seq2graph: Discovering Dynamic Non-linear Dependencies from Multivariate Time Series

Xuan-Hong Dang, Syed Yousaf Shah and Petros Zerfos
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
Email: xuan-hong.dang@ibm.com, {syshah,pzerfos}@us.ibm.com

***Abstract*—Discovering temporal lagged and inter-dependencies in multivariate time series data is an important task. However, in many real-world applications with big data, such as commercial cloud management or predictive maintenance in manufacturing, such dependencies can be time-variant and non-linear, which makes it more challenging to extract such dependencies through traditional methods like Granger causality or statistical models. In this work, we present a novel deep learning model that uses multiple layers of adapted gated recurrent units (GRUs) for discovering both time lagged behaviors and inter-timeseries dependencies, representing them in the form of directed weighted graphs. Each individual time series is first analyzed by a pair of encoding-decoding GRUs in order to discover the time lagged dependencies and representing its samples as high dimensional vectors. Such vectors collected from all component time series are then analyzed by a decoding network component to discover inter-dependencies across all time series while forecasting their next values in the multivariate time series. Though the discovery of two types of dependencies are separated at two levels of our neural network, they are tightly connected and jointly trained in an end-to-end manner. With this joint training, improvement in learning of one type of dependency immediately impacts the learning process of the other one, leading to the overall highly accurate dependencies discovery. We empirically test our model on synthetic time series data in which the exact form of dependencies are known. We also practically evaluate its performance on two real-world applications, (i) dynamic multivariate performance monitoring data with high volatility from a commercial cloud provider and, (ii) multivariate time series generated by sensors for a manufacturing plant. We show how our approach is capable of capturing these dependency behaviors via intuitive and interpretable dependency graphs and use them to generate forecasting values.**

***Index Terms*—Big Data, Neural Network, Multivariate Time Series**

I. Introduction

Multivariate time series (MTS) modeling and understanding is an important task in machine learning and data mining, with numerous applications ranging from science and manufacturing [1], [2], economics and finance [3], [4], to medical diagnosis and monitoring [5], [6]. The main objective of time series modeling is to choose an appropriate model and train it based on the observed data such that it can capture the inherent structure and behavior of the time series. While majority of studies in the literature focus on the task of time series *forecasting* [7], in this work we focus on an another equally important and challenging task of *discovering dependencies* that exist in the time-variant multivariate times series. This task, though challenging, is important and particularly desirable for a variety of big data applications including early identification of components that require maintenance servicing in manufacturing facilities [8], [9], predicting causalities for resource management in cloud service applications [1], or discovering leading performance indicators for financial analysis of stocks to name a few.

In the literature, techniques such as learning coefficients from vector auto regression (VAR) models [10], Granger causality [11] and its variant of graphical Granger [12], clustering [13], and the recently proposed analysis of the weights embedded in a trained neural network [14] can be used to explore inter-dependencies among time series. However, one of the key drawback of all these methods is that, once the models are trained, their learnt coefficients/weights stay *fixed* during the *inference phase*, and hence intrinsically lack the desirable capability of discovering *time-variant* dependencies among time series. Their applicability are thus limited in the aforementioned real-world applications which desire for an innovative solution to unearth both time lagged and inter-dependencies in a timely manner in order to enable automated control or decision making. Figure 1 illustrates an example of cloud service with the plot of two component time series, the cpu-utilization and memory-utilization. Between time points of 208th and 214th (noted by the green rectangle), the time series system behaves dynamically. There is desirable to forecast and understand what causes such dynamic change(s) and how the dependencies among time series vary during such period of time. This sort of knowledge not only alerts a critical situation but also immensely assist the expert to investigate the causes of dynamic changes observed in the multivariate time series.

In this paper, we address the above challenging problem of discovering time variant dependencies by developing a novel deep learning architecture, which is built upon the core of multi-layer customized recurrent neural networks (RNNs). Our model is capable of timely discovering the inter-dependencies and temporal-lagged dependencies that exist in multivariate time series data and represent them as directed weighted graphs. By means of varied inter-dependencies, our model aims at learning the relationships existed among the time series, especially such relationships/dependencies can vary

The copyright notice: 978-1-7281-0858-2/19/$31.00 2019 IEEE

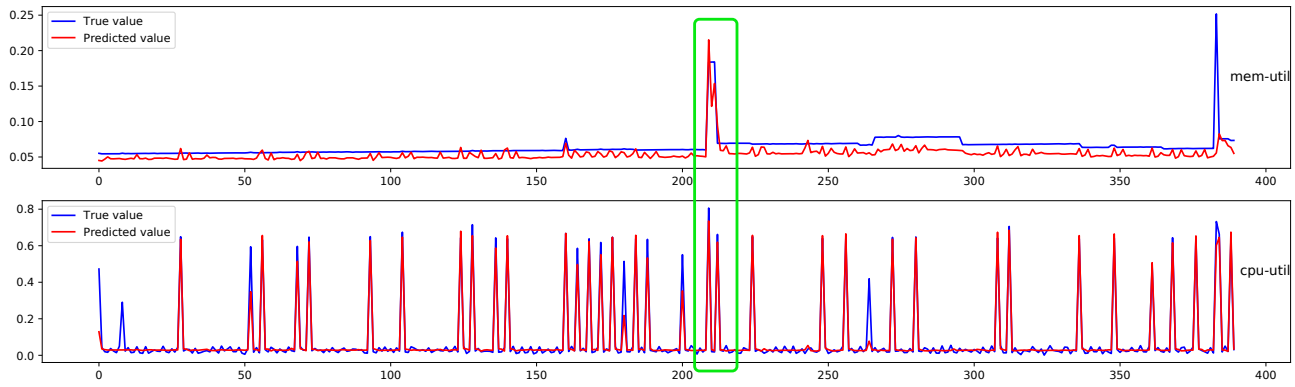

Fig. 1. Two selected time series, the cpu-utility and mem-utility, from a Cloud Service system. They are plotted during the *inference* phase of our model with the blue curve showing the ground truth values while the red one showing the forecasted values of our model. As noted, both time series keep changing and behave dynamically. It is hence desirable to discover the time-varying dependencies among these component time series, for example, as the ones (marked in the green box) during the period between 208 and 214 time points (details are explained in text and further in the Experiment section).

with time as the generating process of the multivariate series evolves. By varied temporal dependency, our model aims at discovering the time-lagged dependency within each individual time series, to identify those historical time points that highly influence future values of the multivariate time series. Our proposed model is a multi-layer RNN network developed based on the gated recurrent units (GRUs) [15] which excel in capturing long term dependencies in sequential data, are less susceptible to the presence of noise, and readily learn both linear and non-linear relationships embedded in the data sequences. Our model extracts hidden time-lagged and inter-dependencies in the MTS and present them in form of dependency graphs that help to comprehend the time series behaviors along the temporal dimension. In particular, this paper makes the following contributions:

(1) We pose and address the challenging yet important problem of discovering dynamic temporal-lagged and inter-dependencies from a system generating multivariate time series during the inference phase in a timely manner with the RNNs deep learning approach. (2) We present a novel neural architecture that is capable of discovering non-linear and non-stationary dependencies in the multivariate time series, which are not captured by the Granger causality. Of particular importance is its capability in identifying non-stationary, time-varying dependencies during the *inference* phase of the time series, which differentiates our method from state-of-the-art techniques. (3) We evaluate the performance of our proposed approach on synthetic data, for which we have control and knowledge of the (non-linear) dependencies that exist therein, as well as two real-world applications: (i) the utilization measurements of a commercial cloud service provider and (ii) the sensor data series monitoring from a semiconductor manufacturing facility, which exhibit highly dynamic and volatile behavior. (4) We compare our model with well-established multivariate time series algorithms, including the widely used statistical VAR model [10], [16], the recently developed RNNs [17] and the hybrid residual based RNNs [18], showing its superiority in discovering dependencies that are also not captured by the Granger causality [11].

