Pattern-adaptive Time Series Prediction via Online Learning and Paralleled Processing using CUDA

Zhifeng He[‡], Meng Han[†*], Bing Han[†]
[‡]Department of Electrical and Computer Engineering, Auburn University
[†]Data-driven Intelligence Research (DIR) Lab, Kennesaw State University
[*]Corresponding Author: menghan@kennesaw.edu

Abstract—In this paper we study the problem of prediction of time series data in a computational efficient way. We proposed a SARIMA (Seasonal AutoRegressive Integrated Moving Average) based-online learning algorithm to update the model parameters via Gradient Descent based algorithm every time when a new data comes, based on the gap between prediction and true value of the time series data. In this way, our model can adapt to the new pattern of the time series if the pattern changes, and generates more accurate prediction results comparing with existing SARIMA implementations. Besides, it has the capability of learning the true pattern and converge to optimal model even if the start point is far from optimal. We also implement our algorithm on CUDA (Compute Unified Device Architecture) platform to support training millions of different online learning models concurrently by leveraging the GPU's capability of parallel computing. The performance of our online learning algorithm is evaluated with simulation results.

I. Introduction

In the past decades, time series forecasting played an important role in extensive application domains such as financial market analysis, supply chain management, intelligent transportation system, and speech analysis. Typically, time series forecasting identified the underlying pattern from the historical observations, and used the pattern to predict future observations with the assumption that the data will resemble themselves in the future. In trying to forecast various of time series data, statistical methods [1], traditional machine learning [2], [3], and neural networks are developed [5]. Among the existing time series forecasting methods, the Box and Jenkins method proposed in 1970 is the fundamental role that popularize of a group of classical models such as AutoRegressive Moving Average (ARMA), AutoRegressive Integrated Moving Average (ARIMA), and Seasonal AutoRegressive Integrated Moving Average (SARIMA) models.

In practice, ARMA can describe of a noisy linear system [7] and short-term time series [9], and have been used in a wide of areas, such as forecasting of short-term traffic flow and household electric consumption [9]. The ARIMA containing the integrated term is a more general version of ARMA, which can make non-stationary time series to stationary time series by differencing technique. Typically, differencing technique is used for eliminate the influence trends and thus making data stationary [7]. With the integrated term, ARIMA has higher accuracy to represent some types of time series than ARMA [10]. However, the ARIMA model still has many limitations, especially for online learning with large dataset. First, the ARIMA model can't predict the new data well after the model is trained with historical data, if the new data doesn't follow a pattern similar with the historical data. In that case, we need to retrain the model, will be very time consuming. Second, existing algorithms for estimating the parameters of ARIMA model such as least squares and maximum likelihood still need to access the entire dataset, which is not suitable for online learning with streaming data. Third, the most variants of ARIMA are essentially batch learning, which are computational expensive, limiting its application in Big Data or real-time situations [7].

Existing implementations of SARIMA algorithms fail on capturing the new pattern and generates unsatisfying prediction results. Due to the high computational complexity of running existing SARIMA implementations, it is unrealistic to run the algorithm every time when a new data comes in, to fit the most current pattern. To overcome these difficulties, here we propose an Online Learning SARIMA algorithm which can update the trained model each time when a new data comes in via Gradient Descent, aiming to enhance both long-term prediction accuracy and speed performance, with only linear complexity [4], [6], [8]. In this way, our model can adapt to the new pattern by tuning model parameters based on the gap between prediction and true value in real time, and thus generates predictions that reflects new pattern. Our online ARIMA model consider the seasonality and integrated terms. In this aspect, our model relaxes the stationary assumption and could deal with non-stationary time series prediction problems. To deal with real-time and Big Data time series prediction, we move forward to propose our model in a distribute computing system leverage of GPU parallel computational power. Our online ARIMA models could be distributed in GPU cells, update weights independently, and achieve different parameters for different dataset efficiently.

