 doi:10.20944/preprints202101.0326.v1

Article

AN ENTERPRISE TIME SERIES FORECASTING SYSTEM FOR CLOUD APPLICATIONS USING TRANSFER LEARNING

Arnak Poghosyan [1,2,†,‡*], Ashot Harutyunyan [1,‡*], Naira Grigoryan [1,‡], Clement Pang [1,‡], George Oganesyan [1,‡], Sirak Ghazaryan [1,‡], and Narek Hovhannisyan [3,‡]

1 VMware, Inc., Palo Alto, CA 94304, USA
2 Institute of Mathematics of NAS RA, Yerevan 0019, Armenia
3 TeamViewer Armenia, Yerevan 0018, Armenia
† This paper is an extended version of our paper published in Second CODASSCA Workshop on Collaborative Technologies and Data Science in Artificial Intelligence Applications, Yerevan, Armenia 2020.
‡ These authors contributed equally to this work.

Abstract: One of the key components of application performance monitoring (APM) software is AI/ML empowered data analytics for predictions, anomaly detection, event correlations and root cause analysis. Time series metrics, logs and traces are three pillars of observability and the valuable source of information for IT operations. Accurate, scalable and robust time series forecasting and anomaly detection are desirable capabilities of the analytics. Approaches based on neural networks (NN) and deep learning gain increasing popularity due to their flexibility and ability to tackle complex non-linear problems. However, some of the disadvantages of NN-based models for distributed cloud applications mitigate expectations and require specific approaches. We demonstrate how NN-models pretrained on a global time series database can be applied to customer specific data using transfer learning. In general, NN-models adequately operate only on stationary time series. Application to non-stationary time series requires multilayer data processing including hypothesis testing for data categorization, category specific transformations into stationary data, forecasting and backward transformations. We present the mathematical background of this approach and discuss experimental results from the productized implementation in Wavefront by VMware (an APM software) while monitoring real customer cloud environments.

Keywords: time series analysis; anomaly detection; neural networks; hypothesis testing; trend analysis; periodicity analysis; cloud applications; pretrained models; transfer learning

MSC: 37M10, 62M45, 62M20

1. Introduction

One of the main goals of IT infrastructure and application monitoring/management solutions is full visibility into those system with increasingly more intelligence. Prediction of performance degradations, their root cause analysis, as well as self-remediation of issues before they affect a customer environment are anticipated features of modern cloud management solutions. Self-driving data centers require the availability of proactive Analytics with AI for IT operations (AIOps) [1] in view of nowadays very large and distributed cloud environments. The key capabilities of the AIOps are predictions, anomaly detection, correlations and root cause analysis on all acquired data including traces, logs and time series (see [2–10] with references therein).

Time series collection and analysis is of great importance for various reasons like anomaly detection, anomaly prediction, correlations and capacity planning [11–15]. Administrators of cloud environments require automated forecasts of future metric-data values for prediction of future states of applications or infrastructure components. Capacity planning requires trend analysis for resource consumption like CPU, memory, etc. for prediction of additional needed processor bandwidth or mass-storage capacity in order to prevent delays and failures. Time series data can also be used for correlation analysis and as a source of anomaly events for further root cause analysis.

Time-series analysis is a significant branch of mathematics and computing that includes a variety of different types of analytical procedures, computational tools, and forecasting methods. It is sufficient to mention the well-known and powerful approaches like Fourier analysis, time series decompositions, forecasting by SARIMA and Holt-Winters' methods (see [16–20] with references therein). However, distributed cloud infrastructures and applications require relatively quick forecasts and are associated with significant temporal constraints, forestalling lengthy and computationally intensive analyses.

In this paper (see also [2]), we focus our attention to time series forecasting with further application to anomaly detection problem. Application of NN-models and other ML techniques may produce efficient methods [21,22], but naïve implementation in a cloud-computing environment fails to provide adequate response times and would likely be far too expensive for most clients. Training and storing of neural networks are both time-consuming and expensive with respect to the necessary resources (CPU, GPU and memory). Hence, it is not feasible to train those models in demand for the specified time series data. Moreover, it would not be attainable to train and store special-purpose neural networks for all of the different possible types of time series. From the other side, a naïve attempt to train a single neural network to analyze all of the various different types of time-series data would also likely fail, since different types of time-series data exhibit different types of behaviors and temporal patterns. A single neural network would need a vast number of nodes and even vaster sets of training data to produce reasonable forecasts for global time-series data. The truth should be somewhere in the middle.

The purpose of the paper is application of NN-based models to time series forecasting in cloud applications. The main idea is training of a generic NN-model and transferring the acquired knowledge to a customer specific time series data never seen before. This should be the only way of overcoming the challenges regarding the resource utilization (GPU trainings of the networks) as the application of the pretrained model doesn't require on-demand network training. We already have criticized the naive approach to this problem due to overwhelming complexity. Such a solution is feasible if we can narrow down the problem to some classes of time series data with specific behaviors for which application of pretrained models are attainable. Moreover, those classes should be enough large to cover the sufficient portion of unseen customer data and enough specific by the behavior to deal with moderate network configurations. We found that class of stationary time series can be properly handled by NN-models. Unfortunately, this class is not common in the discussed domain. Conversely, the majority of time series data contain non-stationary patterns like trend, seasonality or stochastic behavior. However, the class of stationary time series data can be extended to time series categories which can be transformed into the needed class by some simple transformations.

This outlines the main idea of our approach - perform time series classification, find those needed transformations and utilize the pretrained model. We develop the theoretical foundation of the approach and show the results of its realization in real cloud-computing environments. Implementation and testing are performed in Wavefront by VMware [23]. Wavefront offers a real-time metrics monitoring and analytics platform designed for optimization of cloud and modern applications.

Worth noting that our main goal is the performance of the approach for cloud environments rather than accuracy compared to the well-known techniques that perform individual training for each specified time series data in the GPU accelerated environments. For us, the performance is balance between accuracy and resource utilization. We observed that the accuracy of the forecasts is comparable to the classical ARMA related approaches while preserving resource consumption on acceptable levels. In particular, application of the pretrained network to a specified time series in a

 doi:10.20944/preprints202101.0326.v1

cloud environment can be performed without GPU acceleration and with moderate number of CPU cores.

One of the important applications of time-series forecasts is detection or prediction of infrastructure and application performance degradations or failures. Accurate and fast anomaly/outlier detection leads to proactive problem resolution and remediation before it affects a customer environment. It means that timeliness and preciseness of anomaly detection are of great importance for distributed systems. However, worth noting that forecasts based anomaly detection may be associated with low response times especially for longer forecast horizons. Moreover, precautionary procedures should be taken for reduction of false positive anomalies that can unnecessary disturb users with alarms.

Another important aspect tightly related to the problem is association of time series outliers with system anomalies which is roughly saying not always true. In any case, such problems are unsolvable without intrusion of domain expertise into mathematical models or their outcomes. Our solution to anomaly detection utilizes a test window which is smaller than the forecast window for providing adequate response times and meanwhile contains enough data points to reduce the possible false positives. The fraction of violations of the confidence bounds of the forecasts in the test window generate an anomaly signal (time series). Whenever the anomaly score rises above a particular threshold, the anomaly monitor can generate alarms and warnings or launch preventive procedures.

2. Related Work

Application of pretrained NN-models to solution of different problems is a well-founded approach for many domains like classification, image processing, voice recognition, text mining, etc. (see [24–28] with references therein). It is known as transfer learning for some applications [28] and is a natural approach for knowledge generalization and complexity reduction. Such pretrained networks (VGG, ResNet, BERT, etc.) have deep learning sophisticated architectures requiring serious resources and datasets for their trainings. Application of this idea to time series forecasting is a novel approach. We only consider the first steps and many questions still need clarifications. We trained the simplest networks like MLP or LSTM, but the exact required architecture remains unknown and extended research will be carried out elsewhere.

Time series forecasting is an important area for many diverse areas such as econometrics, signal processing, astrophysics, etc. The classical theory of forecasting [16–18] deals with time series data with wide range of properties. ARMA models are very powerful for stationary time series data. However, in many problems (e.g. economy, business) time series exhibit non-stationary variations due to trend or seasonality (deterministic or stochastic). Models that analyze non-stationary data require knowledge of those patterns. Some models assume that variations are deterministic and apply regression analysis to handle both trend and seasonality. One of the interesting approaches is time series decompositions known as STL [20]. Other approaches model data as having stochastic trend as in ARIMA and stochastic seasonality as in SARIMA. Holt-Winters' seasonal and SARIMA models represent a broad and flexible class relevant for many applications. It has been found empirically that many time series can be adequately fit by those models, usually with a small number of parameters.

Naturally, models based on artificial neural networks should have better performance due to their non-linearity, flexibility and ability of generalization [21,29,30]. It was assumed that no any specific assumptions need to be made about the model which should be one of the most important advantages. Different authors showed in their studies and experiments [31] that better results compared to SARIMA and related models could be achieved only by combination of transformations that 'stabilize' the behavior (e.g. detrending, deseasoning) of the specified time series [32]. However, the results regarding the forecasting of non-stationary time series data via NN models are very controversial [31,33–35].

It turned out that time series data categorization is the crucial milestone for the analyses. Application of NN-models and deep learning for time series classification should be explored elsewhere [36]. In the current research, we restrict ourselves by the classical hypothesis testing methods for trend

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

and seasonality detection. Each data category identifies the set of transformations that will convert any class-representative into a stationary time series. For example, deterministic and stochastic trends can be detected via KPSS [37] and ADF [38–41] tests, respectively, that have the best combined power for data with moderate length. Seasonality detection has some additional peculiarities. It requires two-fold investigations. First, the periodicity lag should be explored and for many applications multi-seasonality is preferable with different lags. One of the powerful methods for period-lag determination is Fourier analysis treating in the frequency domain [19]. Another interesting and very simple approach is phase dispersion minimization (PDM) [42–44] treating the time domain with some advantages over the classical Fourier analysis. It is admitting efficient optimal implementation and is applicable for data with few observations, with non-regular sampling, with gaps and with non-sinusoidal periodicities. Secondly, some tests should be applied to reveal either deterministic or stochastic seasonality. PDM approach can test both categories based on some importance measures. Valuable tests for detecting deterministic versus stochastic seasonality are CH [45], HEGY [46], OCSB [47] and more [48–51]. In this article, we restrict ourselves by the determinisitic seasonality. Stochastic seasonality will be considered elsewhere.