II. Related work

State-of-the-art studies on time series analysis and modeling can generally be divided into three categories. The first one involves statistical models such as the autoregressive integrated moving average *(ARIMA)* [7], the latent space based Kalman Filters [19], the HoltWinters technique [20], and the vector autoregression (VARs) [7], [16] which are extended to deal with multivariate time series. Although having been the widely applied tools, the usage of these models often requires the strict assumption regarding the linear dependencies of the time series. Their approximation to complex real-world problems does not always lead to satisfactory results [21].

The second category consists of techniques based on artificial neural networks (ANN) [22], [23]. One of the earliest attempts [24] employs the 1-hidden feedforward neural network with markedly improved performance as compared to the ARIMIA. More recent approaches include the usage of multilayered perceptron network [25], Elman recurrent neural network [26], and the RNNs dealing classification task over short sequences [27]–[29]. Compared to statistical models, ANN-based models have a clear advantage of well approximating non-linear complex functions. However, they are also criticized for their black box nature and limited explanatory information. RNNs have also been applied to the sequence classification as recently developed in.

The third category is a class of hybrid methods that combine several models into a single one. [30] is among the premier studies that first applies the ARIMA to model the linear part of time series, and then adopts a feedforward neural network to model the data residuals returned by the ARIMA. Subsequent studies are often the variants of this approach, by replacing the feedforward network with other learning models [31], such as the long-short term memory (LSTM) [18], or non-linear learning techniques [32]. These existing techniques including Granger causality [11], [12] either lack the ability to discover complex dependencies or are unable to capture the

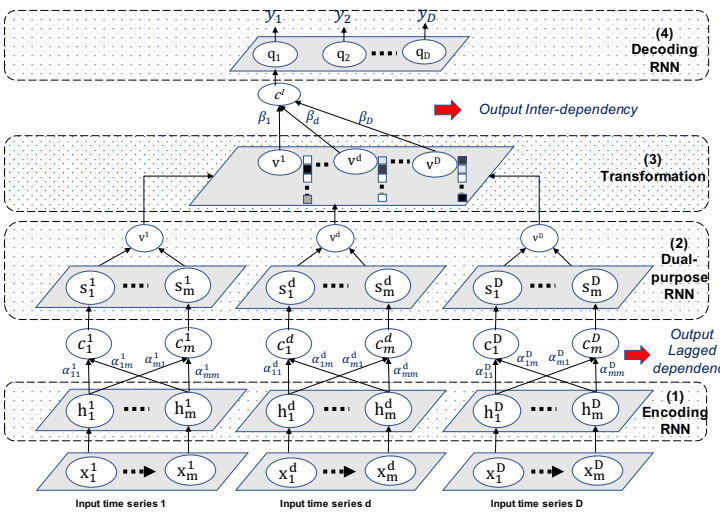

Fig. 2. Overview of our deep learning model. Given an MTS at input, it learns to discover time lagged and inter-dependencies, which are outputted as dependency graphs while making prediction over the future time point $\mathbf{y}$.

instantaneous dynamic dependencies in the data, as contrasted to our studies that learn and discover these time-variant and complex dependencies explicitly through the deep recurrent neural network model.

III. Methodology

Our studies in this paper focus on a multivariate time series (MTS) system with D time series (variables) of length T, denoted by $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$, where each $\mathbf{x}_t \in \mathcal{R}^D$ is the measurements of MTS at timestamp t. The component d-th time series is denoted by $X^d = \{x_1^d, x_2^d, \ldots, x_T^d\}$, in which $x_t^d \in \mathcal{R}$ represents its measurement at time t. Given such an MTS system, we aim to analyze it from two aspects: (i) how the future measurements are dependent on the past values of each individual time series, we call them temporal lagged dependencies; (ii) how the measurements of different time series are dependent on each other, we call them inter-timeseries dependencies. The inter-dependencies discovered at a specific timestamp can be naturally represented as a directed weighted graph whose nodes correspond to individual time series, and an edge reveals which time series is dependent on which time series. An edge's weight encodes how strong the corresponding dependency is (for example, such graphs are shown in Fig.3, Fig.6). It is important to note that our focus is to capture the *time-variant* dependencies that are not limited to the training phase but further during the inference phase. Thus, the developed model timely discovers the dynamically kept-changing dependencies at any future timestamp t, and utilizes these unearthed dependencies to accurately forecast future values of the MTS.

A. Overall model

We outline our network model in Figure 2 which consists of 4 main components/layers: (1) Given a multivariate time series (MTS) at input layer, the `Encoding RNN` component comprises a set of D RNN networks, each processes an individual time series (of the MTS) by encoding it into a sequence of encoding vectors. (2) The next `Dual-purpose RNN` component also consists of a set of D customized RNNs, each discovers the temporal lagged dependencies from one component/constituted time series and subsequently encodes them to a sequence of output states. Together with `Encoding RNN`, they form D pairs of encoding-decoding RNNs, reasoning on D input time series. (3) Sequences of output states from all component `Dual-purpose RNNs` are then gathered together in which each sequence is transformed into a high dimensional vector by the `Transformation layer`. These feature vectors act as the encoding representations of the corresponding component time series and are fed into the next network layer. (4) The next and also final `Decoding RNN` layer learns to discover the inter-dependencies across all component time series through identifying the most informative input feature vectors toward forecasting the next value of each component time series at the final output of the model. This layer hence outputs both the dependency graphs and the predictive next value of the multivariate time series. Before presenting each layer of our model in details, we provide a brief background on the gated recurrent units (GRUs) [33] to which our network components are the extended variants intrinsically designed for the multivariate time series setting.

Gated recurrent units (GRUs): have been among the most successful neural networks in learning from sequential data [17], [33]. Similar to the long-short term memory units (LSTMs) [34], GRUs are cable of capturing the long term dependencies in sequences through the memory cell mechanism. Specifically, there are two gates, reset $\mathbf{r}_t$ and update $\mathbf{z}_t$, defined in GRUs which are computed with the following equations.

