pre-print

Dependency Analysis of Cloud Applications for Performance Monitoring using Recurrent Neural Networks

Syed Yousaf Shah*, Zengwen Yuan†, Songwu Lu†, Petros Zerfos*
* IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598
Email: {syshah, pzerfos}@us.ibm.com
† University of California, Los Angeles, Los Angeles, CA 90095
Email: {zyuan,slu}@cs.ucla.edu

***Abstract*—Performance monitoring and logging systems generate huge amounts of time series data and are a major component of the cloud infrastructure. Cloud providers rely heavily on accurate forecasting and analysis of the monitoring data for tasks such as resource and performance management, anomaly detection and meeting customer SLAs etc. Traditionally analytics systems use statistical forecasting models for such resource management tasks without utilizing the runtime interdependencies that exists among different monitoring metrics. Finding such interdependencies is challenging but when known they can highly increase the accuracy of the analytics systems. In this paper, we propose a deep learning based management system that first discovers the hidden dependencies in large scale monitoring data and uncovers the runtime dependency graph of the cloud applications. The system then feeds such information as extra features to a deep learning based multivariate forecasting model to more accurately forecast time series values for analytics tasks. We propose new additions to the Long Short Term Memory (LSTM) networks that enable us to extract the hidden relationships in the data. We have run our system on data sets from two different commercial cloud platforms and the results show that such interdependency information is very crucial to improve the accuracy of analytics tasks such as application performance forecasting and anomaly detection. Moreover, our experimental results show that our proposed deep learning based system outperforms traditional statistical models based system by accurately forecasting time series values, particularly for highly variable data.**

***Index Terms*—Big Data; Machine Learning; Statistical learning; Supervised learning; Time series analysis; Neural networks;**

I. Introduction

Modern cloud-native applications [1] follow a micro-services based design [2] that decomposes the application logic into a several interacting component services, which are often independently developed and deployed in a cloud hosting environment. This approach to cloud application development aims to offer isolation in terms of component reliability and scaling, increased portability regarding the hosting (execution) environment, as well as continuous development-to-deployment for maximum agility. It enables rapid development and iteration throughout the cloud application lifecycle, resulting in reduced delivery times and improvements that reach end users in a more timely manner.

However, cloud-native applications that make use of micro-services also turn the task of application performance monitoring substantially more complicated and challenging [3]: visibility into key performance indicators and monitoring metrics of the various application components becomes very limited, as they are developed and operated by disparate teams. The reduced visibility into the application structure is compounded by the fact that the deployment environment is a cloud hosting provider, with often seperately owned and managed infrastructure, platform and application layers. Hence, domain-knowledge regarding the deployment environment and the network application structure might simply not be available. This is further complicated by the fact that resource management objectives from the cloud infrastructure, platform and application layers might be conflicting and continuously changing, resulting in a dynamic environment that, for monitoring purposes, quickly renders obsolete any *a priori*, static [4] application dependency configuration. A monitoring system that *learns* the runtime dependencies amongst the cloud application components is necessary in order to monitor the performance of the heterogenous, dynamically changing and opaque cloud-native environments.

Towards achieving the above goal, various statistical analysis and machine learning-based techniques have been proposed in the literature. They typically follow a black-box (i.e., without *a priori* domain knowledge) approach through log analysis and mining, and discover relationships amongst key performance indicators using various established statistical characteristics such as Granger causality [5], pair-wise correlations [6] [7], and clustering [8]. In this work, we expand upon this line of research by exploring the application of advances in machine-learning models, namely recurrent neural networks [9], which have shown great promise in addressing the limitations of prior approaches: ability to model only linear relationships ([5]), need for extensive feature pre-processing ([6], [7]), and sensitivity to aberrant data measurements ([8]).

More specifically, we propose a novel use of Long-Short Term Memory (LSTM) [10] recurrent neural networks (RNNs), which excel in capturing temporal relationships in multi-variate time series data, accurately modeling long-term

dependencies, and being resilient to noisy pattern representations. We apply LSTM modeling using the time series data that is collected as part of the performance monitoring of various key performance indicators, which spans over any of the infrastructure/platform/application layers of the cloud applications software stack. We further develop two new techniques for analyzing dependencies in cloud application components using LSTMs: first, a generic method for extending the feature vectors used as input to LSTMs, which directs the neural model to learn relationships of interest between the monitoring metrics. Second, a novel approach that looks into the *weights* of the neural connections that are learned through the training phase of LSTMs, to uncover the actual dependencies that have been learned. In a departure from classical application of LSTM modeling, the latter technique examines the *neural model itself* as opposed to its output, in order to uncover dependencies of performance metrics that characterize the various application component services.

We evaluate our approach through controlled experiments that we conduct by deploying and monitoring the performance of a sample cloud application serving trace-driven workloads, as well as by analyzing a data set of measurements obtained from the monitoring infrastructure of a public cloud service provider. Three monitoring use cases were considered: (1) finding strongest performance predictors for a given metric, (2) discovering lagged/temporal dependencies, and (3) improving accuracy of forecasting and dynamic baselining for a given metric using its strongest predictors (early performance indicators). Our results demonstrate that the use of LSTMs not only improves accuracy of dynamic baselining and forecasting by a factor of 3–10 times, as compared to more classical statistical forecasting techniques such as ARIMA and Holt-Winters seasonal models [11], but is also able to discover service component dependencies that concur with the findings of the well-established Granger analysis [5].

In summary, this work makes the following contributions:

- a novel application of Long-Short Term Memory networks for dynamically analyzing cloud application component dependencies, which is robust to noisy data patterns and able to model non-linear relationships.
- an analysis on three use cases for the use of LSTMs in the performance monitoring setting, namely the identification of early performance indicators for a given metric, discovery of lagged/temporal relationships among performance metrics, as well as improvements in dynamic baselining & forecasting accuracy through the use of additional indicators.
- performance evaluation of the approach through controlled experiments in a cloud-deployed testbed, as well as through real-world monitoring measurements from an operational public cloud service provider.