Time series anomaly/outlier detection have been investigated by numerous authors for many applications [11–14,52–57]. It is known as very hard problem with many diverse ramifications. NN-based methods and deep learning are now becoming very popular and powerful [54,58].

3. Main Idea

Application of NN-based models to time series forecasting in cloud applications faces several challenges. One of the main ones is the restriction on the computing resource utilization. Complex network trainings require powerful GPUs and sufficient volume of data. Those are real issues in cloud environments, and the solution is in transfer learning, or in other words, in utilization of pretrained NN-models. We train a network on a global dataset collected across different customers and store it for further application to a specific time series data definitely never seen before. It means application of the NN-models on-demand for a specified time series via several CPU cores without GPU acceleration. The training of the models will be performed in private on-premises powerful data centers with enough GPU resources. Figure 1 shows a high level schema of this idea.

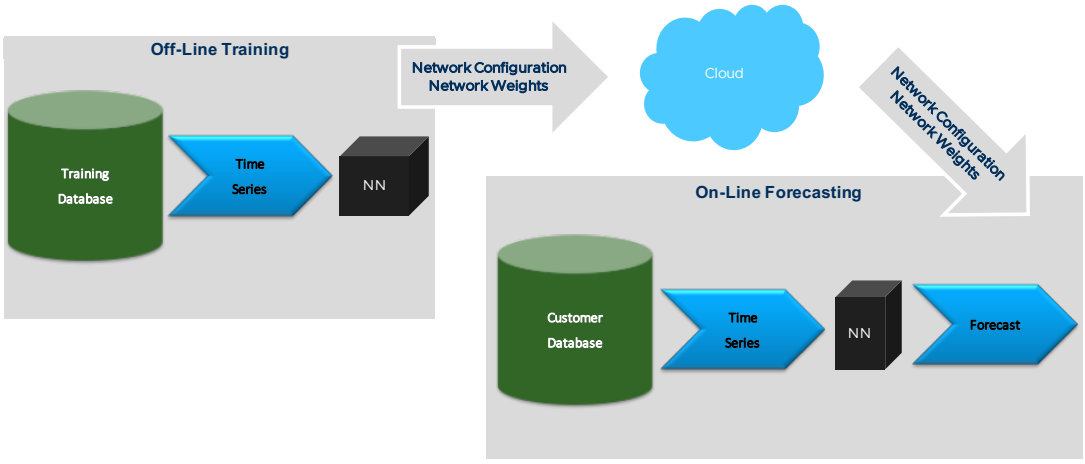

Figure 1. Utilization of pretrained NN-models for cloud applications.

The entire system consists of two separate and totally independent subsystems called as off-line and on-line modes. The off-line mode performs model training for a global training database containing time series data with different behaviors. We store the network configuration and weights in a cloud (as a file in "json" format) for on-demand access. The on-line mode corresponds to a customer cloud-computing environment. The weights and configuration of the pretrained network can be restored from the file and applied to the specified time series data. Off-line mode requires GPU

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

empowered data centers while on-line mode is the customer common computing space without GPU acceleration.

The diversity of time series data behaviors is a crucial milestone connected with the system of Fig. 1 that probably will not allow to handle all of them with a unique trained model. We already mentioned the role of data categorization for proper model construction. One of the scenarios is selection of data classes and the corresponding class-specific network models. Those pretrained networks can be stored and called on-demand. Preliminary data categorization should be performed in both on-line and off-line modes for treating with the required models. This scenario should be considered elsewhere.

Another scenario developed in this paper is selection of a single class that can be adequately treated by some NN-models and transformation of other time series into it. This scenario should be more optimal as only one model should be trained, stored and applied. How to find the class with the best trainable and transformable characteristics? Our previous discussion indicated that the class of stationary time series should be the first candidate for experiments. They can be properly modeled by NN-models, and the techniques of transformation of a non-stationary time series into a stationary (called before as stabilization) are theoretically well-founded. The set of stabilizing transformations is time series class specific. A deterministic trend can be stabilized via detrending by a regression (linear or non-linear), a stochastic trend by a differencing of the proper order, etc. Our implementation applies different well-known hypothesis testing algorithms for time series classification. Deterministic versus stochastic trend classification can be performed via KPSS and ADF tests. Deterministic versus stochastic periodicity analysis can be performed via PDM and CH tests.

As a result, we perform model trainings only for stationary time series data. We have two possibilities. Either collect only stationary time series data for a global database where selection similarly can be performed by a hypothesis testing or collect all available time series and feed the models after preliminary stabilization.

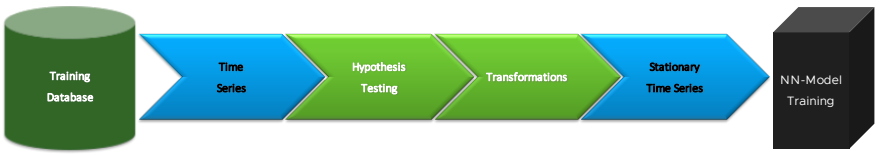

Figure 2. Off-line training of an NN-model for stationary time series.

Flowchart of Figure 2 describes the second possibility. The training dataset contains time series of any behavior. Hypothesis testing engine performs data categorization. A stationary time series will be used directly for the model training. In case of non-stationary time series, the engine also identifies the set of transformations that will transform it into a stationary one and only then feed the network. In case of unknown data type, the time series will be skipped from the process of training.

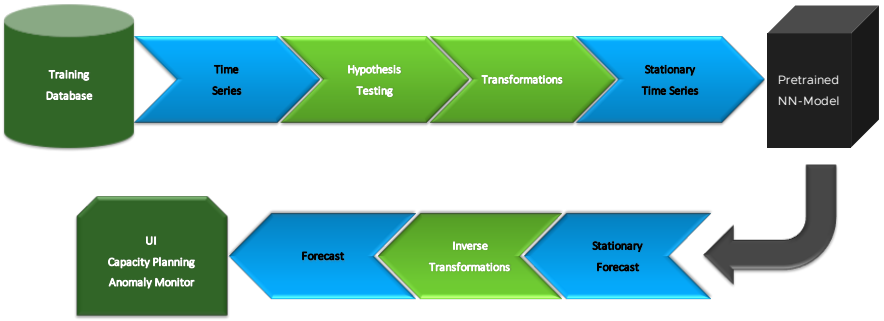

Figure 3. On-line forecast for a user specified time series data.

Application of the pretrained neural networks to a user specified time series data will similarly pass through the hypothesis testing engine for data categorization (see Figure 3). The model will be

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

straightforwardly applied to a stationary data. Non-stationary time series should be transformed into a stationary one, and then the corresponding inverse transformations should be applied to the forecast for recovering the original scale and behavior.

The next challenge is limited number of predefined input/output nodes of the network models which means limited number of history and forecast data points included in the process of a prediction. In the current implementation, we use networks with 40 inputs and 20 outputs. It means utilization of 40 history points to get 20 forecast values which restricts the ability of models to utilize bigger number of points even when they are available. Figure 4 shows the sparse grid as red dots selected from the entire history windows consisting of black and red dots. As a consequence, the forecast grid will be sparse.

Our idea is in mobilization of the entire available data points in a history window. NN-models require uniformly sampled history points. The forecast points will appear with the same monitoring interval. We take N uniformly sampled history points multiple to the size of a network input. In a specific implementation, it should be multiple to 40 and $N = k * 40$ where $k = 1, 2, \cdots$ can be selected from the complexity considerations. This is the entire history window (see the full grid in Figure 4). We divide the full grid into k different uniformly sampled sparse grids containing 40 data points and all sub-grids with the same monitoring interval. All those sparse grids can be fed into the network model, get the corresponding forecast values (20 points in each) and by gathering them together derive the forecast window with $k * 20$ points.

Worth noting that the hypothesis testing should be applied to the entire full grid to have enough data points for understanding the behavior of a time series, then the forecast should be applied to each sparse grid.

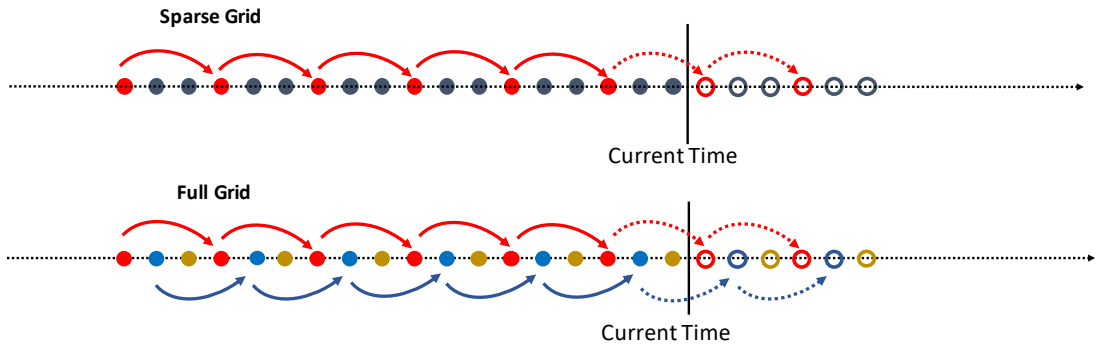

Figure 4. The construction of the full grid for the forecast models with a comb-like selection approach.

4. Hypothesis Testing for Data Categorization

In this paper, we restrict ourselves by some specific data categories that contain deterministic and stochastic trends, deterministic and stochastic periodicities. In all those cases, we are aware how to transform a non-stationary data into a stationary one with further application of NN-models. The list of categories can be enlarged if the corresponding transformations are available. It should be reasonable to add more domain specific data categories based on some domain expertise. Flowchart of Figure 5 illustrates the workflow of data categorization engine.

The engine starts with the periodicity analysis. It tests for three data categories - stationary periodic, trendy periodic and stochastic periodic. The PDM test inspects for the first two categories and CH test is for the last one. PDM test runs across different lags and measures the importance. The value of the importance above a threshold is the clear evidence of the deterministic periodicity with the corresponding lag. We call it as stationary periodic time series. PDM examines data for trendy periodicity if all previous importances were below the threshold. It removes a possible deterministic trend via linear regression and test for the period once more. The positive result identifies trendy periodic time series. CH test runs for the stochastic periodicity if both tests fail to categorize data. Time

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

series is non-periodic if all periodicity tests fail. A non-periodic time series data should be scanned for a trend. Combination of KPSS and ADF tests will classify data as stationary or trend stationary (trend is deterministic) or unit root time series (trend is stochastic). Data type is unknown if all tests fail. We don't utilize unknown types.