The remainder of this paper is organized as follows. The prediction problem formulation and SARIMA model is given in Section II. A high level description of the steps of proposed online learning algorithm is presented

978-1-7281-4121-3/19/$31.00 ©2019 IEEE
DOI 10.1109/MASSW.2019.00013

TABLE I
Notation table

Notation	Meaning
B	lag operator
p	non-seasonal AR order
q	non-seasonal MA order
n	non-seasonal difference order
a	seasonal AR order
d	seasonal MA order
m	seasonal difference order
s	seasonality, i.e., time span of repeating seasonal pattern
t	current time slot. $t-i, i=1,2,\ldots,t$ means previous i-th time slot
$x_\tau, \tau = t, t-1, \ldots, 0$	the true value of variable of interest at τ
$\vec{X}_t$	vector representation of time series $\{x_t, x_{t-1}, \ldots, x_0\}$
$\hat{x}_t$	the predicted value of variable of interest at t
$\vec{Z}_t$	vector representation of time series $\{z_t, z_{t-1}, \ldots, z_0\}$
$\hat{z}_t$	the predicted value of error term at t

in Section III, which is followed by a detailed explanation about prediction and parameter tuning steps in Section IV and Section V, respectively. The idea of implementing the proposed algorithm on GPU is discussed in Section VII, and the performance evaluation of proposed algorithm is presented in Section VIII. Section IX concludes the paper.

II. Problem Statement and SARIMA model

We first define the following major variables that will be used in our proposed algorithm in Table.I.

The error terms $\vec{Z}_t$, which is the gap between the prediction value and the actual value of the variable of interest, are generally assumed to be independent, identically distributed variables sampled from a normal distribution with zero mean. Given a $SARIMA(p,n,q)(a,m,d)_s$ process and the notations defined in Table.I, the problem is to, at the beginning of t, calculate the predicted value of variable of interest $\hat{x}_t$ before getting the true value of variable of interest x_t:

$$\begin{aligned}&(1-\sum_{i=1}^{a}\theta_i B^{i\times s})(1-\sum_{j=1}^{p}\phi_j B^j)\Delta_s^m\Delta^n\hat{x}_t\\&=(1+\sum_{i=1}^{d}\sigma_i B^{i\times s})(1+\sum_{j=1}^{q}\rho_j B^j)\hat{z}_t\end{aligned}\tag{1}$$

where θ_i is the parameter for seasonal AR part, ϕ_j is for non-seasonal AR part, σ_i is for seasonal MA part, and ρ_j is for non-seasonal MA part.

Existing software implementations such as $StatsModels$ can give us optimal model parameters (p,n,q,a,m,d,s) and optimal parameters $\theta_i, \phi_j, \sigma_i$, and ρ_j for our time series, which will be used to initialize our proposed online learning algorithm.

III. Process of Online Learning Algorithm

Before starting our proposed algorithm, we need to collect sufficient amount of historical data, and then use existing software implementations such as $StatsModels$ to find optimal model parameters, which will then be used to initialize our proposed algorithm. At time slot t, there are 5 major steps in our online learning algorithm:

1) Difference: transform the original time series into stationary time series with seasonal difference & difference
2) Prediction: predict the value of stationary time series
3) Undifference: transform the predicted value of stationary time series into predicted value of original time series with seasonal undifference & undifference. Output the predicted value of original time series
4) Get true value: get true value of original time series which will be transformed into true value of stationary time series by seasonal difference & difference
5) Parameter tuning: update model parameters based on the gap between predicted value of stationary time series & true value of stationary time series.

In the following we will explain each step listed above in detail.

IV. Difference, Prediction, and Undifference

A. Difference

After getting seasonal difference order n and non-seasonal difference order m, we do transformation on our time series data $\vec{X}_t$ to make it stationary. Denote $\vec{Y}_t = \Delta_s^m\Delta^n\vec{X}_t$ as the stationary time series data after doing seasonal difference and non-seasonal difference on $\vec{X}_t$, then we have:

$$\begin{aligned}y_\tau &= (1-B^s)^m(1-B)^n x_\tau\\&=\left(\sum_{i=0}^{m}(-1)^i C_m^i B^{i*s}\right)\left(\sum_{j=0}^{n}(-1)^j C_n^j B^j\right)x_\tau\\&=\left(\sum_{i=0}^{m}\sum_{j=0}^{n}(-1)^{i+j}C_m^i C_n^j B^{i*s+j}\right)x_\tau\\&=\sum_{i=0}^{m}\sum_{j=0}^{n}(-1)^{i+j}C_m^i C_n^j x_{\tau-i*s-j},\\&\forall\tau = t, t-1, \ldots, m*s-n\end{aligned}\tag{2}$$

where $C_i^j = \frac{j!}{(j-i)!i!}$, $0 \le i \le j$ denotes the combination formula.