The remainder of this paper is organized as follows: Section II provides background information on recurrent neural networks (RNNs), the basic modeling technique used in this work. Section III describes our proposal for cloud application monitoring using RNNs, whereas Section IV provides performance evaluation results through controlled experiments and analysis of real-world cloud measurement datasets. Section V discusses various issues that arise through the application of our proposed technique. Section VI provides review of the related research literature and finally Section VII concludes this paper.

II. Background

Deep neural networks such as *Recurrent Neural Networks (RNNs)* have been shown to be very effective in modeling time series and sequential data. The *Long Short-Term Memory (LSTM)* networks [10] are a type of RNNs that are suitable for capturing long-term dependencies in sequential data, a property that makes them particular appealing for our application domain. A brief description of their design follows, while the interested reader is referred to [10] for more details. Neurons (also called "cells" in neural networks) are organized in layers that are connected with one another. The first (input) layer receives the input data, computes weighted sums on it and then applies an activation function (e.g. $tanh$, $sigmoid$, etc.) that produces an output, which is then consumed by the neurons or cells in the next layer and/or itself, as in the case of RNNs. During the training phase, the network tries to learn the weights that minimize the error between the final output of the network and the real value of the data.

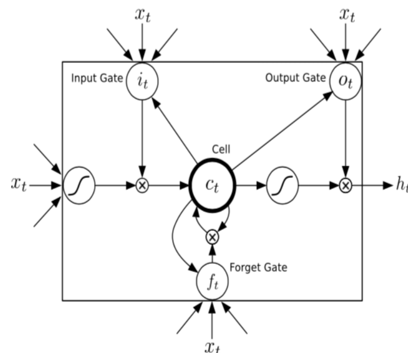

Fig. 1. Architecture of a cell in LSTM networks

In the case of LSTMs, the basic building block is the cell as shown in Fig. 1 [12], which has multiple gates i.e., *Input*, *Output*, and *Forget* gates that control what information is maintained, forgotten and provided as output by the cell, respectively. Multiple cells are chained together in a repeating structure and each connection carries an entire vector. The *Cell* stores values and acts as memory that is reused over time. Of specific interest to our work is the *Input* gate, which decides on what new values are important to be stored in the *Cell*. It is essentially a non-linear function σ applied to the weighted sum of the current input values (x_t), feedback from the previous stage (h_{t-1}) and a bias b_i (see eq. 1).

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (1)$$

III. Method and System Design for Dependency Analysis using RNNs

To use LSTMs for dependency analysis of service components, we first extend the LSTM model to extract information relevant to this task. In the following, more details are provide on these LSTM extensions, as well as an overview of the system architecture that implements the overall approach.

A. Extensions to LSTM Model

The main idea is to use the training mechanism of *LSTM* to learn dependencies among performance metrics of service components that the user might be interested in. In order to achieve this, we introduce a new addition to the *Input gate*. This extra input consists of a *Probe Matrix 'V'* and the features data matrix *'d'*. We construct the *'d'* matrix using a user defined *Feature Learning Function* that represents the type of dependency the user is trying to discover for a target variable. A target variable is the variable for which we would like to find dependencies. Equation 2 shows the updated *Input gate* that we use to extract dependencies.

$$i_t = \sigma(W_i x_t + V_i d_t + U_i h_{t-1} + b_i) \tag{2}$$

With the new design of the *Input gate*, we feed the network with input data and the *'d'* matrix. The matrix *'d'* works as matrix of extra features that we feed into the network and during training network uses more relevant features accordingly. The output of the network is the forecasted values of target variable for which the user wants to discover dependencies. Once the network is trained, we extract back the *'V'* matrix and analyze it. During the training process, the LSTM network assigns higher weights to the features in the *'d'* matrix that are more relevant to the target variable. This way we can infer that the target variable has stronger dependency on features or variables *'d'* with higher weights assigned to them. In the following subsections, we explain how to build the matrix *'d'* using user defined *Feature Learning Function* and how we summarize the *'V'* matrix called as *Probe Matrix*.

1) Feature Learning Function: We name the function that is used to build feature matrix *'d'* as *Feature Learning Function* because that is the function used to generate features from the data that represent dependencies in the data. Generally speaking, the learning function can be written as in eq 3.

$$d = g(X, Y) \tag{3}$$

X and *Y* represent time series data sets on which learning function 'g(.)' is applied.

In scope of this paper, we compute *'d'* for two types of dependencies, (i) Finding strongest predictors or direct dependency among variables (ii) Finding time-lagged dependency among variables. For (i) and (ii) the learning functions and matrix *'d'* notations are shown by eq 4 and eq 5 respectively.

$$\begin{aligned} g(a, b, .., z) &= [a_t, b_t, .., z_t] \quad where\ a, b, ..., z\ are\ predictors \\ d_t &= [metric_t^1, metric_t^2, metric_t^3, ...] \end{aligned} \tag{4}$$

$$\begin{aligned} &d = g(X, Y) \\ &d_t = [metric_t^x, metric_{t-1}^x, ..., metric_{t-h}^x]\ where\ h\ is\ the \\ &\qquad lag\ horizon \end{aligned} \tag{5}$$

2) The Probe Matrix: The *Probe Matrix* is essentially the weight matrix associated with the feature data matrix generated using *Feature Learning Function*. The dimensions of the *Probe Matrix* depend on dimensions of the data matrix *'d'*. During neural network training phase the values of this *Probe Matrix* are updated by the error propagation algorithm e.g., SGD (Stochastic Gradient Dissent) and once the training is finished we extract the *Probe Matrix* with final values. In the *Probe Matrix* we have weights corresponding to each feature in the matrix *'d'* and we have one such *Probe Matrix* for each cell or neuron in the network. These *Probe Matrices* can be analyzed in different ways but for this paper we average out weights for each feature across all the neurons or across all *Probe Matrices*. The eq. 6 shows an example of how problem matrix V_i and data matrix d_t are combined and fed to network using eq 2

$$V_i d_t = \begin{bmatrix} w_1 \\ ... \\ w_n \end{bmatrix} . \begin{bmatrix} metric^1 & metric^2 & ... & metric^n \end{bmatrix} \tag{6}$$