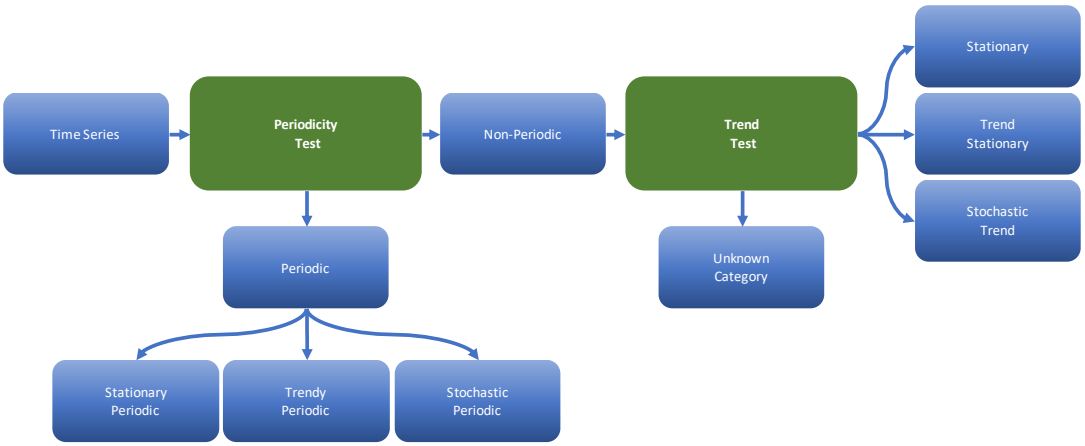

Figure 5. Data categorization engine.

The engine starts with the periodicity analysis. It tests for three data categories - stationary periodic, trendy periodic and stochastic periodic. The PDM test inspects for the first two categories and CH test is for the last one. PDM test runs across different lags and measures the importance. The value of the importance above a threshold is the clear evidence of the deterministic periodicity with the corresponding lag. We call it as stationary periodic time series. PDM examines data for trendy periodicity if all previous importances were below the threshold. It removes a possible deterministic trend via linear regression and test for the period once more. The positive result identifies trendy periodic time series. CH test runs for the stochastic periodicity if both tests fail to categorize data. Time series is non-periodic if all periodicity tests fail. A non-periodic time series data should be scanned for a trend. Combination of KPSS and ADF tests will classify data as stationary or trend stationary (trend is deterministic) or unit root time series (trend is stochastic). Data type is unknown if all tests fail. We don't utilize unknown types.

4.1. Periodicity Detection

We restrict ourselves by stationary periodic and trendy periodic categories. Stochastic periodic time series will be discussed elsewhere although the idea of the NN-model application is identical.

Let us start with period-lag determination. Let y_t, $t = 1, \ldots, T$ be the observed time series data with ℓ_0 period-lag. We define it as the equality

$$y_{n+k*\ell_0} = y_n, \; k = 0, 1, 2, \ldots \tag{1}$$

In reality, we can only expect approximate equality due to noise and instability in a time series

$$y_{n+k*\ell_0} \approx y_n, \; k = 0, 1, 2, \ldots \tag{2}$$

We follow [42–44]. The idea is very simple and Figure 6 illustrates it. It shows pure periodic time series data with $\ell_0 = 37$. We consider two different subsequences uniformly sampled from the data. The sampling rate of the first subsequence is matching the true period-lag ℓ_0 (see the top chart). It is a constant subsequence with zero variance due to the periodicity. The sampling rate of the second subsequence doesn't match the period-lag. The variance of the second subsequence is almost identical to the variance of the entire time series.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

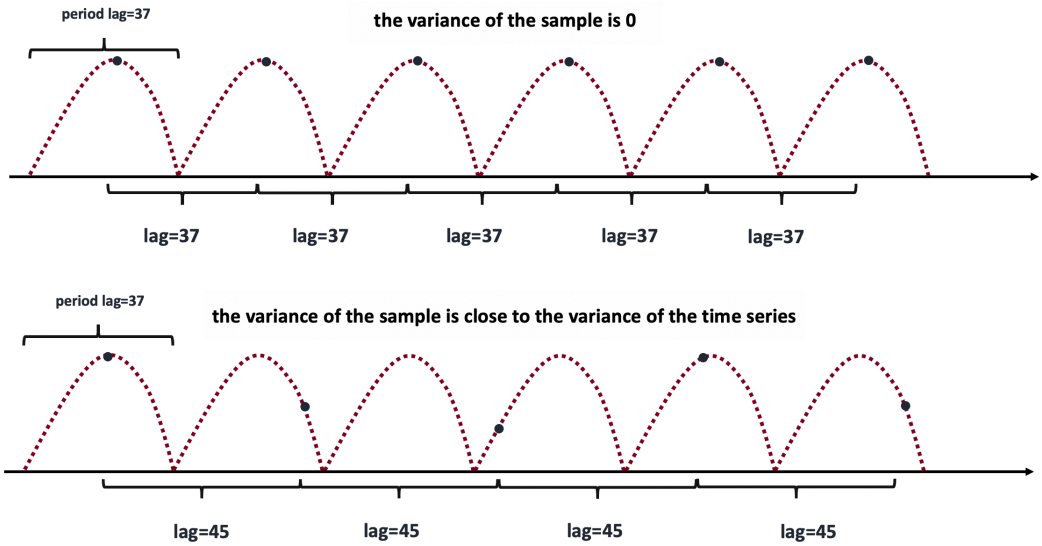

Figure 6. An example illustrating the idea of the PDM method.

More precisely, let time series variance be

$$\sigma^2 = \frac{1}{N-1}\sum_{i=1}^{N}(y_i - \bar{y})^2, \tag{3}$$

where $\bar{y}$ is the average. Assume M distinct samples collected from the time series containing n_j data points with variances s_j^2, $j = 1, \ldots, M$. Note, that all M samples must be collected with uniform and fixed lag. We denote by $s(lag)$ the average variance of the samples as follows

$$s(lag) = \frac{\sum_{j=1}^{M}(n_j - 1)s_j}{\sum_{j=1}^{M}(n_j - M)}. \tag{4}$$

The preliminary goal is minimization of $s(lag)$ via lag selection. Let us reformulate the problem that allows more efficient implementation. We define the phase of each data point y_i at the time stamp t_i by the following expression

$$\Phi_i = \frac{t_i}{lag} - \left[\frac{t_i}{lag}\right], \ \Phi_i \in [0,1), \tag{5}$$

where $[\cdot]$ stands for the integer part. If data points are sampled regularly, then $t_i = i$, $i = 1, 2, \ldots$. In order to detect data points with similar phases, we divide the full phase interval $(0,1)$ into fixed bins (20 in our experiments) and pick M samples from time series from the same bin. Consider the following statistic

$$\theta(lag) = \frac{s(lag)}{\sigma^2}. \tag{6}$$

If $lag \neq \ell_0$

$$s(lag) \approx \sigma^2 \text{ and } \theta(lag) \approx 1. \tag{7}$$

Otherwise, if $lag = \ell_0$, statistic θ will reach a local minimum compared with neighboring periods, hopefully near zero

$$\theta(lag) \approx 0. \tag{8}$$

We define "importance" of each lag as

$$importance(lag) = 1 - \theta(lag). \tag{9}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

Time series called to be periodic if one of the local maximums of the $importance(lag)$ is greater than 0.6. Period-lag can be identified as solution of the following optimization problem

$$\begin{cases} \ell_0 = \operatorname{argmin} s(lag) \\ importance(\ell_0) > threshold \end{cases} \tag{10}$$

PDM method has another interpretation connected with time series decompositions. Assume the following additive decomposition of a time series depending on a lag (details see in [16])

$$Time\ Series = Seasonal\ Component(lag) + Residual\ Time\ Series(lag). \tag{11}$$

Then, the strength of the seasonal component corresponding to a specified lag can be measured by the following fraction of variances

$$\frac{variance(Residual(lag))}{variance(Time\ Series)} \tag{12}$$

with the corresponding importance of the lag

$$importance(lag) = 1 - \frac{variance(Residual(lag))}{variance(Time\ Series)} \tag{13}$$

which exactly coincides with the importance defined via PDM method.

Figure 7 illustrates the process of identification of a trendy periodic time series. The left chart shows almost periodic time series with a slight trend. The red line indicates the trend. The right charts show the behavior of importance versus lag (like periodograms in the Fourier analysis). The second chart corresponds to the original time series. The importance of the largest local maximum is far below the threshold (0.6) and there is no evidence of a period. The third chart corresponds to the detrended time series via linear regression. The first local maximum ($\ell_0 = 19$) has the importance above the threshold. The time series can be categorized as trendy periodic with 19 period-lag.

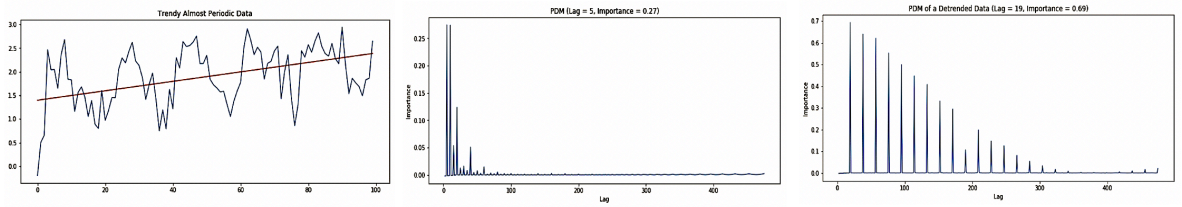

Figure 7. Categorization of a trendy periodic time series.

4.2. Trend Detection

Data categorization engine exploits three tests named as $KPSS_c$, $KPSS_{ct}$ and ADF_c for trend type detection. They test a time series for three possible categories - stationary, trend stationary and unit-root process. We follow [37–39,59].

