is also regarded as the value of one country's currency in relation to another currency. For example, the daily FX rate of EUR/CNY (Euro/Chinese Yuan) on Mar. 19, 2020 was 7.68, which means that 1 Euro was worth 7.68 Yuan. Fig. 1 shows the 5000 days' daily FX rates of EUR/CNY, EUR/USD (US Dollar) and EUR/GBP (British Pound). The horizontal and vertical axes are time and FX rates, respectively. Since standardized currencies around the world float in value with demand, supply and consumer confidence, their relative values change over time, as illustrated in Fig. 1.

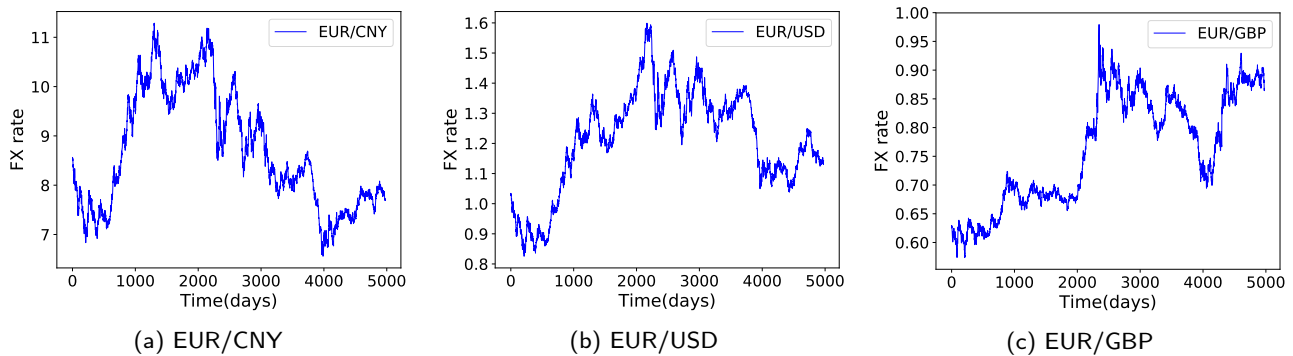


Figure 1: The daily FX rates of EUR/CNY, EUR/USD, and EUR/GBP over 5000 days.

3.2. Stochastic processes

Daily FX rates can be modelled by discrete stochastic processes. The word stochastic is synonym of random. A discrete stochastic process is a system which evolves in time while undergoing random fluctuations over time. We describe such a system by defining a family of random variables $\{X_t\}_{t \in \mathbb{N}}$, where X_t measures at time t the aspect of the system which is of interest.

The discrete Wiener process (Malliari, 1990) W_t is a discrete stochastic process defined for time steps t where t is a positive or null integer. The process is defined by the following properties: i) $W_0 = 0$, and ii) for every $t \geq 1$, the process increment given by the difference $\Delta W_t = W_t - W_{t-1}$ is independently and normally distributed:

$$\Delta W_t \sim \mathcal{N}(0, 1) \quad (1)$$

Thus, the increment of the Wiener process is independent of its past values (Markov property).

Discrete Brownian motion with drift is another discrete stochastic process X_t based on the Wiener process, defined by i) $X_0 \in \mathbb{R}$ and ii) for every $t \geq 1$, $\Delta X_t = X_t - X_{t-1} = \mu + \sigma \Delta W_t$, where the $\mu \in \mathbb{R}$ is a parameter called drift and $\sigma > 0$ is a second parameter called volatility. The Wiener process is a special case of Brownian motion where the drift is null ($\mu = 0$) and the volatility is one ($\sigma = 1$). Here, the drift parameter is interpreted as the deterministic trend of the process and the volatility as the amplitude of the noise or non-deterministic component in the process.

The mean reverting process, also called *Ornstein-Uhlenbeck* (OU) process, is defined by i) $X_0 \in \mathbb{R}$ and ii) $\Delta X_t = X_t - X_{t-1} = \alpha(n - X_{t-1}) + \sigma \Delta W_t$ where $\alpha \in \mathbb{R}$ is a parameter called mean-reversion rate, $n \in \mathbb{R}$ is called mean-reversion

level and $\sigma > 0$ is called volatility. This equation describes the dynamics of a variable that randomly fluctuates around some mean level n , follows random increments with an amplitude controlled by σ and tends to revert back to n at a speed controlled by α (assuming this parameter has a strictly positive value). FX rates are often modelled as mean reverting processes over long periods and as Brownian motion with drift over short periods. Fig. 2 illustrates some trajectories of a Wiener process, a Brownian motion and a mean-reverting process over a period of 500 time steps.

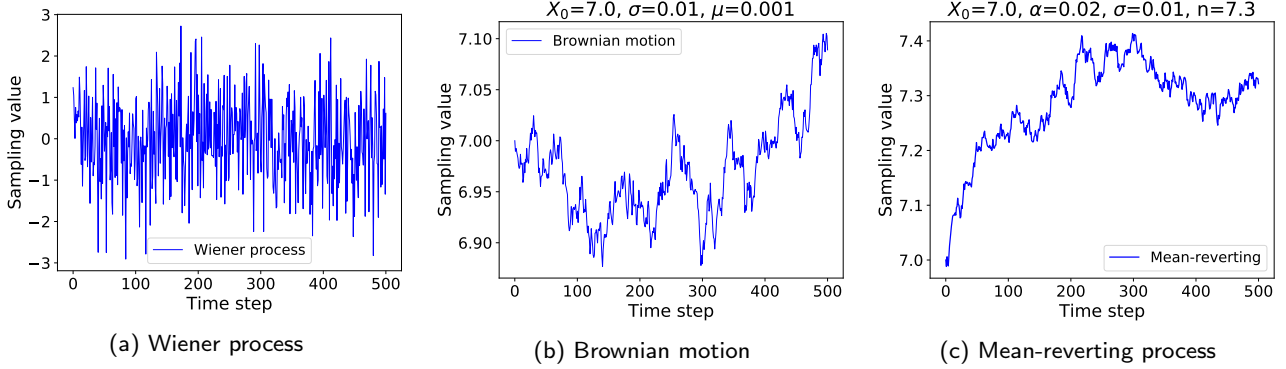


Figure 2: Trajectories of a Wiener process, Brownian motion and mean-reverting process.

3.3. Generalized Ornstein-Uhlenbeck process

We now generalize the OU process X_t of dimension 1 (univariate process) to a multivariate process \mathbf{Y}_t of dimension d . Thus, \mathbf{Y}_t denotes here a d -dimensional vector. The process is defined by $\mathbf{Y}_0 \in \mathbb{R}^d$ and

$$\Delta \mathbf{Y}_t = \mathbf{Y}_t - \mathbf{Y}_{t-1} = \mathbf{A} \cdot \mathbf{Y}_{t-1} + \mathbf{N} + \mathbf{\Sigma} \cdot \Delta \mathbf{W}_t \quad (2)$$

where $\mathbf{A} = (\alpha_{i,j})$ is a real-valued square matrix of dimension $d \times d$, \mathbf{N} is a real-valued vector of dimension d , $\mathbf{\Sigma} = (\sigma_{i,j})$ is a real-valued square matrix of dimension $d \times d$, and $\Delta \mathbf{W}_t = (\Delta W_{i,t})$ is a d -dimensional vector where the components $\Delta W_{i,t} \sim \mathcal{N}(0, 1)$ are identically and independently normally distributed.

Observe that when $\mathbf{A} = 0$, $\mathbf{N} = 0$, and $\mathbf{\Sigma} = I_d$, the generalized OU process becomes a d -dimensional Wiener process. When $\mathbf{A} = 0$, the generalized OU process becomes a d -dimensional Brownian motion with drift \mathbf{N} and volatility $\mathbf{\Sigma}$. Finally when $\mathbf{A} = -\alpha \mathbf{I}$, $\mathbf{N} = \alpha \cdot n$ and $\mathbf{\Sigma} = \sigma$, the generalized OU process is a d -dimensional OU process with parameters α , n and σ .

3.4. Offline regression of a generalized Ornstein-Uhlenbeck process

Assume that we have observations for the values of the vector \mathbf{Y}_t for $t = 1, \dots, T$, and wish to estimate the parameters \mathbf{A} , \mathbf{N} , $\mathbf{\Sigma}$ of the generalized OU process. A so called ‘‘offline’’ method such as Ordinary Least Squares can be used, where offline (Karp, 1992) means that the regression algorithm uses the whole data set of observations $\mathbf{Y}_{1:d, 1:T}$ at once.

To calibrate the parameters in Eq. (2) using data $\mathbf{Y}_{1:d, 1:T}$, we first rewrite Eq. (2) as a linear equation of the form $\mathbf{y}_t = \boldsymbol{\beta} \mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t$, where

$$\begin{aligned} \mathbf{y}_t &= \mathbf{Y}_t - \mathbf{Y}_{t-1} \\ \boldsymbol{\beta} &= \begin{bmatrix} n_1 & a_{1,1} & \dots & a_{1,d} \\ n_2 & a_{2,1} & \dots & a_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ n_d & a_{d,1} & \dots & a_{d,d} \end{bmatrix} \\ \mathbf{x}_{t-1} &= [1, \mathbf{Y}_{1, t-1}, \dots, \mathbf{Y}_{d, t-1}]^T \\ \boldsymbol{\epsilon}_t &= \mathbf{\Sigma} \cdot \Delta \mathbf{W}_t = \begin{bmatrix} \epsilon_{1,t} \\ \vdots \\ \epsilon_{d,t} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^d \sigma_{1,k} \cdot \Delta W_{k,t} \\ \vdots \\ \sum_{k=1}^d \sigma_{d,k} \cdot \Delta W_{k,t} \end{bmatrix} \end{aligned} \quad (3)$$

According to the ordinary least square method, the loss L^{OLS} between observation \mathbf{y} and function value of the linear model $\beta\mathbf{x}$ is:

$$L^{OLS} = \frac{1}{2} \sum_{t=1}^T \left\| \mathbf{y}_t - \beta \mathbf{x}_{t-1} \right\|_2^2 \quad (4)$$

where $\|\cdot\|_2^2$ is the square of matrix norm 2 distance. Since L^{OLS} is a positive quadratic function of β which admits a minimum, the optimal value of β can be computed by solving the quadratic problem:

$$\frac{\partial L^{OLS}}{\partial \beta} = 0 \quad (5)$$

The optimal value of β is:

$$\beta = \left(\sum_{t=1}^T \mathbf{y}_t \mathbf{x}_{t-1}^T \right) \left(\sum_{t=1}^T \mathbf{x}_{t-1} \mathbf{x}_{t-1}^T \right)^{-1} \quad (6)$$

The parameters \mathbf{A} and \mathbf{N} can be retrieved from β by identification with Eq. (3):

$$[\mathbf{N}; \mathbf{A}] = \beta \quad (7)$$

To estimate Σ , we first compute the covariance matrix $\mathbf{K}_{\epsilon\epsilon,t}$ of $\epsilon_t = [\epsilon_{1,t}, \dots, \epsilon_{d,t}]^T$:

$$\mathbf{K}_{\epsilon\epsilon,t} = \begin{bmatrix} \text{cov}(\epsilon_{1,t}, \epsilon_{1,t}) & \text{cov}(\epsilon_{1,t}, \epsilon_{2,t}) & \dots & \text{cov}(\epsilon_{1,t}, \epsilon_{d,t}) \\ \text{cov}(\epsilon_{2,t}, \epsilon_{1,t}) & \text{cov}(\epsilon_{2,t}, \epsilon_{2,t}) & \dots & \text{cov}(\epsilon_{2,t}, \epsilon_{d,t}) \\ \vdots & \vdots & \ddots & \vdots \\ \text{cov}(\epsilon_{d,t}, \epsilon_{1,t}) & \text{cov}(\epsilon_{d,t}, \epsilon_{2,t}) & \dots & \text{cov}(\epsilon_{d,t}, \epsilon_{d,t}) \end{bmatrix}$$

where

$$\text{cov}(\epsilon_{i,t}, \epsilon_{j,t}) = \mathbb{E} \left[(\epsilon_{i,t} - \mathbb{E}(\epsilon_{i,t})) (\epsilon_{j,t} - \mathbb{E}(\epsilon_{j,t})) \right] = \sum_{k=1}^d \sigma_{i,k} \cdot \sigma_{j,k} \quad (8)$$

From Eq. (8) (Appx. B.1) we can see that $\mathbf{K}_{\epsilon\epsilon,t}$ is not related to t , so $\mathbf{K}_{\epsilon\epsilon}$ can be further written as:

$$\mathbf{K}_{\epsilon\epsilon} = \begin{bmatrix} \sum_{k=1}^d (\sigma_{1,k})^2 & \sum_{k=1}^d \sigma_{1,k} \sigma_{2,k} & \dots & \sum_{k=1}^d \sigma_{1,k} \sigma_{d,k} \\ \sum_{k=1}^d \sigma_{2,k} \sigma_{1,k} & \sum_{k=1}^d \sigma_{2,k} \sigma_{2,k} & \dots & \sum_{k=1}^d \sigma_{2,k} \sigma_{d,k} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^d \sigma_{d,k} \sigma_{1,k} & \sum_{k=1}^d \sigma_{d,k} \sigma_{2,k} & \dots & \sum_{k=1}^d (\sigma_{d,k})^2 \end{bmatrix} \quad (9)$$

Since $\mathbf{K}_{\epsilon\epsilon} = \Sigma \Sigma^T$, it is a positive definite matrix and thus can be decomposed using the Cholesky decomposition into a product of the form $\mathbf{M}\mathbf{M}^T$, where \mathbf{M} is a lower triangular matrix. So, we can estimate Σ by choosing $\Sigma = \mathbf{M}$.

3.5. Online regression of a generalized Ornstein-Uhlenbeck process

The parameters \mathbf{A} , \mathbf{N} , Σ of the generalized OU process described in Eq. (2) can also be calibrated in an "online" fashion. The word online (Karp, 1992) refers to any method that estimates the result of an algorithm without having all input data at once, but step-by-step processes the input. In this way, at every time step t , the online algorithm updates

the parameters from the previous time step $\mathbf{A}_{t-1}, \mathbf{N}_{t-1}, \boldsymbol{\Sigma}_{t-1}$ to $\mathbf{A}_t, \mathbf{N}_t, \boldsymbol{\Sigma}_t$. We replace in Eq. (2) the static parameters $\mathbf{A}, \mathbf{N}, \boldsymbol{\Sigma}$ by the time-dependent parameters as follows:

$$\Delta \mathbf{Y}_t = \mathbf{A}_{t-1} \cdot \mathbf{Y}_{t-1} + \mathbf{N}_{t-1} + \boldsymbol{\Sigma}_{t-1} \cdot \Delta \mathbf{W}_t \quad (10)$$

The error term $\boldsymbol{\epsilon}_t$ is defined as:

$$\boldsymbol{\epsilon}_t = \Delta \mathbf{Y}_t - (\mathbf{A}_{t-1} \mathbf{Y}_{t-1} + \mathbf{N}_{t-1}) = \begin{bmatrix} \epsilon_{1,t} \\ \vdots \\ \epsilon_{d,t} \end{bmatrix} \quad (11)$$

To infer the update rule for \mathbf{A}_t and \mathbf{N}_t , we define the quadratic loss $L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})$ at time t as:

$$L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1}) = \boldsymbol{\epsilon}_t^T \boldsymbol{\epsilon}_t \quad (12)$$

The update rules of \mathbf{A}_t and \mathbf{N}_t can be expressed as gradient descent steps:

$$\mathbf{A}_t \leftarrow \mathbf{A}_{t-1} - \eta_A \frac{\partial L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})}{\partial \mathbf{A}_{t-1}} \quad (13)$$

$$\mathbf{N}_t \leftarrow \mathbf{N}_{t-1} - \eta_N \frac{\partial L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})}{\partial \mathbf{N}_{t-1}} \quad (14)$$

where η_A, η_N and η_Σ are learning rates for $\mathbf{A}_t, \mathbf{N}_t$ and $\boldsymbol{\Sigma}_t$, respectively.

The partial derivative of $L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})$ with respect to \mathbf{A}_{t-1} is computed as:

$$\frac{\partial L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})}{\partial \mathbf{A}_{t-1}} = -2 \cdot \boldsymbol{\epsilon}_t \mathbf{Y}_{t-1}^T \quad (15)$$

Similarly, the partial derivative of $L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})$ with respect to \mathbf{N}_{t-1} is:

$$\frac{\partial L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})}{\partial \mathbf{N}_{t-1}} = -2 \cdot \boldsymbol{\epsilon}_t \quad (16)$$

The calculation details are given in Appx. B.2.

Substituting Eq. (15) and (16) into Eq. (13) and (14), we get the following update rules for \mathbf{A}_t and \mathbf{N}_t :

$$\mathbf{A}_t \leftarrow \mathbf{A}_{t-1} + 2\eta_A \boldsymbol{\epsilon}_t \mathbf{Y}_{t-1}^T \quad (17)$$

$$\mathbf{N}_t \leftarrow \mathbf{N}_{t-1} + 2\eta_N \boldsymbol{\epsilon}_t \quad (18)$$

Similarly, we use gradient descent to get the update rule for $\boldsymbol{\Sigma}_t$. We first define the loss $L_t(\boldsymbol{\Sigma}_{t-1})$ at time t as:

$$L_t(\boldsymbol{\Sigma}_{t-1}) = \left\| \boldsymbol{\Sigma}_{t-1} \boldsymbol{\Sigma}_{t-1}^T - \hat{c} \hat{v}(\boldsymbol{\epsilon}_t) \right\|_2^2 \quad (19)$$

where $\hat{c} \hat{v}(\boldsymbol{\epsilon}_t) = ((\hat{c} \hat{v}(\epsilon_{i,t}, \epsilon_{j,t})))$, and $\hat{c} \hat{v}(\epsilon_{i,t}, \epsilon_{j,t})$ is an online estimate at t of the covariance of $\epsilon_{i,t}$ and $\epsilon_{j,t}$.

The derivative of $L_t(\boldsymbol{\Sigma}_{t-1})$ over the whole matrix $\boldsymbol{\Sigma}_{t-1}$ is computed as (see Appx. B.2):

$$\frac{\partial L_t(\boldsymbol{\Sigma}_{t-1})}{\partial \boldsymbol{\Sigma}_{t-1}} = 4 \cdot (\boldsymbol{\Sigma}_{t-1} \boldsymbol{\Sigma}_{t-1}^T - \hat{c} \hat{v}(\boldsymbol{\epsilon}_t)) \cdot \boldsymbol{\Sigma}_{t-1} \quad (20)$$

Thus, $\boldsymbol{\Sigma}_t$ can be updated by:

$$\boldsymbol{\Sigma}_t \leftarrow \boldsymbol{\Sigma}_{t-1} - 4\eta_\Sigma \cdot (\boldsymbol{\Sigma}_{t-1} \boldsymbol{\Sigma}_{t-1}^T - \hat{c} \hat{v}(\boldsymbol{\epsilon}_t)) \cdot \boldsymbol{\Sigma}_{t-1} \quad (21)$$

To estimate the covariance matrix $c\hat{v}(\epsilon_t)$ in Eq. (21), we first estimate the expectation of ϵ_t by using an Exponential Moving Average (EMA) with weight φ :

$$\hat{\mathbb{E}}(\epsilon_t) = EMA_\varphi(\epsilon_t) = \varphi \cdot \epsilon_t + (1 - \varphi) \cdot EMA_\varphi(\epsilon_{t-1}) \quad (22)$$

Since the covariance matrix $cov(\epsilon_t)$ is mathematically defined as

$$cov(\epsilon_t) = \mathbb{E}[(\epsilon_t - \mathbb{E}[\epsilon_t])(\epsilon_t - \mathbb{E}[\epsilon_t])^T] \quad (23)$$

we can again introduce another EMA with weight ρ for estimating also the outer expectation operator and estimate the covariance matrix as:

$$c\hat{v}(\epsilon_t) = EMA_\rho[(\epsilon_t - \hat{\mathbb{E}}(\epsilon_t))(\epsilon_t - \hat{\mathbb{E}}(\epsilon_t))^T] \quad (24)$$

The online regression procedure allows to estimate the parameters $\mathbf{A}_t, \mathbf{N}_t, \mathbf{\Sigma}_t$ of the generalized OU process given only the following hyperparameters:

- initial values of $\mathbf{A}_0, \mathbf{N}_0, \mathbf{\Sigma}_0, \hat{\mathbb{E}}(\epsilon_0)$ and $c\hat{v}(\epsilon_0)$
- a 5-dimensional vector of learning rates $\mathbf{H} = [\eta_A, \eta_N, \eta_\Sigma, \varphi, \rho]$ composed of three learning rates $\eta_A, \eta_N, \eta_\Sigma$ used in gradient descent update rules and the weights φ and ρ of two exponential moving averages for $\hat{\mathbb{E}}(\epsilon_t)$ and $c\hat{v}(\epsilon_t)$.

Different values for the hyperparameters lead to different estimates for the parameters $\mathbf{A}_t, \mathbf{N}_t$ and $\mathbf{\Sigma}_t$, unlike the offline regression which always lead to the same result. For instance, with high values of \mathbf{H} , the parameters adapt very fast to the time series but contain more noise and tend to degrade the accuracy of long term forecasts. Conversely, small values of \mathbf{H} lead to slowly changing stochastic parameter estimates, which is good for long term but tends to degrade the accuracy of short term forecasts.

4. RegPred Network

This section introduces the two networks RegNet and PredNet that compose RegPred Net. We present their respective recurrent cells and network architecture. RegNet and PredNet are then simply juxtaposed to form the overall RegPred Net.

4.1. Regression Cell (RegCell) and Regression Network (RegNet)

In this subsection, we introduce a recurrent network termed RegNet for the online estimation of parameters of a generalized OU process \mathbf{Y}_t . The basic RegNet is a single layer network using a recurrent cell called RegCell. The RegCell at time t and layer k , which is defined in Algorithm 1 and illustrated in Fig. 3, simply encapsulates all the update rules needed for the online regression of the parameters $\mathbf{A}, \mathbf{N}, \mathbf{\Sigma}$ as described in the previous section. Several layers of the basic RegNet can be stacked on top of each other to form a multi-layered RegNet (see Fig. 4). In this case, the k -th layer performs online regression of the parameters $\mathbf{A}^{(k)}, \mathbf{N}^{(k)}, \mathbf{\Sigma}^{(k)}$ of an generalized OU process for the multivariate input series $\mathbf{Z}_t^{(k-1)}$, defined as the flattened vector of regressed parameters from the OU process in $k-1$ -th layer:

$$\forall k = 1, \dots, K \begin{cases} \Delta \mathbf{Z}_t^{(k-1)} = \mathbf{A}_{t-1}^{(k)} \mathbf{Z}_{t-1}^{(k-1)} + \mathbf{N}_{t-1}^{(k)} + \mathbf{\Sigma}_{t-1}^{(k)} \Delta \mathbf{W}_t^{(k)} \\ \mathbf{Z}_t^{(k-1)} = [\mathbf{A}_t^{(k-1)}, \mathbf{N}_t^{(k-1)}, \mathbf{\Sigma}_t^{(k-1)}] \end{cases} \quad (25)$$

with $\mathbf{Z}_t^{(0)} = \mathbf{Y}_t$.