B. System Architecture for RNN-enabled Cloud Monitoring

The system architecture that utilizes *LSTM* network both for extracting dependencies and then using that information in time series forecasting is shown in Figure 3. It ingests performance measurements from monitoring systems collected at various layers of the cloud stack (infrastructure/platform/application). When new monitoring data from any layer is available, the system prepares it for model (re)training. It generates the feature matrix *'d'* using the user defined *Learning Function*, as described in section III-A1 and initializes the probe vector *'V'*. It then feeds the feature matrix *'d'* along with the monitoring metric for which the forecasting is desired to the *LSTM* neural network.

Once the network is trained and the training error converges to a desired minimum level, the system makes available the probe vector weights to the *Probe Matrix Analyzer*, which then produces runtime dependencies, e.g., list of strongest predictors, that exists in the input data for the desired monitoring metric. This dependency knowledge can further be used as additional features for future model training and to produce models that can more accurately forecast a monitoring metric, as will be shown in the respective use case in Section IV. The *Data Preparation for Forecasting* module prepares data for forecasting on a trained model that was produced based on the runtime dependency graph information. The forecasted values are used by the *Analytics Systems* to perform various tasks, such as anomaly detection on application or infrastructure-level metrics, detecting SLA violations, application performance tuning etc. The model is continuously updated and and

retrained at intervals defined by how dynamic is the cloud application and infrastructure. For example, for applications or infrastructures that are highly elastic and their usage is very dynamic, the dependency graphs might need to be extracted more often than those applications that have more stable applications or workloads.

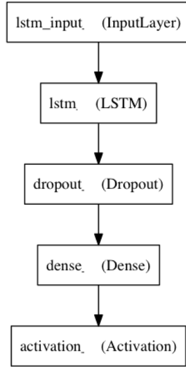

Fig. 2. LSTM Based Neural Network Layers

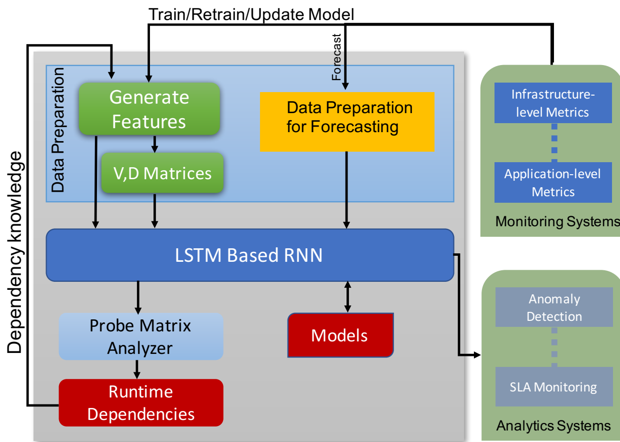

Fig. 3. RNN Enabled Cloud Management System

IV. Experimentation and Performance Evaluation

We conducted a series of experiments on the two datasets using our updated *LSTM* model and compared it to popular statistical techniques. Figure 2 shows our neural network model, our *LSTM* layer consists of 64 cells. For training we use 500 epochs, the activation function as $tanh$, loss function as MSE and for optimizer we use $RMSProp$ [13], all these options are available in *Python Keras* library that we used for implementation. The data was log-dransforemed before we ran the experiments on it.

We performed experiments on two types of datasets explained in the following sections. These datasets are not only different in nature based on their variability but are also obtained from two different infrastructures. Our experiments show that our system performs very well in both the situations.

A. Experiment 1: Measurement Data of Sample Cloud App Deployed in AWS (DataSet-I)

Our first dataset consists of data that is an emulation of a multi-tiered cloud application following traffic patterns obtained from real world traces from an ISP (Internet Service Provider). This dataset has a more predictable traffic pattern and corresponding time series measurement data. Figure 4 shows the architecture of our traffic generation and data collection platform. We use *Amazon Web Services (AWS)* [14] as our cloud provider where we deploy the testbed. The testbed has a front-end *Python Flask* web-application that runs in an *AWS EC2 instance* and a backend *Amazon RDS Database*, which runs in a separate *AWS EC2 instance*. We collected cloud performance data using *AWS Cloud Watch* and implemented a python web service client, which automatically downloads the cloud monitoring data (total of 11 metrics shown in figure 4 for application and database) using the *AWS CloudWatch* APIs. The monitoring data is collected every minute, so the dataset has a data point for every minute.

In order to generate load on the cloud application, we use *Apache JMeter [15]* to send *REST* requests to our *Python-Flask* web-application. For realistic load, we used *Clark-Net HTTP Dataset [16]*, which are traces collected from an *ISP* in the metro Baltimore-Washington DC area. The *Clark-Net HTTP Dataset [16]* contains mostly *Http GET* requests. We transformed the traces so that they can be sent as *REST requests* by the load generator. In transforming traces, we made sure that the *REST* requests are replayed as per the original timestamps and intervals thus maintaining the traffic pattern of the original traces. The *Python-Flask* web-application accepts requests and based on the request type it either retrieves values from the backend database or inserts new values into the database. Since most of the requests in the original data were *Http Get*, we used random sampling to convert some of them to *Http POST* requests so that, we can emulate data insertion in the database as well as retrieval from the database.