Let y_t, $t = 1, \ldots, T$ be the observed time series data. We consider the following decomposition of the time series into the sum of a deterministic trend, a random walk, and a stationary error

$$y_t = \xi t + r_t + \varepsilon_t, \tag{14}$$

where r_t is the random walk

$$r_t = r_{t-1} + u_t \tag{15}$$

and u_t are $iid(0, \sigma_u^2)$. The initial value $r_0 = c$ is fixed and serves as an intercept. Under the null hypothesis y_t is trend-stationary if $\sigma_u^2 = 0$. We call this test as $KPSS_{ct}$. In a special case, $\xi = 0$, in which

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

case under the null hypothesis y_t is stationary around a level c. We call the test as $KPSS_c$. Thus, we consider the following two hypothesis testing scenarios:

$$\begin{gathered}\textbf{KPSS}_{\mathbf{c}}\textbf{ test}: \\ -\ Null\ Hypothesis: level\ stationarity \\ -\ Alternative\ Hypothesis: unit\ root\ process\end{gathered} \tag{16}$$

and

$$\begin{gathered}\textbf{KPSS}_{\mathbf{ct}}\textbf{ test}: \\ -\ Null\ Hypothesis: trend\ stationarity \\ -\ Alternative\ Hypothesis: unit\ root\ process\end{gathered} \tag{17}$$

Tests use ordinary least squares (OLS) to find the coefficients and the corresponding residuals e_t, $t = 1, 2.., T$. They apply one-sided LM statistic for the hypothesis $\sigma_u^2 = 0$ defined as follows

$$S_t = \sum_{i=1}^{t} e_i,\ t = 1, 2, \ldots, T \tag{18}$$

and

$$LM = \sum_{i=1}^{T} \frac{S_i^2}{\hat{\sigma}_\varepsilon^2}, \tag{19}$$

where $\hat{\sigma}_\varepsilon^2$ is the estimate of variance of ε_t. The test is an upper tail test. The corresponding p-values (P_v) can be found in [37].

ADF test uses data model

$$\Delta y_t = y_t - y_{t-1} = c + \alpha_0 y_{t-1} + \sum_{s=1}^{p} \alpha_s \Delta y_{t-s} + \varepsilon_t, \tag{20}$$

where c is the level, ε_t is a stationary process, p is the number of lags used in the model. The value $p = 0$ corresponds to DF test. Test uses OLS to find the coefficients and applies Akaike information criterion for automatic lag selection. The unit root test is then carried out under the null hypothesis $\alpha_0 = 0$. Alternatively, test checks the condition $\alpha_0 < 0$ that corresponds to a stationary process. We call this test as ADF_c:

$$\begin{gathered}\textbf{ADF}_{\mathbf{c}}: \\ -Null\ Hypothesis: unit\ root\ process \\ -Alternative\ Hypothesis: stationary\ process\end{gathered} \tag{21}$$

The value $c = 0$ corresponds to a unit root process without a drift. The ADF_c test calculates the following statistics

$$DF = \frac{\alpha_0}{SE(\alpha_0)} \tag{22}$$

and compare with the critical values known from the DF test to calculate the corresponding p-values (P_v).

Flowchart in Figure 8 illustrates the workflow for the trend and stationarity testing. It shows the priority of test applications. We sequentially apply $KPSS_c$, $KPSS_{ct}$ and ADF_c and inspect the p-values of the tests. If the corresponding $P_v \geq 0.05$, we stop the procedure and categorize data accordingly. The data type is unknown if all tests fail.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

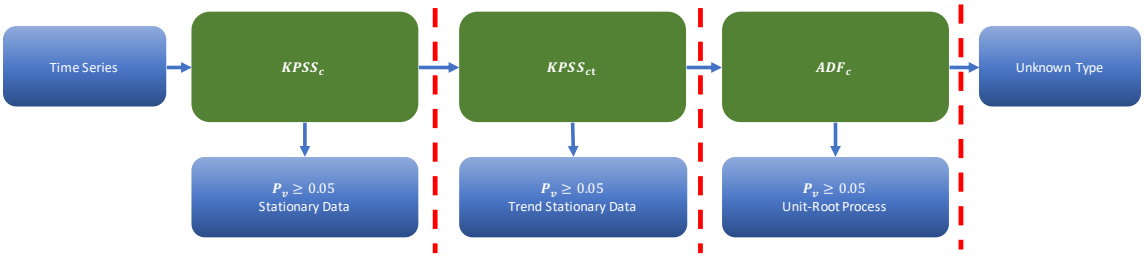

Figure 8. The priority order for a trend detection.

Figure 9 shows how the flow of Figure 8 works for specific examples. We present the corresponding p-values of the tests for inspection. We use in our experiments implementation of $KPSS_c$, $KPSS_{ct}$ and ADF_c from Python module "statsmodels". The p-value of the $KPSS_c$ test for the first example is larger than 0.05. Although, the p-value of the $KPSS_{ct}$ is also bigger than 0.05, the priority order categorizes data as the stationary. By the way, in the production, we don't need the second p-value if the first one is already the winner. The p-value of the $KPSS_c$ for the second example is smaller than 0.05 and the categorization engine tries the next one. The p-value of $KPPS_{ct}$ test equals to 0.05. We accept the null hypothesis and categorize data as trend stationary. However, interesting to see that the p-value of the ADF_c test is also very big. It means that data can be categorized either as trend stationary or as a unit root process, but the priority order (actually based on the simplicity of a category) selects the trend stationarity as the winner. For the third example, the p-values of $KPSS_c$ and $KPSS_{ct}$ tests are smaller than 0.05 while for ADF_c it is bigger. The last example belongs to a unit root process.

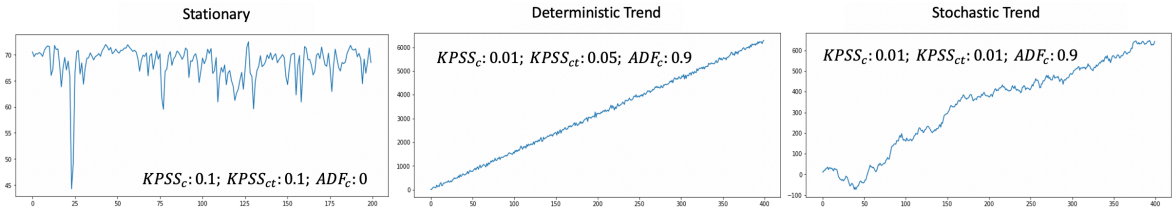

Figure 9. Trend identification by the data categorization engine.

5. Time Series Forecasts with Confidence Bounds

In this section, we discuss application of the pretrained NN-models to time series forecasting in the on-line mode. The entire engine was described in Figure 3. Recall that NN-models were only pretrained for stationary time series. A non-stationary time series should be converted into a stationary one and the corresponding forecast should be reconverted by backward transformations. We restricted ourselves by five specific data types: stationary, trend stationary (deterministic trend), unit root process (stochastic trend), stationary periodic and trendy periodic. More extended version of data categorization will be considered elsewhere.

5.1. Stationary Time Series

We treat with this class without any forward and backward transformations. We only perform scaling into the interval $[0, 1]$ as the standard procedure for NN-models. We apply the following transformation

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}}, \tag{23}$$

where X is the original time series and X_{min}, X_{max} are its minimum and maximum values, respectively. We apply reverse scaling to the corresponding forecast by multiplying with $(X_{max} - X_{min})$ and summing by X_{min}.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

The confidence bounds of the forecasts are one of the important pieces of any approach. Ideally, we need rather long historical data with different forecasts that will help to extract the bounds based on some criticality levels (say 95%). Unfortunately, we have only one history window with the corresponding forecast window and forced to extrapolate that available information to future data points without strong evidence.

Assume y_k, $k = 1, \ldots, T$ be the observed data points from a stationary time series, $\hat{y}_k$, $k = T+1, \ldots, T+M$ be the corresponding forecast values and $\mu = average\{y_k\}$. Let y_k^{high}, $k = 1, \ldots, N_{high}$ and y_k^{low}, $k = 1, \ldots, N_{low}$ be history window data points that are bigger or equal and lower or equal than μ, respectively. Then, we calculate higher and lower standard deviations as follows

$$\sigma_{high} = \left(\frac{1}{N_{high} - 1} \sum_{k=1}^{N_{high}} \left(y_k^{high} - \mu \right)^2 \right)^{\frac{1}{2}}, \tag{24}$$

and

$$\sigma_{low} = \left(\frac{1}{N_{low} - 1} \sum_{k=1}^{N_{low}} \left(y_k^{low} - \mu \right)^2 \right)^{\frac{1}{2}}, \tag{25}$$

respectively. We define upper and lower bounds (UB and LB as confidence bound vectors) for each of the point in the forecast window as follows

$$UB_k = \hat{y}_k + z * \sigma_{high}, \; LB_k = \hat{y}_k - z * \sigma_{low}, \tag{26}$$

respectively, where parameter z stands for the criticality levels $z = 1, 2, 3, \cdots$.

Confidence bounds for any data category can be found similarly. We perform forward transformations for converting a time series into a stationary, calculate the corresponding forecast with the confidence bounds and apply backward transformations.

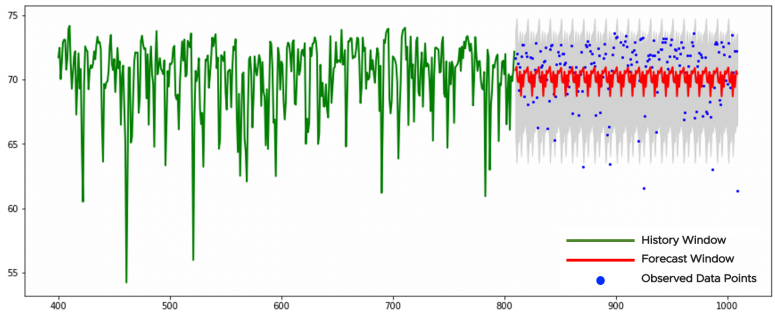

Figure 10. The forecast of a stationary data.

Figure 10 illustrates an example of a forecast for a stationary time series. History window contains 400 points (full grid). It consists of 10 sparse grids with 40 points that we used for independent forecasts. They provide with 10 forecasts with 20 points in each. We collect those forecasts together and get 200 points in the forecast window. The forecast is smoother (less variable) than the really observed data points which is common for the NN-based models.