In a RegNet (Fig. 4), the RegCell is replicated T times along the time axis at each time step from $t = 1$ to T , which allows for an iterative regression of the coefficients of an OU process modeling the univariate input time series Y_1, \dots, Y_T . These T RegCells form the first layer $k = 1$ of the RegNet. The outputs of layer $k = 1$ are the regressed coefficients $\mathbf{A}_t, \mathbf{N}_t, \mathbf{\Sigma}_t$ (1-dimensional for each) for the time steps $t = 1, \dots, T$. We append and flatten these coefficients

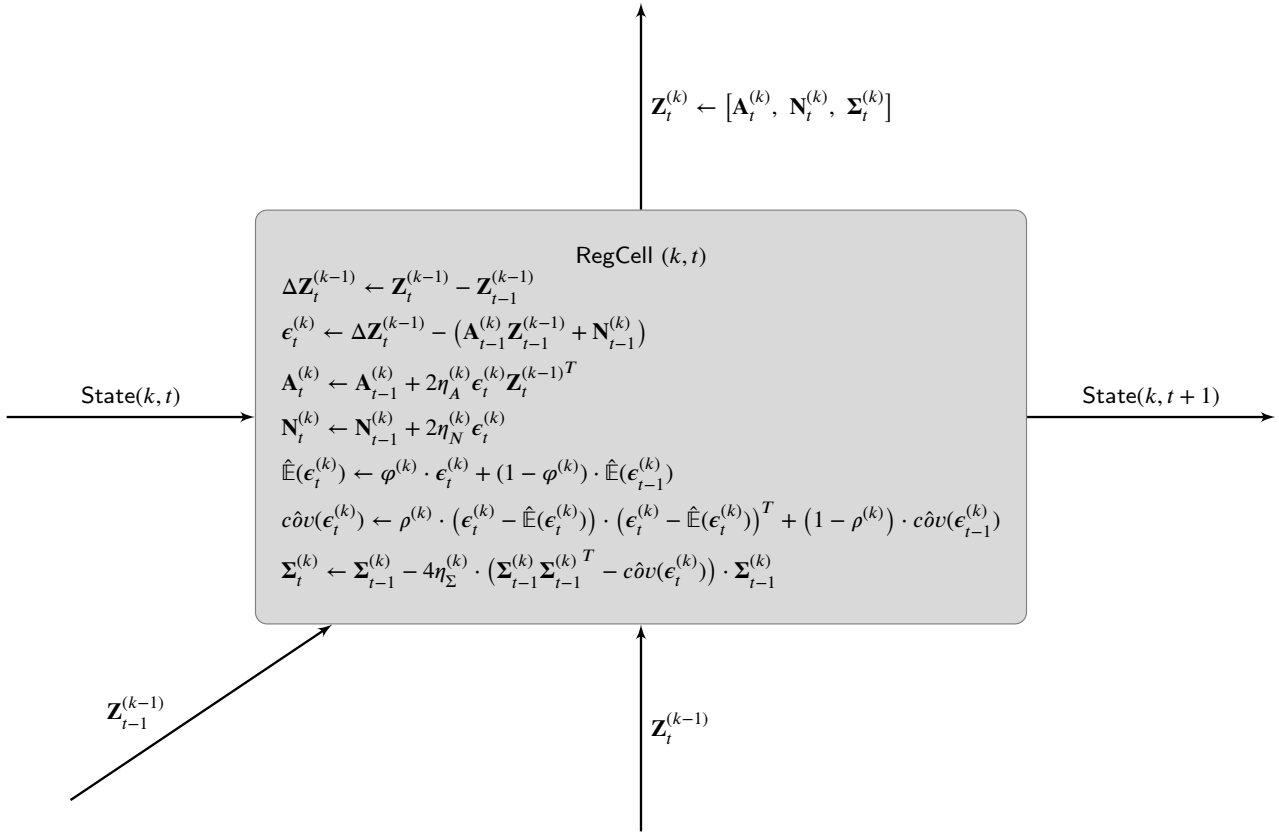


Figure 3: The RegCell (k, t) updates the parameters $\mathbf{A}, \mathbf{N}, \Sigma$ of the online regression model: $\Delta \mathbf{Z}_t^{(k-1)} = \mathbf{A}_{t-1}^{(k)} \mathbf{Z}_{t-1}^{(k-1)} + \mathbf{N}_{t-1}^{(k)} + \Sigma_{t-1}^{(k)} \Delta \mathbf{W}_t^{(k)}$. $\text{State}(k, t) := \mathbf{Z}_{t-1}^{(k)}, \hat{\mathbb{E}}(\epsilon_{t-1}^{(k)}), c\hat{\sigma}v(\epsilon_{t-1}^{(k)})$.

Algorithm 1 RegCell in layer k at time t

Input: $\text{State}(k, t) := \left\{ \mathbf{Z}_{t-1}^{(k)} := [\mathbf{A}_{t-1}^{(k)}, \mathbf{N}_{t-1}^{(k)}, \Sigma_{t-1}^{(k)}], \hat{\mathbb{E}}(\epsilon_{t-1}^{(k)}), c\hat{\sigma}v(\epsilon_{t-1}^{(k)}) \right\}$,

$$\mathbf{Z}_{t-1:t}^{(k-1)} := \begin{cases} [\mathbf{A}_{t-1:t}^{(k-1)}, \mathbf{N}_{t-1:t}^{(k-1)}, \Sigma_{t-1:t}^{(k-1)}], & \text{if } k > 1, \\ \mathbf{Y}_{t-1:t}, & \text{elif } k = 1. \end{cases}$$

Output: $\text{State}(k, t+1) := \left\{ \mathbf{Z}_t^{(k)} := [\mathbf{A}_t^{(k)}, \mathbf{N}_t^{(k)}, \Sigma_t^{(k)}], \hat{\mathbb{E}}(\epsilon_t^{(k)}), c\hat{\sigma}v(\epsilon_t^{(k)}) \right\}$

- 1: $\Delta \mathbf{Z}_t^{(k-1)} \leftarrow \mathbf{Z}_t^{(k-1)} - \mathbf{Z}_{t-1}^{(k-1)}$
 - 2: $\epsilon_t^{(k)} \leftarrow \Delta \mathbf{Z}_t^{(k-1)} - (\mathbf{A}_{t-1}^{(k)} \mathbf{Z}_{t-1}^{(k-1)} + \mathbf{N}_{t-1}^{(k)})$
 - 3: $\mathbf{A}_t^{(k)} \leftarrow \mathbf{A}_{t-1}^{(k)} + 2\eta_A^{(k)} \epsilon_t^{(k)} \mathbf{Z}_t^{(k-1)T}$
 - 4: $\mathbf{N}_t^{(k)} \leftarrow \mathbf{N}_{t-1}^{(k)} + 2\eta_N^{(k)} \epsilon_t^{(k)}$
 - 5: $\hat{\mathbb{E}}(\epsilon_{t-1}^{(k)}) := \text{EMA}_\varphi(\epsilon_{t-1}^{(k)})$
 - 6: $\hat{\mathbb{E}}(\epsilon_t^{(k)}) \leftarrow \varphi^{(k)} \cdot \epsilon_t^{(k)} + (1 - \varphi^{(k)}) \cdot \hat{\mathbb{E}}(\epsilon_{t-1}^{(k)})$
 - 7: $c\hat{\sigma}v(\epsilon_{t-1}^{(k)}) := \text{EMA}_\rho(c\hat{\sigma}v(\epsilon_{t-1}^{(k)}))$
 - 8: $c\hat{\sigma}v(\epsilon_t^{(k)}) \leftarrow \rho^{(k)} \cdot (\epsilon_t^{(k)} - \hat{\mathbb{E}}(\epsilon_t^{(k)})) \cdot (\epsilon_t^{(k)} - \hat{\mathbb{E}}(\epsilon_t^{(k)}))^T + (1 - \rho^{(k)}) \cdot c\hat{\sigma}v(\epsilon_{t-1}^{(k)})$
 - 9: $\Sigma_t^{(k)} \leftarrow \Sigma_{t-1}^{(k)} - 4\eta_\Sigma^{(k)} \cdot (\Sigma_{t-1}^{(k)} \Sigma_{t-1}^{(k)T} - c\hat{\sigma}v(\epsilon_t^{(k)})) \cdot \Sigma_{t-1}^{(k)}$
 - 10: $\mathbf{Z}_t^{(k)} \leftarrow [\mathbf{A}_t^{(k)}, \mathbf{N}_t^{(k)}, \Sigma_t^{(k)}]$
-

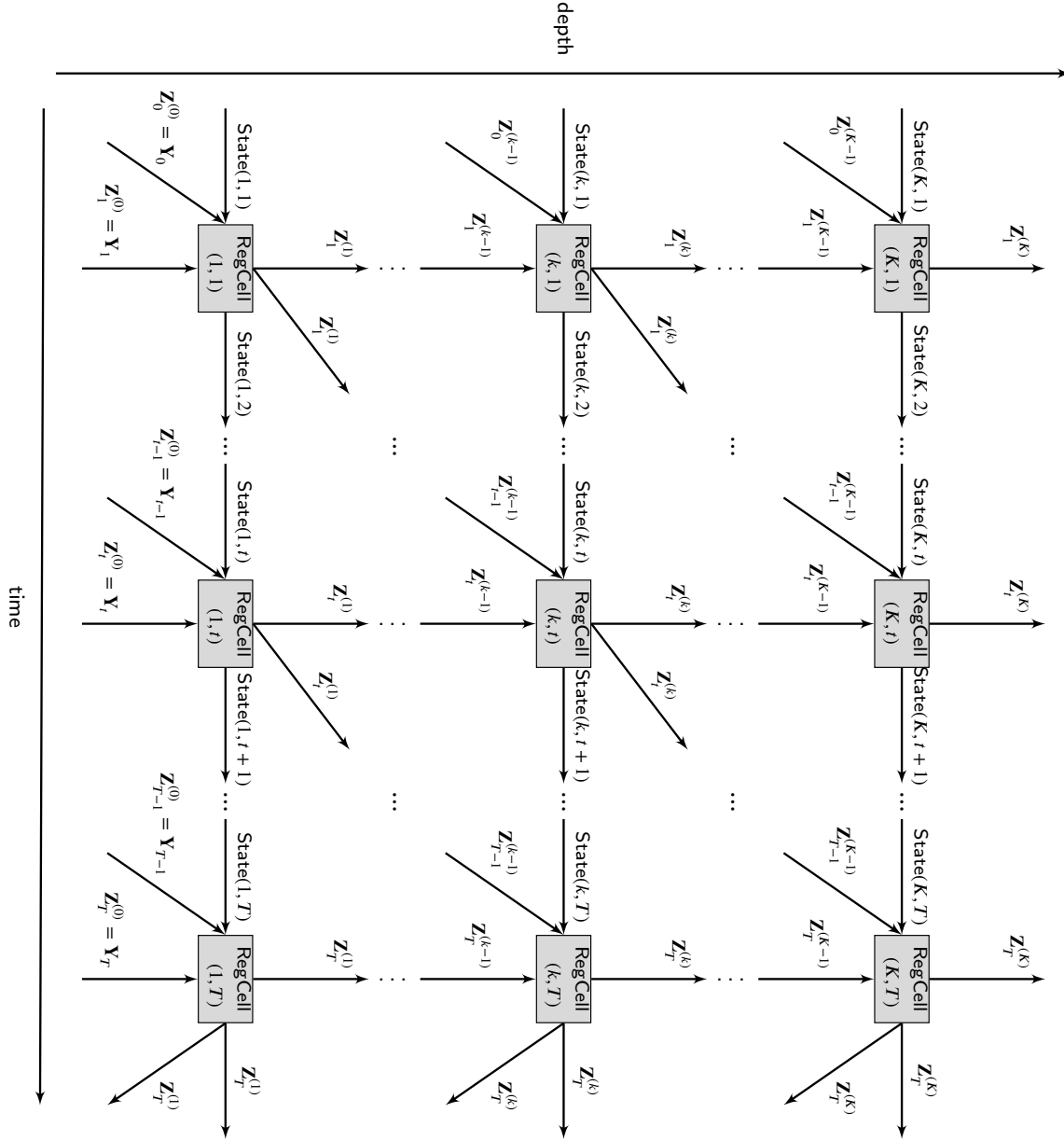


Figure 4: The structure of RegNet. $\text{State}(k, t) = \mathbf{Z}_{t-1}^{(k)}, \hat{\mathbb{E}}(\epsilon_{t-1}^{(k)}), c\hat{v}(\epsilon_{t-1}^{(k)})$.

into a 3-dimensional vector of coefficients denoted $\mathbf{Z}_t^{(1)} = [\mathbf{A}_t^{(1)}, \mathbf{N}_t^{(1)}, \boldsymbol{\Sigma}_t^{(1)}]$. The output of layer $k = 1$ at t is now a multivariate time series $\mathbf{Z}_t^{(1)}$ of dimension 3.

The dynamics of $\mathbf{Z}_t^{(1)}$ for the time steps $t = 1, \dots, T$ can be analyzed in the same way as Y_t . Thus, we treat $\mathbf{Z}_t^{(1)}$ as a multivariate OU process and regress its coefficients in an online fashion using a second layer composed of T RegCells. Several layers can then be stacked on top of each other as shown in Fig. 4, allowing each layer k to regress the vector of parameters $\Delta \mathbf{Z}_t^{(k)} = [\mathbf{A}_t^{(k)}, \mathbf{N}_t^{(k)}, \boldsymbol{\Sigma}_t^{(k)}]$ of an OU process $\Delta \mathbf{Z}_t^{(k-1)} = \mathbf{A}_{t-1}^{(k)} \mathbf{Z}_{t-1}^{(k-1)} + \mathbf{N}_{t-1}^{(k)} + \boldsymbol{\Sigma}_{t-1}^{(k)} \Delta \mathbf{W}_t^{(k)}$, modeling the dynamics $\Delta \mathbf{Z}_t^{(k-1)} = \mathbf{Z}_t^{(k-1)} - \mathbf{Z}_{t-1}^{(k-1)}$ of the time series output $\mathbf{Z}_t^{(k-1)}$ by the previous layer $k - 1$. Since we denote generally $\mathbf{Z}_t^{(k)}$ the output of a RegCell at time step t in layer k for any integer $k \geq 0$, we adopt the convention $\mathbf{Z}_t^{(0)} = Y_t$. Observe that each layer k increases the dimensionality of the vector of coefficients from d_{k-1} to $d_k = 2d_{k-1}^2 + d_{k-1}$. The sequence of dimensions $(d_k) = 1, 3, 21, 903, 1631721, \dots$ quickly diverges towards infinity, thus limiting in practice RegNet to a maximum of 2 or 3 layers.

The motivation for using multiple layers in the RegNet is that it allows us to extract more information from the time

series Y_t . Since this information is passed on to the PredNet, the resulting RegPred Net can potentially yield better forecasts in the long run. This can be best understood perhaps by analogy with a function f defined on a time interval $[0, T]$ that we would like to extrapolate to $[T, +\infty]$. If the function is 2 times continuously differentiable on $[0, T]$, we could extrapolate it for $t > T$ with the Taylor series expansion of order 2. The higher the order of the Taylor series, the more accurate the extrapolation becomes. Similarly, the higher the number of layers used the RegPred Net, the better the forecasts may get, as primarily, each layer k is modeling the k -th discrete derivative of Y with respect to t .

4.2. Prediction Cell (PredCell) in Prediction Network (PredNet)

For prediction, we need another type of cell that uses the information extracted by the RegCell, which we call PredCell. Assume K is the number of total layers of the RegNet, T is the last time step of an input time series \mathbf{Y} , and at time T RegNet outputs $\mathbf{Z}_T^{1:K}$. PredNet starts making predictions of the multivariate process $\mathbf{Z}_t^{(k)}$ at the last layer $k = K$ and ends making predictions for the process at the first layer $k = 0$ where the process $\mathbf{Z}_t^{(0)} = \mathbf{Y}_t$. In the last layer $k = K$, since the process to forecast is not stochastically modeled, we assume that the outputs of the PredCell $\mathbf{Z}_T^{(K)}$ is constant for any time step $T + i, i > 0$:

$$\mathbf{Z}_{T+i}^{(K)} \leftarrow \mathbf{Z}_T^{(K)} \quad (26)$$

The PredCells in all other layers where $k < K$ follow the update rule:

$$\mathbf{Z}_t^{(k)} \leftarrow \mathbf{Z}_{t-1}^{(k)} + \mathbf{A}_{t-1}^{(k+1)} \mathbf{Z}_{t-1}^{(k)} + \mathbf{N}_{t-1}^{(k+1)} + \boldsymbol{\Sigma}_{t-1}^{(k+1)} \Delta \mathbf{W}_t^{(k)} \quad (27)$$

which simply randomly generates a new value for the process using its previous value and the equation for the increment of an OU process.

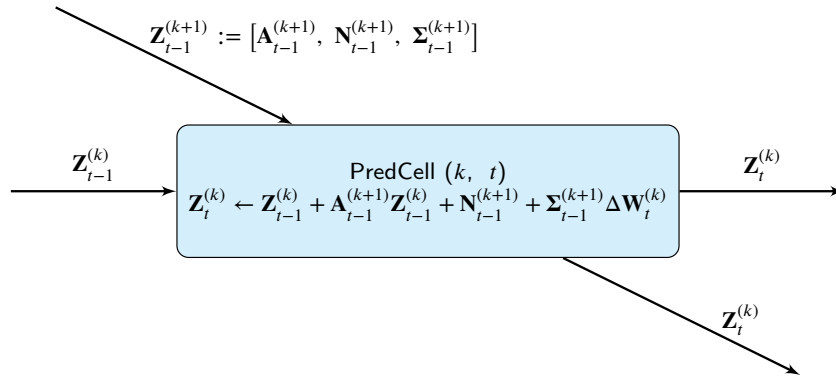
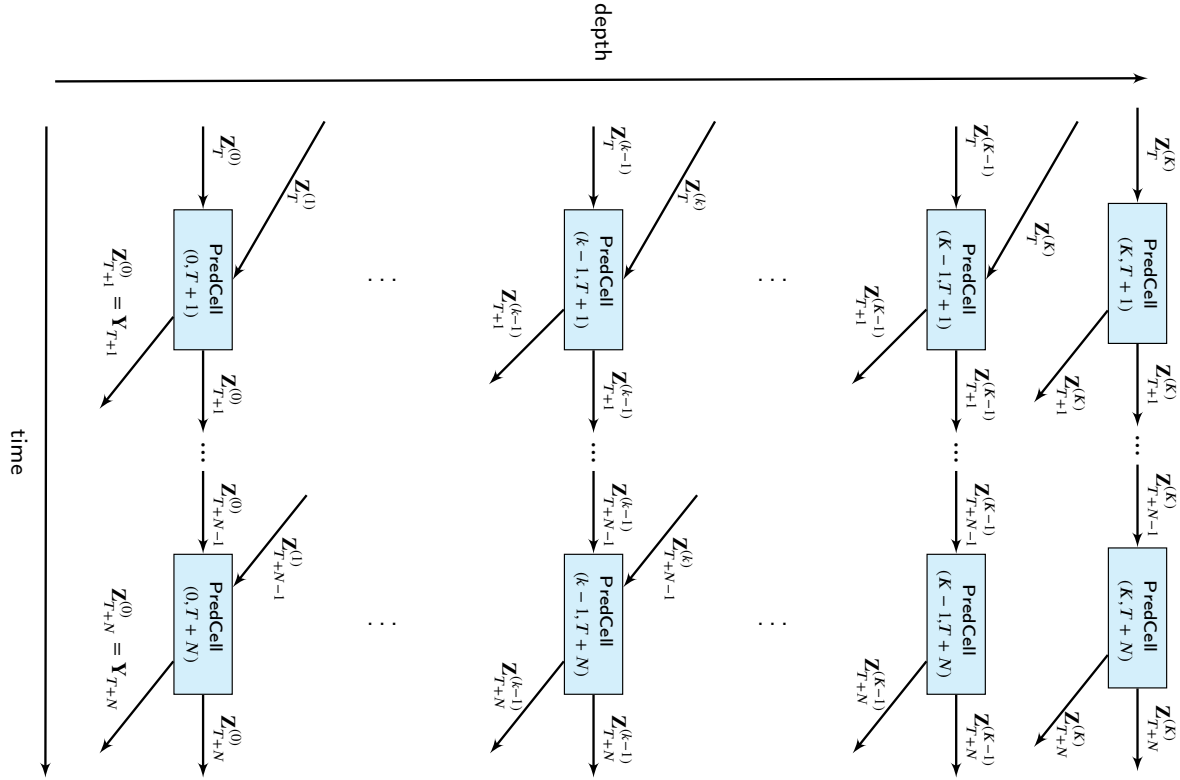


Figure 5: The PredCell (k, t) .

Fig. 5 illustrates the function of a PredCell in layer k at time t and Algorithm 2 shows how to implement such a cell. In the calculation of the output $\mathbf{Z}_t^{(k)}$ of PredCell (k, t) , the term $\Delta \mathbf{W}_t^{(k)}$ is a randomly generated vector of the same dimension as $\mathbf{Z}_t^{(k)}$, which we denote by d_k . Each component of this vector follows an independent standard normal distribution. It is called a *random factor* of \mathbf{Z}_t and can be seen as the source of randomness of the process. So remember that $\Delta \mathbf{W}_t^{(k)}$ has dimension d_k and its components are i.i.d. with distribution $\mathcal{N}(0, 1)$. Thus, the PredCells generate random outputs. In that sense, the PredNet constitutes a generative network. This is why the PredNet can only be used for simulation of trajectories and not directly for prediction. The diagonal arrow above the cell represents the input $\mathbf{Z}_{t-1}^{(k+1)}$, which is from layer $k + 1$ and time $t - 1$. $\mathbf{Z}_{t-1}^{(k+1)}$ contains the parameters $\mathbf{A}_{t-1}^{(k+1)}, \mathbf{N}_{t-1}^{(k+1)}, \boldsymbol{\Sigma}_{t-1}^{(k+1)}$ that are needed in Eq. (27). The left horizontal arrow indicates the input $\mathbf{Z}_{t-1}^{(k)}$ from layer k , time $t - 1$. Notice here at the initial prediction step $T + 1$, $\mathbf{Z}_{t-1}^{(k)}$ is the outputs of RegCell in layer k at step T . The diagonal arrow below the cell indicates the output $\mathbf{Z}_t^{(k)}$. This will be divided into $\mathbf{A}_t^{(k)}, \mathbf{N}_t^{(k)}, \boldsymbol{\Sigma}_t^{(k)}$, and be used for the PredCell of layer $k - 1$, time $t + 1$. $\mathbf{Z}_t^{(k)}$ is also transferred to the next time step for PredCell $(k, t + 1)$ at the right horizontal arrow. The multi-layer PredNet is illustrated in Fig. 6.

Algorithm 2 PredCell in layer k at time t
Input: $\mathbf{Z}_{t-1}^{(k+1)}$, $\mathbf{Z}_{t-1}^{(k)}$, $\Delta \mathbf{W}_t^{(k)}$
Output: $\mathbf{Z}_t^{(k)}$

- 1: **if** layer k is the last layer **then**
- 2: $\mathbf{Z}_t^{(k)} \leftarrow \mathbf{Z}_{t-1}^{(k)}$
- 3: **else**
- 4: $\mathbf{Z}_t^{(k)} \leftarrow \mathbf{Z}_{t-1}^{(k)} + \mathbf{A}_{t-1}^{(k+1)} \mathbf{Z}_{t-1}^{(k)} + \mathbf{N}_{t-1}^{(k+1)} + \boldsymbol{\Sigma}_{t-1}^{(k+1)} \Delta \mathbf{W}_t^{(k)}$
- 5: **end if**


Figure 6: The structure of PredNet.

4.3. Regression-Prediction Network (RegPred Net)

We combine the RegNet in Fig. 4 and PredNet in Fig. 6 together to get the overall RegPred Network, as in Fig. 7. The regression part of the network starts from RegCell (1, 1) at the bottom left and ends at RegCell (K , T).

Algorithm 3 describes how a K layer(s) RegPred Net in Fig. 7 predicts the future N steps given an input time series. The inputs of Algorithm 3 include: $\mathbf{Y}_{1:T}$ as input time series, $\mathbf{H}^{(1:K)}$ as the learning rates of layer 1 to K , $\mathbf{Z}_0^{(1:K)}$ are the initial input vector of layer 1 to K . We initialize State(1 : K , 0) as the concatenation of $\mathbf{H}^{(1:K)}$, $\mathbf{Z}_0^{(1:K)}$, $\hat{\mathbf{E}}(\epsilon_0^{(1:K)})$, and $c\hat{\nu}(\epsilon_0^{(1:K)})$. Among them $\hat{\mathbf{E}}(\epsilon_0^{(1:K)})$, $c\hat{\nu}(\epsilon_0^{(1:K)})$ are set as vector and matrix of $\mathbf{0}$. The errors $\epsilon_0^{(1:K)}$ are also initialized as $\mathbf{0}$. $\mathbf{Z}_{0:T}^{(0)}$ are always equal to $\mathbf{Y}_{0:T}$. At the start of Algorithm 3, we generate $K \times N$ normally distributed random noises $\Delta \mathbf{W}_{T+1:T+N}^{(0:K-1)}$, then take \mathbf{Y} as the input series of RegNet and run RegCell described in Algorithm 1 from layer 1 to layer K and times 1 to T . After the regression, we get State(1 : K , $T + 1$). They are the inputs for the prediction. The predictions are calculated in the reverse order of the regression: run Algorithm 2 downwards from layer K to 0, time $T + 1$ to $T + N$. At layer 0, we obtain a simulated trajectory $\mathbf{Z}_{T+1:T+N}^{(0)} = \mathbf{Y}_{T+1:T+N}$ as output. Like any Recurrent Neural Network, the RegPred Net can handle input series with different input lengths T and make predictions of arbitrary length N .