B. Prediction

After converting original time series $\vec{X}_t$ into stationary time series $\vec{Y}_t$, we can start prediction using $\vec{Y}_t$. In the following we will do some simplification of Eq.(1) for the easy of presentation. For the left hand side of Eq.(1), we have:

$$\left(1-\sum_{i=1}^{a}\theta_i B^{i\times s}\right)\left(1-\sum_{i=1}^{p}\phi_i B^i\right)\Delta_s^m\Delta^n\hat{x}_t$$
$$=\hat{y}_t-\sum_{j=1}^{p}\psi_j y_{t-j}-\sum_{i=1}^{a}\sum_{j=0}^{p}\eta_{i,j}y_{t-(i*s+j)} \quad (3)$$

where ψ_j and $\eta_{i,j}$ are parameters converted from θ_i and ϕ_j. The conversion can be inferred based on the subscript of y_{t-j} and $y_{t-(i*s+j)}$. After getting θ_i and ϕ_j from existing implementations of SARIMA, we can calculate ψ_j and $\eta_{i,j}$ based on θ_i and ϕ_j.

Let's denote vector $\vec{\Gamma}$ as:

$$\begin{aligned}\vec{\Gamma}=&(1,2,...,p,\\&s,s+1,...,s+j,...,s+p,...,\\&i*s,i*s+1,...,i*s+j,...,i*s+p,...,\\&a*s,a*s+1,...,a*s+j,...,a*s+p)\end{aligned} \quad (4)$$

Denote ℓ as the number of elements of $\vec{\Gamma}$. We can see that $\ell=(a+1)*(p+1)-1$. Denote vector $\vec{\beta}_t$ as:

$$\vec{\beta}_t=\left(\beta_0,\beta_1,\beta_2,...,\beta_{(a+1)*(p+1)-2}\right) \quad (5)$$

and the number of elements of $\vec{\beta}_t$ is ℓ.

Based on Eqs.(3), (4), and (5), we can convert the left hand side of Eq.(1) to:

$$(1-\sum_{i=1}^{a}\theta_i B^{i\times s})(1-\sum_{i=1}^{p}\phi_i B^i)\Delta_s^m\Delta^n\hat{x}_t$$
$$=\hat{y}_t-\sum_{j=0}^{\ell-1}\beta_j*y_{t-\Gamma_j} \quad (6)$$

where Γ_j is the j-th element of $\vec{\Gamma}$. Similarly, let's denote vector $\vec{H}$ as:

$$\begin{aligned}\vec{H}=&(1,2,...,q,\\&s,s+1,...,s+j,...,s+q,...,\\&i*s,i*s+1,...,i*s+j,...,i*s+q,...,\\&d*s,d*s+1,...,d*s+j,...,d*s+q)\end{aligned} \quad (7)$$

and vector $\vec{\alpha}_t$ as

$$\vec{\alpha}_t=\left(\alpha_0,\alpha_1,\alpha_2,...,\alpha_{(d+1)*(q+1)-2}\right) \quad (8)$$

Then we can convert the right hand side of Eq.(1) to the following:

$$(1+\sum_{i=1}^{d}\sigma_i B^{i\times s})(1+\sum_{j=1}^{q}\rho_j B^j)\hat{z}_{t+1}=\hat{z}_t+\sum_{i=0}^{\kappa-1}\alpha_i*z_{t-H_i} \quad (9)$$

where H_i is the i-th element of vector $\vec{H}$, and $\kappa=(d+1)*(q+1)-1$ is the number of elements of vector $\vec{\alpha}$