5.2. Trend Stationary Time Series

This class contains time series data with deterministic linear trend that can be removed via ordinary least squares (linear regression). Linear regression identifies the corresponding slope (k) and intercept (b). We remove the trend $(kt + b)$ and apply the pretrained NN-model to the resulting stationary data, get the forecast and recover the trend by the same slope and intercept. Figure 11 shows an example for a trend stationary time series.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

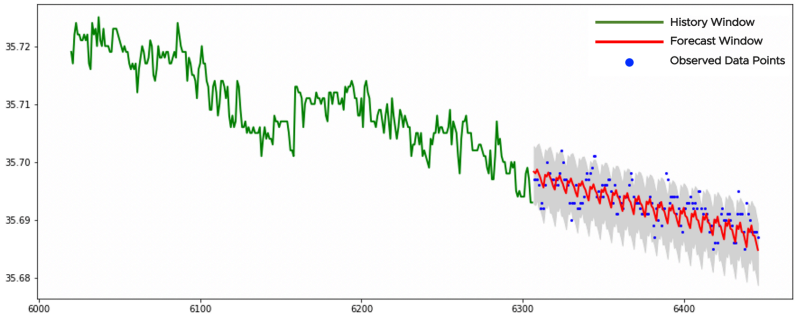

Figure 11. The forecast of a trend stationary time series.

5.3. Unit Root Processes

We apply differencing to the time series of this type and convert them to a stationary process. After application of the pretrained NN-model, we apply backward transformation (cumulative sums) and restore the original trend and scale. Figure 12 shows an example for a trend stationary time series.

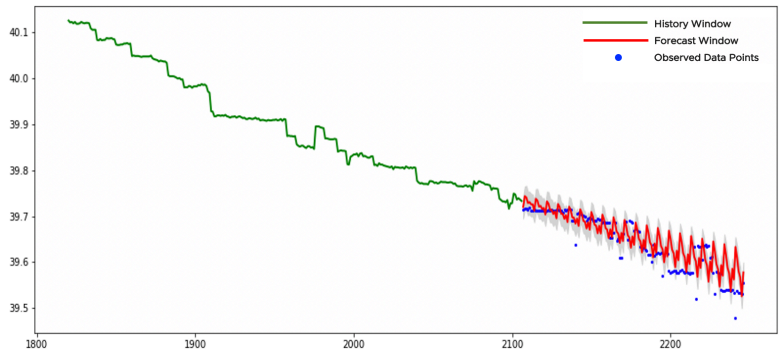

Figure 12. The forecast of a unit root process.

5.4. Stationary Periodic Time Series

We assume that for any time series from this class, the period-lag ℓ_0 is known. We consider two different approaches for the corresponding forecasts. The first approach is connected with the grid construction described above. History window consists of data points multiple to the size of the input of the NN-model. We already mentioned that the current model has 40 inputs and the history window will have $40 * k$ data points. Hence, we have k different sub-grids for separate forecasts. The idea of the first approach is to take $k = \ell_0$, get k different stationary (almost constant) data samples due to the periodicity and directly apply NN-model to each sub-grid without any forward-backward transformations. Figure 13 shows an example of a stationary periodic time series with $\ell_0 = 19$. We show also data points of the history window for better visual perception.

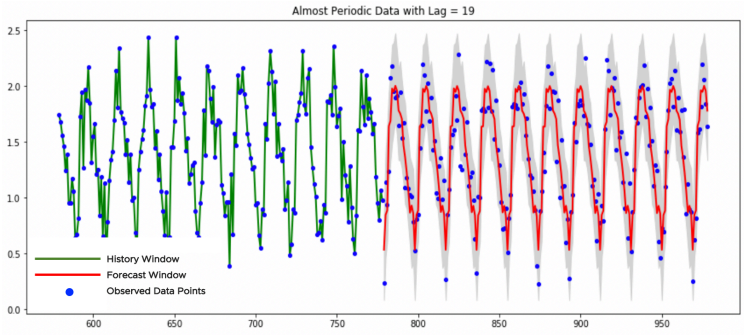

Figure 13. The forecast of a stationary periodic data.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

The mentioned approach will cause problems if the process of grid extraction is connected with time consuming database calls and the period-lag is unknown for a time series. In that cases, it is impossible to detect the period-lag with further resampling of the time series. In such situations, we can apply different deseasoning approaches like seasonal means. Hence, we can subtract the periodicity, forecast the corresponding stationary time series and return the seasonal-means.

5.5. Trendy Periodic Time Series

The procedure is identical to the previous one as after removal of a deterministic trend by the linear regression, we get a stationary time series. At the end, we just return the trend back.

Figure 14 illustrates the forecast for a trendy periodic time series with application of the seasonal-means approach.

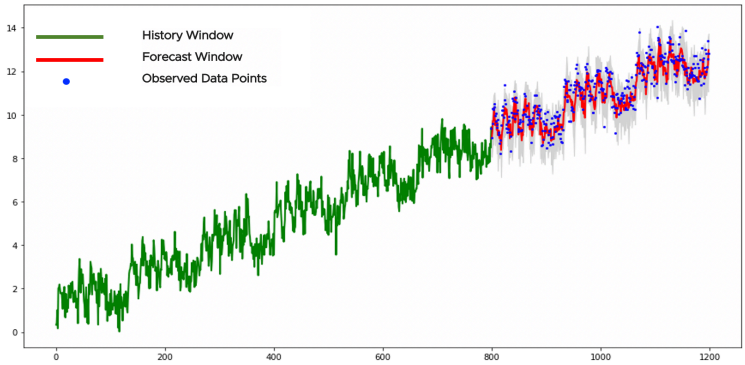

Figure 14. The forecast of a trendy periodic data.

6. Anomaly Signals From Time Series Data Forecasts

In this section, we discuss an approach to anomaly signal time series generation based on the confidence bounds of the forecasts. Each datapoint in the anomaly signal shows the percentage (fraction) of observed data points in a test window that violate upper and/or lower confidence bounds. The anomaly signal may detect or predict anomalous conditions whenever its values exceed a particular threshold. In such situations, an anomaly monitor will generate alarms indicating some behavioral changes in a specified time series. We consider details for NN-based forecasting methods described in the previous sections, although the approach is applicable to any predictive model.

One of the principle problems in time series data anomaly/outlier detection is setting of the proper trade-off between the timeliness and confidence of the detections. From the one side, the alarms should be detected as faster as possible for preventive actions before the alarms will impact customers' environments. From the other side, the big number of false positive alarms overwhelms system administrators and decreases the confidence towards the anomaly detection system. The trade-off may be resolved by the proper selection of underlying parameters for the anomaly signal generation.

An initial indication that the state of a monitored system has begun to change in an unexpected fashion is that an observed data point diverges from its forecast. However, no one is expecting that this single indication will be used as a detectable signal due to a significant noise in time series data and its non-deterministic nature which makes very accurate predictions impossible. Another indication can be violation of a confidence bound by an observed data point. Nevertheless, no one will pay attention to that signal if the subsequent observed time series data are within the bounds or even close to the predicted values. The violation may possibly be an outlier due to noise or some sudden instability rather than an indication of a serious malfunctioning of a system. It is likely that many false-positive alarms will appear if alarms and warnings will be generated based on single-data-point or short-term departures of observed time series data values from the forecast ones. However, by waiting until a pattern of detected preliminary behavioral change will emerge, the problem may have already cascaded to a point when proactive actions can no longer be possible due to some catastrophic

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

impacts on the system. The period of time between the initial indication of an anomaly and the onset of serious degradation depends on the nature of time series and the process that it describes.

Figures 15 and 16 illustrate the basis of the solution to the mentioned problems. Recall that there are three forecast time series data that should be used for anomaly signal generation. The first one is the predicted time series (forecast window) and the next two ones are predicted upper and lower bounds. We are not showing the last two time series data in the figures for the simplicity but the term violation always refers to the bounds. Also, we refer to a history window which contains time series data points from which the mentioned forecasts were calculated. Moreover, observation window contains actually monitored time series data points. It is assumed that the history and forecast windows contain uniformly sampled time series data points with the same monitoring intervals.

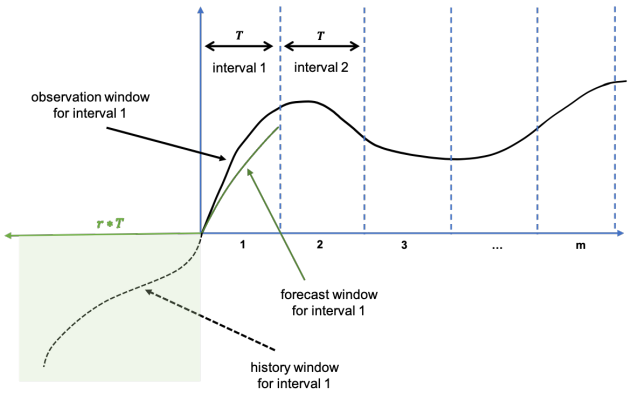

Figure 15. The hidden background of an anomaly monitor.

Figure 15 represents the hidden background of an anomaly monitor while following a specific time series data. The monitoring will be started by indication of the length of the forecast window. We describe below the process of parameter selection in more details. Now, we assume that the forecast window contains T uniformly sampled data points. To be more precise, parameter T must be a multiple of the size of the output of the pretrained NN-model. Moreover, in the previous sections it was indicated the strict connection between the sizes of a history window and the corresponding forecast window.

The current pretrained network uses 40 historical points to predict 20 future points. It means that the history window is twice longer than the forecast window for the current model. For generality, assume that a history window is r times longer than a forecast window (see Figure 15). The user will not see the history window. His UI chart will contain several forecast windows, as much as possible to fit. We show m such intervals in Figure 15. The engine will calculate the forecast for the first window and the corresponding anomaly signal will be estimated for all observed data points. Then, the engine will repeat the process for the other forecast windows by shifting the history window to the right by T data points until it will reach the final forecast window. There are different reasons why we didn't calculate a unique forecast for the entire UI chart (for $m \cdot T$ data points). The first reason is the complexity of data preprocessing. If a user opens rather big UI chart (T is big) than the forecast engine will fail to process $r \cdot T$ data points. The second reason is the desire of immediate incorporation of the latest observed data points into the process of anomaly signal generation.

Figure 16 shows the process of calculation of the anomaly signal for each of the forecast windows. Moreover, the anomaly score must be assigned to each data point in a forecast window. As the entire forecast window can be rather large, and by recalling the requirement for the timeliness, we incorporate a test window (smaller or equal to the forecast window) for the percentage calculation (see the "blue" rectangles in Figure 16) for faster reaction to possible anomalies. To each just observed time series data point a test window should be assigned extending to the left by the time axis where the point of interest is the last point of the window. The percentage of violations in the test window is the anomaly

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

score of that last point. Then, the anomaly monitor can visualize the anomaly signal or trigger an alarm based on some threshold value (say 0.8).