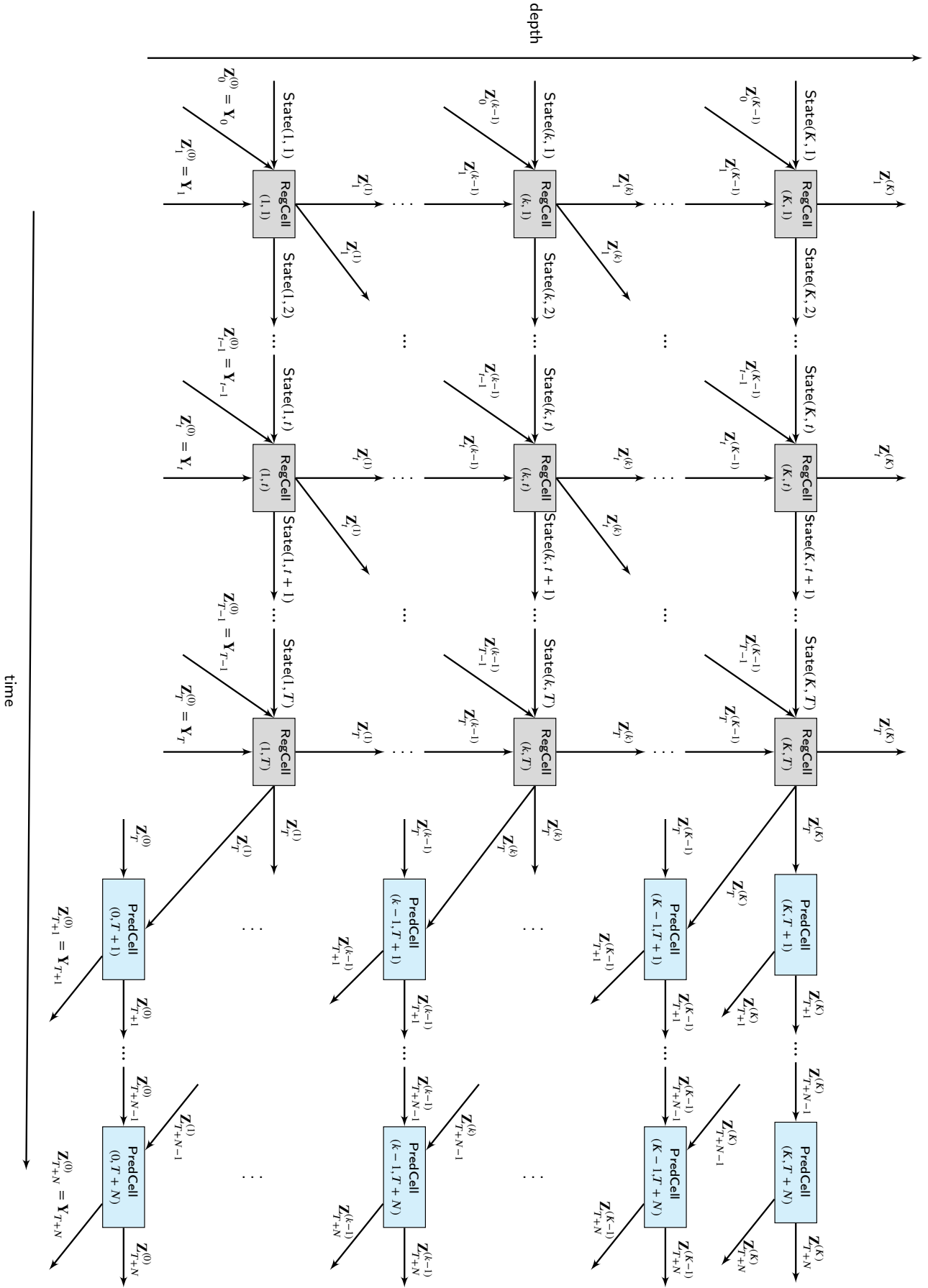


Figure 7: RegPred Net. State $(k, t) = Z_{t-1}^{(k)}, \hat{\mathbb{E}}(\epsilon_{t-1}^{(k)}), c\hat{v}(\epsilon_{t-1}^{(k)})$

Algorithm 3 K layer(s) RegPred Net (prediction)

Input: Input series \mathbf{Y} with length T , number of prediction steps N , $\mathbf{H}^{(1:K)}$, $\mathbf{Z}_0^{(1:K)}$, i.i.d. noises $\Delta\mathbf{W}_{T+1:T+N}^{(0:K-1)}$ (size $[K, N]$) following the standard normal distribution

Output: Simulated trajectory $\mathbf{Z}_{T+1:T+N}^{(0)} = [Y_{T+1}, \dots, Y_{T+N}]$

- 1: **Initialization:** State(1 : K , 0) = $[\mathbf{Z}_0^{(1:K)}, \hat{\mathbb{E}}(\epsilon_0^{(1:K)}) = \mathbf{0}, c\hat{\sigma}(\epsilon_0^{(1:K)}) = \mathbf{0}], \epsilon_0^{(1:K)} = \mathbf{0}, \mathbf{Z}_{0:T}^{(0)} = \mathbf{Y}_{0:T}$
 - 2: **for** $t = 1, \dots, T$ **do**
 - 3: **for** $k = 1, \dots, K$ **do**
 - 4: Run Algorithm 1 with corresponding inputs to get State(1 : k, t)
 - 5: **end for**
 - 6: **end for**
 - 7: **for** $t = T + 1, \dots, T + N$ **do**
 - 8: **for** $k = K, \dots, 0$ **do**
 - 9: Run Algorithm 2 with corresponding inputs to get $\mathbf{Z}_t^{(k)}$
 - 10: **end for**
 - 11: **end for**
-

Algorithm 4 Calculating the loss of mean $L^{\mathbb{E}}$ and the loss of variance $L^{\mathbb{V}}$ using Monte Carlo simulation

Input: The number n_W of simulated trajectories, input time series $\mathbf{Y}_1, \dots, \mathbf{Y}_T, \dots, \mathbf{Y}_{T+N}$, $\mathbf{H}^{(1:K)}$, $\mathbf{Z}_0^{(1:K)}$

Output: $L^{\mathbb{E}}, L^{\mathbb{V}}, \mathbb{E}_{T+1:T+N}, \mathbb{V}_{T+1:T+N}$

- 1: Compute $\mathbf{Z}_T^{(1:K)}$ by online regression with Algorithm 1
- 2: Generate n_W random noises $\Delta\mathbf{W}_{T+1:T+N}^{(0:K-1)}$
- 3: **for** $i = 1, \dots, n_W$ **do**
- 4: Use Algorithm 2 to generate the i -th trajectory $\mathbf{Z}_{T+1,i}^{(0)}, \dots, \mathbf{Z}_{T+N,i}^{(0)}$
- 5: **end for**
- 6: Calculate mean \mathbb{E}_t and variance \mathbb{V}_t at every time step using the n_W trajectories
- 7: Calculate the losses for the mean and the variance:

$$L^{\mathbb{E}} = \sqrt{\frac{1}{N} \sum_{t=T+1}^{T+N} (Y_t - \mathbb{E}_t)^2}, \quad L^{\mathbb{V}} = \sqrt{\frac{1}{N} \sum_{t=T+1}^{T+N} [(Y_t - \mathbb{E}_t)^2 - \mathbb{V}_t]^2}$$

Algorithm 4 explains how to calculate losses for the mean and variance of the trajectories using Monte Carlo simulation. Assume we simulate n_W trajectories, the loss of mean $L^{\mathbb{E}}$ between the mean of samples \mathbb{E} and the target series $\mathbf{Y}_{T+1:T+N}$, and the loss of variance $L^{\mathbb{V}}$ between the variance of samples \mathbb{V} and the target series $\mathbf{Y}_{T+1:T+N}$ can be calculated as:

$$L^{\mathbb{E}} = \sqrt{\frac{1}{N} \sum_{t=T+1}^{T+N} (Y_t - \mathbb{E}_t)^2}$$

$$L^{\mathbb{V}} = \sqrt{\frac{1}{N} \sum_{t=T+1}^{T+N} [(Y_t - \mathbb{E}_t)^2 - \mathbb{V}_t]^2}$$
(28)

where \mathbf{Y}_t is the value of series \mathbf{Y} at time t . The loss of mean $L^{\mathbb{E}}$ is evaluated by taking the average of the difference between values of the target series and their corresponding statistical mean of predicted samples. Similarly, the loss of variance $L^{\mathbb{V}}$ is computed by taking the mean of the difference between calculated variance $(Y_t - \mathbb{E}_t)^2$ and the statistical variance of samples \mathbb{V}_t .

We improve the basic definition of the loss in Algorithm 4 to a more statistically robust and meaningful loss by computing an average loss over several prediction horizons, as in Algorithm 5. In Algorithm 5, $\mathbf{Y}_{1:t}$ are the first

t values in series \mathbf{Y} , where $t = 2, \dots, T$. Use each sub-series $\mathbf{Y}_{1:t+N}$ as input, RegPred Net can predict the mean $\mathbb{E}_{t+1:t+N}$ and the variance $\mathbb{V}_{t+1:t+N}$ of the next N time steps, the label for $\mathbb{E}_{t+1:t+N}$ is actually the steps $t + 1 : t + N$ of series $\mathbf{Y}_{1:t+N}$. The average loss of mean $L_{avg}^{\mathbb{E}}$ and the average loss of variance $L_{avg}^{\mathbb{V}}$ are then calculated by averaging $T - 1$ losses computed by running Algorithm 4 with $Y_{1:2+N}, \dots, Y_{1:T+N}$ as inputs.

Algorithm 5 Calculating the average loss of mean $L_{avg}^{\mathbb{E}}$ and the average loss of variance $L_{avg}^{\mathbb{V}}$

Input: number n_W of trajectories per Monte Carlo simulation, input time series $\mathbf{Y}_1, \dots, \mathbf{Y}_T, \dots, \mathbf{Y}_{T+N}, \mathbf{H}^{(1:K)}, \mathbf{Z}_0^{(1:K)}$

Output: $L_{avg}^{\mathbb{E}}, L_{avg}^{\mathbb{V}}$

1: **for** $t = 2, \dots, T$ **do**

2: Run Algorithm 4 with $\mathbf{Y}_{1:t+N}$ as input series and get loss of mean $L_{t+1:t+N}^{\mathbb{E}}$ and loss of variance $L_{t+1:t+N}^{\mathbb{V}}$ for a regression window $[1, t]$, and prediction window $[t, t + N]$ estimated over n_W trajectories

3: **end for**

4: Calculate the average loss of mean and the average loss of variance:

$$L_{avg}^{\mathbb{E}} = \frac{1}{T-1} \sum_{t=2}^T L_{t+1:t+N}^{\mathbb{E}}, \quad L_{avg}^{\mathbb{V}} = \frac{1}{T-1} \sum_{t=2}^T L_{t+1:t+N}^{\mathbb{V}}$$

5. Optimization of hyperparameters in RegPred Net

As explained in Sec. 4, RegPred Net generates trajectories and ultimately forecasts which (besides randomly generated numbers) only depend on the initial value of the parameters $\mathbf{A}_0^{(1:K)}, \mathbf{N}_0^{(1:K)}, \mathbf{\Sigma}_0^{(1:K)}$ and the learning rates $\mathbf{H}^{(1:K)} = [\eta_A^{(1:K)}, \eta_N^{(1:K)}, \eta_{\Sigma}^{(1:K)}, \varphi^{(1:K)}, \rho^{(1:K)}]$. In comparison to cells in conventional RNNs, the RegCell and PredCell have no weight or parameter to learn (via backpropagation through time). Despite this apparent simplicity, we observe in the case of FX rate time series, that the RegPred Net's regressed parameters and as a matter of consequence simulations and forecasts are all very sensitive to the values of the hyperparameters. The selection of the RegPred Net's hyperparameters is a thorny minimization problem of a loss function with many local minima for which global optimization is required. Note that the loss function $L_{avg} = L_{avg}^{\mathbb{E}} + L_{avg}^{\mathbb{V}}$ to minimize is a) noisy, as it is calculated by Monte Carlo simulation, and b) costly to compute. Consequently, the optimization method chosen must be able to handle noise in the objective function f , be parsimonious in the number of evaluations of f , and ideally shall not require the evaluation of the derivative of f . All of these reasons make Bayesian optimization an adequate method to find optimal values of the hyperparameters.

5.1. Bayesian optimization

Bayesian optimization is a heuristic algorithm to solve a maximization problem:

$$\mathbf{x}_* = \underset{\mathbf{x} \in \mathbb{R}^d}{\operatorname{argmax}} f(\mathbf{x}) \quad (29)$$

where f is an objective function taking its values in \mathbb{R} . Bayesian optimization is a sequential decision strategy for the efficient global optimization of black-box functions which does not require estimation of the function's derivative. Bayesian optimization used in this article to maximize the RegPred Net's negative loss $f(x) = -L_{avg} = -(L_{avg}^{\mathbb{E}} + L_{avg}^{\mathbb{V}})$ where \mathbf{x} represents the network's hyperparameters $\mathbf{x} = [\mathbf{A}_0^{(1:K)}, \mathbf{N}_0^{(1:K)}, \mathbf{\Sigma}_0^{(1:K)}, \mathbf{H}^{(1:K)}]$.

Bayesian optimization sequentially improves its estimates \mathbf{x}_n of the maximizer \mathbf{x}_* of f . At each step n , the value of $f(\mathbf{x}_n)$ is calculated and collected in the set of observations $\mathcal{D}_{1:n} = \left\{ (\mathbf{x}_i, f(\mathbf{x}_i)) \mid i = 1, \dots, n \right\}$. This data set is used to model the posterior distribution $p(f(\mathbf{x}') | \mathcal{D}_{1:n}, \mathbf{x}')$ of the unknown and random value $f(\mathbf{x})$ for any arbitrary \mathbf{x}' . In Gaussian Process Regression, it is assumed that the joint distribution of $f(\mathbf{X})$ and $f(\mathbf{x}')$ is multivariate Gaussian with mean function zero and a covariance function or kernel $k : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$:

$$\begin{bmatrix} f(\mathbf{X}) \\ f(\mathbf{x}') \end{bmatrix} \sim \mathcal{N} \left(0, \begin{bmatrix} \mathbf{K}(\mathbf{X}, \mathbf{X}) & \mathbf{K}(\mathbf{X}, \mathbf{x}') \\ \mathbf{K}(\mathbf{x}', \mathbf{X}) & k(\mathbf{x}', \mathbf{x}') \end{bmatrix} \right) \quad (30)$$

where $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ and $\mathbf{K}(\mathbf{X}, \mathbf{X})$ represents the $n \times n$ covariance matrix:

$$\mathbf{K}(\mathbf{X}, \mathbf{X}) = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} \quad (31)$$

and similarly, $\mathbf{K}(\mathbf{X}, \mathbf{x}')$ is the $n \times 1$ covariance matrix computed for all possible combinations between vectors in \mathbf{X} and \mathbf{x}' . $\mathbf{K}(\mathbf{x}', \mathbf{X}) = \mathbf{K}(\mathbf{X}, \mathbf{x}')^T$, and $k(\mathbf{x}', \mathbf{x}') = 1$. A commonly used kernel is the Squared Exponential (SE), which is a function-space expression of Radial Basis Function (RBF) (A.4):

$$\text{cov}(f(\mathbf{x}), f(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2l^2} \|\mathbf{x} - \mathbf{x}'\|^2\right) \quad (32)$$

where l is a parameter that denotes the kernel's width. A small l makes the covariance smaller, and vice versa. Notice that the covariance between outputs $f(\mathbf{x})$ and $f(\mathbf{x}')$ is described as a function of the inputs \mathbf{x} and \mathbf{x}' . It implies that the covariance between variables tends to 1 if their inputs are similar and tends to 0 if their inputs are different. Another commonly used covariance function is the Matern class (Rasmussen and Williams, 2005, Sec. 4.2.1), defined as:

$$k_{\text{Matern}}(\|\mathbf{x} - \mathbf{x}'\|) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|\mathbf{x} - \mathbf{x}'\|}{l}\right)^\nu K_\nu\left(\frac{\sqrt{2\nu}\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \quad (33)$$

where ν is a positive parameter that controls the smoothness of the function and l is a positive scale parameter. $\Gamma(\cdot)$ is a gamma function (A.5) and K_ν is the modified Bessel function (Abramowitz, 1974, Sec. 9.6). When $\nu \rightarrow \infty$, Eq. (33) is exactly the SE covariance function described in Eq. (32). The most commonly used values for the Matern class in Machine Learning are $\nu = 3/2$ and $\nu = 5/2$:

$$\begin{aligned} k_{\nu=3/2}(\|\mathbf{x} - \mathbf{x}'\|) &= \left(1 + \frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \exp\left(-\frac{\sqrt{3}\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \\ k_{\nu=5/2}(\|\mathbf{x} - \mathbf{x}'\|) &= \left(1 + \frac{\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|}{l} + \frac{\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|}{l}\right) \end{aligned} \quad (34)$$

Other covariance functions can be found in (Rasmussen and Williams, 2005, Chap. 4).

Under the Gaussian Process assumption, it can be proven (Rasmussen and Williams, 2005) using Bayes' theorem that the posterior distribution of $f(\mathbf{x}')$ follows a normal distribution with mean μ_n and variance σ_n^2 :

$$\begin{aligned} p(f(\mathbf{x}') | \mathcal{D}_{1:n}, \mathbf{x}') &= \mathcal{N}(\mu_n(\mathbf{x}'), \sigma_n^2(\mathbf{x}')) \\ \mu_n(\mathbf{x}') &= \mathbf{K}(\mathbf{x}', \mathbf{X})^T \mathbf{K}(\mathbf{X}, \mathbf{X})^{-1} f(\mathbf{X}) \\ \sigma_n^2(\mathbf{x}') &= k(\mathbf{x}', \mathbf{x}') - \mathbf{K}(\mathbf{x}', \mathbf{X})^T \mathbf{K}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{K}(\mathbf{X}, \mathbf{x}') \end{aligned} \quad (35)$$

After evaluating the posterior distribution of $f(\mathbf{x}')$, Bayesian optimization requires the use of an acquisition function $utility(\cdot)$ to guide the search of the maximizer \mathbf{x}_* . A high value of the acquisition function implies a potentially high value of the objective function. The maximizer of the acquisition function provides the next estimate \mathbf{x}_{n+1} :

$$\mathbf{x}_{n+1} = \text{argmax}_{\mathbf{x}} \text{utility}(\mathbf{x} | \mathcal{D}_{1:n}) \quad (36)$$

Many acquisition functions have been proposed in the past. One commonly used function is the Expected Improvement (EI) from (Mockus et al., 2014). It defines first an improvement function I :

$$I(\mathbf{x}) = \max\{0, f(\mathbf{x}) - f(\mathbf{x}_*)\} \quad (37)$$

where $f(\mathbf{x}_*)$ denotes the best estimate the objective function so far. The expected improvement $\mathbb{E}(I)$ is defined by $\mathbb{E}(\max\{0, f(\mathbf{x}) - f(\mathbf{x}_*)\} | \mathcal{D}_{1:n})$. $\mathbb{E}(I)$ is then calculated by:

$$\begin{aligned} \mathbb{E}(I) &= \int_{I=0}^{I=\infty} I \frac{1}{\sqrt{2\pi}\sigma(\mathbf{x})} \exp\left(-\frac{(\mu(\mathbf{x}) - f(\mathbf{x}_*) - I)^2}{2\sigma^2(\mathbf{x})}\right) dI \\ &= \sigma(\mathbf{x}) \left[\frac{\mu(\mathbf{x}) - f(\mathbf{x}_*)}{\sigma(\mathbf{x})} \Phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}_*)}{\sigma(\mathbf{x})}\right) + \phi\left(\frac{\mu(\mathbf{x}) - f(\mathbf{x}_*)}{\sigma(\mathbf{x})}\right) \right] \end{aligned} \quad (38)$$

Eq. (38) can be analytically evaluated as:

$$\mathbb{E}(I) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}_*))\Phi(Z) + \sigma(\mathbf{x})\phi(Z), & \text{if } \sigma(\mathbf{x}) > 0 \\ 0, & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \quad (39)$$

$$Z = \frac{\mu(\mathbf{x}) - f(\mathbf{x}_*)}{\sigma(\mathbf{x})}$$

where Φ and ϕ indicate the CDF and PDF of the standard normal distribution, respectively.

A more general acquisition function, which allows controlling the balance between the exploitation and exploration of the optimum of f is:

$$\mathbb{E}(I) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}_*) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z), & \text{if } \sigma(\mathbf{x}) > 0 \\ 0, & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \quad (40)$$

$$Z = \frac{\mu(\mathbf{x}) - f(\mathbf{x}_*) - \xi}{\sigma(\mathbf{x})}$$

where $\xi \geq 0$ is a parameter encouraging exploration in regions where the variance $\sigma(\mathbf{x})$ is large. By maximizing $\mathbb{E}(I)$ with ξ , we find the next \mathbf{x}_{n+1} that can lead to a higher value of f :

$$\mathbf{x}_{n+1} = \begin{cases} \operatorname{argmax}_{\mathbf{x}} (\mu(\mathbf{x}) - f(\mathbf{x}_*) - \xi)\Phi(Z) + \sigma(\mathbf{x})\phi(Z), & \text{if } \sigma(\mathbf{x}) > 0 \\ 0, & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \quad (41)$$

$$Z = \frac{\mu(\mathbf{x}) - f(\mathbf{x}_*) - \xi}{\sigma(\mathbf{x})}$$

Fig. 8 shows the Bayesian optimization of the function $f(x) = e^{-(x-2)^2} + e^{-(x-6)^2/10} + 1/x^2 + 1$ that has a global maximum in $x = 2$. The blue curve indicates the target (true function f), the red squares are the observations $(x_n, f(x_n))$, the dashed line represents the predicted mean of f and the purple area is a 95% confidence interval for the target. The acquisition function in the lower part of the figure suggests a new maximizer x_{28} (red star) at iteration $n = 27$ that is very close to the global optimum.

The Bayesian optimization algorithm transforms the optimization problem described in Eq. (29) to the optimization problem presented in Eq. (41), which is easier to solve. Many methods can be used to optimize Eq. (41), such as Quasi-Newton methods (Hennig and Kiefel, 2013). Quasi-Newton methods like the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) (Nocedal and Wright, 2006) can iteratively solve unbounded optimization problems which the function is non-smooth. As can be understood from its name, Limited-memory BFGS with Bounds (L-BFGS-B) (Byrd et al., 1995) is another commonly used variant of BFGS with limited memory and bounds. Algorithm 6 summarizes the procedure of Bayesian optimization to maximize the objective function f with imposed bounds on \mathbf{x} .

5.2. Layerwise training of RegPred Net with Bayesian optimization

Since the number of dimensions for the variable \mathbf{x} that can be handled by the Bayesian optimization procedure is practically limited to roughly 20 (Frazier, 2018, Sec. 1) and $d(1) = 3$, $d(2) = 21$ but $d(3) = 903$, we are constrained to a maximum of $K = 2$ layers. Also, it is difficult to train simultaneously the hyperparameters of layers $k = 1$ and $k = 2$, as this represents a total of 24 parameters. Thus, we adopt a layerwise training strategy and tune the hyperparameters of the RegPred Net for only one layer at a time. We first consider a single-layer network ($k = 1$) and use Bayesian optimization to find the optimal hyperparameters in that layer. Then, the learned parameters of the first layer are fixed and we add a second layer to the network ($k = 2$) and train only the newly added hyperparameters. We get the optimal parameters for the second layer and stop. The layerwise training procedure is summarized in Algorithm 7.

6. Experimental validation

In this section, we evaluate the performance of RegPred Net and compare it to other time series forecasting models. The data set used for FX rates is described in Sec. 6.1. The models compared to RegPred Net include Deep Learning models (LSTM, Auto-LSTM) as well as traditional time series models (ARMA, ARIMA). We gather and analyze

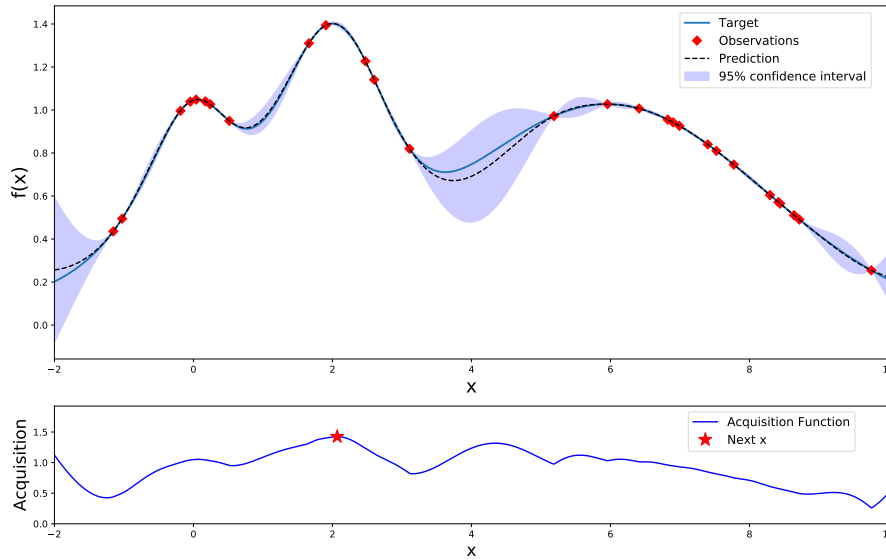


Figure 8: Bayesian optimization of the function $f(x) = e^{-(x-2)^2} + e^{-(x-6)^2/10} + 1/x^2 + 1$ in the search range $[-2, 10]$. The global maximum of f is $x = 2$ and is found after 27 iterations.

all performance results in Sec. 6.5. The RegPred Net, LSTM and Auto-LSTM were implemented in Python’s Deep Learning Framework *Tensorflow* (Abadi et al., 2015) and ARMA and ARIMA were implemented using *statsmodels* (Seabold and Perktold, 2010).

6.1. Data set

The data set used covers three major FX rates: EUR/CNY, EUR/USD, and EUR/GBP. The historical data for the corresponding time series span from 2000.01.04 to 2019.01.29, amounting to 19 years of data and 4975 daily observations per time series. The data used in the experiments are daily closing values of the FX rates provided by Bloomberg.

6.2. Experimental setting

For each FX rate considered, the historical data are used to generate $n = 95$ samples in window steps of 30 days, as illustrated in Fig. 9. Each sample contains a portion of the data over a window of $N_{train,valid} + N_{test}$ days, whereby the first $N_{train,valid}$ days are used for training and validating the model considered in the experiment and the last $N_{test} = 100$ days are used for testing the performance of the model. The first $N_{train,valid}$ days of a sample are in turn sub-divided into $N_{train} = 1830$ days for training and $N_{valid} = 200$ days for validation. Then, for any given model and performance metric considered, the arithmetic mean of the metric is estimated on the basis of calculated values in the n samples and serves as performance statistics for the experimental validation.

6.3. Computing Infrastructure

The computing infrastructure used in this work are one computer with Intel Core(TM) i7-6700K (4.00 GHz) CPU and Nvidia GeForce GTX 970 (6GB) GPU.

6.4. Setting for the models compared

In this section, we explain how we set the experiments for different model comparisons.