Based on Eqs.(6) and (9), we can convert Eq.(1) to the following:

$$\hat{y}_t-\sum_{j=0}^{\ell-1}\beta_j*y_{t-\Gamma_j}=\hat{z}_t+\sum_{i=0}^{\kappa-1}\alpha_i*z_{t-H_i}$$
$$\Rightarrow\hat{y}_t=\sum_{j=0}^{\ell-1}\beta_j*y_{t-\Gamma_j}+\hat{z}_t+\sum_{i=0}^{\kappa-1}\alpha_i*z_{t-H_i} \quad (10)$$

C. Undifference

Recall that $\hat{y}_t$ is the predicted value of the next slot of the stationary time series $\vec{Y}_t$ after transformation. To get $\hat{x}_t$ which is the predicted value of the original time series $\vec{X}_t$, we will need to do the reverse operation of seasonal difference and non-seasonal difference, i.e., $undifference$ as we call it here, on $\vec{X}_t$. According to Eq.(2), we have:

$$\hat{y}_\tau=\hat{x}_\tau+\sum_{i=1}^{m}\sum_{j=1}^{n}(-1)^{i+j}C_m^iC_n^jx_{\tau-(i*s+j)}$$
$$\Rightarrow\hat{x}_\tau=\hat{y}_\tau-\sum_{i=1}^{m}\sum_{j=1}^{n}(-1)^{i+j}C_m^iC_n^jx_{\tau-(i*s+j)},$$
$$\forall\tau=t,t-1,\ldots,m*s-n \quad (11)$$

which is the predicted value of the variable of interest and can be used as the output.

V. Parameters tuning

At the beginning of time slot t, after getting the predicted value $\hat{y}_t$, we will wait for the true value y_t. Due to the uncertainty of time series data, it is possible that the trend or the pattern of the time series data will change over time, and current model is not able to capture the true pattern of future time series data. Therefore, it becomes necessary for us to recalibrate our model at each time slot by evaluating the loss of the model, which can be measured by the difference between the predicted value and the true value that we observed. Our goal is to minimize the loss, and Gradient Descent is proved to be an effective optimization algorithm for minimization problems. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point [12], which means that we can use the negative of the gradient to update each of our model parameters so that the loss can be minimized in the fastest direction.

Our parameter tuning algorithm is based on Gradient Descent. It not only considers the loss of current time slot t, but also considers the loss of previous time slots $t-1,t-2,\ldots$, which means that we calculate the gradient of the loss of both current time slot t and previous time slots. In the following we will illustrate mathematically how to use Gradient Descent to minimize the loss.

Denote $L(\cdot)=\frac{1}{2}(\cdot)^2$ as the loss function which uses $L2$ norm to calculate the loss. Denote ∇_ω as the partial derivative with respect to a variable ω, λ as the learning rate of gradient descent. For each model parameter $\beta_v, 0\leq$

$v \leq \ell - 1$, according to Eq.(10) and loss function, we can update it accordingly.

At the end of each time slot t, the model parameters will be updated according to the above equation, and the updated model parameters is used for prediction of next time slot $t+1$. In this way, our model will be updated iteratively in the direction of minimizing the difference between predicted value and observed value.

And as our model is based on continuous online learning, where we have essentially an infinite number of data points and every data point is a training data, we are not usually in danger of overfitting the data, comparing with current implementations of $SARIMA$ [13].

VI. Computational Complexity Analysis

From Eqs.(2),(10),(11),and gradient descent equation for $\alpha_v, 0 \leq v \leq \kappa - 1$, we can see that the computational complexity of our online learning algorithm is $\mathcal{O}\left(mn + (\ell+\kappa)^2\right) = \mathcal{O}\left(mn + ((a+1)(p+1) + (d+1)(q+1) - 2)^2\right)$, which is of linear complexity. In most real life scenarios, the value of these model parameters are small. Therefore, the online learning algorithm is computational efficient.