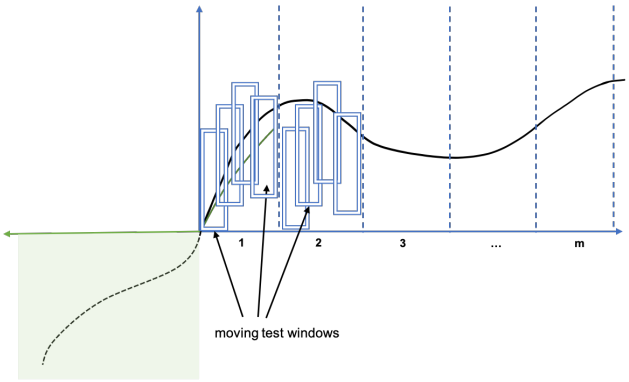

Figure 16. Utilization of test windows for the anomaly signal calculation.

Figures 17, 18 and 19 show some specific time series data with the corresponding anomaly signals. "Blue" curves correspond to time series data and "red" ones to the anomaly signals. The values of time series data are shown on the left y-axes, and the values of the anomaly scores on the right. Anomaly scores take values from interval $[0, 1]$. Value 0 means that all observed data points in the test window arrived within the confidence bounds. Value 1 means that all observed data points in the test window violated the confidence bounds.

Let us explain some peculiarities regarding time series visualization in Wavefront. Figures 17, 18 and 19 refer to the Wavefront UI. The UI can't handle all time series data points available in a database and it applies a method known as summarizing. The figures show that in the current situation the summarizing function uses averaging of data points within a bucket with 7200 seconds duration. However, NN-model utilizes totally different data points derived from the database via interpolation for uniform sampling. Unfortunately, it means that the actual time series data utilized by the NN-model is not the one that we see in the UI. This was one of the big challenges for the current implementation as the situation should be explained to our product users for increasing the confidence towards the forecasts and anomaly detection visualization.

Figure 17 shows the example of a stationary time series data without visible outliers/anomalies and the corresponding anomaly score is almost flat near the zero value. Small fluctuations in the anomaly score are outcomes of random outliers that go out of confidence bounds but are not visible due to Wavefront data summarization procedure. Figure 18 shows piecewise constant data with two change points. In both cases the anomaly score detect the behavioral changes with the values bigger than 0.8 (the threshold for an alarm generation). It is important that the jumps in the anomaly scores ideally coincide with the jumps in time series data. Figure 19 shows almost constant time series data with some spikes. The behavior of the anomaly signal mimics those spikes. In two cases, the scores became bigger than 0.8, so alarms should be announced. In other cases, the changes and spikes should be ignored.

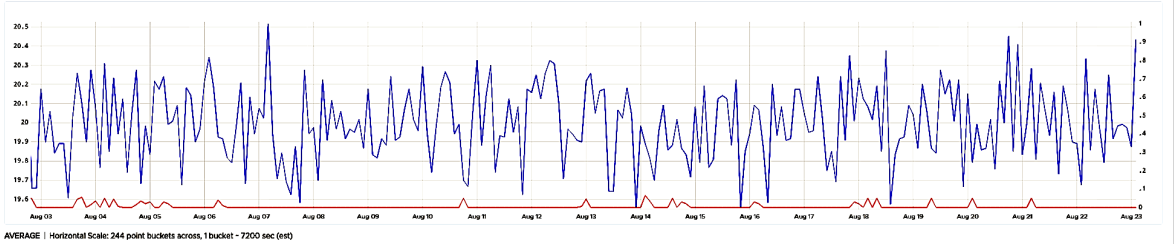

Figure 17. Example of a stationary time series data with the anomaly signal.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

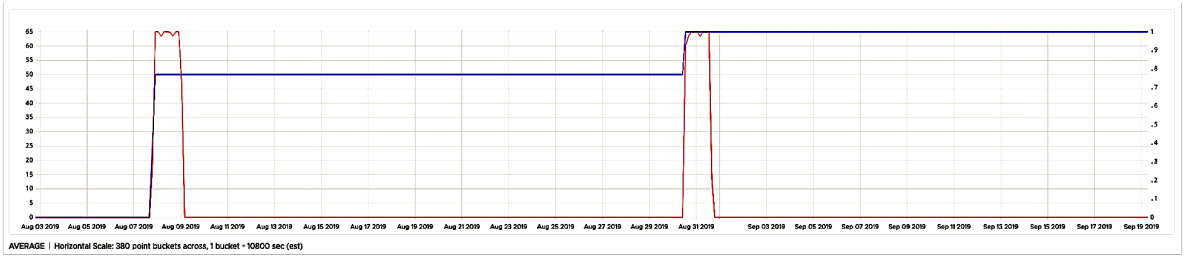

Figure 18. Example of a piecewise-constant time series data with the anomaly signal.

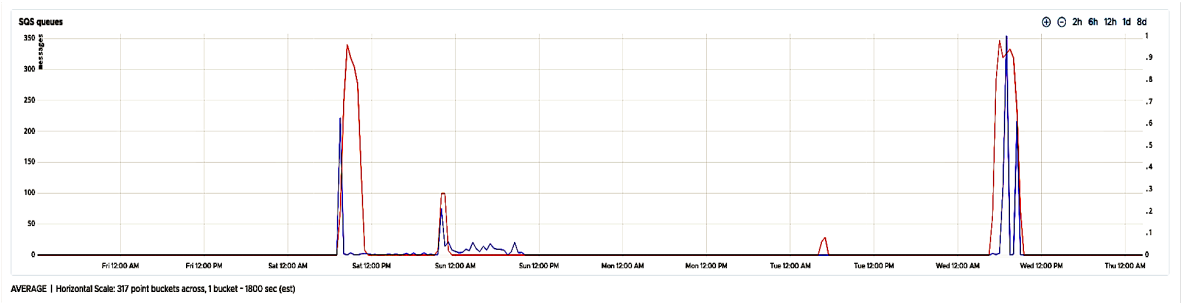

Figure 19. Example of a constant time series data with random spikes with the anomaly signal.

The biggest problem that the Wavefront customers encountered while consuming the described system for anomaly detection was the large number of false positive alarms. Our experience shows that the customers agree with the reduction of false positives even at the expense of the rising number of false negatives. The common approach to reduction of false positives is through smoothing methods. Paper [60] describes such a kernel-smoothing simple procedure. The kernel smoothing can be applied both to time series data and/or anomaly scores. It actually performs a weighted averaging of data points or anomaly scores where the weights can be extracted via some kernel function. The Gaussian kernel is the most common kernel

$$K_h(x,y) = exp\left(-\alpha \frac{||x-y||^2}{h}\right), \tag{27}$$

where h is the width (window) of the kernel, $\alpha > 0$ is some sensitivity parameter and $\|\cdot\|$ stands for the Euclidian distance. Assume that x_i are time series data points and s_i are the corresponding anomaly scores. A new anomaly score $\hat{s}_i$ ([60]) is estimated as follows (the weighted mean of anomaly scores detected before the current time)

$$\hat{s}_i = \frac{\sum_{j=i-n}^{i} K_h(x_j, x_i) s_j}{\sum_{j=i-n}^{i} K_h(x_j, x_i)},$$

where n is the number of points within the window h. We can set h to be equal to the test window mentioned above. It is possible to calculate two-sided averages if time allows us to wait for new data points to arrive. Similarly, instead of the anomaly scores, we can smooth time series data points. Let $\hat{x}_i$ be the estimate:

$$\hat{x}_i = \frac{\sum_{j=i-n}^{i} K_h(x_j, x_i) x_j}{\sum_{j=i-n}^{i} K_h(x_j, x_i)}.$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

Then a new anomaly score estimate $\hat{s}_i$ based on $\hat{x}_i$ can be calculated. Experiments showed that the first approach is preferable. However, more experiments should be performed for the final decision. Probably, time series category (semi-constant, trendy, etc.) should be important for the approach.

7. Materials and Methods

In this section, we introduce the NN-model training process in the off-line mode. The trainings were performed in VMware private data centers equipped with powerful GPUs. However, our experimental training database is not big. It includes around 3300 time series, taken from real customer cloud environments. The database contains around 1500-"cpu" , 400-"disk", 110-"IOps", 320-"memory", 450-"network bandwidth", 100-"network packets" and 410-"workload" metrics. Metrics in the database have 1-minute monitoring interval and, in average, 1-month duration. It doesn't contain any specific information crucial for model trainings and similar results should be possible to get via other datasets of time series. Moreover, interesting should be application of synthetic datasets of time series.

The current network has 40 inputs and 20 outputs. We experimented with different dimensions without any serious difference. We noticed that longer input compared to the output result in better forecast accuracy. Taking into account the grid structure, we can utilize $40 * k$ data points in history window to estimate $20 * k$ points in the forecast window with $k = 1, 2, \cdots$. We applied a sliding window containing 600 points to each time series. The sliding window had $400 = 10 * 40$ history points and $200 = 10 * 20$ forecast points. We performed hypothesis testing to the entire sliding window, identified needed transformations, and applied those transformations to each sub-grid containing 60 training data points (presumably stationary) for the network input (40 points) and output (20 points).

We tried different network architectures. The first was LSTM networks with stateless configuration, 2 hidden layers with 256 nodes in each. The next was MLP networks with identical configuration. We didn't find significant differences between LSTM and MLP networks for our small dataset. The current model is MLP network which is very easy to implement without special libraries. We used 'relu' activation function for the hidden layers and "linear" activation for the output layer. 'Adam' optimizer and mean average error ('mae') as a loss function were used. We applied 5 epochs for each time series and 20 epochs for the entire database and $batchsize = 1500$. The idea was in getting a generic model for the entire database. The trainings took from several hours to a day depending on the available GPUs.

8. Discussion

We tried different implementations of the on-line mode in Java as enterprise cloud service. The first attempt was utilization of Deep Learning for Java (DL4J) library [61]. It caused some problems due to bigger memory consumption and longer response time. The second attempt was total independent implementation of the MLP network without external libraries. The latest approach is more reasonable as on-line mode doesn't require on-demand trainings and complete deep learning functionality of DL4J is wasteless. Figure 20 shows comparison of timings for both implementations while forecasting a stationary time series. The y-axis shows the timings in milliseconds. The x-axis illustrates different runs for averaging purposes. We see that "DL4J" is far behind compared to "MLP" especially while loading the library.