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r) \quad (1)$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z) \quad (2)$$

with $\sigma(.)$ as the non-linear sigmoid function, $\mathbf{x}_t$ is the data point input at time t, $\mathbf{h}_{t-1}$ is the previous hidden state of the GRUs. $\mathbf{W}$'s,$\mathbf{U}$'s, and $\mathbf{b}$'s (subscripts are omitted) are respectively the input weight matrices, recurrent weight matrices, and the bias vectors, which are collectively the GRUs' parameters to be learnt. Based upon these two gates, GRUs compute the current memory content $\tilde{\mathbf{h}}_t$ and then update their hidden state (memory) to the new one $\mathbf{h}_t$ as follows:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h x_t + \mathbf{U}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h) \quad (3)$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t \quad (4)$$

where $\odot$ is the element-wise product, and $\mathbf{W}_h, \mathbf{U}_h, \mathbf{b}_h$ are again the GRU's parameters to learn. As seen, $\tilde{\mathbf{h}}_t$ is determined by the new data input $\mathbf{x}_t$ and the past memory $\mathbf{h}_{t-1}$, and its content directly impacts $\mathbf{h}_t$ through Eq.(4). It is due to the usage of $\mathbf{r}_t$ and $\mathbf{z}_t$ in these equations that defines their specific functions. In specific, $\mathbf{r}_t$ decides how much information in the past $\mathbf{h}_{t-1}$ should be forgotten (Eq.(3), hence named reset gate), while $\mathbf{z}_t$ determines how much of the past information

to pass along to the current state (Eq.(4), hence named update gate). Due to this gating mechanism, GRUs can effectively keep most relevant information at every single step. For the sake of brevity, we denote all these computational steps briefly by $\mathbf{h}_t = GRU(\mathbf{x}_t, \mathbf{h}_{t-1})$ implying it as a function with inputs and outputs while temporarily skipping the detailed internal gating computations.

B. Discovering temporal lagged dependencies

At a single time point, our model receives D sequences as inputs, each consisting of m historical values from each component time series, and produces a graph of inter-dependencies among all time series, and temporal lagged dependencies, along with a sequence $\mathbf{y} = \{y_1, \ldots, y_D\}$ as the forecasted values of the next timestamp in the multivariate time series at output (Fig.2). It is worth mentioning that these dependencies can be complex, non-trivial, and especially time-variant. For instance, future performance of CPUs and memory usage from a cloud management service can be anticipated based on their own historical values in most normal situations. Nonetheless, when the requesting data from outside world is abnormally high (e.g., under attacks or at peak hours) and overwhelms the CPU capacity, poor performance on computation can be experienced. It can consequently influence the performance of memory due to caching data issues. Early detection and accurately discovering these fast changing dependencies is particularly important since it provides in details the mutual impacts among time series, based upon which appropriate actions can be performed, or it also can enable an automated system by allocating more resources to the right component and in an adaptive manner, ensuring the sustainability of the cloud service.

Our deep learning model is intrinsically designed for modeling and discovering these complex time-varying dependencies among time series. We next present the network components designed for discovering the time lagged dependencies which tells us what timepoints in the past of a specific d-th time series that crucially impact the future performance of MTS.

(1) Encoding RNN: At the d-th time series, the Encoding RNN component receives a sequence of the last m historical values $\{x_1^d, x_2^d, \ldots, x_m^d\}$, and it encodes them into a sequence of hidden states $\{\mathbf{h}_1^d, \mathbf{h}_2^d, \ldots, \mathbf{h}_m^d\}$. Its main objective is to learn and encode the long-term dependencies among historical timepoints of the given time series into a set of the RNNs' hidden states. As our model read in an entire sequence of length m timepoints from the d-th time series, we utilize the bidirectional GRU_E (E for encoding) in order to better exploit information from both directions of the sequence, as shown by the following equation, for each $t = 1 \ldots m$:

$$\mathbf{h}_t^d = [\overrightarrow{\mathbf{h}_t^d}, \overleftarrow{\mathbf{h}_t^d}] = [\overrightarrow{GRU_E}(x_t^d, \overrightarrow{\mathbf{h}_{t-1}^d}), \overleftarrow{GRU_E}(x_t^d, \overleftarrow{\mathbf{h}_{t+1}^d})]$$

It is worth mentioning that, our model makes use an individual GRU to learn each component time series at the input, which is crucial to discover time lagged dependencies within each series (as shortly presented next). This is opposed to other techniques [18], [26] that often feed entire multivariate time series into a single model and hence are limited in discovering such patterns.

(2) Dual-purpose RNN: At this network component, there is a corresponding variant GRU_{DP} (DP for dual-purpose) network for each GRU_E in the previous Encoding RNN layer. Unlike a GRU_E which outputs its hidden states, GRU_{DP} explicitly returns m vectors v_t^d's based on vector $\mathbf{h}_t^d$, its own hidden states $\mathbf{s}_t^d$, and the novelly introduced temporal context vectors $\mathbf{c}_t^d$'s. In calculating each $\mathbf{c}_t^d$, GRU_{DP} generates non-negative, normalized coefficients α_{tj}^d w.r.t. each input vector $\mathbf{h}_j^d$ outputted from the GRU_E. While still retaining the time order among the output values, this mechanism enables our network component to focus on specific timestamps of the d-th time series at which the most relevant information is located. Mathematically, our GRU_{DP} network computes the following Eqs.(5)-(8), for each $t = 1, \ldots, m$:

$$\alpha_{tj}^d = \frac{\exp(tanh(\mathbf{W}_\alpha[\mathbf{s}_{t-1}^d; \mathbf{h}_j^d])^\top \mathbf{u}_\alpha)}{\sum_{k=1}^m \exp(tanh(\mathbf{W}_\alpha[\mathbf{s}_{t-1}^d; \mathbf{h}_k^d])^\top \mathbf{u}_\alpha)} \tag{5}$$

$$\mathbf{c}_t^d = \sum_{j=1}^m \alpha_{tj}^d \mathbf{h}_j^d \tag{6}$$

$$\mathbf{s}_t^d = GRU_{DP}(v_{t-1}^d, \mathbf{s}_{t-1}^d, \mathbf{c}_t^d) \tag{7}$$

$$v_t^d = tanh(\mathbf{W}_o^d v_{t-1}^d + \mathbf{U}_o^d \mathbf{s}_t^d + \mathbf{C}_o^d \mathbf{c}_t^d + \mathbf{b}_o^d) \tag{8}$$

where $\mathbf{W}_\alpha$ and $\mathbf{u}_\alpha$ are parameters to learn α_{tj}^d's coefficients, and are jointly trained with the entire model. $\mathbf{W}_o^d, \mathbf{U}_o^d, \mathbf{C}_o^d, \mathbf{b}_o^d$ are the network's parameters in learning to output v_t^d's. As seen, α_{tj}^d's are utilized to weight the importance of each encoding vector $\mathbf{h}_j^d$ toward learning the temporal context vector $\mathbf{c}_t^d$, which directly makes impact on updating the current hidden state $\mathbf{s}_t^d$ and the output v_t^d. It is worth mentioning here that our weight mechanism through α_{tj}^d's at this temporal domain follows the general attention mechanism adopted in neural machine translation (NMT) [35], [36], yet it is fundamentally different in two aspects. First, we do not have ground-truth (e.g., target sentences in NMT) for the outputs v_t^d's but let GRU_{DP} learn them automatically. Their embedding information directly governs the process of learning inter-timeseries dependencies in the upper layers in Fig.2. Indeed, v_t^d's act as the variables bridging information between two levels of discovering the time lagged and the inter-dependencies. Second, our model explores the $tanh$ function rather than the $softmax$ one, which gives it more flexility to work on the *continuous* domain of time series, making real values embedded in v_t^d's, similar to the input and output time series. Hence, our extension of GRU essentially performs two tasks: (i) decoding information in the temporal domain to discover time lagged dependencies within each time series, (ii) encoding this temporal information into a set of outputs v_t^d's which collectively are seen as the representation of the corresponding original time series.