VII. Implementation on CUDA

We implemented our online learning algorithm on CUDA (Compute Unified Device Architecture) platform to support running millions of independent time series models in parallel [14]. Existing time series libraries don't support running multiple time series models concurrently. However, consider some real life scenarios such as predicting the sales of millions of different commodities. The pattern of the time series of each commodity is different with others and thus each commodity requires an independent time series prediction model to predict its sales. CUDA is a parallel computing platform and application programming interface model created by Nvidia. GPUs had evolved into highly parallel multi-core systems allowing very efficient manipulation of large blocks of data. This design is more effective than general-purpose CPUs for algorithms in situations where processing large blocks of data is done in parallel such as machine learning [14]. These characteristics make CUDA an ideal platform to implement our online learning algorithm for paralleled prediction and parameter tuning on millions of independent time series models. In the simulation section we will evaluate the performance of CUDA implementation of our proposed algorithm.

VIII. Experiment Results

In this section we evaluate the performance of our proposed algorithm with simulation results.

As described above, we first get the optimal model parameters and model parameters of the time series data using $StatsModels$. Here we ignore the MA part in order to reduce the number of variables, due to the fact the more variables to be tuned, the more number of iterations it takes for Gradient Descent to converge to optimal. To search for the optimal model parameters without Moving Average part equals search for optimal values for model $SARIMA(p,n,q)(a,m,d)_s$ with $q=0$ and $d=0$, which can be achieved by setting the search range for q and d during grid search for the optimal parameters.

By ignoring the Moving Average part, all the elements of vector $\vec{Z}_t$ is set to 0, thus prediction equation Eq.(10) becomes:

$$\hat{y}_t = \sum_{j=0}^{\ell-1} \beta_j * y_{t-\Gamma_j} \tag{12}$$

Then we use the obtained parameters to initialize our online learning algorithm based on above simplifications. At each iteration, i.e., each time stamp of the time series data, we follow the procedures listed above to generate prediction and update prediction model.

In Fig.1 and Fig.2 we compared the performance of our online learning algorithm with the existing implementation of SARIMA in terms of prediction accuracy under two scenarios: (1) the pattern of time series doesn't change; (2) the pattern of time series changes. Time series data used in scenario (1) and (2) are from [15], [16], respectively, which are real-world examples and have 144 and 1235 data points respectively. We plot these two time series in Fig.1(a) and Fig.2(a). The performance metrics used are Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). To test the performance, We used last 40 data points (totally 144 data points) for scenario (1) and last 400 data points (totally 1235 data points) for scenario (2), respectively. Fig.1(b) and Fig.2(b) plot the prediction versus true value, Fig.1(c) and Fig.2(c) plot MAE and RMSE. We can see that under scenario (1) the performance of SARIMA and our proposed algorithm are close, while under scenario (2) our algorithm outperforms existing SARIMA significantly, which suggests that our proposed algorithm is suitable for time series whose pattern changes. While there are some minor changes in the pattern of the time series, since the model parameters are fixed for SARIMA, the model is unable to adjust to the minor changes; in our online learning algorithm, whenever there is a gap between the prediction and true value, model parameters can be fine tuned to adjust to the minor changes by updated at the direction of minimizing the gap.

TABLE II
RMSE and MAE comparison of scenario (1)

average RMSE of SARIMA	average RMSE
542.09	420.89
average MAE of SARIMA	average MAE
18.62	17.42

The online learning algorithm also has the capability of converging to optimal solution even if it's start point drifts