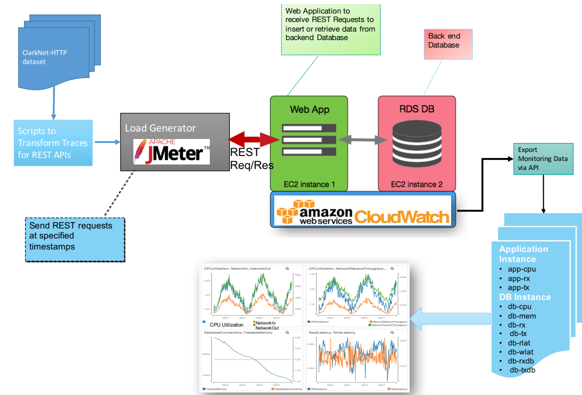

Fig. 4. Data Generation Testbed

The figure 5 shows the plots of collected metrics data from our deployed testbed.

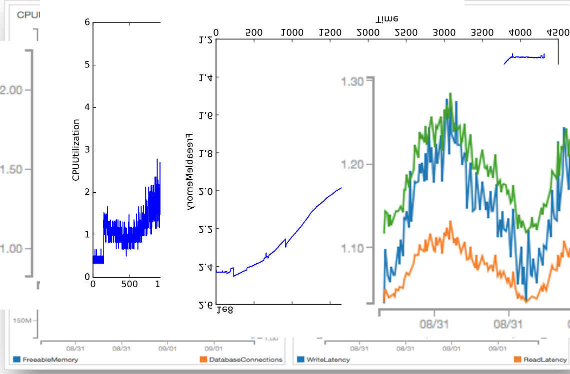

Fig. 5. DataSet-I plot

1) Use case 1: finding strongest predictors: In our DataSet-I, we want to infer the application graph and find out strongest predictor for our metric of interest, which in this case is *AWS RDS's Network Rx Throughput (db-rx)*. Using our proposed system we trained the neural network model with the aim to reduce *Mean Squared Error (MSE)* for forecasting *AWS RDS's Network Rx Throughput (db-rx)*. We extracted the *Probe Matrix* and analyzed it. Figure 6 shows a heat map of our analyzed *Probe Matrix* with neurons (or cells) on the x-axis and predictors on y-axis. The average weights across all 64 neurons for a particular predictor are also shown along y-axis legend. The purple box shows the strongest predictors for *db-rx* namely, *app-cpu,app-rx,app-tx*. The network assigned on average high weights to these features/metrics as compared to other features that were supplied to the network. We generated our feature matrix *'d'* using function *'g'* shown in eq. 4 and provided all the performance metrics that we collected.

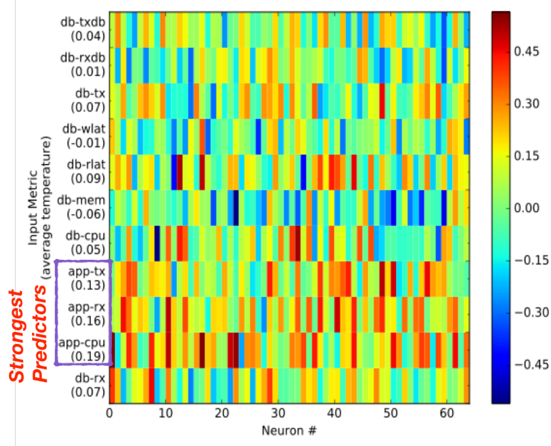

Fig. 6. Strongest Predictors for db-rx

From figure 6, we can also infer the application performance runtime dependency graph for *AWS RDS's Network Rx Throughput (db-rx)* to be dependent on the strongest predictors. Figure 7 shows the application graph inferred based on *Probe Matrix* analysis.

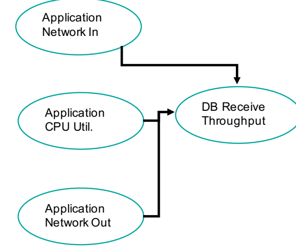

Fig. 7. Runtime Application Performance Graph

In order to verify our findings regarding the *strongest predictor*, we analyzed the metrics data using the well-known statistical technique of *Granger's Causality*. Since, the actual *Granger Causality* finds dependences between only pairs of variables, we computed *Granger Causality* between *db-rx* and every other variable. The top three predictors for *db-rx* found by *Granger Causality* method overlaps with findings from our method. The results are shown in table in figure 8. Note that our *Granger Causality* test was based on *F-Statistics*, so we are selecting top predictors based on the *F-Statistic* value. The results indicate that the top predictors from our mechanism match quite well with those obtained through the *Granger Causality* test.

Independent variable	F Statistic Value	p-value or Pr(>F)
App CPU Utilization (app-cpu)	4794.8	2.2e-16 ***
App Packet Received(app-rx)	14346	2.2e-16 ***
App Packet Transmitted (app-tx)	10994	2.2e-16 ***
DB Server CPU Utilization (db-cpu)	166.94	2.2e-16 ***
DB Server Memory Utilization (db-mem)	184.9	2.2e-16 ***
DB Server Read Latency (db-rlatency)	5.4615	0.0196 *
DB Server Network Transmit Throughput (db-tx)	11.714	0.0006413 ***
DB Read Throughput (db-rxdb)	2.0672	0.1508
DB Write Throughput (db-txdb)	46.489	1.458e-11 ***
DB Server Write Latency (db-wlatency)	12.827	0.0003554 ***

Strongest Predictors

Significance codes: 0 '***' 0.1% '**' 1% '*' 5%

Fig. 8. Granger's Causality Test Results

2) Use case 2: finding temporal & lagged depenencies: For this use case, the goal is to learn lagged dependencies of the metrics in the DataSet-I. We trained the neural network model with the aim to reduce *Mean Squared Error (MSE)* for forecasting *AWS RDS's Network Rx Throughput (db-rx)*. The goal is to find out if there is any lagged dependency between *db-rx* and *Application Packet Received(app-rx)*. We extracted the *Probe Matrix* and analyzed it. Figure 9 shows a heat map of our analyzed *Probe Matrix* with neurons or cells on x-axis and lagged predictors on y-axis, the average weights across all 64 neurons for particular lagged predictor are also shown along y-axis legend. We generated our feature matrix *'d'* using eq 5, with *h=5*. As one can see from the results, *db-rx* has stronger lagged dependency on *app-rx* at *t-1* and *t-2* in past 5 values of *app-rx*.