6.4.1. RegPred Net

We set the number of layers to $K = 2$ and train RegPred Net layerwise according to Algorithm 7. The number of generated trajectories for each Monte Carlo simulation is $n_W = 50$. For optimizing the acquisition function, L-BFGS-B is used with 5 restart times and the bounds used are those detailed in Tab. 1. We set the number of iterations for

Algorithm 6 Bayesian optimization of $f(\mathbf{x})$ with bounds on \mathbf{x}

Input: objective function f to maximize, N as the number of iterations, bounds for each element in \mathbf{x} for L-BFGS-B, number of different initial guesses of L-BFGS-B N_s , ξ as the exploration parameter of expected improvement

Output: \mathbf{x}_* , $f(\mathbf{x}_*)$

1: **Initialization:** Initialize \mathcal{D} as an empty list, the maximal function value so far as $f(\mathbf{x}_*) = -\infty$, N_s initial input vectors within bounds for L-BFGS-B

2: **for** $i = 1, \dots, N$ **do**

3: Use L-BFGS-B with N_s different initial inputs to optimize $\mathbb{E}(I)$ and assign the optimum to \mathbf{x}_i :

$$\mathbb{E}(I) = \begin{cases} (\mu_{i-1}(\mathbf{x}) - f(\mathbf{x}_*) - \xi)\Phi(Z) + \sigma_{i-1}(\mathbf{x})\phi(Z), & \text{if } \sigma_{i-1}(\mathbf{x}) > 0 \\ 0, & \text{if } \sigma_{i-1}(\mathbf{x}) = 0 \end{cases}$$

$$Z = \frac{\mu_{i-1}(\mathbf{x}) - f(\mathbf{x}_*) - \xi}{\sigma_{i-1}(\mathbf{x})}$$

4: $\mathbf{x}_i = \operatorname{argmax}_{\mathbf{x}} \mathbb{E}(I)$

5: Update the mean $\mu_i(\mathbf{x}_i)$ and covariance $\sigma_i^2(\mathbf{x}_i)$ of the posterior distribution:

$$p(f(\mathbf{x}_i) | \mathcal{D}_{1:i-1}, \mathbf{x}_i) = \mathcal{N}(\mu_i(\mathbf{x}_i), \sigma_i^2(\mathbf{x}_i))$$

$$\mu_i(\mathbf{x}_i) = \mathbf{K}(\mathbf{x}_i, \mathbf{X}_{1:i-1})^T \mathbf{K}(\mathbf{X}_{1:i-1}, \mathbf{X}_{1:i-1})^{-1} f(\mathbf{X}_{1:i-1})$$

$$\sigma_i^2(\mathbf{x}_i) = k(\mathbf{x}_i, \mathbf{x}_i) - \mathbf{K}(\mathbf{x}_i, \mathbf{X}_{1:i-1})^T \mathbf{K}(\mathbf{X}_{1:i-1}, \mathbf{X}_{1:i-1})^{-1} \mathbf{K}(\mathbf{X}_{1:i-1}, \mathbf{x}_i)$$

6: the kernel k used is the Matern class function with $\nu = 5/2$ and $l = 1$:

$$k_{\nu=5/2, l=1}(\mathbf{x}, \mathbf{x}') = \left(1 + \sqrt{5}\|\mathbf{x} - \mathbf{x}'\| + \frac{\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|^2}{32}\right) \exp\left(-\sqrt{5}\|\mathbf{x} - \mathbf{x}'\|\right)$$

$$\mathbf{K}(\mathbf{X}_{1:i-1}, \mathbf{X}_{1:i-1}) = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_{i-1}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_{i-1}, \mathbf{x}_1) & \dots & k(\mathbf{x}_{i-1}, \mathbf{x}_{i-1}) \end{bmatrix}$$

7: Calculate $f(\mathbf{x}_i)$ from objective function f

8: **if** $f(\mathbf{x}_i) > f(\mathbf{x}_*)$ **then**

9: $\mathbf{x}_* = \mathbf{x}_i$

9: $f(\mathbf{x}_*) = f(\mathbf{x}_i)$

10: **end if**

11: Add $(\mathbf{x}_i, f(\mathbf{x}_i))$ to observations data set $\mathcal{D}_{1:i-1}$

12: Update the covariance matrix of the Gaussian process model by calculating:

$$\mathbf{K}(\mathbf{X}_{1:i}, \mathbf{X}_{1:i}) = \begin{bmatrix} \mathbf{K}(\mathbf{X}_{1:i-1}, \mathbf{X}_{1:i-1}) & \mathbf{K}(\mathbf{X}_{1:i-1}, \mathbf{x}_i) \\ \mathbf{K}(\mathbf{x}_i, \mathbf{X}_{1:i-1}) & k(\mathbf{x}_i, \mathbf{x}_i) \end{bmatrix}$$

13: **end for**

14: Return the maximizer \mathbf{x}_* and maximum value $f(\mathbf{x}_*)$

Algorithm 7 Layerwise training of RegPred Net with $K \leq 2$ layers by Bayesian optimization

Input: number of layers K , input time series $\mathbf{Y}_{0:T+N}$

Output: optimal hyperparameters $\mathbf{x}_*^{(1:K)} = \{\mathbf{H}_*^{(1:K)}, \mathbf{Z}_{0*}^{(1:K)}\}$ for the K layers of RegPred Net

1: **for** $k = 1, \dots, K$ **do**

2: Add layer k to RegPred Net with $\mathbf{x}^{(k)}$ as hyperparameters

3: Use Bayesian optimization in Algorithm 6 with $f = -L_{avg} = -(L_{avg}^E + L_{avg}^V)$ as objective function to find the k -th layer's optimal parameters $\mathbf{x}_*^{(k)}$, where the average losses L_{avg}^E and L_{avg}^V are computed by Algorithm 5

4: **end for**

Bayesian optimization to $D = 200$. For the exploration parameter, we use $\xi = 0.01$ for EUR/CNY and EUR/GBP and $\xi = 0.05$ for EUR/USD. Tab. 1 shows the bounds we used in L-BFGS-B algorithm for finding the optimal hyperparameters of RegPred Net for different samples of time series by Bayesian optimization.

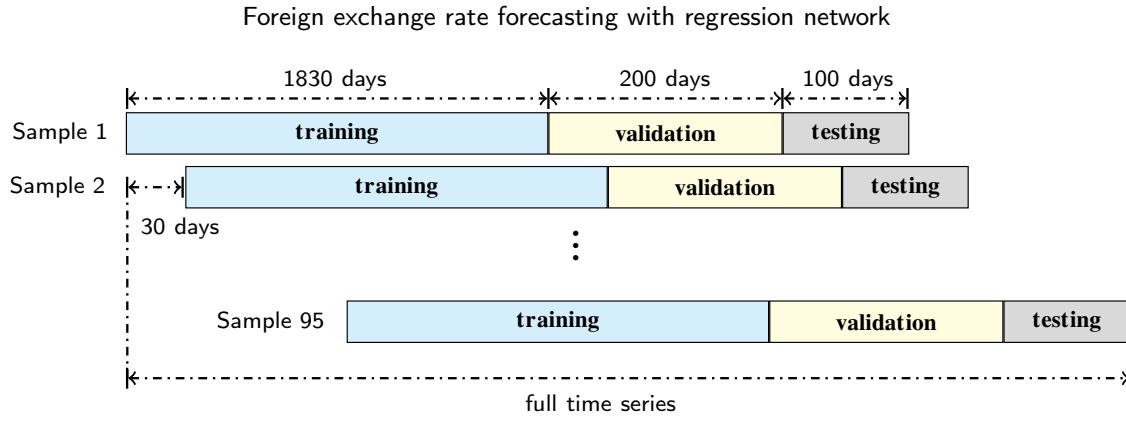


Figure 9: Sampling of the historical data used for computing performance statistics.

Table 1

Bounds in the form of [min, max] used for finding the optimal hyperparameters of RegPred Net by Bayesian optimization.

	A_0, N_0	Σ_0	$\eta_A, \eta_N, \eta_\Sigma$	φ, ρ
Layer 1	[-0.3, 0.3]	[0.001, 0.01]	[0.001, 0.3]	[0.1, 1.0]
Layer 2	[-0.1, 0.1]	[-0.001, 0.001]	[0.001, 0.3]	[0.1, 1.0]

6.4.2. LSTM and Auto-LSTM

Two Deep Learning models are considered: the LSTM and Auto-LSTM. For the LSTM, we test the architecture of (Gensler et al., 2016) in both single-shot (predict in once) and autoregressive way (predict stepwisely), the single-shot way failed on predicting long-term multi-steps time series forecasting task. We performed optimization of the LSTM's hyperparameters by grid search, considering a number of layers ranging from 1 to 5, learning rates of 10^{-2} , 10^{-3} and 10^{-4} , LSTM cells with 32, 64 and 128 units. The LSTM with the best results is illustrated in part (a) of Fig. 10, where each layer is described by layer type and index / layer size (units) / activation function. We connect the output of the last time steps of the LSTM 3 with a dense layer to generate the prediction. In autoregressive mode the network only predicts 1 step at each time and predicts $N_{valid} = 200$ and $N_{test} = 100$ times. We compared the sigmoid and relu activation functions for the last dense and chose the sigmoid for its superior performance. During training, an early stopping technique with patience equals to 50 was used.

To build an Auto-LSTM, we stacked the auto-encoder illustrated in part (b) of Fig. 10 on top of the LSTM shown in part (a). The auto-encoder was pre-trained and used to extract features from the input time series. We then fed the time series into the auto-encoder and used the extracted features from the middle hidden layer (part (b), bottleneck) as inputs for the LSTM.

6.4.3. ARMA and ARIMA

Two of the most important statistical models for time series forecasting are considered: Autoregressive Moving Average (ARMA) (Whittle, 1983) and Autoregressive Integrated Moving Average (ARIMA) (McKenzie, 1984). An ARMA model with orders p and q as hyperparameters is denoted ARMA(p, q) and is of the following form:

$$X_t = c + \epsilon_t + \sum_{i=1}^p \varphi_i X_{t-i} + \sum_{i=1}^q \theta_i \epsilon_{t-i} \quad (42)$$

where X_t is the value of time series at time t , c is a constant, ϵ_t is a noisy term whose values are assumed to be i.i.d. and normally distributed, $\varphi_1, \dots, \varphi_p$ are p parameters for the autoregressive part (AR) and $\theta_1, \dots, \theta_q$ are q parameters for the moving average part (MA) of the time series. ARIMA is a generalized version of ARMA. ARIMA uses an additional hyperparameter d that plays the role of number of differencing steps required to make the time series stationary. The differencing computes the differences between consecutive observations. This helps stabilize the time series and eliminate the trend. Therefore, an ARIMA is represented in the form ARIMA(p, d, q). For the choice of (p, d, q) order for the ARIMA, we referred to (Ho et al., 2002) and also compared several values for p, d , and q .

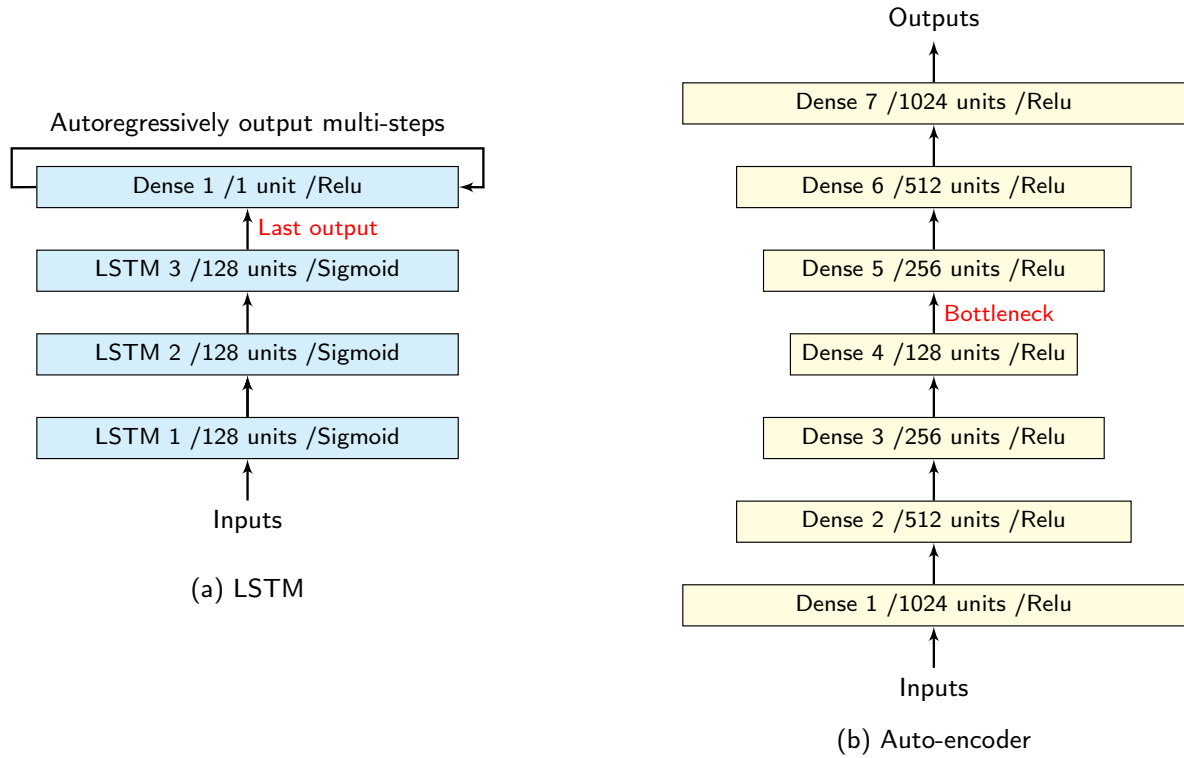


Figure 10: Architecture of the LSTM and Autoencoder part of the Auto-LSTM used in the experimental validation.

6.5. Experimental results

The performance statistics are here-after presented separately for the three FX rates EUR/CNY, EUR/USD and EUR/GBP in Sec. 6.5.1, 6.5.2, and 6.5.3, respectively. The metrics we use to evaluate the forecasting performance of the models are the *Pearson* correlation coefficient (*Pearson's R*), R-squared (R^2), Root mean square error (RMSE) and Mean directional accuracy (MDA). For RegPred Net, it took 5 minutes to train each single sample of each currency type in the data set using the 2 layers infrastructure described in Section 6.3. We use a batch size of the data set size of each currency to train LSTM and Auto-LSTM, it took in average 1000 iterations and 120 minutes to finish the training. For ARMA and ARIMA, we search the parameters p, d, q from 0 to 20 and take the best of them for each sample, which costs around 5 minutes.

6.5.1. Experiment results of EUR/CNY

According to the results reported in Tab. 2, RegPred Net outperforms the other four models (LSTM, Auto-LSTM, ARMA and ARIMA) regardless of the performance metric considered. The RedPred Net's forecasts have a correlation (R) with the true value of the FX rate that is 2.2 times higher than the second best correlated model (LSTM). RegPred Net has an error (RMSE) that is about 30% lower than the error of the second most accurate model (ARIMA). For MDA the gap between the methods is within 10%. For R-squared, all methods except RegPred are negative, which means that the long-term forecasts do not follow the trend of the actual values. Our dataset has several instances where exchange rates fall or rise sharply in the short term due to unexpected events (e.g., financial crises, wars). None of the models involved in the experiments incorporate mechanisms to deal with such situations and therefore can not predict such trends. When the forecasted FX rate has the same tendency as the actual exchange rate, R square is a number in the interval (0, 1]. In contrast, the R square will be a negative number with a large absolute value when the two trends are opposite.

Fig. 11 illustrates the forecasts obtained with RegPred Net, LSTM and ARIMA for 3 of the 95 data samples used. In the examples, we see the RegPred Net's better ability to predict the time-dependent shape and level of the FX rate's estimated trend (plotted in red) and the possibility offered by RegPred Net to interpret and visualize the FX rate's volatility via the 95% confidence zone (colored in gray).

Table 2

Comparison of forecasting performance obtained for EUR/CNY. The results are presented in form of mean \pm std., the best results are indicated in bold font.

	<i>Pearson's R</i>	<i>R-squared</i>	RMSE	MDA
RegPred Net	0.344 \pm 0.443	0.141 \pm 0.862	0.225 \pm 0.141	0.568 \pm 0.046
LSTM	0.156 \pm 0.638	-4.888 \pm 6.264	0.366 \pm 0.234	0.517 \pm 0.080
Auto-LSTM	0.141 \pm 0.691	-5.120 \pm 6.330	0.410 \pm 0.272	0.488 \pm 0.084
ARMA	0.047 \pm 0.667	-2.359 \pm 2.571	0.323 \pm 0.230	0.502 \pm 0.048
ARIMA	0.036 \pm 0.668	-2.279 \pm 2.740	0.318 \pm 0.225	0.460 \pm 0.089

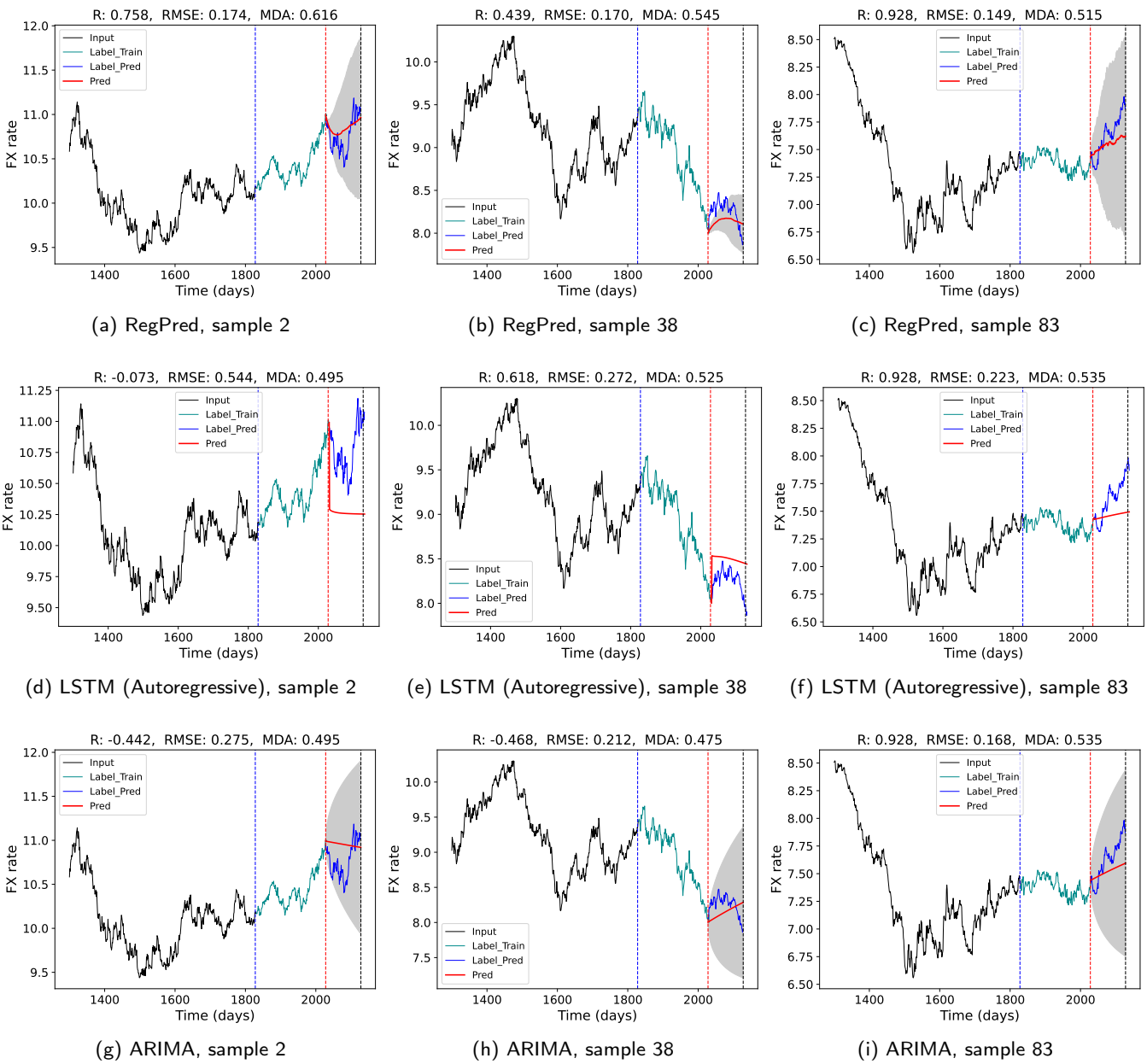


Figure 11: Forecasts obtained for EUR/CNY with RegPred Net, LSTM (Single-shot) and ARIMA in 3 situations.

6.5.2. Experiment results of EUR/USD

In our second experiment, we consider the EUR/USD: an interesting case of erratic time series characterised by the existence of major trend reversals, unstable and often high volatility levels and the frequent occurrence of large (positive or negative) jumps. According to the results reported in Tab. 3, RegPred Net also outperforms the other four models (LSTM, Auto-LSTM, ARMA and ARIMA) regardless of the performance metric considered. The RegPred Net's forecasts have a correlation R with the true value of the FX rate that is 7 times higher than the second best correlated model (LSTM). RegPred Net has an error (RMSE) that is about 25% lower than the error of the second most accurate model (ARIMA). A few representative situations of the predictions of RegPred Net, LSTM and ARIMA can be found in Fig. 12, as previously done for EUR/CNY.

Table 3

Comparison of forecasting performance obtained for EUR/USD. The results are presented in form of mean \pm std., the best results are indicated in bold font.

	<i>Pearson's R</i>	<i>R-squared</i>	RMSE	MDA
RegPred Net	0.342 \pm 0.453	0.108 \pm 0.964	0.038 \pm 0.021	0.544 \pm 0.048
LSTM	0.043 \pm 0.656	-21.983 \pm 30.188	0.093 \pm 0.050	0.501 \pm 0.052
Auto-LSTM	-0.013 \pm 0.062	-27.212 \pm 30.544	0.116 \pm 0.083	0.344 \pm 0.051
ARMA	0.051 \pm 0.684	-2.695 \pm 3.110	0.053 \pm 0.033	0.498 \pm 0.055
ARIMA	-0.020 \pm 0.700	-2.042 \pm 1.744	0.051 \pm 0.035	0.476 \pm 0.095

6.5.3. Experiment results of EUR/GBP

The experimental results for EUR/GBP are shown in Tab. 4. From the table we observe results consistent with those of EUR/CNY and EUR/USD, with a correlation to the true value of the target variable increased by a factor 4 and a reduction of error of 30%. Again, some examples of predictions are shown in Fig. 13.

Table 4

Comparison of forecasting performance obtained for EUR/GBP. The results are presented in form of mean \pm std., the best results are indicated in bold font.

	<i>Pearson's R</i>	<i>R-squared</i>	RMSE	MDA
RegPred Net	0.434 \pm 0.384	0.193 \pm 0.871	0.019 \pm 0.011	0.537 \pm 0.052
LSTM	0.168 \pm 0.680	-9.569 \pm 14.620	0.035 \pm 0.020	0.507 \pm 0.063
Auto-LSTM	-0.001 \pm 0.440	-10.150 \pm 16.233	0.050 \pm 0.034	0.407 \pm 0.071
ARMA	0.049 \pm 0.644	-2.051 \pm 2.575	0.027 \pm 0.017	0.495 \pm 0.054
ARIMA	-0.111 \pm 0.630	-2.106 \pm 2.436	0.025 \pm 0.016	0.447 \pm 0.084

7. Conclusion

In this article, we proposed a novel regression network baptised RegPred Net to forecast daily FX rates in the long term (100 days or more) in an explainable way, by exploiting the regressed and time-dependent parameters of a generalized mean-reverting or *Ornstein-Uhlenbeck* (OU) process. A layerwise procedure based on Bayesian optimization was designed to efficiently train the network in this hard domain of application of Machine Learning.

Despite the strong non-stationarity of FX rates (absence of clear trends and unstable volatility levels), RegPred Net allows to robustly derive via Monte Carlo simulation some accurate and interpretable long term forecasts. In the experiments conducted with 3 of the most traded currencies worldwide (US dollar, Euro and Chinese Yuan) over a history of 19 years, a RegPred Net with 2 layers significantly outperformed other Deep Learning-based models (LSTM, Auto-LSTM) and traditional time series forecasting models (ARMA, ARIMA), reducing the forecasting error (RMSE) by 25-30% and increasing the statistical correlation (R) between forecast and actual value of FX rate by a factor of 2 to 7. The RegPred Net's R -squared coefficient is positive while the others are negative and its MDA is in average 10% higher than the others.