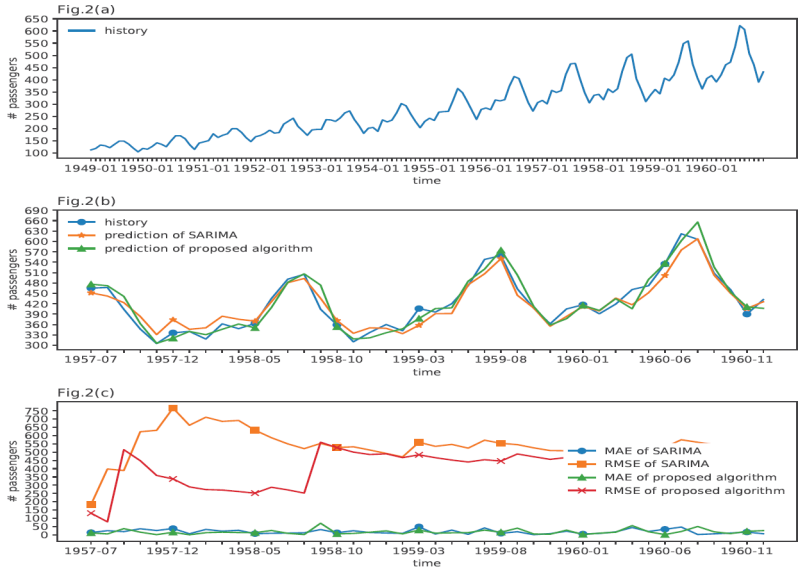

Fig. 1. Performance comparison of scenario (1)

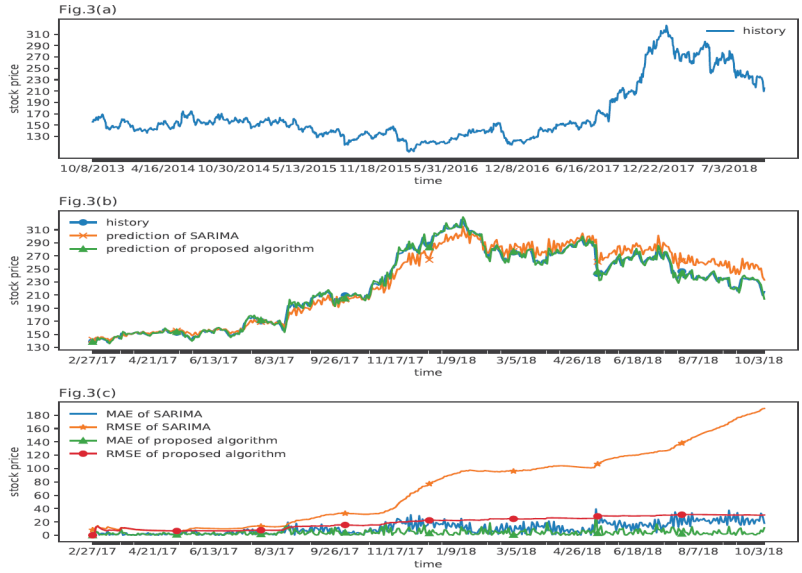

Fig. 2. Performance comparison of scenario (2)

TABLE III
RMSE and MAE comparison of scenario (2)

average RMSE of SARIMA	average RMSE
70.75	19.17
average MAE of SARIMA	average MAE
10.72	4.02

seriously from the optimal. Fig.3 shows such an example. We use the same time series as in Fig.1. We can see from Fig.3(a) that at first the model parameters are far from optimal and thus prediction is also far from true value. As the iteration goes, the distance between predictions and true values becomes smaller, which means the online learning model is being able to capture the pattern of the time series,implying that the model parameters are converging to the optimal. This observation tells that even there are serious changes in the pattern of the time series, our model is still able to adapt to the new pattern iteratively and make correct predictions. We plot the value of model parameters changes of our online learning algorithm over time to visualize this process in Fig.3(b).

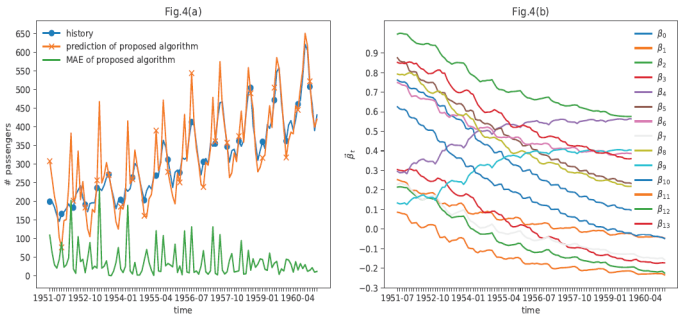

Fig. 3. Algorithm Convergence

Table.IV shows the performance comparison in terms of time efficiency of running our algorithm on CPU and GPU, using Java and CUDA respectively. As for the hardware configuration of these two platforms, the CPU on which the Java implementation runs is 2.3 GHz Intel Core i9, and the GPU on which the CUDA implementation runs is NVIDIA Tesla P100 which has 16GB memory. We can see that the paralleled execution of multiple independent models makes CUDA implementation on GPU exceeds the Java implementation on CPU significantly in terms of run time efficiency. Please note that the above time is measured with one iteration.