Figure 10 shows results of the *Granger Causality* test and these results confirm that *db-rx* has a strong dependency on

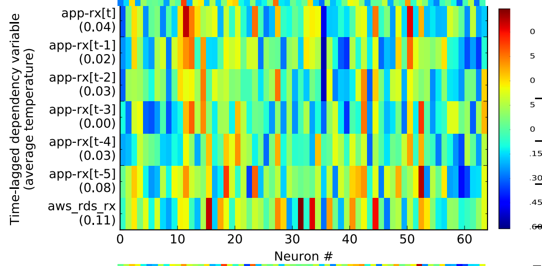

Fig. 9. Lagged Dependencies for db-rx on ap-rx

app-rx and its past values. For *t-3* our mechanism does not show strong dependence, as the average of the weights is 0Note that if there is causal relationship between two variables, *Granger Causality* might show diminishing effect on past values.

Independent variable	Order	F Statistic Value	p-value or Pr(>F)
App Packet Received(app-rx)	1	14346	2.2e-16 ***
	2	6504.8	2.2e-16 ***
	3	4244.8	2.2e-16 ***
	4	3245.8	2.2e-16 ***
	5	2593.6	2.2e-16 ***

Fig. 10. Granger's Causality Test Results for Lagged Dependencies in DataSet-I

3) Use case 3: more accurate forecasting using strongest predictors: Our proposed system can analyze application run-time dependency information and we can use this information to further improve the forecasting accuracy on time series data. We applied well-known statistical forecasting algorithms on our DataSet-I, to evaluate their performance. More specifically, we used *ARIMA, BATS, HoltWinters Additive and HoltWinters Multiplicative* [17] algorithms to forecast values of *AWS RDS's Network Rx Throughput (db-rx)*, the results of which are shown in Table I. As one can observe from the results, the statistical models perform well on this particular data set, since the pattern in the data is predictable so most of the statistical models can forecast values with around *22% MAPE (Mean Absolute Percentage Error)*. We also used *LSTM* recurrent neural network model named as *LSTM(U)* in the table for forecasting and, as shown in the table, it performs slightly better than the statistical models. However, the *LSTM(U)* model acts as univariate model in this particular experiment and makes use of only one variable, i.e., *db-rx*, to forecast its future values. Figure 11 shows the plot of true value and predicted values by *BATS* algorithm.

TABLE I
FORECASTING USING STATISTICAL MODELS AND RNN

Error Measure	ARIMA	BATS	Holt-Winters Additive	Holt-Winters Multiplicative	LSTM (univariate)
MSE	206867	215049	211939	194122	196678
MAPE	22.17%	23.09%	22.82%	21.92%	21.44%

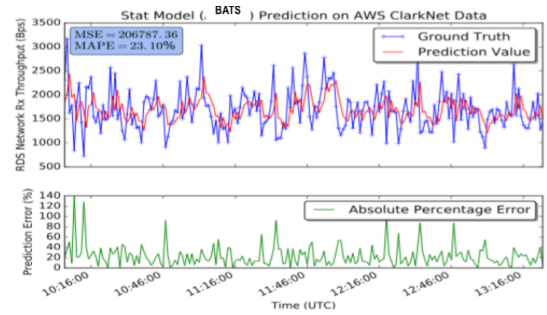

Fig. 11. Forecasting values using BATS Algorithm

We applied our followed our LSTM-based approach to the DataSet-I and added all the variables as input to the model, essentially treating is as multi-variate; we immediately notice that it improves upon the prior results. The details are shown in Table II and Figure 12 shows the plot of the performance of the multivariate LSTM model, when forecasting future values for *db-rx*; it decreases *MAPE* to 7% from the best univariate value of 21%. Note that the results shown in Table II make use of all (11) the metrics as features that were included in matrix *'d'*.

TABLE II
FORECASTING ERROR COMPARISON BETWEEN DIFFERENT MODELS

Error Measure	Statistical Models (BATS)	LSTM (univariate)	LSTM (multivariate)
MSE	200 K	196 K	19 K
MAPE	23.09%	21.44%	6.88%

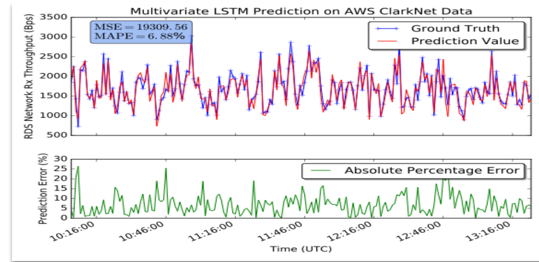

Fig. 12. Forecasting values using RNNs

Using our mechanism, we can learn the strongest predictors for *db-rx* and use only those to improve the forecasting error, while reducing the amount of data that is provided as input to the neural network. By inferring application runtime dependency information, we introduce only the strongest predictors and the results show that we can significantly lower the *MAPE* value and achieve high accuracy in forecasting of the time series data. Table III shows results from reduced number of predictors based on dependency information that we extracted using our mechanism. Results in Table III show that we can achieve approximately same low *MAPE* with 4 effective metrics and 3 strongest predictors namely *app-cpu, app-rx and app-tx*. Table III also shows that we can forecast better than

the best statistical model with lower *MAPE*, using only 1 other strongest predictor which was extracted using our method.