C. Discovering inter-dependencies

(3) Transformation layer: Following the Dual-purpose RNN layer, which generates a sequence $\mathbf{v}^d = \{v_1^d, v_2^d, \ldots, v_m^d\}$ for d-th time series, this Transformation layer gathers these sequences from all D time series and concatenate each as a high dimensional feature vector. These representative vectors are stacked into a sequence $\{\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^D\}$ and serves as the input to the next Decoding RNN component (Fig.2). As noted, though there is no specific temporal dimension among these vectors, their order in the stacked sequence should be specified prior to the model training to ensure the right interpretation when our model explores to discover the inter-timeseries dependencies.

(4) Decoding RNN: This component consists of a single variant GRU_D network (D for decoding) that performs the inter-dependencies discovery while also making prediction over $\mathbf{y}$ at the model's output. During the training phase, an $y_i \in \mathbf{y}$ is the next value of the i-th time series, and its computation is relied on all vectors $\{\mathbf{v}^1, \mathbf{v}^2, \ldots, \mathbf{v}^D\}$ released by the previous Dual-purpose RNN layer. Its calculation steps are as follows:

$$\beta_i^d = \frac{\exp(tanh(\mathbf{W}_\beta[\mathbf{q}_{i-1}; \mathbf{v}^d])^\top \mathbf{u}_\beta))}{\sum_{k=1}^{D} \exp(tanh(\mathbf{W}_\beta[\mathbf{q}_{i-1}; \mathbf{v}^k])^\top \mathbf{u}_\beta))} \tag{9}$$

$$\mathbf{c}_i = \sum_{d=1}^{D} \beta_i^d \mathbf{v}^d \tag{10}$$

$$\mathbf{q}_i = GRU_D(\mathbf{q}_{i-1}, \mathbf{c}_i) \tag{11}$$

$$y_i = tanh(\mathbf{C}_o \mathbf{c}_i + \mathbf{U}_o \mathbf{q}_i + \mathbf{b}_o) \tag{12}$$

where $\mathbf{W}_\beta$ and $\mathbf{u}_\beta$ are parameters to learn β_i^d's weights, and $\mathbf{C}_o, \mathbf{U}_o$ and $\mathbf{b}_o$ are the parameters w.r.t. the output y_i's. As seen, this network component first computes the alignment of each of vectors $\mathbf{v}^d$ (featured for each input time series at this stage) with the hidden state $\mathbf{q}_{i-1}$ of the GRU_D through the coefficient β_i^d. Using these generated normalized coefficients, it obtains the context vector $\mathbf{c}_i$ that is used to update the current GRU_D's hidden state $\mathbf{q}_i$ and making prediction of output y_i. Let y_i be the next value of the i-th time series, β_i^d will tell us how important the corresponding d-th time series (represented by $\mathbf{v}^d$) in predicting y_i. In other words, it captures the dependency of i-th series on the d-th series. The stronger this dependency is, the larger the β_i^d (note that $\sum_d \beta_i^d = 1$ and $\beta_i^d \geq 0$). The whole vector β_i therefore tells us the dependencies of i-th time series on all D time series in the system. These coefficients along with the predictive values are the outputs of our entire model, and based on β_i^d's, it is straightforward to construct the graph of inter-dependencies among all time series.

Optimization: We train our entire model end-to-end via the adaptive moment estimation algorithm [37] using mean squared errors as the train loss function. Let us denote the entire model as a function F with Θ as its all parameters, the train loss function is:

$$L(X; \Theta) = \sum_{t=m}^{T-1} \sum_{d=1}^{D} (y_d - F_d(X_t; \Theta))^2 \tag{13}$$

with X as the entire training MTS data, X_t as a sequence of the last m timepoints in MTS at timestamp t, y_d as the value of d-th time series at the next timestamp $t+1$ and $F_d(X_t; \Theta)$ as the predictive value of the model toward y_d.

a) Discussion: Our model though explores the temporal lagged and inter-dependencies among time series, it can also be more generally seen as performing the task of transforming multiple input sequences into one output sequence. As presented above, the output sequence is the values of the next timestamp of all time series constituted in MTS, but one can easily replace this sequence with the next n values of *one* time series of interest. That means, we want to discover dependencies of this particular series on the other ones while forecasting its multiple time points ahead. With this case, Eq.(11) and (12) can be replaced by adding a term accounting for the previous predictive value, i.e., $\mathbf{q}_i = GRU_D(y_{i-1}, \mathbf{q}_{i-1}, \mathbf{c}_i)$ and $y_i = tanh(y_{i-1}, \mathbf{C}_o \mathbf{c}_i + \mathbf{U}_o \mathbf{q}_i + \mathbf{b}_o)$ in order to further exploit the temporal order in the output sequence. Interpretation over the inter-dependencies based on β_i's remains unchanged yet it is for this solely interested time series and over a window of its n future time points.

IV. Experiments

We name our model `seq2graph` and its training were performed on a machine with two Tesla K80 GPUs, running Tensorflow 1.3.0 as backend. We compare performance of `seq2graph` against: (i) `VAR` model [10] which is one of the most widely used statistical techniques for analyzing MTS; (ii) `RNN-vanilla` which receives all time series at inputs and predicts $\mathbf{y}$ at output (this model can be considered as an ablation study of our network by collapsing all layers into a single one). (iii) `RNN-residual` [18], a hybrid model that trains an RNN on the residuals returned by the statistical ARIMA. To match our model complexity, we use GRUs for both `RNN-vanilla` and `RNN-residual` with similar number of hidden neurons used in our model, and their number of stacked layers are tuned from one to three layers. For `seq2graph`, its optimal number of hidden neurons for each network component is tuned from the pool of $\{16, 32, 48, 64\}$. For each examined dataset presented below, we split it into training, development, and test sets with the ratio of 70:15:15. The development set is used to tune models' hyper-parameters while the test set is used to report their performance.

A. Empirical analysis on synthetic data

Discovering time lagged and inter-dependencies is challenging and there is no available dataset with known time-variant ground truths for these dependencies. Hence, we make use of synthetic data and attempt to answer the following

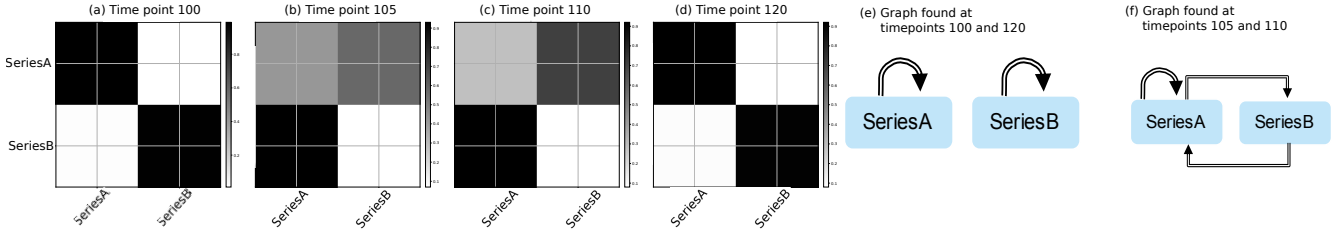

Fig. 3. Inter-dependencies discovered by seq2graph in the bivariate time series at 4 examined time points shown on top of (a)-(d). In each of figures (a)-(d), a darker square reveals a stronger dependency of time series shown in *row* on time series shown in *column*. For instance, in predicting value of SeriesA at time point 100 (first row in (a)), the model strongly relies on historical time points of SeriesA, while at time point 105 (first row in (b)), it relies on the historical time points of both SeriesA and SeriesB. Such inter-dependencies (and others) are represented in the directed graphs shown (e)-(f), which are consistent with the ground truth rules used to generate both time series.

key questions: (i) Can `seq2graph` discover nonlinear time-varying dependencies (e.g., introduced in data via *if-then* rules) among time series during inference phase? (ii) Once dependencies of one time series on the others are known, can `seq2graph` accurately discover the lagged dependencies? (iii) How accurate is our model in predicting future values of the multivariate time series?