Foreign exchange rate forecasting with regression network

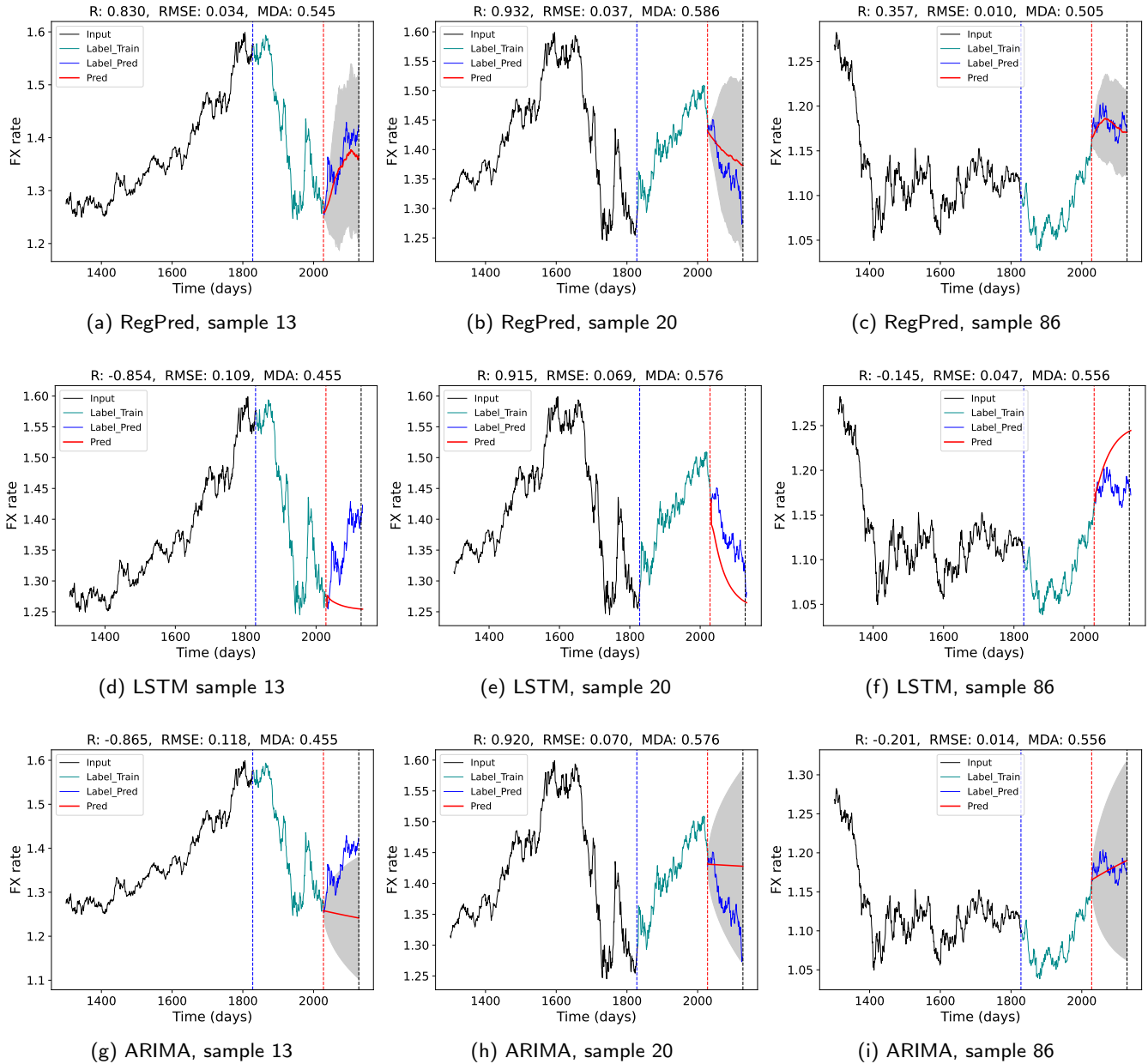


Figure 12: Forecasts obtained for EUR/USD with RegPred Net, LSTM and ARIMA in 3 situations.

This Deep Learning and generative model of FX rates can in principle be used for simulating general stochastic and non-stationary environment variables following Brownian motion or mean-reverting processes, such as is often considered to be the case in Finance with bond or stock prices and FX rates. Such a model can thus be employed for solving a range of risk analysis and sequential decision making problems.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported in this paper.

Acknowledgements

This work was supported by the University of Munich (LMU) and the Daimler AG.

Foreign exchange rate forecasting with regression network

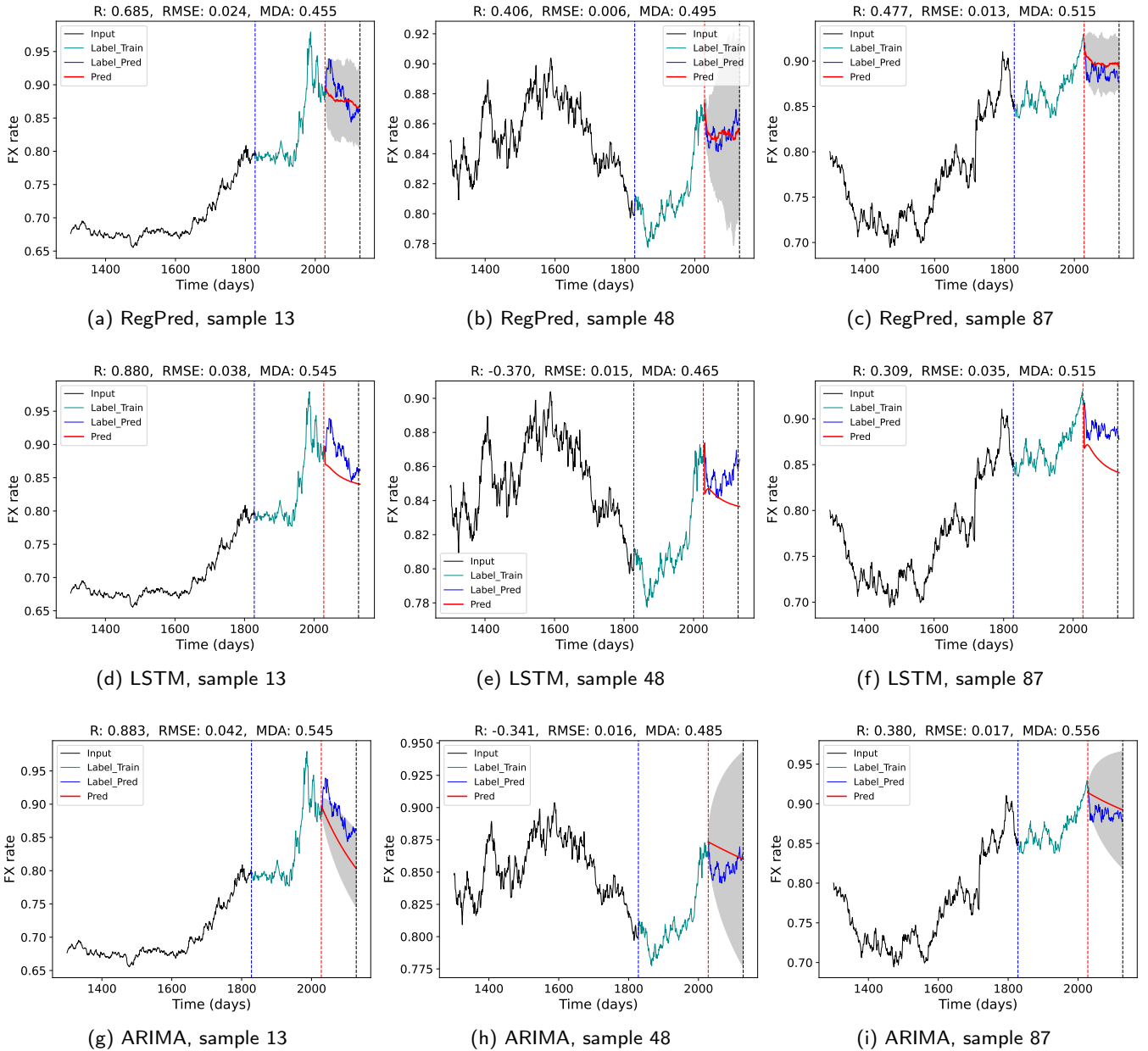


Figure 13: Forecasts obtained for EUR/GBP with RegPred Net, LSTM and ARIMA in 3 situations.

A. Mathematical Background

A.1. Matrix / Vector Differentiation

A Matrix

a Vector (column-vector)

a Scalar

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a} \quad (43)$$

$$\frac{\partial \mathbf{a}^T \mathbf{X} \mathbf{a}}{\partial \mathbf{a}} = (\mathbf{X} + \mathbf{X}^T) \mathbf{a} \quad (44)$$

$$\frac{\partial \mathbf{a}^T \mathbf{X} \mathbf{b}}{\partial \mathbf{X}} = \mathbf{a} \mathbf{b}^T \quad (45)$$

$$\frac{\partial \mathbf{a}^T \mathbf{X}^T \mathbf{b}}{\partial \mathbf{X}} = \mathbf{b} \mathbf{a}^T \quad (46)$$

$$\frac{\partial \mathbf{a}^T \mathbf{X}^T \mathbf{X} \mathbf{b}}{\partial \mathbf{X}} = \mathbf{X}(\mathbf{a} \mathbf{b}^T + \mathbf{b} \mathbf{a}^T) \quad (47)$$

$$\frac{\partial \mathbf{a}^T \mathbf{X} \mathbf{a}}{\partial \mathbf{X}} = \frac{\partial \mathbf{a}^T \mathbf{X}^T \mathbf{a}}{\partial \mathbf{X}} = \mathbf{a} \mathbf{a}^T \quad (48)$$

A.2. Gaussian Distirbution

The Gaussian or normal distribution is given by the following probability density function:

$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (49)$$

where μ is the mean or expectation of the distribution, σ is the standard deviation and σ^2 the variance.

A.3. Multivariate Gaussian Distirbution

The multivariate Gaussian (normal) distribution generalizes the univariate normal distribution to higher dimensions. A vector-valued random variable $x \in \mathbb{R}^n$ is considered as multivariate normal distribution of mean $\mu \in \mathbb{R}^n$ and covariance matrix $\Sigma \in \mathbb{S}_{++}^n$ if its probability density distribution follows

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right) \quad (50)$$

Eq. (50) can be written as $x \sim \mathcal{N}(\mu, \Sigma)$. \mathbb{S}_{++}^n refers to the space of symmetric positive definite $n \times n$ matrices.

A.4. Radial Basis Function

The Radial Basis Function (RBF) kernel, commonly used in kernelized learning algorithms, e.g. SVMs is defined as:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\theta^2} \|\mathbf{x} - \mathbf{x}'\|^2\right) \quad (51)$$

where $\|\mathbf{x} - \mathbf{x}'\|^2$ is the squared Euclidean distance between two feature vectors \mathbf{x} and \mathbf{x}' . θ is a free parameter which indicates the width of the kernel. Since the output range of RBF is in $[0, 1]$, which is inversely proportional to the distance between vectors, RBF is often used as a similarity measure.

A.5. Gamma function

The Gamma function is the generalization of the factorial function to complex numbers and is defined as:

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx \quad (52)$$

where z is a complex number with positive real part ($\text{Re}(z) > 0$). The function has the follow property:

$$\Gamma(z + 1) = z\Gamma(z) \quad (53)$$

B. Mathematical Derivations

B.1. Simplification of the Covariance Matrix

Each element $cov(\epsilon_{i,t}, \epsilon_{j,t})$ of the $d \times d$ covariance matrix $\mathbf{K}_{\epsilon\epsilon,t}$ is equal to:

$$\begin{aligned}
 cov(\epsilon_{i,t}, \epsilon_{j,t}) &= \mathbb{E} \left[(\epsilon_{i,t} - \mathbb{E}(\epsilon_{i,t})) (\epsilon_{j,t} - \mathbb{E}(\epsilon_{j,t})) \right] \\
 &= \mathbb{E}(\epsilon_{i,t} \cdot \epsilon_{j,t}) \quad \text{since } \mathbb{E}(\epsilon_{i,t}) = 0 \text{ (Eq. (3))} \\
 &= \mathbb{E} \left(\sum_{k=1}^d \sum_{k'=1}^d \sigma_{i,k} \cdot \sigma_{j,k'} \cdot \Delta W_{k,t} \cdot \Delta W_{k',t} \right) \\
 &= \mathbb{E} \left(\sum_{k=1}^d \sigma_{i,k} \cdot \sigma_{j,k} \Delta W_{k,t}^2 \right) \quad \text{since } \mathbb{E}(\Delta W_{k,t} \cdot \Delta W_{k',t}) = 0 \text{ when } k \neq k', \\
 & \quad \text{because } \Delta W_{i,t} \text{ are independent and } \mathbb{E}(\Delta W_{i,t}) = 0 \\
 &= \sum_{k=1}^d \sigma_{i,k} \cdot \sigma_{j,k} \cdot \mathbb{E}(\Delta W_{k,t}^2) \\
 &= \sum_{k=1}^d \sigma_{i,k} \cdot \sigma_{j,k} \quad \text{since } \Delta W_{k,t} \sim \mathcal{N}(0, 1), \text{ (Sec. 3.3)}
 \end{aligned} \tag{54}$$

B.2. Computation of the Gradients of \mathbf{A} , \mathbf{N} , Σ

The loss $L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})$ in Eq. (12) can be further computed as:

$$\begin{aligned}
 L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1}) &= \left[\Delta \mathbf{Y}_t - (\mathbf{A}_{t-1} \cdot \mathbf{Y}_{t-1} + \mathbf{N}_{t-1}) \right]^T \left[\Delta \mathbf{Y}_t - (\mathbf{A}_{t-1} \cdot \mathbf{Y}_{t-1} + \mathbf{N}_{t-1}) \right] \\
 &= \Delta \mathbf{Y}_t^T \Delta \mathbf{Y}_t - \Delta \mathbf{Y}_t^T \mathbf{A}_{t-1} \mathbf{Y}_{t-1} - \Delta \mathbf{Y}_t^T \mathbf{N}_{t-1} - \mathbf{Y}_{t-1}^T \mathbf{A}_{t-1}^T \Delta \mathbf{Y}_t \\
 & \quad + \mathbf{Y}_{t-1}^T \mathbf{A}_{t-1}^T \mathbf{A}_{t-1} \mathbf{Y}_{t-1} + \mathbf{Y}_{t-1}^T \mathbf{A}_{t-1}^T \mathbf{N}_{t-1} - \mathbf{N}_{t-1}^T \Delta \mathbf{Y}_t + \mathbf{N}_{t-1}^T \mathbf{A}_{t-1} \mathbf{Y}_{t-1} + \mathbf{N}_{t-1}^T \mathbf{N}_{t-1}
 \end{aligned} \tag{55}$$

According to the matrix differentiation rules in Eq. (45), (46), (47) and the definition of ϵ_t in Eq. (11), the partial derivative of $L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})$ with respect to \mathbf{A}_{t-1} is calculated as:

$$\begin{aligned}
 \frac{\partial L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})}{\partial \mathbf{A}_{t-1}} &= -2 \cdot \Delta \mathbf{Y}_t \mathbf{Y}_{t-1}^T + 2 \cdot \mathbf{A}_{t-1} \mathbf{Y}_{t-1} \mathbf{Y}_{t-1}^T + 2 \cdot \mathbf{N}_{t-1} \mathbf{Y}_{t-1}^T \\
 &= -2 \cdot \epsilon_t \mathbf{Y}_{t-1}^T
 \end{aligned} \tag{56}$$

Similarly, the partial derivative of $L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})$ with respect to \mathbf{N}_{t-1} is:

$$\begin{aligned}
 \frac{\partial L_t(\mathbf{A}_{t-1}; \mathbf{N}_{t-1})}{\partial \mathbf{N}_{t-1}} &= -2 \cdot \Delta \mathbf{Y}_t + 2 \cdot \mathbf{A}_{t-1} \mathbf{Y}_{t-1} + 2 \cdot \mathbf{N}_{t-1} \\
 &= -2 \cdot \epsilon_t
 \end{aligned} \tag{57}$$

To calculate the derivative of the loss $L_t(\Sigma_{t-1})$ (Eq. (19)) with respect to Σ_{t-1} , first calculate the derivative of

$L_t(\mathbf{\Sigma}_{t-1})$ with respect to a single coefficient $\sigma_{ij,t-1}$ of the matrix $\mathbf{\Sigma}_{t-1}$:

$$\begin{aligned}
 \frac{\partial L_t(\mathbf{\Sigma}_{t-1})}{\partial \sigma_{ij,t-1}} &= \frac{\partial \left\| \mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T - c \hat{v}(\epsilon_t) \right\|_2^2}{\partial \sigma_{ij,t-1}} \\
 &= \frac{\partial \sum_{l,m} \left((\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{l,m} - c \hat{v}(\epsilon_t)_{l,m} \right)^2}{\partial \sigma_{ij,t-1}} \\
 &= \sum_{l,m} \frac{\partial \left((\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{l,m} - c \hat{v}(\epsilon_t)_{l,m} \right)^2}{\partial \sigma_{ij,t-1}} \\
 &= \sum_{l,m} 2 \cdot \left((\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{l,m} - c \hat{v}(\epsilon_t)_{l,m} \right) \cdot \left(\frac{\partial \left((\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{l,m} - c \hat{v}(\epsilon_t)_{l,m} \right)}{\partial \sigma_{ij,t-1}} \right) \tag{1}
 \end{aligned} \tag{58}$$

where the subscript l, m represents the position at line l and column m . $(\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{l,m}$ and $cov(\epsilon_t)_{l,m}$ are:

$$(\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{l,m} = \sum_{k=1}^d \sigma_{lk,t-1} \cdot \sigma_{mk,t-1} \tag{59}$$

$$cov(\epsilon_t)_{l,m} = cov(\epsilon_{l,t}, \epsilon_{m,t}) \tag{60}$$

respectively.

Substituting Eq. (59) into term (1) of Eq. (58), we get:

$$\begin{aligned}
 \frac{\partial \left((\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{l,m} - c \hat{v}(\epsilon_t)_{l,m} \right)}{\partial \sigma_{ij,t-1}} &= \frac{\partial \sum_{k=1}^d \sigma_{lk,t-1} \cdot \sigma_{mk,t-1}}{\partial \sigma_{ij,t-1}} \\
 &= \delta_{(i,l)} \cdot \sigma_{mj,t-1} + \delta_{(i,m)} \cdot \sigma_{lj,t-1} \quad \delta_{(a,b)} = 1 \text{ when } a = b, \text{ else } \delta_{(a,b)} = 0
 \end{aligned} \tag{61}$$

Thus, Eq. (58) simplifies as:

$$\begin{aligned}
 \frac{\partial L_t(\mathbf{\Sigma}_{t-1})}{\partial \sigma_{ij,t-1}} &= \sum_{l,m} 2 \cdot \left((\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{l,m} - c \hat{v}(\epsilon_t)_{l,m} \right) \left(\delta_{(i,l)} \cdot \sigma_{mj,t-1} + \delta_{(i,m)} \cdot \sigma_{lj,t-1} \right) \\
 &= \sum_m 2 \cdot \left((\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{i,m} - c \hat{v}(\epsilon_t)_{i,m} \right) \cdot \sigma_{mj,t-1} + \sum_l 2 \cdot \left((\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T)_{l,i} - c \hat{v}(\epsilon_t)_{l,i} \right) \cdot \sigma_{lj,t-1} \tag{62} \\
 &\quad \tag{1} \qquad \tag{2}
 \end{aligned}$$

Term (1) in Eq. (62) is $2 \cdot (\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T - c \hat{v}(\epsilon_t))_{\text{line } i} \cdot (\sigma_{t-1})_{\text{column } j}$ and term (2) is $2 \cdot (\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T - c \hat{v}(\epsilon_t))_{\text{line } i}^T \cdot (\sigma_{t-1})_{\text{column } j}$, where $(\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T - c \hat{v}(\epsilon_t))^T = \mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T - c \hat{v}(\epsilon_t)$. Therefore,

$$\frac{\partial L_t(\mathbf{\Sigma}_{t-1})}{\partial \sigma_{ij,t-1}} = 4 \cdot \left((\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T - c \hat{v}(\epsilon_t)) \cdot \mathbf{\Sigma}_{t-1} \right)_{i,j} \tag{63}$$

and the derivative of $L_t(\mathbf{\Sigma}_{t-1})$ with respect to the matrix $\mathbf{\Sigma}_{t-1}$ is finally

$$\frac{\partial L_t(\mathbf{\Sigma}_{t-1})}{\partial \mathbf{\Sigma}_{t-1}} = 4 \cdot (\mathbf{\Sigma}_{t-1} \mathbf{\Sigma}_{t-1}^T - c \hat{v}(\epsilon_t)) \cdot \mathbf{\Sigma}_{t-1} \tag{64}$$

References

- M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL <https://www.tensorflow.org/>.
- M. Abramowitz. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables*. Dover Publications, Inc., New York, NY, USA, 1974. ISBN 0486612724.
- W. Bao, J. Yue, and Y. Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLOS ONE*, 12(7):1–24, 2017. doi: 10.1371/journal.pone.0180944. URL <https://doi.org/10.1371/journal.pone.0180944>.
- E. Brochu, M. V. Cora, and N. de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modelling and Hierarchical Reinforcement Learning. *CoRR*, abs/1012.2, 2010. URL <http://arxiv.org/abs/1012.2599>.
- R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, sep 1995. ISSN 1064-8275. doi: 10.1137/0916069. URL <http://dx.doi.org/10.1137/0916069>.
- A. Dingli and K. Fournier. Financial Time Series Forecasting – A Deep Learning Approach. *International Journal of Machine Learning and Computing*, 7:118–122, 2017. doi: 10.18178/ijmlc.2017.7.5.632.
- B. Donnelly. *The Art of Currency Trading: A Professional’s Guide to the Foreign Exchange Market*. Wiley Trading, Wiley, 2019. ISBN 9781119583554. URL <https://books.google.de/books?id=I7J8vwEACAAJ>.
- T. Fischer and C. Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270, 2017. doi: 10.1016/j.ejor.2017.11.054.
- P. I. Frazier. A Tutorial on Bayesian Optimization. *ArXiv*, abs/1807.0, 2018.
- A. Gensler, J. Henze, B. Sick, and N. Raabe. Deep Learning for solar power forecasting — An approach using AutoEncoder and LSTM Neural Networks. *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2858–2865, oct 2016. doi: 10.1109/SMC.2016.7844673.
- Z. Guo, H. Wang, Q. Liu, and J. Yang. A Feature Fusion Based Forecasting Model for Financial Time Series. *PLOS ONE*, 9(6):1–13, 2014. doi: 10.1371/journal.pone.0101113. URL <https://doi.org/10.1371/journal.pone.0101113>.
- S. Gyamerah. Trend forecasting in Financial time series with indicator system. *Journal of Applied Statistics*, 2019.
- P. Hennig and M. Kiefel. Quasi-Newton Methods: A New Direction. *J. Mach. Learn. Res.*, 14(1):843–865, mar 2013. ISSN 1532-4435. URL <http://dl.acm.org/citation.cfm?id=2502581.2502608>.
- S. L. Ho, M. Xie, and T. N. Goh. A Comparative Study of Neural Network and Box-Jenkins ARIMA Modeling in Time Series Prediction. *Comput. Ind. Eng.*, 42(2–4):371–375, jun 2002. ISSN 0360-8352. doi: 10.1016/S0360-8352(02)00036-0. URL [https://doi.org/10.1016/S0360-8352\(02\)00036-0](https://doi.org/10.1016/S0360-8352(02)00036-0).
- S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL <http://dx.doi.org/10.1162/neco.1997.9.8.1735>.
- J. Hull. *Options, Futures, and Other Derivatives*. Pearson, 2018. ISBN 9780134472089. URL <https://books.google.de/books?id=vpIYvgAACAAJ>.
- L. L. Jacque. *International Corporate Finance: Value Creation with Currency Derivatives in Global Capital Markets*. Wiley Finance, Wiley, 2014. ISBN 9781118783696. URL <https://books.google.de/books?id=cjUKAwAAQBAJ>.
- M. Jonas. *Bayesian Approach to Global Optimization: Theory and Applications*. Springer Netherlands, 1989. ISBN 978-94-010-6898-7.
- K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2016–2025. Curran Associates, Inc., 2018. URL <http://papers.nips.cc/paper/7472-neural-architecture-search-with-bayesian-optimisation-and-optimal-transport.pdf>.
- R. M. Karp. On-Line Algorithms Versus Off-Line Algorithms: How Much Is It Worth to Know the Future? Technical Report TR-92-044, ICSI International Computer Science Institute, 1992. URL <http://www.icsi.berkeley.edu/pubs/techreports/TR-92-044.pdf>.
- Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL <https://doi.org/10.1038/nature14539>.
- H. Liu, X. Mi, and Y. Li. Smart multi-step deep learning model for wind speed forecasting based on variational mode decomposition, singular spectrum analysis, LSTM network and ELM. *Energy Conversion and Management*, 159:54–64, 2018. doi: 10.1016/j.enconman.2018.01.010.
- D. J. Lizotte. *Practical Bayesian Optimization*. PhD thesis, University of Alberta, Edmonton, Alta., Canada, 2008.
- A. G. Malliaris. Wiener Process. In *Time Series and Statistics*, pages 316–318. Palgrave Macmillan UK, 1990. doi: 10.1007/978-1-349-20865-4_43.
- E. D. McKenzie. General exponential smoothing and the equivalent ARMA process. *Forecasting*, 3:333–344, 1984.
- J. Mockus, V. Tiesis, and A. Zilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimization 2*, 2:117–129, 2014.
- J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.
- S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski. Deep State Space Models for Time Series Forecasting. *Advances in Neural Information Processing Systems 31*, pages 7785–7794, 2018. URL <http://papers.nips.cc/paper/8004-deep-state-space-models-for-time-series-forecasting.pdf>.
- C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.
- S. Seabold and J. Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.
- S. Siami-Namini, N. Tavakoli, and A. Siami Namin. A Comparison of ARIMA and LSTM in Forecasting Time Series. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401, dec 2018. ISSN null. doi: 10.1109/ICMLA.2018.00227.
- J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou,

Foreign exchange rate forecasting with regression network

- and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012. URL <http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf>.
- P. Whittle. *Prediction and Regulation by Linear Least-Square Methods*. University of Minnesota Press., 1983. ISBN 0-8166-1148-3.
- L. Yunpeng, H. Di, B. Junpeng, and Q. Yong. Multi-step Ahead Time Series Forecasting for Different Data Patterns Based on LSTM Recurrent Neural Network. *2017 14th Web Information Systems and Applications Conference (WISA)*, pages 305–310, nov 2017. ISSN null. doi: 10.1109/WISA.2017.25.