TABLE IV
Running time comparison

num. of time series	Java running time	CUDA running time
80000000	222.25 seconds	10.18 seconds

IX. Conclusions

In this paper we present our SARIMA based-online learning algorithm which can learn the new pattern of time series with linear computational complexity. The model is first initialized using parameters found from existing implementations. After that, for each time slot, the model will generate prediction result and when a new data comes in, the model parameters will be updated via a Gradient Descent based-algorithm based on the difference between prediction and true value. Compared with existing solutions, it is computationally efficient and has the ability to adapt to new pattern to generate better prediction results when the pattern of time series changes. Simulation results show that our online learning algorithm outperforms existing SARIMA implementation under different scenarios. We also implemented the online learning algorithm in CUDA platform for parallel processing of large amount of different time series and compare the time efficiency of CUDA implementation on GPU and Java implementation on CPU.

References

[1] Kalpakis, Konstantinos, Dhiral Gada, and Vasundhara Puttagunta. "Distance measures for effective clustering of ARIMA time-series." in Proc. 2001 IEEE international conference on data mining, pp.273-280, 2001.

[2] Ahmed, Nesreen K., Amir F. Atiya, Neamat El Gayar, and Hisham El-Shishiny. "An empirical comparison of machine learning models for time series forecasting." Econometric Reviews, 29(5-6), pp.594-621, Aug 2010.

[3] Cao, Li-Juan, and Francis Eng Hock Tay. "Support vector machine with adaptive parameters in financial time series forecasting." IEEE Transactions on neural networks, vol.14, no.6, pp.1506-1518, 2003.

[4] Xurong Li, Shouling Ji, Meng Han, Juntao Ji, Zhenyu Ren, Yushan Liu, Chunming Wu. "Adversarial Examples Versus Cloud-based Detectors: A Black-box Empirical Study" IEEE Transactions on Dependable and Secure Computing, pp.1-16, 2019.

[5] Qiu, Xueheng, Le Zhang, Ye Ren, Ponnuthurai N. Suganthan, and Gehan Amaratunga. "Ensemble deep learning for regression and time series forecasting." in Proc. 2014 IEEE symposium on computational intelligence in ensemble learning (CIEL), pp.1-6, 2014.

[6] Liyuan Liu, Meng Han, Yiyun Zhou, and Reza Parizi. "E^2C-chain: A two-stage incentive education employment and skill certification blockchain." in The 2nd IEEE International Conference on Blockchain (Blockchain-2019), 2019., pp.1-9, 2019.

[7] Liu, Chenghao, Steven CH Hoi, Peilin Zhao, and Jianling Sun. "Online arima algorithms for time series prediction." in Proc. Thirtieth AAAI conference on artificial intelligence, 2016.

[8] Meng Han, Dongjing Miao, Jinbao Wang, and Liyuan Liu. "Defend the clique- based attack for data privacy" in International Conference on Combinatorial Optimization and Applications. Springer, Cham, 2018, pp. 262–280.

[9] Chujai, Pasapitch, Nittaya Kerdprasop, and Kittisak Kerdprasop. "Time series analysis of household electric consumption with ARIMA and ARMA models." in Proc. the International MultiConference of Engineers and Computer Scientists, vol.1, pp.295-300, 2013.

[10] Chen, Peiyuan, Troels Pedersen, Birgitte Bak-Jensen, and Zhe Chen. "ARIMA-based time series model of stochastic wind power generation." IEEE transactions on power systems, vol.25, no.2, pp.667-676, 2009.

[11] https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace. sarimax.SARIMAX.html

[12] https://en.wikipedia.org/wiki/Gradient_descent.

[13] https://cnl.salk.edu/ schraudo/teach/NNcourse/overfitting.html

[14] https://en.wikipedia.org/wiki/CUDA.

[15] https://www.kaggle.com/rakannimer/air-passengers.

[16] https://www.analyticsvidhya.com/blog/2018/10/predicting