Data description: For illustration purposes, we use the bivariate time series (MTS with more variables are evaluated on the real datasets) that simulates the *time-varying* interaction between two time series namely SeriesA and SeriesB. We generate 10,000 data points with real-values between 0 and 1 using following two rules:

$$\{A[t{+}1], B[t{+}1]\} = \begin{cases} \{A[t{-}4], f(B[t{-}3], B[t{-}6])\} \text{if} & A[t] < 0.5 \\ \{f(A[t{-}6], B[t{-}3]), A[t{-}3]\} \text{if} & A[t] \geq 0.5 \end{cases}$$

Rule 1 (1st row) specifies that, if the current value of SeriesA is smaller than 0.5, values in the next future timestamp $t+1$ of both series are autoregressive from themselves. Rule 2 (2nd row) specifies that if this value is equal or greater than 0.5, there are mutual impacts between two series. Specifically, the next value of SeriesA is decided by its value at the 6th lagged timestamp and that of the 3rd lagged in SeriesB, while the next value of SeriesB is determined by the 3rd lagged time point in SeriesA. f is the average function over these lagged values. Also, in order to ensure the randomness and volatility in the input sequence, we frequently re-generated values for the lagged time points.

a) Discovery of dependencies: We trained all models with window size $m = 8$, just above the furthest lagged points defined in our data generating rules. We evaluate the capability of `seq2graph` in discovering dependencies through investigating the time-varying vectors α's and β's during the inference phase (i.e. on test data). Recall that, unlike model's parameters which were fixed after the training phase, these coefficient vectors keep changing according to the input sequences. We show in Fig.3(a)-(d) the coefficients of β's (Eq.(9)) which reflects the inter-dependencies between two time series at 4 probed timestamps on the test data. Within each figure, a darker color shows a stronger dependency of the time series named in the row axis on those named in the column axis (more concretely, each row in the figure encodes a vector β_i whose entries are computed by Eq.(9)).

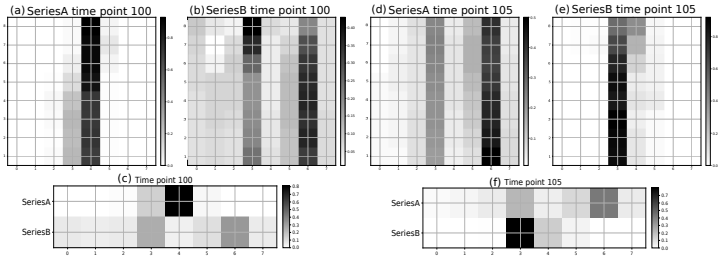

Fig. 4. Temporal lagged dependencies discovered by our model at 2 time points 100 ((a)-(b)) and 105 ((d)-(e)). Plots (c) and (f) are respectively the *row-aggregated* versions of Plots (a)-(b) and (d)-(e). For instance, (a) is row-aggregated to the 1st row of (c), (b) is row-aggregated to the 2nd row of (c). Such aggregated attention maps show the temporal lagged dependencies on each individual input time series sequence (more detailed interpretation is given in text).

Investigating the input sequences, we knew that data points at timestamps 100 and 120 were generated by rule 1, while those at 105 and 110 were generated by rule 2.[1] It is clearly seen that `seq2graph` has accurately discovered the ground truth inter-dependencies between SeriesA and SeriesB. Particularly, at timestamps 100 and 120 at which the two time series were generated by rule 1, our model found that each time series auto-regresses from itself, as evidenced by the dark color at the diagonal entries in Figs.3(a) and (d). However, at timestamps 105 and 110, `seq2graph` found that SeriesA's next value is dependent on the historical values of both SeriesA and B, while SeriesB's next value is determined by the historical values of SeriesA (detailed by the lagged dependencies shortly shown below), as observed from Figs.3(b) and (c). These inter-dependencies between the two time series can be intuitively visualized via a directed weighed graph generated at each time point. In Fig.3(e), we show the graph found at timestamps 100 and 120, while in Fig.3(f). we plot the one discovered at timepoints 105 and 110. As seen, these directed graphs clearly match our ground truth rules that have generated the bivariate time series.

In evaluating the correctness of temporal lagged dependencies discovered by `seq2graph`, we select 2 time points

[1]For simplicity, we have re-indexed the timepoints during the inference phase, i.e. in test dataset, started from 0.

TABLE I

PREDICTION ACCURACY OF ALL METHODS MEASURED IN THE ROOT MEAN SQUARED ERROR (RMSE), AND MEAN ABSOLUTE ERROR (MAE). ORIGINAL VALUES FROM THE CLOUD SERVICE AND MANUFACTURING DATA ARE QUANTIFIED IN THE LOG-SCALE AND HENCE THEY ARE SMALL. HOWEVER, THEY DO NOT IMPACT THE PERFORMANCE COMPARISON ACROSS EXAMINED METHODS. A BETTER PERFORMANCE MEANS A VALUE CLOSER TO ZERO OF RMSE AND MAE.

		Synthetic Data		Cloud Service Data				Manufacturing Data				
		SeriesA	SeriesB	pkg-tx	pkg-rx	mem-util	cpu-util	CurrentHTI	PowerHTI	PowerSPHTI	VoltHTI	TempHTI
RMSE	VAR	0.22	0.215	0.04	0.024	0.021	0.053	0.022	0.028	0.029	0.019	0.012
	RNN-vanilla	0.017	0.014	0.039	0.023	0.018	0.049	0.021	0.029	0.027	0.021	0.011
	RNN-residual	0.026	0.025	0.038	0.021	0.015	0.06	0.02	0.024	0.027	0.018	0.012
	seq2graph	0.012	0.013	0.037	0.022	0.015	0.028	0.018	0.022	0.022	0.016	0.009
MAE	VAR	0.184	0.178	0.024	0.009	0.007	0.044	0.006	0.007	0.006	0.009	0.005
	RNN-vanilla	0.09	0.08	0.021	0.009	0.006	0.042	0.006	0.005	0.007	0.008	0.005
	RNN-residual	0.11	0.12	0.021	0.006	0.007	0.044	0.005	0.006	0.005	0.008	0.005
	seq2graph	0.08	0.07	0.023	0.007	0.005	0.015	0.004	0.005	0.004	0.006	0.004