II. Optimal liquidation of foreign currencies when FX rates follow a generalised Ornstein-Uhlenbeck process

Bibliographic Information

Li, L., Matt, PA., Heumann, C. (2022, April). Optimal liquidation of foreign currencies when FX rates follow a generalised Ornstein-Uhlenbeck process. Applied Intelligence. DOI:10.1007/s10489-022-03280-2.

Author's contribution

Linwei Li and Paul-Amaury Matt jointly design the models and the experiments for performance comparison on this task. The implementation of the model and experiments were done by Linwei Li and jointly revised with Paul-Amaury Matt. The paper was reviewed by Paul-Amaury Matt and Christian Heumann, then revised by Linwei Li.

Copyright Notice

©2023 Springer Nature. This is an accepted version of this article published in DOI:10.1007/s10489-022-03280-2. Copyright license number: 5305401350967.



Optimal liquidation of foreign currencies when FX rates follow a generalised Ornstein-Uhlenbeck process

Linwei Li^{1,2} · Paul-Amaury Matt² · Christian Heumann¹

Accepted: 19 January 2022

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

In this article, we consider the case of a multinational company realizing profits in a country other than its base country. The currencies used in the base and foreign countries are referred to as the domestic and foreign currencies respectively. For its quarterly and yearly financial statements, the company transfers its profits from a foreign bank account to a domestic bank account. Thus, the foreign currency liquidation task consists formally in exchanging over a period T a volume V of cash in the foreign currency f for a maximum volume of cash in the domestic currency d . The foreign exchange (FX) rate that prevails at time t is denoted $X_{d/f}(t)$ and is defined as the worth of one unit of currency d in the currency f . We assume in this article that the natural logarithm of the FX rate $x_t = \log X_{d/f}(t)$ follows a discrete generalized Ornstein-Uhlenbeck (OU) process, a process which generalizes the Brownian motion and mean-reverting processes. We also assume minimum and maximum volume constraints on each transaction. Foreign currency liquidation exposes the multinational company to financial risks and can have a significant impact on its final revenues, since FX rates are hard to predict and often quite volatile. We introduce a Reinforcement Learning (RL) framework for finding the liquidation strategy that maximizes the expected total revenue in the domestic currency. Despite the huge success of Deep Reinforcement Learning (DRL) in various domains in the recent past, existing DRL algorithms perform sub-optimally in this task and the Stochastic Dynamic Programming (SDP) algorithm – which yields the optimal strategy in the case of discrete state and action spaces – is rather slow. Thus, we propose here a novel algorithm that addresses both issues. Using SDP, we first determine numerically the optimal solution in the case where the state and decision variables are discrete. We analyse the structure of the computed solution and derive an analytical formula for the optimal trading strategy in the general continuous case. Quasi-optimal parameters of the analytical formula can then be obtained via grid search. This method, simply referred to as "Estimated Optimal Liquidation Strategy" (EOLS) is validated experimentally using the Euro as domestic currency and 3 foreign currencies, namely USD (US Dollar), CNY (Chinese Yuan) and GBP (Great British Pound). We introduce a liquidation optimality measure based on the gap between the average transaction rate captured by a strategy and the minimum rate over the liquidation period. The metric is used to compare the performance of EOLS to the Time Weighted Average Price (TWAP), SDP and the DRL algorithms Deep Q-Network (DQN) and Proximal Policy Optimization (PPO). The results show that EOLS outperforms TWAP by 54%, and DQN and PPO by 15 – 27%. EOLS runs in average 20 times faster than DQN and PPO. It has a performance on par with SDP but runs 44 times faster. EOLS is the first algorithm that utilizes a closed-form solution of the SDP strategy to achieve quasi-optimal decisions in a liquidation task. Compared with state-of-the-art DRL algorithms, it exhibits a simpler structure, superior performance and significantly reduced compute time, making EOLS better suited in practice.

Keywords Liquidation · Foreign exchange rates · Stochastic process · Reinforcement learning · Stochastic dynamic programming

✉ Linwei Li
Linwei.Li@campus.lmu.de

Extended author information available on the last page of the article.

1 Introduction

In this article, we consider the case of a multinational company realizing profits in a country other than its base country. The currency in use in the base and foreign

countries are referred to as the *domestic* and *foreign* currencies respectively. For its quarterly and yearly financial statements, the company transfers its profits from a foreign bank account to a domestic bank account. Foreign Exchange (forex or FX) is the trading of one currency for another. For example, one can swap the U.S. Dollar for the Euro. Foreign exchange transactions can take place on the foreign exchange market, also known as the forex market. We call *foreign currency liquidation* the task of exchanging a given total amount of cash V expressed in a given foreign currency f for the specified domestic currency d at the prevailing FX rates $X_{d/f}(t)$ during a specified period $t = 0, 1, \dots, T$ of T days. Here $X_{d/f}(t)$ denotes the value of 1 unit of currency d expressed in currency f at t . The *revenue* generated by exchanging n_t units of the foreign currency at t is thus given by $R_t = n_t / X_{d/f}(t)$ and is expressed in the domestic currency. At the end of the liquidation period, the *total revenue* generated is $W_T = \sum_{t=0}^{T-1} R_t$. The method followed to determine the traded volumes n_t is called *liquidation strategy* and the goal is to find a liquidation strategy that maximizes the expected total revenue $\mathbb{E}(W_T)$ for the company.

The FX rate is influenced by many factors in the domestic and foreign countries, such as inflation, interest rates, public debt, political stability, economic health, balance of trade, current account deficit, market confidence and speculation. In Quantitative Finance, they are thus often modeled using stochastic processes such as drifted Brownian motion or the mean-reverting process, also known as Ornstein-Uhlenbeck process. Such models can be employed to simulate FX rates trajectories and evaluate the future performance of a liquidation strategy.

Reinforcement Learning (RL) [1] is an area of Machine Learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. RL is one of three basic Machine Learning paradigms, alongside Supervised Learning and Unsupervised Learning and is a natural paradigm for solving the foreign currency liquidation problem. Recently, RL algorithms have been combined with Deep Neural Networks (DNN) [2, 3], as for example Deep Q-Networks (DQN) [4], Proximal Policy Optimization (PPO) [5] and Deep Deterministic Policy Gradient (DDPG) [6]. They have been quite successful in complex sequential decision problems in the domains of Robotics and Video Games for instance.

We observe however that these methods do not perform optimally in our setting, where the main environment variable (FX rate) is very hard to predict and can be neither controlled nor influenced by the trading agent. Fortunately, Dynamic Programming (DP) is not only tractable in our setting, but also yields the optimal liquidation strategy if the

state and action spaces are discretized with a sufficient level of granularity. Using Stochastic Dynamic Programming (SDP), we first determine the optimal solution in the case where the problem variables are discrete. Plotting the solutions obtained allows then to guess the analytical form of the optimal trading strategy in the general case of continuous trading volumes. Near-optimal parameters of the analytical formula can finally be obtained via grid search. This method, simply referred to as “Estimated Optimal Liquidation Strategy” (EOLS) is validated experimentally using the Euro as domestic currency and 3 foreign currencies, namely the USD (US Dollar), CNY (Chinese Yuan) and GBP (Great British Pound) and the performance and compute time are compared to SDP, TWAP, DQN and PPO. To evaluate the performance of a strategy, we also propose a metric called liquidation optimality O_{liq} in Section 6.4, which measures the gap between between the average transaction rate captured by the strategy and the minimum rate over the liquidation period. A value of $O_{liq} = 1$ indicates that the entire volume has been liquidated at the lowest possible rate and that the collected revenues are maximal, and $O_{liq} = 0$ means that the volume has been liquidated at the maximum rate and the revenue generated are thus minimal.

In summary, our contribution is a new algorithm which 1) is quasi-optimal and outperforms state-of-the-art DRL approaches and 2) is simpler and runs faster than SDP in the foreign currency liquidation task. The rest of this article is organized as follows. Section 2 is dedicated to a discussion of related work. In Section 3 we introduce and define the generalized OU process used to model FX rates. In Section 4, we formally define the foreign currency liquidation problem and the corresponding RL framework. Section 5 introduces SDP and exposes our proposed EOLS algorithm. Section 6 offers a detailed experimental validation and we conclude in Section 7.

2 Related work

The problem of optimal trade execution is concerned with the optimal acquisition or liquidation of large asset positions. In doing so, it is usually beneficial to split up the large order into a sequence of partial orders, which are then spread over a certain time horizon, so as to reduce the overall price impact and the trade execution costs. Optimal liquidation has become since a few decades an important problem in Finance Research and the Corporate Finance of multinational firms.

One of the simplest acquisition/liquidation strategies that exists is usually referred to as the *naive* strategy. It consists in buying/selling at every time step t of the trading

period of length T an equal portion V/T of the total volume V to acquire/liquidate, regardless of the market price or value. The average price thus captured is roughly the arithmetic average of the prices over the entire period and constitutes an important metric for benchmarking trade execution strategies. It was introduced by [7] and is referred to as *Time Weighted Average Price* (TWAP).

Bertsimas and Lo [8] consider the problem of acquiring (rather than liquidating) an asset and seek to minimize the expected execution cost, which is a mathematically equivalent problem to the one of maximizing the expected liquidation revenue. They assume that the asset price follows a simple random walk (Brownian motion without drift) but include also a linear impact of the traded volume on the asset price. Using SDP they derive a closed-form solution for the optimal acquisition strategy.

Almgren and Chriss [9] address the problem of liquidating an asset and seek to minimize the mean-variance $\mathbb{E} + \lambda \mathbb{V}$ of the execution cost, thereby including in the objective function the risk-aversion of the trading agent. They assume that the asset price follows a Brownian motion with drift and include linear temporary and permanent market impact of trading in their model. Solving the mean-variance minimization problem analytically, they obtain the closed-form solution for the optimal liquidation strategy as well as the “efficient frontier”, i.e. the curve formed by all optimal mean and variance value pairs (\mathbb{V}, \mathbb{E}) for various levels of the risk-aversion parameter λ .

Other works on optimal trade execution use the Limit Order Book (LOB), which fully describes the micro-structure of the market at any time. The authors in [10] extend the framework of [9] with elements from the market micro-structure for real-time trade execution. They adopt a RL approach and train a Q-learning agent to adapt the analytical solution derived by [9] to the current trading conditions of the market. The agent is thus able to improve the trading performance by up to 10% in average on 3 stocks. Théate and Ernst [11] and [12] apply Q-learning for the optimal liquidation on millisecond time-scale NASDAQ market microstructure data sets. They show that Q-learning can result in significant improvements over simple strategies such as the “submit-and-leave” policy. Ning et al. [13, 14] and [15] also applied RL models for optimal trade execution using the LOB as market model. Schnaubelt [16] presents a RL application for cryptocurrency exchanges by learning optimal limit order placement strategies. They use the Implementation Shortfall (IS) as metric and compare the performance of double-DQN and PPO to TWAP, the submit-and-leave execution strategy and backwards-induction Q-learning on high-frequency cryptocurrency exchange data. The results show that PPO performs best and

can reduce in average by more than 36% the IS in a single market order execution.

Other model-free RL-based approaches to trading problems require the extraction of intrinsic characteristics of the market to learn a trading policy. For instance, [17] embed a DL model into Direct RL for real-time financial trading. The DL part is a deep recurrent neural network trained for extracting informative features from the market data on which to base the trading policy. Their approach is tested on both stock-indices and commodity future contracts. The results are evaluated in terms of Profit and Loss ($P\&L$) and Sharpe ratio. They show that the proposed framework generally outperforms the original Direct RL model on these trading tasks. Wu et al. [18] adopt a similar approach to [17], but instead of using traditional recurrent neural network for feature learning and a Direct RL model for decision-making, they combine Gated Recurrent Units with DQN and Deterministic Policy Gradient (DPG) to make stock trading decisions. With the metrics of R score and Sharpe ratio, their models outperform the Turtle trading strategy (purchasing a stock or contract during a breakout and quickly selling on a retracement or price fall) and obtain more stable returns than with Direct RL in volatile market conditions. Finally, [19] address the challenge of continuous action and multi-dimensional state spaces and propose the so-called Stacked Deep Dynamic Recurrent Reinforcement Learning (SDDRRL) architecture to construct a real-time optimal portfolio. The approach is tested on 10 stocks from the S&P500 and the Sharpe ratio is used as performance measure. Their results indicate that the proposed model outperforms the rolling horizon Mean-Variance Optimization (MVO) model, the rolling horizon risk parity model, and the uniform buy-and-hold (UBAH) index.

3 Generalized Ornstein-Uhlenbeck process

In this paper, we consider times series of daily FX rates. Concretely, the market worth at time t of one unit of the domestic currency d expressed in units of the currency f is denoted $X_{d/f}(t)$. Since FX rates are always strictly positive, we can take their natural logarithm. Let us then denote $x_t = \log X_{d,f}(t)$. The series x_t is now real valued, i.e. can be either positive or negative. We choose to model the natural logarithm of the FX rate x_t by a *discrete stochastic process*, i.e. a sequence of random variables $\{x_t\}_{t \in \mathbb{N}}$ where t is a positive or null integer.

The discrete Wiener process or Brownian motion [20] is a discrete stochastic process w_t defined by the following properties: i) $w_0 = 0$, and ii) for every $t \geq 1$, the increments

given by the differences of the form $\Delta w_t = w_t - w_{t-1}$ are independent and normally distributed $\Delta w_t \sim \mathcal{N}(0, 1)$.

Discrete Brownian motion with drift is another discrete stochastic process x_t based on the Wiener process, defined by i) $x_0 \in \mathbb{R}$ and ii) for every $t \geq 1$, $\Delta x_t = x_t - x_{t-1} = \mu + \sigma \Delta w_t$, where the $\mu \in \mathbb{R}$ is a parameter called drift and $\sigma > 0$ is a second parameter called volatility. The Wiener process is a special case of Brownian motion where the drift is null ($\mu = 0$) and the volatility is one ($\sigma = 1$). Here, the drift parameter is interpreted as the deterministic trend of the process and the volatility as the amplitude of the noise or non-deterministic component in the process.

The mean-reverting process, also called *Ornstein-Uhlenbeck* (OU) process, is defined by i) $x_0 \in \mathbb{R}$ and ii) $\Delta x_t = x_t - x_{t-1} = \alpha(n - x_{t-1}) + \sigma \Delta w_t$ where $\alpha \in \mathbb{R}$ is a parameter called mean-reversion rate, $n \in \mathbb{R}$ is called mean-reversion level and $\sigma > 0$ is the volatility. This equation describes the dynamics of a variable that randomly fluctuates around some mean level n , follows random increments with an amplitude controlled by σ and tends to revert back to n at a speed controlled by α (assuming this parameter has a strictly positive value).

FX rates are often modelled as mean reverting processes over long periods and as Brownian motion with drift over short periods. Figure 1 illustrates some trajectories of a Wiener process, a Brownian motion and a mean-reverting process over a period of 500 time steps.

The mean-reverting process and drifted Brownian motion can thus be generalised to a process x_t defined by $x_0 \in \mathbb{R}$ and the increments

$$\Delta x_t = A \cdot x_{t-1} + N + \Sigma \cdot \Delta w_t \quad (1)$$

where A, N, Σ are a real-valued coefficients. Observe that when $A = 0, N = 0$, and $\Sigma = 1$, the generalized OU process is a Wiener process. When $A = 0$, the generalized OU process is a Brownian motion with drift $\mu = N$ and volatility $\sigma = \Sigma$. Finally when $A = -\alpha, N = \alpha \cdot n$ and $\Sigma = \sigma$, the generalized OU process is an OU process with mean-reversion rate α , mean-reversion level n and volatility σ .

The generalized OU process thus encompasses both the Brownian motion with drift and the mean-reverting process.

4 Optimal liquidation

In this section, we formally define the foreign currency liquidation problem and the corresponding RL framework.

4.1 Foreign currency liquidation

Assume we have the task of exchanging a total volume of V units of a foreign currency f with the domestic currency d within a period of length $T: t = 0, \dots, T$. We define the decision variable n_t as the amount of foreign currency to exchange for the domestic currency immediately after t and strictly before $t + 1$. The decision variable is subject to the following transaction volume and total liquidation constraints: for every $t = 0, \dots, T - 2$ either $n_t = 0$ or $n_{min} \leq n_t \leq n_{max}$, and $n_{T-1} = v_{T-1}$. The FX rate $X_{d/f}(t)$ denotes the value of 1 unit of currency d expressed in currency f at t . Thus, the revenue generated by exchanging n_t units of the foreign currency at t is $R_t = n_t / X_{d/f}(t)$ and is expressed in units of the domestic currency. We denote the natural logarithm of the FX rate $x_t = \log X_{d/f}(t)$ and will assume that it follows a generalised OU process with known parameters A, N, Σ . We call *liquidation trajectory* any finite decreasing sequence $v_0 = V, \dots, v_T = 0$, where v_t corresponds to the remaining volume to liquidate at time t and is given by

$$v_t = V - \sum_{j=0}^{t-1} n_j, \quad t = 1, \dots, T$$

The procedure that determines the sequence of traded volumes n_0, \dots, n_{T-1} is called *liquidation strategy*. The sum of revenues generated until step t is given by

$$W_t = \sum_{j=0}^{t-1} R_j$$

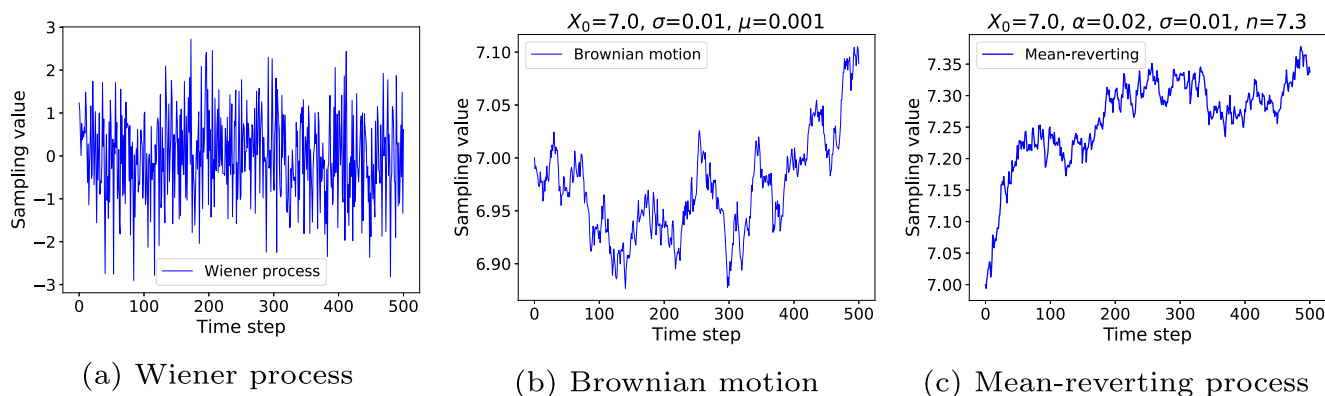


Fig. 1 Trajectories of a Wiener process, Brownian motion and mean-reverting process

and the problem is to find a liquidation strategy that maximizes the expected total revenue

$$\mathbb{E}(W_T) = \mathbb{E}\left(\sum_{t=0}^{T-1} R_t\right)$$

4.2 RL environment for foreign currency liquidation

The stated liquidation problem can be naturally formulated as a RL problem. The corresponding RL framework is for described here-after in terms of the state s of the environment, the set of possible actions a and the reward r_t collected by the agent after performing some action a in state s at time t . Note that since the problem has a finite time horizon T , we can set the discount factor for future rewards to $\gamma = 1$.

The state $s_t = [x_t, v_t, t]$ observed at time t includes the natural logarithm of the FX rate x_t , the volume v_t that remains to be liquidated and the time step t . The action a_t in our problem is defined as the previously introduced decision variable n_t subject to the transaction volume and liquidation constraints. The reward at t is $r_t = W_{t+1} - W_t = R_t$ if $t \in \{0, \dots, T - 1\}$ and $r_T = 0$.

In the state s_t , the agent executes action $a_t = n_t$ and the environment returns the reward r_t . Then the state makes a transition from $s_t = [x_t, v_t, t]$ to $s_{t+1} = [x_{t+1}, v_{t+1}, t + 1]$, where x_{t+1} follows the dynamic described in (1) and

$v_{t+1} = v_t - n_t$. The coefficients A, N, Σ of the generalized OU process remain unchanged over the entire liquidation period.

5 Methodology

In this section, we introduce SDP and expose our proposed EOLS algorithm.

5.1 Stochastic dynamic programming

First, SDP requires a discrete version of the RL framework. Since the parameters A, N, Σ of the OU process are assumed to be known (by calibration from past data) and the current value x_0 is known at the beginning of the liquidation period, we use the definition of the increment Δx_t and simulate a sufficiently large number M of random trajectories for x_t over $t = 0, \dots, T$. The generated trajectories allow us to estimate upper and lower bounds x_{max}, x_{min} for x_t . The variable x_t and subsequently the state space are then discretized using indices i, j, t as follows: $s_{i,j,t} = [x_i, v_j, t] = [x_{min} + i \cdot \Delta x, j \cdot \Delta v, t]$, where $i \in \{0, 1, \dots, N_x\}$, $\Delta x = \frac{x_{max} - x_{min}}{N_x}$, $j \in \{0, 1, \dots, N_v\}$, $\Delta v = V/N_v$ and $t \in \{0, 1, \dots, T\}$.

The set of possible actions in any state $s_{i,j,t}$ is then

$$\mathcal{A}(s_{i,j,t}) = \begin{cases} \{j' \cdot \Delta v, 0 \leq j' \leq j \text{ and } n_{min} \leq j' \cdot \Delta v \leq n_{max}, \text{ when } j' > 0\} \\ \text{for } t \neq T \\ \{j \cdot \Delta v\} \text{ for } t = T \end{cases} \tag{2}$$

The probability of transition from state $s_{i,j,t}$ and action $a_{i,j,t}$ to state $s_{i',j',t+1}$ is decomposed as follows, where we rely on the fact that x_t is not influenced by the performed action $a_{i,j,t}$. I is the indicator function and is equal to 1 if the condition given as argument is true and is equal to 0 otherwise.

$$\begin{aligned} P(s_{i',j',t+1} | s_{i,j,t}, a_{i,j,t}) &= P(x_{i'} | x_i)P(v_{j'} | v_j, a_{i,j,t}) \\ &= P(x_{i'} | x_i)I(v_{j'} = v_j - a_{i,j,t}) \end{aligned} \tag{3}$$

The probability of transition from x_i at t to $x_{i'}$ at $t + 1$ is obtained by discretizing the normal distribution $\mathcal{N}(\mu, \sigma^2)$, where $\mu = (A + 1) \cdot x_i + N$ and $\sigma = \Sigma$. Using the notation $\phi(t_a) = \int_{-\infty}^{t_a} \frac{e^{-\frac{t^2}{2}}}{\sqrt{2\pi}} dt$ for the cumulative normal distribution, we get

$$P(x_{i'} | x_i) = \begin{cases} \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x_{min}} e^{-\frac{[x-\mu]^2}{2\sigma^2}} dx = \phi\left(\frac{x_{min} - \mu}{\sigma}\right), & \text{if } i' = 0 \\ \frac{1}{\sigma\sqrt{2\pi}} \int_{x_{min}+(i'-1)\cdot\Delta x}^{x_{min}+i'\cdot\Delta x} e^{-\frac{[x-\mu]^2}{2\sigma^2}} dx \\ = \phi\left(\frac{x_{min} + i' \cdot \Delta x - \mu}{\sigma}\right) - \phi\left(\frac{x_{min} + (i' - 1) \cdot \Delta x - \mu}{\sigma}\right) & \\ \text{if } 1 \leq i' \leq N_x - 1 \\ \frac{1}{\sigma\sqrt{2\pi}} \int_{x_{max}-\Delta x}^{+\infty} e^{-\frac{[x-\mu]^2}{2\sigma^2}} dx = 1 - \phi\left(\frac{x_{max} - \Delta x - \mu}{\sigma}\right) & \\ \text{if } i' = N_x \end{cases} \tag{4}$$

Algorithm 1 SDP for the optimal liquidation of a foreign currency.