TABLE III
FORECASTING USING STRONGEST PREDICTORS

Error Measure	All metrics (11)	4 effective metrics (db-rx, app-cpu, app-rx, app-tx)	2 effective metrics (db-rx, app-cpu)
MSE	19 K	22 K	44 K
MAPE	6.88%	7.27%	10.10%

The improved forecasting achieved using our LSTM-based approach can be used for various cloud performance monitoring tasks such as anomaly detection. An example of anomaly detection for *RDS Network Receive Throughput* is plotted in figure 13, which shows that we can detect anomalies with higher accuracy in *RDS Network Receive Throughput* and act accordingly. In figure 13, we used $mean \pm 3\sigma(StandardDeviation)$ values to calculate *upper* and *lower* bounds for error margins.

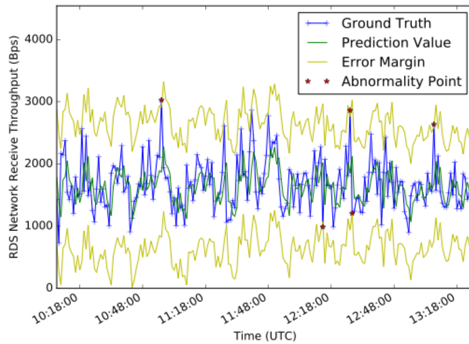

Fig. 13. Anomaly detection in DataSet-I using RNN Enabled Cloud Application Management

B. Experiment 2: Analysis of Operational Measurements from a Public Cloud Service Provider (DataSet-II)

Our second dataset contains real world cloud performance data obtained from a major cloud service provider. This dataset contains application/service-level traces, measured from the container infrastructure of the cloud provider. The data is highly variable, as micro-services are invoked by other micro-services, as per real world usage. Figure 14 shows the plots for *data set II*, consisting of *Cpu Utilization*, *Network Tx*, *Network Rx* and *Memory Utilization*. The metrics were collected every 30 seconds.

We evaluated the same set of use cases as in Section IV-A, using *data set II*. As we can see from figure 14, time series data in *data set II* is very volatile and not as predictable as in *data set I*, even though the data shown in figure 14 log-transformed, a common technique in time series analysis that reduces variability.

1) Use case 1: finding strongest predictors: From data set II, we are interested in finding the strongest predictor for *cpu utilization*. Figure 15 shows the heat map of analyzed *Probe Matrix* with neurons (or cells) on x-axis and predictors on y-axis. The average weights across all 64 neurons for a particular

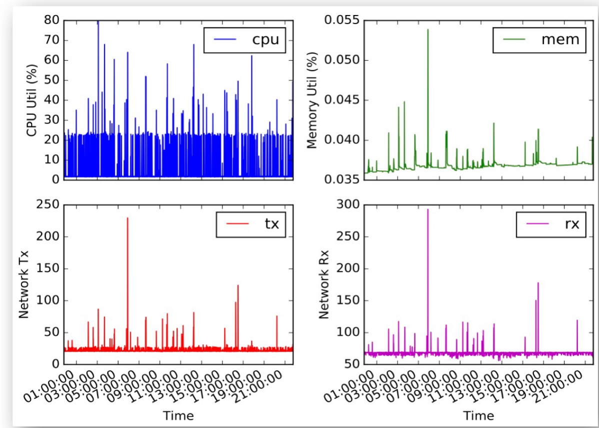

Fig. 14. Data Set-II plot

predictor are also shown along y-axis legend. We can see from the plot that the *network-tx* is the strongest predictor for *cpu*. The neural network learning process assigned high weights to the *network-tx* metric, as compared to other features that were used as input. We generated our feature matrix *'d'* using eq. 4 and provided all the four performance metrics that we received in *data set II*. We also performed *Granger Causality* test to verify the strongest predictor and we can see from figure 17 that *cpu* has indeed the strongest dependency on *network-tx*.

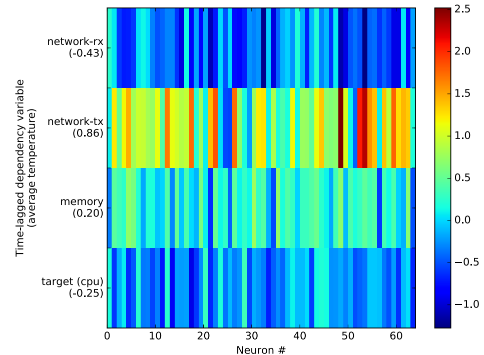

Fig. 15. Strongest Predictors for cpu

2) Use case 2: finding temporal & lagged depenencies: Similar to *use case 2* in Section IV-A, we are interested in finding the lagged dependency of *cpu* on *network-tx*. As shown by the heat map of *Probe Matrix* in Figure 16, *cpu* has lagged dependency on past values of *network-tx* except for *t-3*, for which our model shows weak dependency (we only checked for last 5 values i.e., *h=5*). For validation, the *Granger Causality* test results shown in figure 17 seem to confirm our findings.

3) Use case 3: more accurate forecasting using strongest predictors: As in *use case 3* in the previous Section, we generated well-known statistical forecasting models and as well as *LSTM* on data set II and measured their predictive

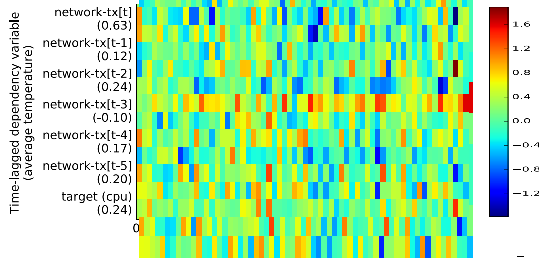

Fig. 16. Lagged Dependencies for cpu on network-tx

Independent variable	Order	F Statistic Value	p-value or Pr(>F)
#Packets Transmitted (network-tx)	1	2303.1	2.2e-16 ***
	2	1164.2	2.2e-16 ***
	3	789.87	2.2e-16 ***
	4	570.88	2.2e-16 ***
	5	464.63	2.2e-16 ***
#Packets Received (network-rx)	1	0.0673	0.7953
	2	7.1696	0.0007896 ***
	3	5.7471	0.0006516 ***
	4	5.0198	0.000502 ***
	5	6.5022	5.248e-06 ***
Memory Utilization (memory)	1	8.3326	0.003936 **
	2	6.5575	0.00145 **
	3	4.8716	0.002229 **
	4	4.4349	0.001432 **
	5	3.2734	0.006007 **