We performed some comparisons of the current model and classical ARIMA (our internal implementation without the periodicity analysis for both approaches). We applied both models to a database of time series data from our internal cloud environments with different history windows sliding across the time axis. We experimented with 120 points (2 hours), 1440 points (1 day), 11520 points (1 week) and 30240 points (2 weeks). We calculated the corresponding root mean square relative errors (RMSRE) for each forecast. Table of Figure 21 summarizes the results. It shows overall 279148 forecast cases. Each column shows the number and percentage of the forecasts for which the corresponding RMSRE is smaller than the mentioned value $0.5, 1, 2, 5$ and 10. For example, the last column of the table shows that 126682 forecasts via NN-model or 45% of all cases have errors smaller

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 18 January 2021 doi:10.20944/preprints202101.0326.v1

than 0.5 while for the ARIMA the same number is bigger by 1%. The difference is insignificant. In average, both methods perform similar, although ARIMA is slightly better as it was expected.

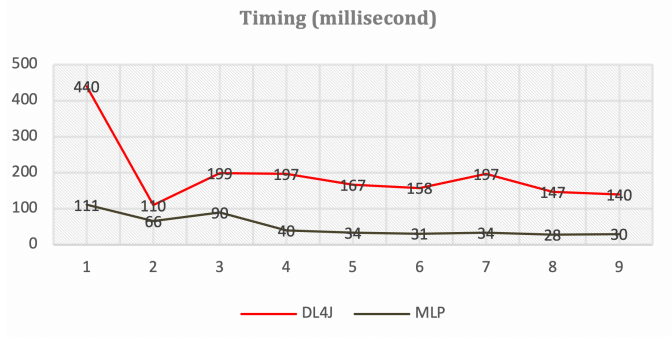

Figure 20. Comparison of different implementations.

It should be interesting to compare the average errors across all metrics from the same class. For example, for the class of stationary metrics, NN-model shows an average error 1.2 while ARIMA shows 1.3. For the class of trend stationary time series, NN-model has an average error 1.57 while ARIMA has 1.52.

Models	Total Forecasts	RMSRE < 10	RMSRE < 5	RMSRE < 2	RMSRE < 1	RMSRE < 0.5
NN	279148	268924 (96%)	262833 (94%)	243169 (87%)	144684 (52%)	126682 (45%)
ARIMA	279148	270267 (97%)	264994 (95%)	246483 (88%)	166155 (60%)	127776 (46%)

Figure 21. Comparison of root mean square relative errors for different models.

Figure 22 represents an example from the Wavefront AI Genie UI [62].

Figure 22. An example of a trend stationary time series in the Wavefront AI Genie UI.

It illustrates the on-line mode for a specific time series data. AI Genie UI of Wavefront simplifies and automates time series forecasting and anomaly detection capabilities. It requires minimal set of parameters to start running the AI engine. A user can specify (or use defaults) a time series, select the forecast period, and the corresponding sensitivity of the confidence bounds. In Figure 22, the red curve corresponds to the historical data, the black curve to the forecast, and the green area to the confidence bounds. The forecast window is 1 week. It means that the history window is 2 weeks as the pretrained network works with 2:1 ratio. Confidence bounds correspond to “moderate” setting (the others are “conservative” and “aggressive”).

The current NN-model uses 4000 data points (uniformly sampled via interpolation) for 2 weeks history, 8640 points for 2 months history and 12960 points for 6 months. Those selections are the

 doi:10.20944/preprints202101.0326.v1

trade-offs between the complexity and grid density. We think that those numbers can be reduced without affecting the accuracy especially for some data categories.

9. Conclusions and Future Work

We considered application of NN-based models to time series forecasting and anomaly detection in cloud applications. Throughout the paper, we discussed approaches for overcoming some of the challenges.

The first and main challenge is restrictions on resource consumption in distributed cloud environments. Neural networks require intensive GPU utilization and sufficient data volume which make on-demand training and application of NN-based models unrealistic due to additional costs and unacceptable response times. We proposed a solution along with the ideas of transfer learning. We generate a global database for time series data collected across different cloud environments and customers, train a model in a private GPU-accelerated data centers and apply the acquired knowledge in the form of a pretrained model to a user specified time series data never seen before without GPU utilization. The weights and configuration of the pretrained network are stored in a cloud and monitoring tools can easily access the corresponding files and retrieve the required information for on-demand application to forecasting and anomaly detection.

The second challenge is the weakness of NN-models for analyzing non-stationary time series data. It is a well-known problem and many researchers suggest application of stabilizing procedures like detrending and deseasoning before feeding the network. The stabilizing transformations convert a non-stationary time series into a stationary one, and properly trained NN-models can adequately treat those metrics. We utilize this common idea and train models only for stationary time series. We detect the stabilizing transformations via hypothesis testing. In the off-line mode, we perform hypothesis testing to all time series data within the database for finding the set of required transformations for all examples. Those transformations convert all non-stationary time series data into stationary ones before sending to a model for the training. In the on-line mode, we transform a user specified time series into a stationary data, calculate the corresponding forecast and by application of the backward transformations return to the original scale and behavior. Throughout the paper, we demonstrated the main capabilities of the approach. Moreover, the approach was implemented as a SaaS solution for Wavefront by VMware and it passed full validation in real cloud environments. Our customers mainly utilize the service for anomaly detection.

However, many questions need further investigations. One of the key problems is improvement of the current approach via different networks and configurations. The second interesting problem should be hypothesis testing via NN-based models. We already received some results with one-dimensional convolutional neural networks for data classification. It should be natural to combine both networks to automate data categorization and forecasting. Another interesting problem is designing of new models for some new classes of time series data that should improve the accuracy. We also need to check wether bigger datasets will improve the accuracy of the current models.

It is not fair to compare the proposed approach with the methods that train network models in demand for a specific time series data. Undoubtedly, the latest will be more accurate or at least comparable to our approach. Our main goal is the balance between the power and resource utilization. We aimed to develop methods for cloud environments without consumption of valuable resources and with acceptable accuracy.

10. Patents

Pang, C. Anomaly detection on time series data. Filed: Aug 22, 2018. Application No.: 16/109,324. Published: Jan 30, 2020. Publication No.: 2020/0034733A1.

Pang, C. Visualization of anomalies in time series data. Filed: Jan 30, 2020. Application No.: 16/109,364. Published: Jan 30, 2020. Publication No.: 2020/0035001A1.

Poghosyan, A.V.; Pang, C.; Harutyunyan, A.N.; Grigoryan, N.M. Processes and systems for forecasting metric data and anomaly detection in a distributed computing system. Filed: Jan 17, 2019. Application No: 16/250,831. Published: February 27, 2020. Publication No.: 2020/0065213A1.

Poghosyan, A.V.; Hovhannisyan, N.; Ghazaryan, S.; Oganesyan, G.; Pang, C.; Harutyunyan, A.N.; Grigoryan, N.M. Neural-network-based methods and systems that generate forecasts from time-series data. Filed: Jan 14, 2020. Application No.: 16/742,594.

Poghosyan, A.V.; Harutyunyan, A.N.; Grigoryan, N.M.; Pang, C.; Oganesyan, G.; Ghazaryan, S.; Hovhannisyan, N. Neural-network-based methods and systems that generate anomaly signals from forecasts in time-series data. Filed: Dec 19, 2020. Application No.: 17/128,089. This application is a continuation-in-part of US Application No. 16/742,594, filed Jan 14, 2020.

Author Contributions: All the authors participated in almost all stages of the research project, but their more detailed contribution to the project is described below. Conceptualization, A.P., A.H., N.G., C.P and G.O.; methodology, A.P., A.H., N.G., C.P, G.O. and N.H.; software, G.O., S.G. and N.H.; validation, A.P., A.H., N.G., C.P., G.O., S.G. and N.H..; formal analysis, A.P., A.H., N.G. and C.P.; investigation, A.P., A.H. and N.G.; resources, C.P. and G.O.; data curation, N.G., G.O., S.G. and N.H.; writing–original draft preparation, A.P. and A.H; writing–review and editing, A.P., A.H., N.G., C.P., G.O., S.G. and N.H.; visualization, A.P., G.O., S.G. and N.H.; supervision, A.P., G.O. and C.P.; project administration, C.P.; funding acquisition, A.P. and C.P. All authors have read and agreed to the published version of the manuscript.

Funding: Arnak Poghosyan was partially funded by RA Science Committee, in the frames of the research project 20TTAT-AIa014".

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

ADF test	Augmented Dickey-Fuller test
AI	Artificial Intelligence
AIOps	AI for IT operations
APM	Application Performance Monitoring
ARMA model	Auto Regressive Moving Average model
ARIMA model	Auto Regressive Integrated Moving Average model
CH test	Canova-Hansen test
CPU	Central Processing Unit
DF test	Dickey-Fuller test
DL4J	Deep Learning For Java
GPU	Graphical Processing Unit
HEGY test	Hylleberg-Engle-Granger-Yoo test
IT	Information Technologies
KPSS test	Kwiatkowski-Phillips-Schmidt-Shin test
LB	Lower Bound
LSTM network	Long Short Term Memory network
ML	Machine Learning
MLP network	Multi Layer Perceptron network
NN	Neural Network
OCSB test	Osborn–Chui-Smith-Birchenhall test
OLS	Ordinary Least Squares
PDM test/method	Phase Dispersion Minimization test/method
RMSRE	Root Mean Square Relative Error
SaaS	Software as a Service
SARIMA model	Seasonal ARIMA model
STL decomposition	Seasonal and Trend decomposition using Loess
UB	Upper Bound
UI	User Interface