Input: Minimum and maximum logarithm of the FX rates x_{min} and x_{max} , log-FX rate increment Δx , volume increment $\Delta v = \frac{V}{N_v}$, liquidation volume V , time horizon of liquidation T , values of the FX rate and volume increments N_x and N_v , parameters of the generalized OU process A, N, Σ

Output: Optimal liquidation strategy a^*

```

1: FX rate indices  $\mathcal{I} = \{0, 1, \dots, N_x\}$ 
2: Volume indices  $\mathcal{J} = \{0, 1, \dots, N_v\}$ 
3: for  $t = T, \dots, 0$  do
4:   for  $i \in \mathcal{I}$  do
5:     for  $j \in \mathcal{J}$  do
6:        $s_{i,j,t} = [x_i, v_j, t]$ 
7:        $s_{i',j',t+1} = [x_{i'}, v_{j'}, t + 1]$ 
8:        $\mathcal{A}(s_{i,j,t})$  as defined in (2)
9:        $P(s_{i',j',t+1} | s_{i,j,t}, a_{i,j,t})$  as defined in (3) and (4)
10:       $r_t(s_{i,j,t}, a_{i,j,t}) = \frac{a_{i,j,t}}{e^{x_i}}$ 
11:       $f_t^*(s_{i,j,t}) = \begin{cases} \max_{a_{i,j,t} \in \mathcal{A}(s_{i,j,t})} [r_t(s_{i,j,t}, a_{i,j,t}) + \\ \sum_{i' \in \mathcal{I}, j' \in \mathcal{J}} P(s_{i',j',t+1} | s_{i,j,t}, a_{i,j,t}) \cdot f_{t+1}^*(s_{i',j',t+1})], \\ \text{if } t \neq T \\ r_t(s_{i,j,t}, a_{i,j,t}), \text{ if } t = T \end{cases}$ 
12:       $a_{i,j,t}^* = \begin{cases} \arg \max_{a_{i,j,t} \in \mathcal{A}(s_{i,j,t})} [r_t(s_{i,j,t}, a_{i,j,t}) + \\ \sum_{i' \in \mathcal{I}, j' \in \mathcal{J}} P(s_{i',j',t+1} | s_{i,j,t}, a_{i,j,t}) \cdot f_{t+1}^*(s_{i',j',t+1})], \\ \text{if } t \neq T \\ j \cdot \Delta v, \text{ if } t = T \end{cases}$ 
13:     end for
14:   end for
15: end for
16: Return  $a^*$ 

```

The detailed SDP for optimal liquidation of a volume V of a foreign currency is given in pseudo-code as Algorithm 1. It iterates backwards in time over $t = T, T - 1, \dots, 0$ and at each time step, iterates over $i = 1, \dots, N_x$ and $j = 0, \dots, N_v$ to calculate the maximal expected returns $f_t^*(s_{i,j,t})$ for each state $s_{i,j,t}$ using the Bellman equation. The corresponding optimal action $a_{i,j,t}^*$ is the action from the set of possible actions $\mathcal{A}(s_{i,j,t})$ that achieves the maximal expected reward $f_t^*(s_{i,j,t})$. At the end, the algorithm returns the optimal strategy a^* in the form of a tensor that associates each tuple (i, j, t) to the optimal transaction volume $a_{i,j,t}^* = n_t^*(s_{i,j,t})$ to liquidate in state $s_{i,j,t}$.

5.2 Structure of the optimal SDP solution

By running the SDP Algorithm 1 with some chosen total volume V , horizon T , initial log-FX rate x_0 and parameters A, N, Σ for the OU process, we obtain the optimal

liquidation strategy a^* . The optimal liquidation strategy can then be visualized in the form of a video sequence, where for each $t = 0, \dots, T$, the image displayed is a 3D plot of the function of 2 variables $f_t : (x, v) \mapsto a^*(x, v, t)$. Some examples are provided in Fig. 2, where we set $V = 100, T = 100, N_x = 600, N_v = 100$ and use 3 different OU processes following respectively an uptrend, sideways movement and downtrend, as shown in Fig. 3. These parameters are calibrated from historical daily EUR/USD rates.

We observe that the surfaces of the optimal strategy shown in the plots and videos have the same structure:

- Firstly, we have $a^*(x, v, T - 1) = v$ for all x , which comes from the constraint that the volume must be totally liquidated at the end of the liquidation period.
- Secondly, for any $t < T - 1$, we observe the existence of a threshold value x_{thres} for x above which $a^*(x, v, t) = 0$. This also seems intuitive, because as x increases, the revenue a/e^x decreases and above a certain value

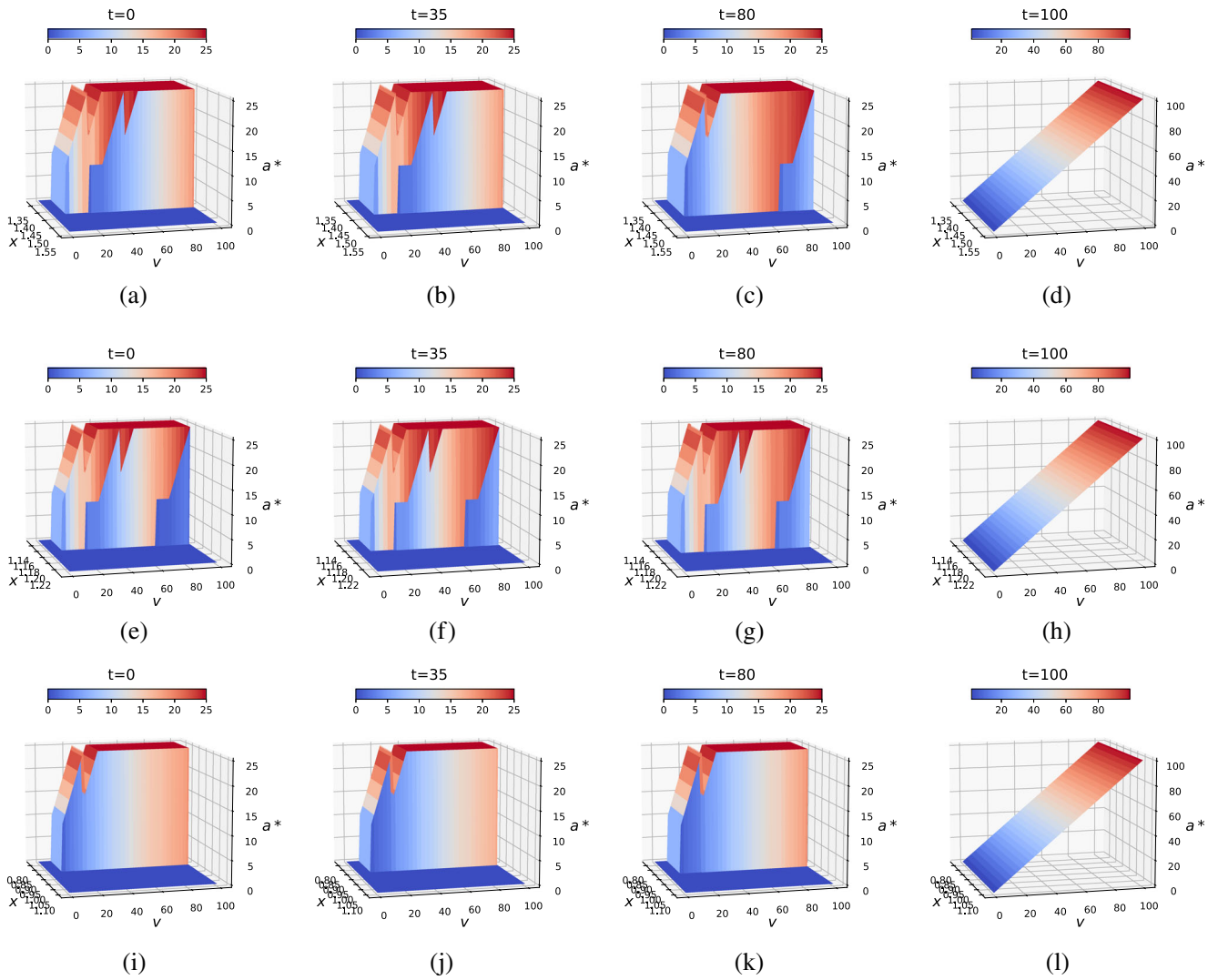


Fig. 2 Optimal liquidation strategy $a^*(x, v, t)$ computed by SDP for $V = 100, T = 100, N_x = 600, N_v = 100$. (a) - (d), (e) - (h), (i) - (l) correspond to 3 different OU processes following an uptrend, sideways movement and a downtrend

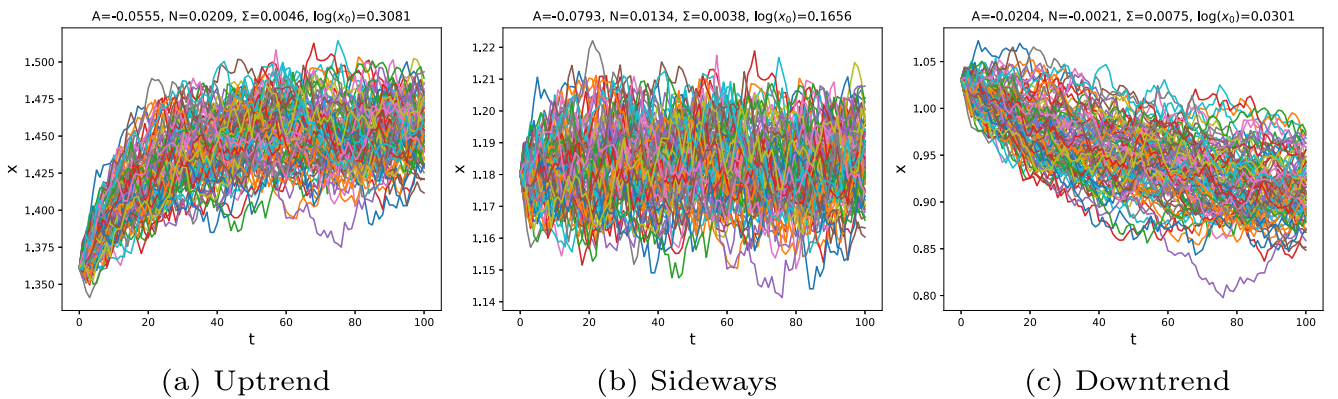


Fig. 3 Simulated FX rate trajectories (EUR/USD) of 3 different OU processes. For each process, we plot 100 trajectories of length $T = 100$. The OU process parameters are indicated above each plot

of x , the revenue becomes lower than what can be expected at a later stage and thus it is better to wait for a more favourable (lower) FX rate. But as t evolves, less time remains for the liquidation, and we would expect that the x_{thres} gets higher (i.e. that the minimum revenue for trading decreases). By plotting separately the evolution of $x_{thres}(t)$ (see Fig. 4), we observe that this is indeed the case. Interestingly, the plot seems to suggest that $x_{thres}(t)$ can be approximated by a quadratic function $x_{thres}(t) = p \cdot t^2 + q \cdot t + c$. To ensure that $x_{thres}(t)$ open upwards, increases with t and remains bounded between x_{min} and x_{max} on $[0, T]$, the following conditions must be met,

$$\begin{cases} p \geq 0, q \geq 0 \\ c \geq x_{min} \\ p \cdot T^2 + q \cdot T + c \leq x_{max} \end{cases} \quad (5)$$

We use Algorithm 2 to discretize p, q and c and search for tuples (p, q, c) that meet these conditions.

- Finally, at any given time $t < T - 1$ and for any $x < x_{thres}(t)$, the function of one variable $f_{t,x} : v \mapsto a^*(x, v, t)$ is a piecewise linear function equal to

$$g(v) = \begin{cases} 0 & \text{for } v \in [0, n_{min}[\\ v & \text{for } v \in [n_{min}, n_{max}] \\ \max(n_{min}, v - n_{min}) & \text{for } v \in]n_{max}, n_{max} + n_{min}] \\ n_{max} & \text{for } v \in]n_{max} + n_{min}, V] \end{cases} \quad (6)$$

It is easy to verify that the function g is either null or takes values in the interval $[n_{min}, n_{max}]$, thereby ensuring that the liquidation trajectory is decreasing and that the trading volume constraints are satisfied. The function $g(v)$ gives in fact the maximum volume that it is possible to liquidate with a remaining volume of v given the constraints imposed, while avoiding that after the trade is executed a volume that is strictly smaller than n_{min} remains. Indeed, due to the volume constraints, such a remaining volume v can only be

liquidated at $T - 1$, at a rate that risks to be less favourable than the current threshold FX rate.

In conclusion, the optimal liquidation strategy $a^*(x, v, t)$ can be expressed as in (7), where $H(x) := \mathbf{1}_{x \geq 0}$ denotes the Heaviside function.

$$a^*(x, v, t) = \begin{cases} H(p \cdot t^2 + q \cdot t + c - x) \cdot g(v), & \text{if } t < T \\ v, & \text{if } t = T \end{cases} \quad (7)$$

Algorithm 2 Finds feasible tuples (p, q, c) .

Input: Horizon T , number of discretization steps N_{pqc} , bounds x_{min} and x_{max}

Output: Tuples (p, q, c) that meet the condition in (5)

- 1: Initialize $p_{min} = 0, p_{max} = \frac{x_{max} - x_{min}}{T^2}, q_{min} = 0, q_{max} = \frac{x_{max} - x_{min}}{T}, c_{min} = x_{min}, c_{max} = x_{max}, \Delta p = \frac{x_{max} - x_{min}}{N_{pqc} \cdot T^2}, \Delta q = \frac{x_{max} - x_{min}}{N_{pqc} \cdot T}, \Delta c = \frac{x_{max} - x_{min}}{N_{pqc}}$, empty list R
- 2: Make a grid search over p, q, c with intervals $\Delta p, \Delta q, \Delta c$ for each step, respectively. Append combination of p, q, c that meet the condition in (5) to R
- 3: Return R

The proposed EOLS algorithm is shown in Algorithm 3. The algorithm estimates the expected total revenue by Monte Carlo simulation for each admissible set of coefficients (p, q, c) for the quadratic function $x_{thres}(t)$ and returns the liquidation strategy that maximizes the expectation.

6 Experimental validation

In this section, we evaluate the performance of EOLS and compare it to SDP, TWAP, DQN and PPO. The data set used for the FX rates is described in Section 6.1. The experimental setting is detailed in Section 6.2 and the hyperparameters of the algorithms are listed in Section 6.3. We then define a performance measure in Section 6.4. We gather and analyze all performance results in Section 6.5.

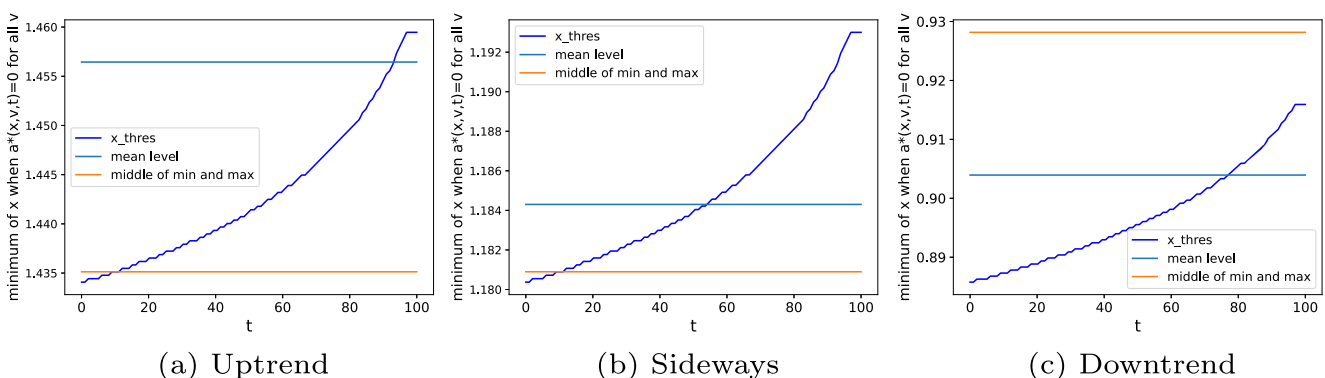


Fig. 4 Evolution of the threshold value $x_{thres}(t)$ for the 3 OU processes

The EOLS, SDP, TWAP, DQN and PPO algorithms were implemented in Python using Tensorflow [21].

Algorithm 3 Estimated Optimal Liquidation Strategy (EOLS).

```

Input: Liquidation volume  $V$ , maximum and minimum volume  $n_{max}, n_{min}$ , horizon of liquidation  $T$ , time series matrix  $\mathbf{X}$  from MC simulation of size  $[M, T + 1]$ , list  $R$  of coefficients  $(p, q, c)$  of length  $len_R$ 
Output: Estimated optimal liquidation strategy  $a^*$ 
1: Initialize zero matrix  $\mathbf{X}_{thres}$  of size  $[len_R, T + 1]$ , zero vector  $\mathbf{x}_{thres}$  of length  $T + 1$ , vector  $\mathbf{t} = [0, 1, \dots, T]$ 
2: for  $i = 0, \dots, len_R - 1$  do
3:    $(p, q, c) = R[i]$ 
4:    $\mathbf{X}_{thres}[i, :] = p \cdot \mathbf{t}^2 + q \cdot \mathbf{t} + c$ 
5: end for
6:  $f^* = -\infty$ 
7: for  $i = 0, \dots, len_R - 1$  do
8:    $\mathbf{f} = [0, \dots, 0]_M$ 
9:   for  $m = 0, \dots, M - 1$  do
10:     $v = v_0$ 
11:    for  $t = 0, \dots, T$  do
12:       $a = \begin{cases} H(\mathbf{X}_{thres}[i, t] - \mathbf{X}[m, t]) \cdot g(v) & \text{if } t < T \\ v & \text{if } t = T \end{cases}$ 
13:       $\mathbf{f}[m] = \mathbf{f}[m] + \frac{a}{exp(\mathbf{X}[m, t])}$ 
14:       $v = v - a$ 
15:    end for
16:   end for
17:   if  $mean(\mathbf{f}) > f^*$  then
18:      $f^* = mean(\mathbf{f})$ 
19:      $\mathbf{x}_{thres} = \mathbf{X}_{thres}[i, :]$ 
20:   end if
21: end for
22: Return  $a^*(x, v, t) = \begin{cases} H(\mathbf{x}_{thres}[t] - x) \cdot g(v) & \text{if } t < T \\ v & \text{if } t = T \end{cases}$ 

```

6.1 Data set

The data set used covers three major FX rates: EUR/USD, EUR/CNY and EUR/GBP. The historical data for the corresponding time series span from 2000.01.04 to 2019.01.29, amounting to 19 years of data and 4975 daily observations per time series. The data used in the experiments are daily closing values of the FX rates as provided by Bloomberg. For each FX rate, we use a moving window of length 250 shifted every 100 days to calibrate A, N, Σ using ordinary least squares regression, resulting in a total 49 sets of parameters A, N, Σ . We then generate $M = 2000$ random trajectories $(x_t)_{t=0, \dots, T}$ of length $T = 100$ for each tuple of parameters (A, N, Σ) . In the end, we split the M series into train and test sets of size 1000 each. The same data set will be used to evaluate the performance of every algorithm to ensure a fair comparison.

6.2 Experimental setting

For each tuple (A, N, Σ) of the data set, we determine the minimum and maximum rates x_{min} and x_{max} over the corresponding 1000 generated training trajectories $(x_t)_{t=0, \dots, T}$ and over the T time steps. We can then run Algorithm 1 and derive the SDP solution. In the next step, we run Algorithms 2 and 3 to get the EOLS solution. Then run DQN and PPO on the same training data and derive the corresponding solutions. The TWAP liquidation strategy is determined directly as it does not depend on any data.

All liquidation strategies thus obtained are then tested on the other 1000 test trajectories generated for each tuple (A, N, Σ) . The cumulative reward W_T on each FX rate trajectory is computed and used to compute the liquidation optimality O_{liq} , which will be defined in Section 6.4. The performance of each liquidation algorithm for an FX rate is measured as the average of O_{liq} over all test trajectories for this FX rate.

6.3 Hyperparameters of the evaluated algorithms

For both EOLS and SDP, we set the total amount of foreign currency to $V = 100$, the liquidation period to $T = 100$, the maximal trading volume to $n_{max} = 25$, the minimal trading volume constrain to $n_{min} = 10$, the number of simulated trajectories to $M = 2000$ and finally the number of trading volume steps N_v and FX rates N_x to 100.

TWAP is adapted to satisfy the trading volume constraints, by liquidating at equally spaced times the minimum possible volume n_{min} .

For DQN, the value function estimator is a 5 layers fully connected neural network with 64 hidden units in each layer. The first 4 layers use the ReLU activation function and the last layer uses a linear activation. We set the batch size to 32 and the total number of epochs to 800 so as to ensure convergence of the network on the training set. We set the replay buffer size to 10^6 (the buffer stores the experienced state, action, reward and next state pairs as tuples (s_t, a_t, r_t, s_{t+1})). We use the Adam optimizer with an initial learning rate of 0.00025. We update the value function network every 4 time steps and clip the gradient to $[-1, 1]$ so as to avoid gradient explosion during the learning phase. The target network used for generating the label for loss calculation is updated by copying the weights from the value function network every 10000 steps. A soft-update with $\tau = 0.1$ is used when updating the target network: new weights $\leftarrow \tau * \text{value function network's weights} + (1 - \tau) * \text{target network's weights}$. Since our task is episodic, the discount factor in accumulative reward calculation is set to 1. The exploration parameter in

ϵ -greedy is set initially to 0.95 and reduced progressively to 0 during training.

For PPO, we set the clip parameter (prevents excessively large policy updates) of the surrogate objective to 0.2. The buffer size is set to 10^6 and the batch size used is 100. Both the actor network and critic network are 5 layers fully connected neural networks with 64 hidden units. The first 4 layers use the ReLU activation function. The output layer of actor uses a softmax activation for discrete action selection and the one of critic is a linear activation for value function estimation. We use the Adam optimizers with initial learning rates of 0.0006 and 0.002 for actor network and critic network, respectively. The entropy coefficient for exploration control is set to 0.001, higher value causes divergence. The maximum gradient norm for clipping is set to 0.5. The total number of epochs is set to 800 so as to ensure convergence and we update both networks every epoch.

6.4 Performance measure

For a given liquidation strategy and FX rate trajectory, X_{min} and X_{max} are the minimum and maximum values of the trajectory, respectively. The *captured* logarithm of the FX rate X_{cap} is simply calculated and defined as $X_{cap} = \log(V/W_T)$. It is interpreted as the logarithm of the FX rate which would yield the same total reward as W_T if the whole volume V was liquidated in one single transaction. Based

Table 2 Performance and compute time of different algorithms on 3 data sets. For SDP, $N_x = N_v = T = 100$. For EOLS, $N_{pgc} = 10$. The best results are shown in bold

FX rates	Algorithms	Liquidation optimality O_{liq}		Time[s]
		Mean	Std.	
EUR/USD	SDP	0.654	0.212	613.4
	EOLS	0.787	0.094	345.6
	TWAP	0.510	0.031	0.6
	DQN	0.649	0.205	6530.5
	PPO	0.674	0.168	7550.0
CNY/EUR	SDP	0.661	0.192	614.1
	EOLS	0.792	0.089	344.8
	TWAP	0.512	0.034	0.7
	DQN	0.620	0.235	6530.5
	PPO	0.653	0.186	7550.0
GBP/EUR	SDP	0.688	0.187	615.4
	EOLS	0.795	0.078	345.1
	TWAP	0.514	0.029	0.6
	DQN	0.679	0.196	6530.5
	PPO	0.691	0.175	7550.0

on this, we define the liquidation optimality measure O_{liq} as in (8),

$$O_{liq} = 1 - \left[\frac{(X_{cap} - X_{min})}{(X_{max} - X_{min})} \right] \quad (8)$$

Table 1 Performance and compute time of SDP and EOLS with $N_x = \{50, 100, 200, 400, 600\}$ and $N_v = T = 100$. The coefficients A, N, Σ are calibrated from historical EUR/USD rates. The best results are shown in bold

Trend	Coefficients ²	SDP ¹					EOLS ¹
		$N_x = 50$	100	200	400	600	$N_{pgc} = 10$
Downtrend	-0.02038755/						
	-0.00205892/	0.158/	0.640/	0.816/	0.856/	0.858/	0.856/
	0.0074969/	0.078/	0.123/	0.117/	0.122/	0.128/	0.138/
	0.0301412	210.0	613.4	1960.0	6865.4	15227.3	345.6
Sideways	-0.07930878/						
	0.01341533/	0.586/	0.586/	0.586/	0.586/	0.812/	0.811/
	0.0038368/	0.150/	0.150/	0.150/	0.150/	0.136/	0.141/
	0.1655992	210.0	613.4	1960.0	6865.4	15227.3	345.6
Uptrend	-0.05549867/						
	0.02086743/	0.929/	0.929/	0.929/	0.929/	0.929/	0.929/
	0.00464528/	0.037/	0.037/	0.037/	0.037/	0.037/	0.037/
	0.3081463	210.0	613.4	1960.0	6865.4	15227.3	345.6

¹Liquidation optimality O_{liq} (Mean/Std) and compute time [s]

² $A/N/\Sigma/\log(x_0)$

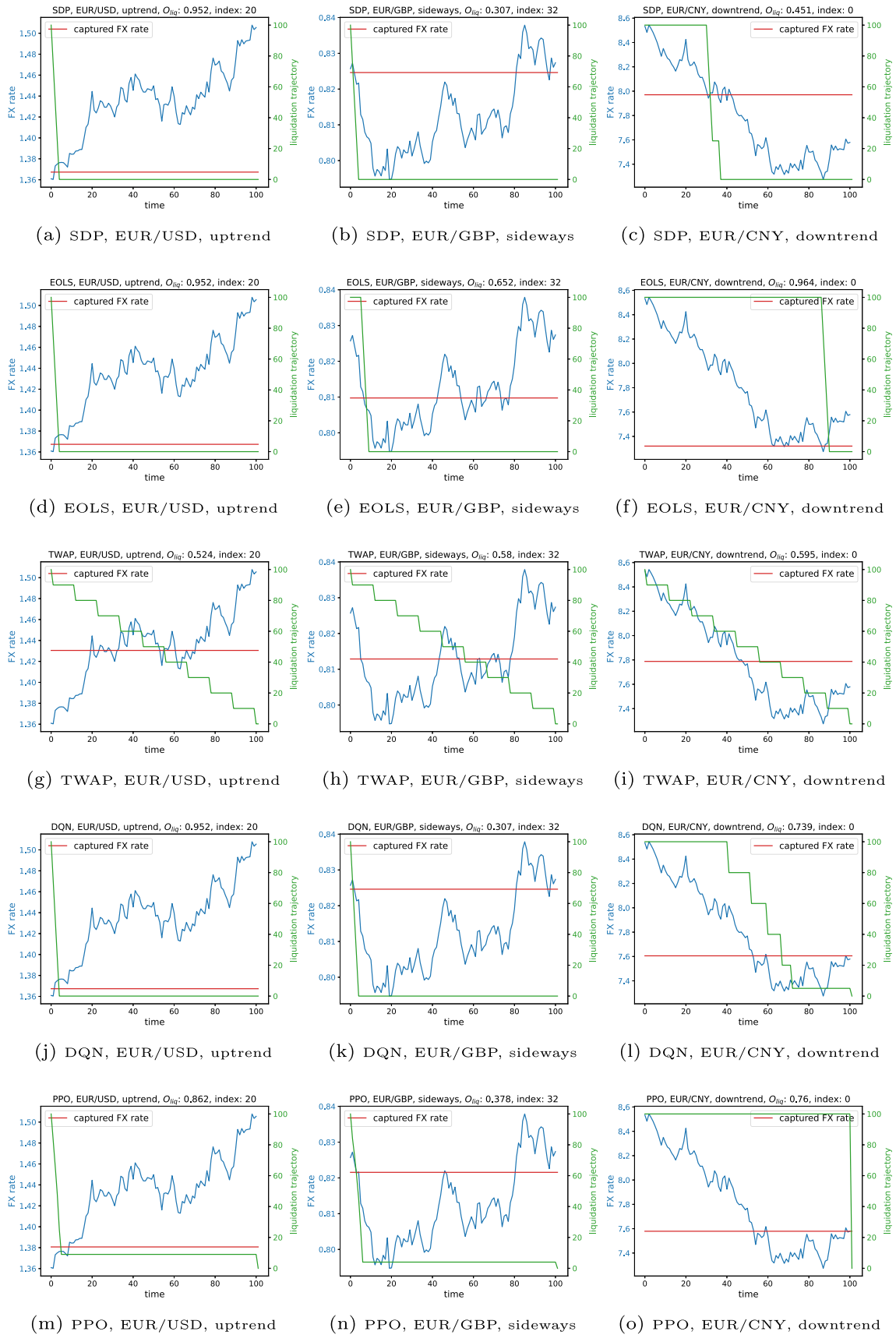


Fig. 5 Liquidation strategies obtained with SDP, EOLS, TWAP, DQN and PPO for simulated FX trajectories

O_{liq} measures how close the captured log-FX rate X_{cap} is to X_{min} . The value ranges from 0 (attained if $X_{cap} = X_{max}$) to 1 (attained if $X_{cap} = X_{min}$).