Strongest Predictors

Significance codes: 0 '***' 0.1% '**' 1% '*' 5%

Fig. 17. Granger's Causality Test Results for Lagged Dependencies

performance. Table IV and Figure 18 show the results that we obtained. *MAPE* error is very high for this data set for all models, while the *ARIMA* model cannot even detect any discernible pattern in the data. The *LSTM(U)* is the univariate version of *LSTM* neural network model and, while it performs slightly better than the classical statistical models,it merely succeeds in capturing the mean of the data as shown in figure 19. From Table IV and Figure 19, *LSTM(U)* fails to properly model the peaks and troughs in the data, which is critical in order to detect anomalies.

TABLE IV
FORECASTING USING STATISTICAL MODELS AND RNN

Error Measure	ARIMA	BATS	Holt-Winters Additive	Holt-Winters Multiplicative	LSTM (univariate)
MSE	58.47	58.72	58.58	58.56	51.05
MAPE	227.20%	226.04%	227.13%	227.30%	61.56%

TABLE V
FORECASTING VALUES USING LSTM MODEL

Error Measure	Statistical Models (BATS)	LSTM (univariate)	LSTM (multivariate)
MSE	58	51	4
MAPE	220%	60%	24%

From the results of *use case 1* for data set II, we know that the strongest predictor for *cpu* is *network-tx*, therefore we make use of this dependency information in our *LSTM* model as an extra feature. The results, as plotted in Table V,

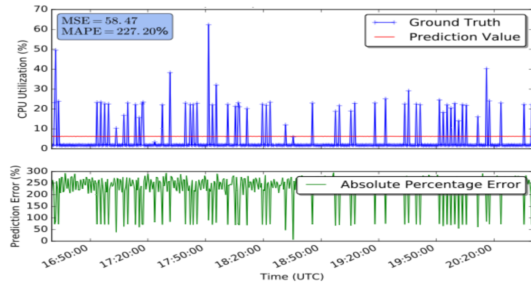

Fig. 18. Forecasting values using ARIMA Algorithm

show that this extra feature drastically improves the forecasting accuracy by decreasing the forecasting error (as measured by *MAPE*) from 220% to 24%. More importantly, they show that the improved model more is able to capture the peaks and troughs in the data, as shown in Figure 20.

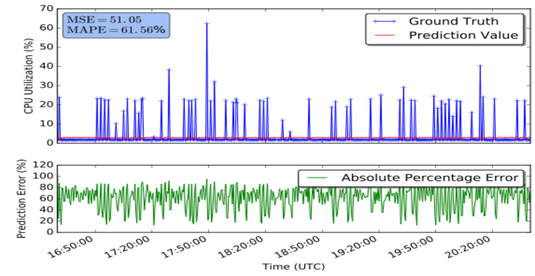

Fig. 19. Forecasting values using LSTM Model

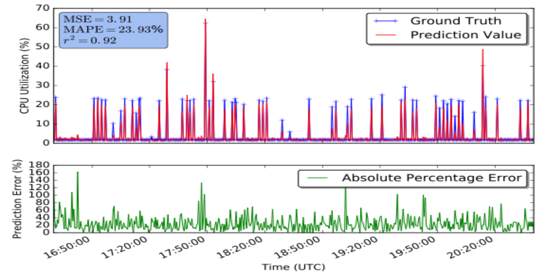

Fig. 20. Forecasting values using RNNs

The improved forecasting capability allows to more accurately perform dynamic baselining for *cpu* utilization and detect anomalies as shown in Figure 21. In Figure 21, we used $mean \pm 3\sigma(StandardDeviation)$ values to calculate upper and lower bounds for error. Accurate performance metric forecasting and ability to detect anomalies can help cloud service providers avoid SLA violations, by allocating more resources to the application in cases where *cpu* utilization is expected to go high and could impact application performance.

V. DISCUSSION

The approach for dependency analysis of cloud applications described in this work requires consideration of some practical

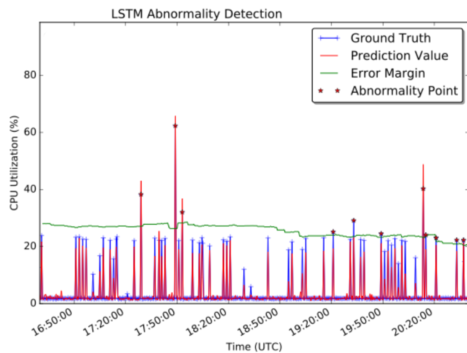

Fig. 21. Anomaly detection using RNN Enabled Cloud Application Management

aspects, towards becoming a fully automated system that can operate with minimal human supervision. In the following, we describe some of these issues and offer potential resolution.

Automating the discovery of service component dependencies: in the LSTM-based approach that we proposed, strong predictors (or indicators) for certain variable(s) of interest are found by providing all the available predictor variables as features to the LSTM network. In practice, there might be a large number of features to consider and a pre-processing mechanism would need to be applied to filter the feature subsets that will be used for further analysis. This pre-processing mechanism can be based on existing techniques e.g., clustering the features based on some criteria, or reduced number of features using autoencoders, which will be as used as input to the LSTM network.