References

1. Magic quadrant for application performance monitoring. https://www.gartner.com/doc/3983892. Accessed: 2021-01-10.
2. Poghosyan, A.V.; Harutyunyan, A.N.; Grigoryan, N.M.; Pang, C.; Oganesyan, G.; Ghazaryan, S.; Hovhannisyan, N. W-TSF: Time series forecasting with deep learning for cloud applications. Proceedings of the Second CODASSCA Workshop, Yerevan, Armenia: Collaborative Technologies and Data Science in Artificial Intelligence Applications; Hajian, A.; Baloian, N.; Inoue, T.; Luther, W., Eds.; Logos Verlag: Berlin, 2020; pp. 152–158.
3. Harutyunyan, A.N.; Poghosyan, A.V.; Grigoryan, N.M.; Hovhannisyan, N.A.; Kushmerick, N. On machine learning approaches for automated log management. *J. Univers. Comput. Sci.* **2019**, *25*, 925–945.
4. Harutyunyan, A.N.; Poghosyan, A.V.; Grigoryan, N.M.; Kushmerick, N.; Beybutyan, H. Identifying changed or sick resources from logs. 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Trento, Italy, September 3-7, 2018. IEEE, 2018, pp. 86–91.
5. Poghosyan, A.V.; Harutyunyan, A.N.; Grigoryan, N.M. Compression for time series databases using independent and principal component analysis. 2017 IEEE International Conference on Autonomic Computing, ICAC 2017, Columbus, OH, USA, July 17-21, 2017; Wang, X.; Stewart, C.; Lei, H., Eds. IEEE Computer Society, 2017, pp. 279–284.
6. Poghosyan, A.V.; Harutyunyan, A.N.; Grigoryan, N.M. Managing cloud infrastructures by a multi-layer data analytics. 2016 IEEE International Conference on Autonomic Computing, ICAC 2016, Wuerzburg, Germany, July 17-22, 2016; Kounev, S.; Giese, H.; Liu, J., Eds. IEEE Computer Society, 2016, pp. 351–356.
7. Marvasti, M.A.; Poghosyan, A.V.; Harutyunyan, A.N.; Grigoryan, N.M. Ranking and updating beliefs based on user feedback: Industrial use cases. 2015 IEEE International Conference on Autonomic Computing, Grenoble, France, July 7-10, 2015. IEEE Computer Society, 2015, pp. 227–230.
8. Marvasti, M.A.; Poghosyan, A.V.; Harutyunyan, A.N.; Grigoryan, N.M. An enterprise dynamic thresholding system. 11th International Conference on Autonomic Computing, ICAC 2014, Philadelphia, PA, USA, June 18-20, 2014; Zhu, X.; Casale, G.; Gu, X., Eds. USENIX Association, 2014, pp. 129–135.
9. Harutyunyan, A.N.; Poghosyan, A.V.; Grigoryan, N.M.; Marvasti, M.A. Abnormality analysis of streamed log data. 2014 IEEE Network Operations and Management Symposium, NOMS 2014, Krakow, Poland, May 5-9, 2014. IEEE, 2014, pp. 1–7.
10. Marvasti, M.A.; Poghosyan, A.V.; Harutyunyan, A.N.; Grigoryan, N.M. Pattern detection in unstructured data: An experience for a virtualized IT infrastructure. 2013 IFIP/IEEE International Symposium on Integrated Network Management, IM 2013, Ghent, Belgium, May 27-31, 2013; Turck, F.D.; Diao, Y.; Hong, C.S.; Medhi, D.; Sadre, R., Eds. IEEE, 2013, pp. 1048–1053.
11. Amarbayasgalan, T.; Pham, V.H.; Theera-Umpon, N.; Ryu, K.H. Unsupervised anomaly detection approach for time-series in multi-domains using deep reconstruction error. *Symmetry* **2020**, *12*, 1251.
12. Carta, S.; Podda, A.S.; Recupero, D.R.; Saia, R. A local feature engineering strategy to improve network anomaly detection. *Future Internet* **2020**, *12*, 177.
13. Burgueño, J.; de-la Bandera, I.; Mendoza, J.; Palacios, D.; Morillas, C.; Barco, R. Online anomaly detection system for mobile networks. *Sensors* **2020**, *20*, 7232.
14. Zhang, M.; Guo, J.; Li, X.; Jin, R. Data-driven anomaly detection approach for time-series streaming data. *Sensors* **2020**, *20*, 5646.
15. Bronner, L. Overview of the capacity planning process for production data processing. *IBM Systems Journal* **1980**, *19*, 4–27.
16. Hyndman, R.; Athanasopoulos, G. *Forecasting: Principles and practice*; OTexts: Melbourne, Australia, 2018.
17. Hamilton, J.D. *Time series analysis*; Princeton University Press, 1994.
18. Cryer, J.D.; Chan, K.S. *Time series analysis: With applications in R*; Springer, 2008.
19. Olson, T. *Applied Fourier analysis*; Springer, 2017.
20. Cleveland, R.B.; Cleveland, W.S.; McRae, J.E.; Terpenning, I. STL: A seasonal-trend decomposition procedure based on Loess (with discussion). *Journal of Official Statistics* **1990**, *6*, 3–73.
21. Lewis, N. *Deep time series forecasting with Python: An intuitive introduction to deep learning for applied time series modeling*; CreateSpace Independent Publishing Platform, 2016.

22. Rhee, M.J. *Nonlinear time series forecasting with neural networks*; ProQuest LLC, Ann Arbor, MI, 1995; p. 143. Thesis (Ph.D.)–University of California, Los Angeles.
23. Enterprise observability for multi-cloud environments. https://tanzu.vmware.com/observability. Accessed: 2021-01-10.
24. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition, 2015, [arXiv:cs.CV/1409.1556].
25. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition, 2015, [arXiv:cs.CV/1512.03385].
26. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions, 2014, [arXiv:cs.CV/1409.4842].
27. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding, 2019, [arXiv:cs.CL/1810.04805].
28. Pan, S.J.; Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering* **2010**, *22*, 1345–1359.
29. Faraway, J.; Chatfield, C. Time series forecasting with neural networks: A comparative study using the airline data. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **1998**, *47*, 231–250.
30. Hansen, J.; Nelson, R. Forecasting and recombining time-series components by using neural networks. *Journal of the Operational Research Society* **2003**, *54*, 307–317.
31. Zhang, G.P.; Qi, M. Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research* **2005**, *160*, 501–514.
32. Wang, X.; Smith, K.; Hyndman, R. Characteristic-based clustering for time series data. *Data Min. Knowl. Discov.* **2006**, *13*, 335–364.
33. Zhang, G.P., Neural networks for time-series forecasting. In *Handbook of Natural Computing*; Springer Berlin Heidelberg: Berlin, Heidelberg, 2012; pp. 461–477.
34. Kolarik, T.; Rudorfer, G. Time series forecasting using neural networks. *SIGAPL APL Quote Quad* **1994**, *25*, 86–94.
35. Nelson, M.; Hill, T.; Remus, W.; O'Connor, M. Time series forecasting using neural networks: Should the data be deseasonalized first? *Journal of Forecasting* **1999**, *18*, 359–367.
36. Fawaz, H.I.; Forestier, G.; Weber, J.; Idoumghar, L.; Muller, P.A. Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery* **2019**, *33*, 917–963.
37. Kwiatkowski, D.; Phillips, P.C.; Schmidt, P.; Shin, Y. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics* **1992**, *54*, 159 – 178.
38. Dickey, D.A.; Fuller, W.A. Distribution of the estimators for autoregressive time series with a unit root. *J. Amer. Statist. Assoc.* **1979**, *74*, 427–431.
39. Dickey, D.A.; Fuller, W.A. Likelihood ratio statistics for autoregressive time series with a unit root. *Econometrica* **1981**, *49*, 1057–1072.
40. Saïd, S.E.; Dickey, D.A. Testing for unit roots in autoregressive-moving average models of unknown order. *Biometrika* **1984**, *71*, 599–607.
41. Phillips, P.C.B. Time series regression with a unit root. *Econometrica* **1987**, *55*, 277–301.
42. Stellingwerf, R.F. Period determination using phase dispersion minimization. *The Astrophysical Journal* **1978**, *224*, 953–960.
43. Davies, S.R. An improved test for periodicity. *Monthly Notices of the Royal Astronomical Society* **1990**, *244*, 93–95.
44. Davies, S.R. Davies periodicity test revisited. *Monthly Notices of the Royal Astronomical Society* **1991**, *251*, 64–65.
45. Canova, F.; Hansen, B.E. Are seasonal patterns constant over time? A test for seasonal stability. *Journal of Business and Economic Statistics* **1995**, *13*, 237–252.
46. Hylleberg, S.; Engle, R.F.; Granger, C.W.J.; Yoo, B.S. Seasonal integration and cointegration. *J. Econometrics* **1990**, *44*, 215–238.
47. Osborn, D.R.; Chui, A.; Smith, J.P.; Birchenhall, C. Seasonality and the order of integration for consumption. *Oxford Bulletin of Economics and Statistics* **1933**, *50*, 4.
48. Hylleberg, S. *Modelling seasonality*; Oxford University Press, 1992.

49. Dickey, D.A.; Hasza, D.P.; Fuller, W.A. Testing for unit roots in seasonal time series. *Journal of the American Statistical Association* **1984**, *79*, 355–367.
50. Dickey, D.A. Seasonal unit roots in aggregate US data. *Journal of Econometrics* **1993**, *55*, 329–331.
51. Darne, O.; Diebolt, C. Note on seasonal unit root tests. *Quality and Quantity* **2002**, *36*, 305–310.
52. Thudumu, S.; Branch, P.; Jin, J.; Singh, J.J. A comprehensive survey of anomaly detection techniques for high dimensional Big Data. *Journal of Big Data* **2020**, *7*, 43 – 57.
53. Blázquez-García, A.; Conde, A.; Mori, U.; Lozano, J.A. A review on outlier/anomaly detection in time series data, 2020, [arXiv:cs.LG/2002.04236].
54. Pang, G.; Shen, C.; Cao, L.; van den Hengel, A. Deep learning for anomaly detection: A review, 2020, [arXiv:cs.LG/2007.02500].
55. He, Q.; Zheng, Y.; Zhang, C.; Wang, H. MTAD-TF: Multivariate time series anomaly detection using the combination of temporal pattern and feature pattern. *Complexity* **2020**, *2020*, 1–9.
56. Chandola, V.; Banerjee, A.; Kumar, V. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)* **2009**, *41*, 1–58.
57. Hodge, V.; Austin, J. A survey of outlier detection methodologies. *Artificial Intelligence Review* **2004**, *22*, 85–126.
58. Geiger, A.; Liu, D.; Alnegheimish, S.; Cuesta-Infante, A.; Veeramachaneni, K. TadGAN: Time series anomaly detection using generative adversarial networks, 2020, [arXiv:cs.LG/2009.07769].
59. Fuller, W.A. *Introduction to statistical time series*; Wiley, 1995; p. 728.
60. Grill, M.; Pevný, T.; Rehak, M. Reducing false positives of network anomaly detection by local adaptive multivariate smoothing. *Journal of Computer and System Sciences* **2017**, *83*, 43 – 57.
61. Deep learning for Java. https://deeplearning4j.org. Accessed: 2021-01-10.
62. Forecasting and anomaly detection with AI Genie. https://docs.wavefront.com/ai_genie.html. Accessed: 2021-01-10.