6.5 Experimental results

In this section, we first show the influence of the granularity of the log-FX rates on the performance of SDP. We then give the performance obtained for each algorithm and FX rate in the data set.

6.5.1 Performance of SDP and influence of the granularity

The performance of SDP increases with the level of granularity used for discretizing log-FX rates $\Delta x = (x_{max} - x_{min})/N_x$ and volumes $\Delta v = V/N_v$. Table 1 illustrates the different results obtained with SDP when $N_x \in \{50, 100, 200, 400, 600\}$, letting N_v fixed at 100.

In general, the larger the value of N_x , the better the liquidation performance, but the worse the compute time becomes (quadratic increase). As one can see, EOLS yields also quasi-optimal results when using only $N_{p,q,c} = 10$ candidate quadratic functions for the threshold $x_{thres}(t)$ and runs with a much lower time complexity.

6.5.2 Experimental results

In our next experiment, we choose $N_x = 100$ so that the compute time of SDP remains of the same order of magnitude as the one needed for EOLS. Table 2 shows the mean and standard deviation of the liquidation performance O_{liq} as well as the compute time obtained with 3 FX rates.

From the table, we observe that EOLS achieves the highest average performance among all the algorithms compared, although SDP could surpass EOLS if N_x was large enough, e.g. by setting $N_x = 600$ or more as discussed in Section 6.5.1. The performance of EOLS is in average 15–20% higher than the one of SDP (for $N_x = 100$), 15–27% higher than DQN and PPO and 54% higher than TWAP algorithm. DQN and PPO offer a similar improvement (27–34%) as SDP compared to TWAP, which indicates that these models can exploit the dynamics of the FX rate under the assumption of the OU stochastic process in some extent. PPO achieves better performance than DQN due to its actor-critic structure, although the training time needed is in average 15% longer.

The standard deviation of the performance for SDP, DQN and PPO is 1.7–2.6 times higher than for EOLS. The EOLS algorithm is the second least time-consuming algorithm after TWAP (which does not need to be trained) and its compute time is only about 4.5–5.2% of DQN and PPO. As mentioned in Section 6.5.1, it takes 44 times more time for SDP to achieve the same performance as EOLS.

Figure 5 provides concrete examples of liquidation trajectories for the different algorithms. For each plot, the simulated FX trajectory is marked in blue, the liquidation trajectory (remaining volume) is marked in green and the captured FX rate is indicated by a red horizontal line. The results implied in the figure are exactly the same as those in the Table 2.

7 Conclusion

In this article, we have considered the problem of liquidating a given volume V of a foreign currency f into a domestic currency d over a period of length T with minimum and maximum transaction volumes n_{min}, n_{max} and the objective of maximizing the expected total revenue $\mathbb{E}(W_T)$. We have assumed that the logarithm of the FX rate follows a generalised OU process with known parameters. We solved numerically the problem using SDP, analysed the structure of the solution and derived the analytical form for the solution. Our approach, baptised Estimated Optimal Liquidation Strategy (EOLS) achieves experimentally a performance that is on par with SDP and thus is quasi-optimal, but with a 44x speed-up. EOLS outperforms by 15–27% DQN and PPO, by 54% TWAP and runs 20 times faster than DQN and PPO.

Funding This study was funded by Daimler AG.

Declarations

Consent for Publication All authors agree to publish the article in the Journal of Applied Intelligence.

Conflict of Interests The authors have no relevant financial or non-financial interests to disclose.

References

1. Sutton RS, Barto AG (2018) Reinforcement Learning: An Introduction. 2nd edn, Cambridge. <http://incompleteideas.net/book/the-book-2nd.html>
2. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444. <https://doi.org/10.1038/nature14539>
3. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
4. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller MA (2013) Playing atari with deep reinforcement learning. arXiv:1312.5602
5. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv:1707.06347
6. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2019) Continuous control with deep reinforcement learning

7. Berkowitz SA (1988) The total cost of transactions on the NYSE. Hoboken, Wiley. NJ [u.a.]. NYSE = New York Stock Exchange
8. Bertsimas D, Lo A (1998) Optimal control of execution costs. *J Financ Mark* 1(1):1–50
9. Almgren R, Chriss N (2000) Optimal execution of portfolio transactions. *J Risk*, 5–39
10. Hendricks D, Wilcox D (2014) A reinforcement learning extension to the almgren-chriss framework for optimal trade execution. In: 2014 IEEE Conference on computational intelligence for financial engineering economics (CIFEr), pp 457–464. <https://doi.org/10.1109/CIFEr.2014.6924109>
11. Théate T, Ernst D (2021) An application of deep reinforcement learning to algorithmic trading. *Expert Syst Appl* 173:114632
12. Nevmyvaka Y, Feng Y, Kearns M (2006) Reinforcement learning for optimized trade execution. In: Proceedings of the 23rd International Conference on Machine Learning. ICML '06. Association for Computing Machinery, New York, pp 673–680. <https://doi.org/10.1145/1143844.1143929>
13. Ning B, Ling FHT, Jaimungal S (2018) Double deep q-learning for optimal execution. arXiv:1812.06600
14. Ye Z, Deng W, Zhou S, Xu Y, Guan J (2020) Optimal trade execution based on deep deterministic policy gradient. In: DASFAA (1), pp 638–654. https://doi.org/10.1007/978-3-030-59410-7_42
15. Lin S, Beling PA (2020) An end-to-end optimal trade execution framework based on proximal policy optimization. In: Bessiere C (ed) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20. International Joint Conferences on Artificial Intelligence Organization, Yokohama, pp 4548–4554. Special Track on AI in FinTech, <https://doi.org/10.24963/ijcai.2020/627>
16. Schnaubelt M (2022) Deep reinforcement learning for the optimal placement of cryptocurrency limit orders. *Eur J Oper Res* 296(3):993–1006
17. Deng Y, Bao F, Kong Y, Ren Z, Dai Q (2016) Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems* 28(3):653–664
18. Wu X, Chen H, Wang J, Troiano L, Loia V, Fujita H (2020) Adaptive stock trading strategies with deep reinforcement learning methods. *Inf Sci* 538:142–158
19. Aboussalah AM, Lee C-G (2020) Continuous control with stacked deep dynamic recurrent reinforcement learning for portfolio optimization. *Expert Syst Appl* 140:112891
20. Malliaris AG (1990) Wiener Process. In: Time Series and Statistics. Palgrave Macmillan, London, pp 316–318. https://doi.org/10.1007/978-1-349-20865-4_43
21. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jia Y, Jozefowicz R, Kaiser L, Kudlur M, Levenberg J, Mané D, Monga R, Moore S, Murray D, Olah C, Schuster M, Shlens J, Steiner B, Sutskever I, Talwar K, Tucker P, Vanhoucke V, Vasudevan V, Viégas F, Vinyals O, Warden P, Wattenberg M, Wicke M, Yu Y, Zheng X (2015) TensorFlow : Large-Scale Machine Learning on Heterogeneous Systems. <https://www.tensorflow.org/>

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



is Machine Learning, Reinforcement Learning and their applications in the field of Finance.



Center for Cyber Cognitive Intelligence at the Fraunhofer Institute for Manufacturing Engineering and Automation IPA in Stuttgart, Germany. His primary research interests are in Artificial Intelligence and Quantum Computing.



ics are methods for missing data, model selection, model averaging, interpretable machine learning and natural language processing and its applications in medical research, epidemiology, insurances, social media analysis and industry.

Linwei Li This author obtained his bachelor's degree from Ningbo University in Ningbo, China in 2014. His major was Communication Engineering. Then he obtained a double master's degrees from Tongji University in Shanghai, China and the Technical University of Munich in Munich, Germany in 2018. In the same year, he began a doctoral degree in the Statistics Department of the University of Munich (LMU) in Munich, Germany. His research focus

Paul-Amaury Matt This author graduated in Electrical and Computer Engineer at CentraleSupélec in Paris, France in 2005. He received his M.Sc. degree in Advanced Computing in 2005 and his Ph.D. degree in Logic & Artificial Intelligence from Imperial College London, United Kingdom in 2009. He has been holding the position of Cognitive Computing Engineer at Daimler AG from 2018 to 2021 and is currently a research associate in the

Christian Heumann This author obtained his diploma degree from LMU Munich, Germany in 1992, his major is Statistics. Then he got his PhD from LMU Munich in 1997 with a topic on likelihood based marginal models for longitudinal categorical data. He is professor at the Department of Statistics at LMU Munich where he teaches basic statistics, statistical inference and deep learning methods for natural language processing. His research topics

Affiliations

Linwei Li^{1,2}  · Paul-Amaury Matt² · Christian Heumann¹

Paul-Amaury Matt
paulmatt67@gmail.com

Christian Heuman
chris@stat.uni-muenchen.de

¹ Department of Statistics, University of Munich (LMU), Ludwigstr.
33, Munich, 80539, Germany

² Daimler AG, Breitwiesenstr. 5, Stuttgart, 70565, Germany

Part III
Conclusions

3. Liquidation performance when tested with historical FX rates

Recall that in contribution I, we evaluated the forecasting performance of RegPred Net on historical FX rate data set. The forecasted parameters A, N, Σ can well reflect the dynamics of FX rate. In contribution II, we evaluate the performance of the optimal strategy estimated by EOLS by assuming that the FX rate during the liquidation period is known while in reality, this is unknown. Therefore, we use the liquidation framework illustrated in Fig. 3.1 to solve the liquidation problem where the FX rate in the liquidation period is unknown and should be forecasted. We derive some conclusive experimental results using the historical FX rate as the data set.

3.1 Experimental setting

We use the same data set described in Sec. 6.1 of contribution I. Each sample in the data set is part of the complete FX rate time series, it includes 3 episodes: training, validation and testing. We follow the setting described in Sec. 6.2 of contribution I to train the RegPred Net, the training and validation episodes are used as input and label, respectively. The testing episode is used as the FX rate of liquidation period. The trained RegPred Net forecasts the dynamics of the testing episode with the concatenation of training and validation episodes as input. The forecasted dynamics change over the liquidation period, we simply take the mean of them as the representative dynamics of this period. We then use EOLS to estimate the optimal liquidation strategy with the forecasted dynamics as inputs. The final results are evaluated on the testing episode with the liquidation optimality metric O_{liq} described in Sec. 6.4 of contribution II. We use the same data set to evaluate the other algorithms involved in the comparison. The experimental setting and hyperparameters selection are the same as described in Sec. 6.2, 6.3 of contribution II, respectively.

3.2 Experimental results

We list the experimental results in Table 3.1. From the table, we observe that the results are generally consistent with the results tested using the simulated data in Table 2 of contribution II.

EOLS still performs best compared to other algorithms in the liquidation optimality metric. It performs on average 7.3% better than SDP (for $N_x = 100$), 14-22% better than DQN and PPO and 18.8% better than benchmark model TWAP. However, the magnitude of EOLS outperforming TWAP here is reduced compared to the result obtained in contribution II where EOLS outperformed TWAP by an average of 54%. The same is true for DQN and PPO, although they still perform 6-8% better than TWAP. These algorithms are not substantially ahead of the TWAP due to the bias in RegPred Net’s forecasting of the FX rate during the liquidation period. Whether the forecast dynamics reflect the actual trend of the exchange rate has a significant impact on estimating the optimal liquidation strategy.

The standard deviation of EOLS’s result is 9.3% lower than SDP and 23% lower than DQN and PPO. The time consumption of the algorithm is the same as described in contribution II, i.e. the EOLS algorithm is still the second least time-consuming algorithm after TWAP, where TWAP does not require any training. When $N_x = 100$, EOLS runs on average 20 faster than DQN and PPO and 44 faster than SDP.

Fig. 3.1 shows the liquidation strategies estimated by all algorithms involved in the comparison. The mean and 95% confidence interval of the FX rate forecasted by RegPred Net are marked as a blue line and purple area, respectively.

FX rates	Algorithms	Liquidation optimality O_{liq}		Time[s]
		Mean	Std.	
EUR/USD	SDP ¹	0.596	0.277	613.4
	EOLS ¹	0.620	0.265	345.6
	TWAP	0.494	0.076	0.6
	DQN ¹	0.525	0.327	6530.5
	PPO ¹	0.541	0.297	7550.0
EUR/CNY	SDP ¹	0.611	0.276	614.1
	EOLS ¹	0.674	0.233	344.8
	TWAP	0.497	0.070	0.7
	DQN ¹	0.541	0.312	6530.5
	PPO ¹	0.553	0.251	7550.0
EUR/GBP	SDP ¹	0.598	0.287	615.4
	EOLS ¹	0.648	0.271	345.1
	TWAP	0.544	0.081	0.6
	DQN ¹	0.567	0.308	6530.5
	PPO ¹	0.582	0.291	7550.0

¹Using the dynamics forecasted by RegPred Net

Table 3.1: Performance and computation time of different algorithms on 3 data sets of real historical FX rate series. The dynamics of FX rate are forecasted by RegPred Net. For SDP, $N_x = N_v = T = 100$. For EOLS, $N_{pgc} = 10$. The best results are shown in bold.

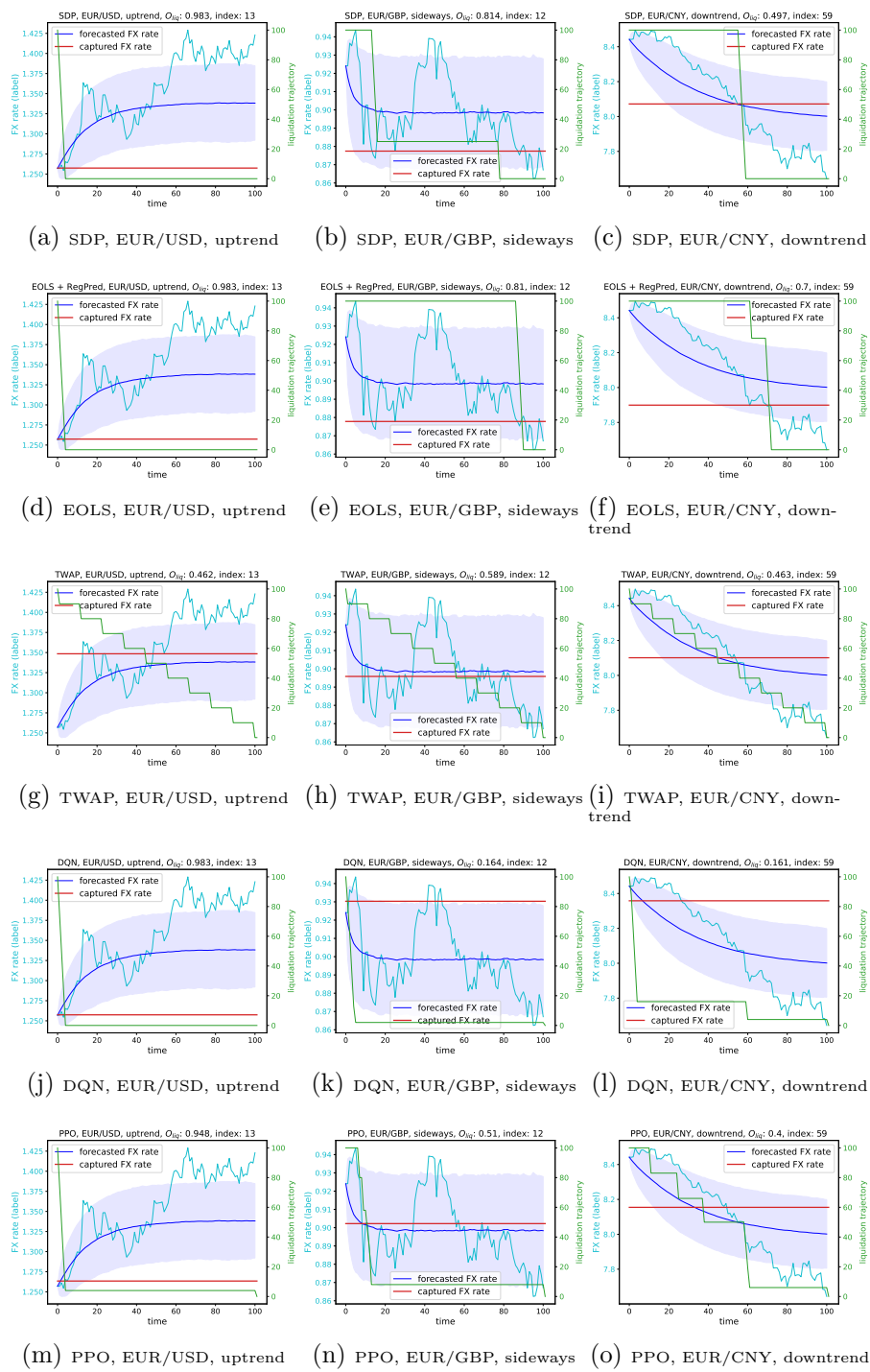


Figure 3.1: Liquidation strategies obtained by testing SDP, EOLS, TWAP, DQN and PPO with historical FX rate trajectories. The dynamics of the FX series are predicted by RegPred Net. The purple area indicates the 95% confidence interval of the prediction.

4. Conclusion and Outlook

This thesis focuses on solving the problem of optimal liquidation of foreign currencies. This task requires the decision-maker to exchange a certain amount of foreign currency for domestic currency during the liquidation period. The decision-maker should seek the optimal liquidation strategy that maximizes the amount of the exchanged domestic currency under exchange rate uncertainty.

The difficulties in solving the optimal liquidation problem include: 1) the foreign exchange rate during the liquidation period is unknown and difficult to predict, 2) this uncertainty makes the RL algorithm difficult to estimate the liquidation strategy. To solve these problems, we have made the following attempts. To address problem 1), we use state-of-the-art forecasting models to predict the foreign exchange rate during the liquidation period. However, since the liquidation period is usually long (one quarter), we found experimentally that the existing algorithms are often inaccurate in predicting daily FX rates for such long periods. Meanwhile, most forecasting models focus on one-step-ahead forecasting or forecasting time series with cyclical regularity. To the best of our knowledge, models capable of forecasting daily FX rates within a quarter are rare. For solving problem 2), we use state-of-the-art RL algorithms to search for the optimal liquidation problem. The RL algorithms are suitable for solving sequential decision problems. However, we find that they perform rather sub-optimally. Their heuristic exploration approach makes the convergence both unstable and slow. SDP can compute the theoretical optimal solution to the sequential decision problem but requires a complete environment model (state transition probabilities), but the dynamics of exchange rate during the liquidation period are unknown.

Inspired by the above experimental attempts, we propose a corresponding algorithm for each problem to solve it. First, we propose the RegPred Net in the contribution I to predict the FX rate in the liquidation period and the dynamic parameters that describe the FX rate. These parameters are from the generalized OU process. The forecasting performance of RegPred Net is tested on three FX rate datasets (EUR/CNY, EUR/USD and EUR/GBP). The experimental results show that RegPred Net outperforms ARMA, ARIMA, LSTM and Auto-LSTM. Compared to them, the RMSE of the forecasted results of RegPred Net is 25 – 30% lower, the correlation coefficient R is 2-7 times higher, the MDA is 10% higher, and R-Squared is positive while the other algorithms are negative. Then, we introduce

in contribution II the EOLS algorithm, which estimates the optimal liquidation strategy based on the dynamic parameters predicted by RegPred Net. We show experimentally that the complete framework of EOLS+RegPred Net successfully solves the optimal liquidation problem for foreign currencies. The liquidation optimality O_{liq} of the strategy estimated by our framework is 14 - 22% higher than DQN and PPO, 7.3% higher than that of SDP (for $N_x = 100$), and 18.8% higher than that of TWAP. EOLS runs on average 44 times faster than SDP and 20 times faster than DQN and PPO.

Our proposed algorithm has the following significance. RegPred Net is the first regression network that can predict the time series of foreign exchange rates over 100 steps and outperforms the state-of-the-art models. Compared to other forecasting models such as LSTM, RegPred Net is an interpretable algorithm that models FX rates with a simple structure and does not require a complex training process. EOLS is the first algorithm to achieve quasi-optimal strategy estimation in liquidation tasks using a closed-form solution computed by SDP. It approximates and simplifies the computed optimal strategy by an analytical formula. A quasi-optimal strategy can then be determined by estimating the parameters of this formula through grid search. EOLS has a more straightforward structure and better performance than state-of-the-art DRL algorithms. Without heuristic exploration, the algorithm’s computational time is significantly reduced. Combined with the dynamics of the liquidation period predicted by RegPred Net, EOLS becomes more efficient and valuable in practical liquidation tasks.

Finally, I would like to summarize the research that is relevant to the approaches in this thesis and that is worth pursuing in the future. Although our proposed optimal liquidation framework can estimate quasi-optimal liquidation strategies, there is still much room for improvement. The following points can be used as a starting point for improving the framework.

- The hyperparameters of RegPred Net will increase with the number of network layers. The Bayesian optimization algorithm used to search for the best hyperparameters has constraints on the input dimension. Therefore, the usual Bayesian optimization will not work when the RegPred network has a large number of layers. For this problem, we 1) can try to use the improved high-dimensional Bayesian optimization [57], [58], which will also bring a larger amount of computation. 2) Replace the layers other than the first layer in RegPred Net with LSTMs, which can handle high-dimensional time series inputs. At this point, since the LSTM is a parameterized network, we can simply adopt the existing hyperparameter initialization method and let the parameters in the LSTM fit the inputs and outputs. In this way, using Bayesian optimization to find the optimal hyperparameters is not necessary.
- Although RegPred Net outperforms other algorithms in predicting 100-step

exchange rates. However, it can not predict rapid, short-term fluctuations in exchange rates caused by emergencies such as financial crises and wars. In this regard, we can introduce a neural network for sentimental analysis to predict the impact of recent news on exchange rate movements. Thereby adjusting the dynamic parameters predicted by RegPred Net or use confidence coefficients to show the possibility of sharp short-term fluctuations in the exchange rate.

- The EOLS algorithm has the potential to be used for other financial liquidation tasks or even other real-world sequential decision-making tasks. The idea of simplifying SDP in this model deserves to be further investigated and applied to other tasks.

Complete List of Publications

Li, L., Matt, PA., Heumann, C. (2022, April). Optimal liquidation of foreign currencies when FX rates follow a generalised Ornstein-Uhlenbeck process. Applied Intelligence. DOI:10.1007/s10489-022-03280-2.

Li, L., Matt, PA., Heumann, C. (2022). Forecasting foreign exchange rates with regression networks tuned by Bayesian optimization. arXiv:2204.12914.

Eidesstattliche Erklärung

(Siehe Promotionsordnung vom 12. Juli 2011, § 8 Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

Stuttgart, January 24, 2023

.....
(Linwei Li)