100 and 105 (each represented for one generated rule), and further inspect the vectors α's (Eq.(5)) calculated at the Dual-purpose RNN layer in our deep neural model. Figs.4(a)-(b) show coefficients of these vectors w.r.t. to input SeriesA and B respectively at time points 100, while Figs.4(d)-(e) show those at time point 105. Within each figure, the x-axis shows the lagged timestamps with the latest one indexed by 0, while the y-axis shows the index of t computed in the Dual-purpose RNN. Again, dark colors encode the dependencies of the next timepoint on each of the last m lagged timepoints in a time series. One can observe that `seq2graph` properly puts more weights (larger coefficients α^d_{tj}'s) to the correct lagged time points in both cases. Particularly, it strongly focuses on 4th lagged point in SeriesA, 3rd and 6th lagged points in SeriesB (Figs.4(a)-(b)) at timestamp 100. Likewise, the model takes much attention on 3rd and 6th lagged points in SeriesA, and 3rd lagged point in SeriesB at time 105 (Fig.4(d)-(e)). Note that, when the number of time series in a system is large (as we shortly present with the real-world time series systems), one can aggregate α's w.r.t each component time series into a single one for better visualization. For example, Figs.4(a) and (b) are row-aggregated respectively into the first and second rows of Fig.4(c). Similarly, Fig.4(d) and (e) are aggregated into the two rows of Fig.4(f). As seen, both figures explain well the lagged dependencies within each component time series (named in the y-axis) which strongly match our ground truth rules.

In comparison against other methods, we found that these *dynamically* changed dependencies were not accurately captured by `VAR`'s and `RNN-residual`'s coefficients. They both tend to give weights on more historical time points than needed. For instance, `VAR` places high (absolute) weights on timepoints $\{1, 3, 6, 7, 8\}$ on SeriesA, and $\{3, 5, 6\}$ on SeriesB when it was trained to forecast the future value of SeriesA; and on timepoints $\{1, 3, 5, 7\}$ on SeriesA, $\{3, 4, 5, 7\}$ on SeriesB when it was trained to forecasts the next value of SeriesB. None of algorithms are able to timely unearth the accurate *time-varying* dependencies in the MTS.

b) Prediction Accuracy: We briefly provide prediction accuracy of `seq2graph` and other techniques on this dataset

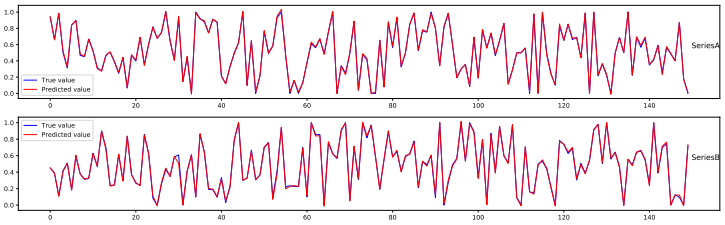

Fig. 5. Forecasting values on the bivariate time series using our model. True values are shown in blue while our model's predicted values are shown in red.

in the first two columns of Table I, and plot in Fig.5 the first 150 forecasted values of our model on the test set of this controlled bivariate time series (plots from other methods are omitted to due space constraints). The root mean squared error (RMSE), and mean absolute error (MAE) have been used for the accuracy measurements. As observed, `VAR` underperforms compared to both `RNN-vanilla` and `RNN-residual`, which is due to its linear nature. The prediction accuracy of `seq2graph` is better than `RNN-vanilla` and `RNN-residual` while further offering insights into the dependencies within and between time series. Its better performance could be justified by the accurate capturing of time-varying time series dependencies.

B. Empirical analysis over real-world applications

Data description: We further evaluate our model on two real-world systems that generate multivariate time series: (i) Cloud Service Data consisting of 4 metrics CPU utilization (cpu-util), memory utilization (mem-util), network receive (pkg-rx), and network-transmit (pkg-tx), collected every 30 seconds from a public cloud service provider over one day. (ii) Manufacturing Data consisting of 5 sensors measured over CurrentHTI, PowerHTI, VoltHTI, TempHTI, and PowerSPHTI (power set point) at a manufacturing plant with 500,000 timepoints.

a) Discovery of dependencies: With regard to the Cloud Service data, we show in the first four plots of Fig.6 the time-varying dependencies of one typical short period during the inference phase, between timepoints 208 and 214 as

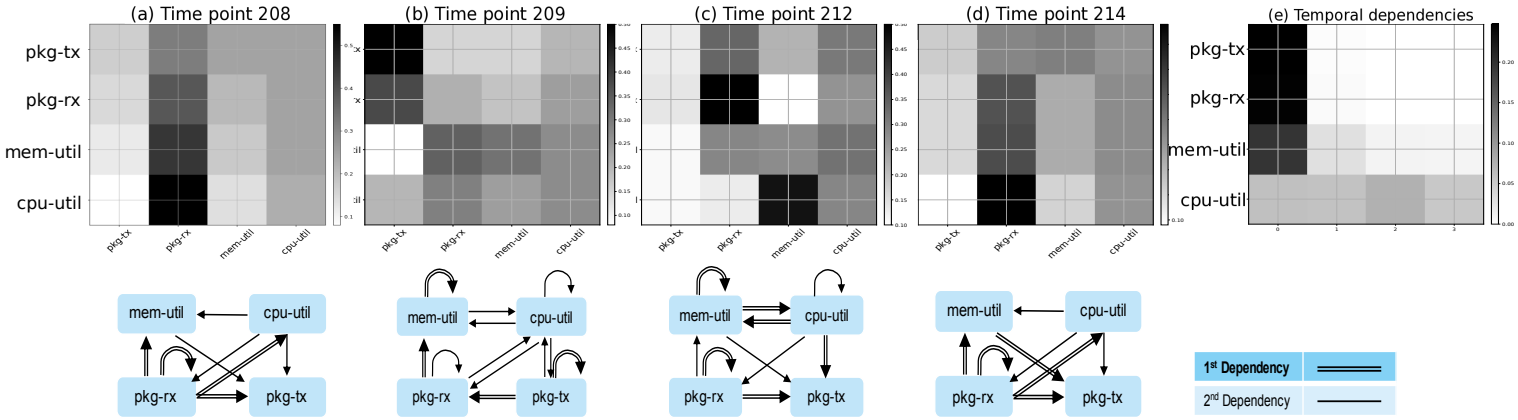

Fig. 6. Cloud Service Data: Popular/common inter-dependencies are shown as at timepoint 208 (a), and timepoint 214(d). Between them, there are dynamic changes of dependencies among the involved time series as shown at timepoint 209(b) and 212(c). Their corresponding representative directed graphs are shown in the second row. Thickness of each edge is proportional to the strength of dependency. Popular time lagged dependencies are shown in (e), the x-axis shows lagged timestamps with the latest one indexed by 0.

highlighted in the green rectangle in Fig.1[2], at which the system is likely suffered from a large amount of requesting data, impacting the performance on both the cpu- and mem-utilization. As observed, there are clear patterns that can be visibly interpreted. Prior and after this period of time (timepoints 208 and 214 respectively), performance of both cpu-util and mem-util are highly dependent on the pkg-rx (Figs.6(a,d)). Nonetheless, when the utilization in the memory time series is notably high (between timepoints 209 and 212), such dependencies significantly change. For instance, performance of cpu-util is determined by almost all series at the early stage of the event (timepoint 209) and lately strongly determined by only the mem-util and itself (timepoint 212). They tend to reflect the situation at which the Cloud system suffers from a large amount of data, being buffered in the memory, and in turn requests intensive performance of the cpu-util. After this period, the system restores to the common state in which the performance of all time series generally depends on the pkg-rx and cpu-util as shown in Figs.6(d).

The detailed inter-dependencies among all time series are illustrated in the weighted directed graph plotted below each time point in Fig.6. Thickness of edges in general is proportional to the strength of dependencies. In the plots, we roughly classify them into 1st and 2nd dependencies with the former one being more significant. The common time lagged dependencies are shown in the 5th plot of Fig.6. As mentioned, the x-axis shows the lagged timepoints with the latest one indexed by 0. As seen, except the cpu-util whose historical lagged values often influence the future performance of all time series, the latest values of the other time series tend to impact the system's future performance.

For Manufacturing data, we show the application of our model in Fig.8, and in Fig.7 we present some typical dependencies (with milder changes). The probed time points at which these dependencies were investigated are shown on top of each plot. As observed, there are strong dependencies of all time series (except TempHTI) on the PowerHTI, which can be justified by the power law in electronics. However, future values of TempHTI are mostly dependent on its historical values, which shows its nature of being less variant and slowly varied as compared to other time series in the system. These dependencies are timely converted into directed weighted graphs in our testbed model as the one illustrated at the bottom-right graph in Fig.8. The common time lagged dependencies are summarized in the last plot of Fig.7, revealing predictive information are mostly located at the latest time points of all time series, except with PowerSPHTI and VoltHTI whose further lagged time points also show impact.

Recall that we do not have the ground truth of dependencies among component time series at any given time point of these real-world multivariate time series. Nonetheless, as an attempt to verify our dependency findings above, we select the Cloud Service Data and further perform the Granger causality tests (based on F-statistic) [11], [12] of all other time series toward predicting 2 principle series of mem-util and cpu-util. The results are reported in Table II. It is clearly observed that the P-value of pkg-rx is extremely small as compared to that of cpu-util toward predicting the mem-util series (top table), or that of mem-util toward predicting cpu-util (bottom table). This means we are more confident in rejecting the null hypothesis that pkg-rx does not Granger-cause mem-util (cpu-util), but less confident in rejecting the hypothesis that cpu-util (mem-util) does not Granger-cause mem-util (cpu-util). These statistical tests confirm the findings of our model `seq2graph` above which shows the strong dependencies of both cpu-util and mem-util on pkg-rx, but weak dependencies between cpu-util and mem-util in common situations (e.g. before time point 208 or after time point 214 in Fig.1). Note that, these pairwise tests also showed that pkg-tx may contain information toward predicting mem-util and cpu-util based on its low P-values. However, `seq2graph` relies mainly on pkg-rx, rather than both pkg-rx and pkg-tx, in order to avoid

[2]Recall that the time series shown in Fig.1 are during the *inference* time of the Cloud Service data.

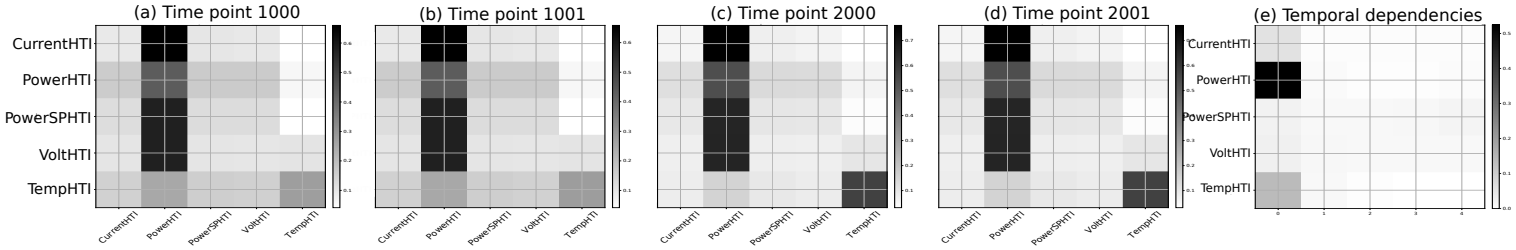

Fig. 7. Manufacturing Data: Inter-dependencies are examined at four selected testing time points shown on top each figure (a)-(d). The common temporal lagged dependencies are plotted in (e).

redundant information, i.e. a new variable is included only if it brings additional information [38]. It is worth mentioning that Granger-causality is highly effective in validating the existence of dependencies dominantly appeared in the time series. It is limited in discovering dependencies appeared in short windows of time. Although subsets of time series can be sampled and tested, it remains open on how large these subsets should be (in order to gain statistical significance), and how often this test should be done. Our developed neural network `seq2graph` is obviously more advantageous from this perspective as it can timely discover dynamic changes in the dependencies of time series as demonstrated in Fig.6.

TABLE II
GRANGER CAUSALITY TESTS TOWARD MEM-UTIL AND CPU-UTIL SERIES IN THE CLOUD SERVICE DATA.

mem-util:

Ind. Time series	F-statistic	P-value
pkg-rx	147.1	9.20E-122
pkg-tx	27.3	1.66E-22
cpu-util	8.6	5.40E-07

cpu-util:

Ind. Time series	F-statistic	P-value
pkg-rx	1763.7	0
pkg-tx	431.1	0
mem-util	2.78	0.025

b) Prediction Accuracy: Table I shows forecasting accuracy comparison among all models on the Cloud Service Data, and Manufacturing Data. Although there are no big gaps among all examined methods, we still observe better performance for RNN models over the `VAR` model in both datasets. Compared to the `RNN-residual` and `RNN-vanilla`, our model performs more accurate on the Cloud Service Data, especially on the two principle time series mem-util and cpu-util.

V. CONCLUSIONS

In this paper, we present a novel neural network model called `seq2graph` that develops recurrent neural networks (RNNs) for discovering both time lagged behaviors as well as inter-timeseries dependencies during the *inference* phase, which contrasts it to majority of existing studies that focus only on the forecasting problem. We extend the conventional RNNs and introduce a novel network component of dual-purpose RNN that decodes information in the temporal domain

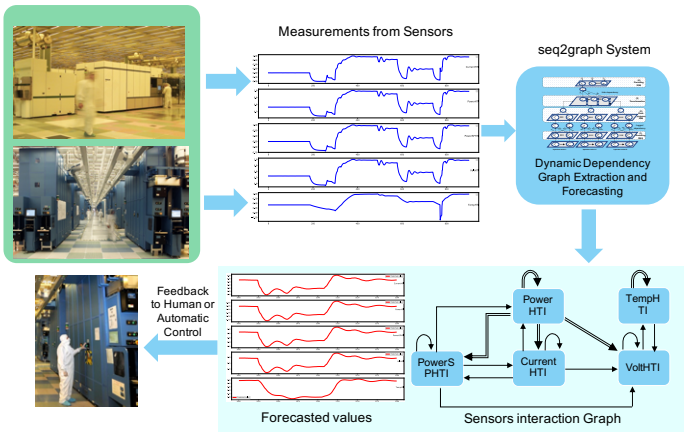

Fig. 8. Application of our model to manufacturing data: It analyzes sensor readings from different components of manufacturing plant and discovers dynamic dependencies among plant's components. The model also forecasts their future behaviors and patterns which are provided as feedback to human or used to automatically take appropriate actions timely, for instance "shutdown" for the safety reason[4].

to discover time lagged dependencies within each time series, and encodes them into a set of feature vectors which, collected from all component time series, to form the informative inputs to discover inter-dependencies. All components in `seq2graph` are jointly trained which allows the improvement in learning one type of dependencies immediately impact the learning of the other one, resulting in the overall dependencies discovery. We empirically evaluate our model on a non-linear, volatile synthetic dataset and two real-world systems generating multivariate time series. The experimental results show that `seq2graph` successfully discovers time series dependencies, uncovers the dynamic insights of the multivariate time series (which are missed by other established models), and provides highly accurate forecasts.

In future, we aim to explore wider applications of our model and scale it to larger numbers of input time series. Though several ablation studies showed that omitting either of components in our RNN-based model did not lead to the satisfied discovery of time lagged and inter-dependencies, it is worth to further investigate this issue with other successful network structures such as the convolution neural networks (CNNs) [39] or the recently developed Transformer [40]. We consider these as potential research directions.

[4]Plant pictures source https://www.zdnet.com/pictures/inside-ibms-300mm-chip-fab-photos/

REFERENCES

[1] S. Y. Shah, Z. Yuan, S. Lu, and P. Zerfos, "Dependency analysis of cloud applications for performance monitoring using recurrent neural networks," in *IEEE Big Data*, 2017, pp. 1534–1543.

[2] F. Karim, S. Majumdar, H. Darabi, and S. Harford, "Multivariate lstm-fcns for time series classification," *Neural Networks*, vol. 116, pp. 237–245, 2019.

[3] Q. Wen, J. Gao, X. Song, L. Sun, H. Xu, and S. Zhu, "Robuststl: a robust seasonal-trend decomposition algorithm for long time series," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 5409–5416.

[4] S. A. Raza, N. Shah, and A. Sharif, "Time frequency relationship between energy consumption, economic growth and environmental degradation in the united states: Evidence from transportation sector," *Energy*, vol. 173, pp. 706–720, 2019.

[5] C. B. Rjeily, G. Badr, A. H. El Hassani, and E. Andres, "Medical data mining for heart diseases and the future of sequential mining in medical field," in *Machine Learning Paradigms*. Springer, 2019, pp. 71–99.

[6] Z. Gao *et al.*, "Complex network analysis of time series," *Europhysics Letters*, vol. 116, no. 5, p. 50001, 2017.

[7] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. Springer-Verlag, 2016.

[8] R. Sipos, D. Fradkin, F. Moerchen, and Z. Wang, "Log-based predictive maintenance," in *ACM SIGKDD*, New York, USA, 2014, pp. 1867–1876.

[9] S. Y. Shah, X. Dang, and P. Zerfos, "Root cause detection using dynamic dependency graphs from time series data," in *2018 IEEE International Conference on Big Data*, Dec 2018, pp. 1998–2003.

[10] L. Kilian and H. Lütkepohl, *Structural vector autoregressive analysis*. Cambridge Uni Press, 2017.

[11] M. Eichler, "Granger causality and path diagrams for multivariate time series," *Journal of Econometrics*, vol. 137, no. 2, pp. 334–353, 2007.

[12] G. Magda *et al.*, "Learning predictive leading indicators for forecasting time series systems with unknown clusters of forecast tasks," *Proc. of ML Research*, 2017.

[13] J. Yin *et al.*, "Cloudscout: A non-intrusive approach to service dependency discovery," *IEEE TPDS*, 2017.

[14] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long-and short-term temporal patterns with deep neural networks," *arXiv:1703.07015*, 2017.

[15] K. Cho *et al.*, "On the properties of neural machine translation: Encoder-decoder approaches," *SSST-8*, 2014.

[16] N. Wichitaksorn, "Analyzing multiple vector autoregressions through matrix-variate normal distribution with two covariance matrices," *Communications in Statistics-Theory and Methods*, pp. 1–13, 2019.

[17] H. Salehinejad *et al.*, "Recent advances in recurrent neural networks," *arXiv preprint arXiv:1801.01078*, 2017.

[18] H. Goel, I. Melnyk, and A. Banerjee, "R2n2: Residual recurrent neural networks for multivariate time series forecasting," *arXiv preprint arXiv:1709.03159*, 2017.

[19] H. Takeda, Y. Tamura, and S. Sato, "Using the ensemble kalman filter for electricity load forecasting and analysis," *Energy*, vol. 104, pp. 184–198, 2016.

[20] L. F. Tratar and E. Strmčnik, "The comparison of holt–winters method and multiple regression method: A case study," *Energy*, vol. 109, pp. 266–276, 2016.

[21] D. Hsu, "Time series forecasting based on augmented long short-term memory," *arXiv preprint arXiv:1707.00666*, 2017.

[22] X. Chen, K. Chau, and A. Busari, "A comparative study of population-based optimization algorithms for downstream river flow forecasting by a hybrid neural network model," *Engineering Applications of AI*, vol. 46, pp. 258–268, 2015.

[23] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[24] K. Chakraborty, K. Mehrotra, C. K. Mohan, and S. Ranka, "Forecasting the behavior of multivariate time series using neural networks," *Neural networks*, vol. 5, no. 6, pp. 961–970, 1992.

[25] B. Choubin *et al.*, "Multiple linear regression, multi-layer perceptron network and adaptive neuro-fuzzy inference system for forecasting precipitation based on large-scale climate signals," *Hydrological Sciences Journal*, vol. 61, no. 6, pp. 1001–1009, 2016.

[26] M. Ardalani-Farsa and S. Zolfaghari, "Chaotic time series prediction with residual analysis method using hybrid elman–narx neural networks," *Neurocomputing*, vol. 73, no. 13-15, pp. 2540–2553, 2010.

[27] A. Rajkomar, "Scalable and accurate deep learning with electronic health records," in *Digital Medicine*, 2018.

[28] E. Choi *et al.*, "Retain: An interpretable predictive model for healthcare using reverse time attention mechanism," in *NIPS*, 2016, pp. 3504–3512.

[29] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzel, "Learning to diagnose with lstm recurrent neural networks," *ICLR*, 2016.

[30] G. P. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model," *Neurocomputing*, 2003.

[31] M. Khashei and M. Bijari, "A novel hybridization of artificial neural networks and arima models for time series forecasting," *Applied Soft Computing*, 2011.

[32] J. Hu, J. Wang, and G. Zeng, "A hybrid forecasting approach applied to wind speed time series," *Renewable Energy*, vol. 60, pp. 185–194, 2013.

[33] K. Cho *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *EMNLP*, 2014.

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.

[35] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ICLR*, 2015.

[36] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *Proc. of the Empirical Methods in NLP*, 2015.

[37] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ICLR*, 2015.

[38] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[39] X. Dang, R. Akella, S. Bahrami, V. Sheinin, and P. Zerfos, "Unsupervised threshold autoencoder to analyze and understand sentence elements," in *2018 IEEE International Conference on Big Data*, Dec 2018, pp. 3267–3276.

[40] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.