Continuous discovery and incremental updates of the dependency model: our proposed method for service component analysis makes use of the learning process of the training phase of an LSTM network. This is somewhat static, in the sense that, to detect changes in the service component dependency graph, a new (re)training would need to be initiated. Since the training phase is computationally intensive and, depending on the available compute infrastructure, potentially time consuming, it would be worth exploring online learning techniques for LSTMs [18], which incrementally adapt the model as new input data (i.e., new measurements of performance metrics in our case) is collected and made available to the monitoring system. As future work, we plan to conduct such experiments and evaluate the ability of the model to track changes in the dependency structure of the cloud application that is being monitored. Further, domain knowledge of the cloud application's operating environment can be used to trigger full re-training, in the case of significant changes in the operating conditions, such as large-scale outages and hardware failures.

VI. Related Work

The cloud services dependency discovery has become a recent interest to research communities. Our work takes a novel RNN based log analysis and mining approach to mine such dependencies. It is thus orthogonal to manual annotation, code injection [19] [20], and network packet inference approaches [21] [22] [23].

Along the log mining line, [6], [7] and [24] adopted the pair-wise correlations of monitor metrics to calculate the distance between service components. [5] utilized Granger causality for modeling, while [8] uses clustering to calculate the service components distance. [25] categorized Google cluster hosts according to their similarity for CPU usage. Our work differs from all of them for two reasons; first, we take the recurrent neural network approach for its effectiveness in modeling non-linear relationships, simple data pre-processing and insensitivity to outliers. Second, our approach does not require *a prior* knowledge on service structure nor does it requires specific metric. It is fine-grained at micro service-level, thus differs from the component-level [24] or network-specific like Microsoft Service Map [26].

Our new LSTM-based approach is very effective and outperforms traditional statistical models in all experimental tasks that we conducted. These include Granger causality [27], [28], [29], autoregressive models such as ARIMA [30], [31], BATS and Holt-Winters method [17]. Our approach further differs from existing works that use neural networks for time series data analysis. [9] uses RNN but is focused on handling data with missing values. [32] forecasts multivariate data using artificial neural network, but the data is stationary and homogenous in their type, and it uses fee-forward network. Our work is not limited to using RNN to perform time series forecasting only [33] [34]; on the contrary, the modified neural network provides dependency for various cloud monitoring and analysis tasks.

VII. Conclusion

In this paper, we presented a new method that makes use of Long-Short Term Memory (LSTM), a popular variant of recurrent neural networks, to analyze dependencies in performance metrics obtained from monitoring of service components used by modern, cloud-native applications. By appropriately engineering the input features and looking into the learned parameters of the LSTM network once its training phase is completed, dependencies among metrics from across the cloud software stack are identified. We demonstrate the versatility of the technique by applying it in three use cases, namely the identification of early indicators for a given performance metric, analysis of lagged & temporal dependencies, as well as improvements in the forecasting accuracy. We evaluated our proposed method both through controlled experiments in a cloud application testbed deployed in AWS, as well as through analysis of real, operational monitoring data obtained from a major, public cloud service provider. Our results show that, by incorporating the dependency information in LSTM-based forecasting models, accuracy of forecasting improves by 3–10 times. In our on-going work, we are investigating techniques that can detect changes in the dependency graph over time in an online fashion that minimizes the need for full re-training, as well as reduce the number of features that are required as

input to the LSTM network, in order to discover dependencies amongst performance metrics.

REFERENCES

[1] A. Wiggins, "The twelve-factor app," https://12factor.net, accessed: 2017-08-14.

[2] S. Newman, *Building Microservices*. O'Reilly Media, 2015.

[3] X. Chen, M. Zhang, Z. Mao, and P. Bahl, "Automating network application dependency discovery: Experiences, limitations, and new solutions," in *ACM Operating Systems Design and Implementation*, 2008, pp. 117–130.

[4] V. Shrivastava, P. Zerfos, K.-w. Lee, H. Jamjoom, Y.-H. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *IEEE Infofom 2011*, 2011, pp. 66–70.

[5] A. Arnold, Y. Liu, and N. Abe, "Temporal causal modeling with graphical granger methods," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, pp. 66–75. [Online]. Available: http://doi.acm.org/10.1145/1281192.1281203

[6] J. Gao, G. Jiang, H. Chen, and J. Han, "Modeling probabilistic measurement correlations for problem determination in large-scale distributed systems," *2009 29th IEEE International Conference on Distributed Computing Systems*, pp. 623–630, 2009.

[7] I. Laguna, S. Mitra, F. A. Arshad, N. Theera-Ampornpunt, Z. Zhu, S. Bagchi, S. P. Midkiff, M. Kistler, and A. Gheith, "Automatic problem localization via multi-dimensional metric profiling," *2013 IEEE 32nd International Symposium on Reliable Distributed Systems*, pp. 121–132, 2013.

[8] J. Yin, X. Zhao, Y. Tang, C. Zhi, Z. Chen, and Z. Wu, "Cloudscout: A non-intrusive approach to service dependency discovery," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1271–1284, May 2017. [Online]. Available: https://doi.org/10.1109/TPDS.2016.2619715

[9] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *arXiv preprint arXiv:1606.01865*, 2016.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[11] P. J. Brockwell and R. A. Davis, *Introduction to time series and forecasting*. springer, 2016.

[12] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[13] "Rmsprop optimizer," https://keras.io/optimizers/, accessed: 2016-08-11.

[14] "Amazon web services," https://aws.amazon.com/, accessed: 2016-08-11.

[15] "Apache jmeter," http://jmeter.apache.org/index.html, accessed: 2017-07-05.

[16] "Clarknet-http," http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html, accessed: 2017-07-05.

[17] A. M. De Livera, R. J. Hyndman, and R. D. Snyder, "Forecasting time series with complex seasonal patterns using exponential smoothing," *Journal of the American Statistical Association*, vol. 106, no. 496, pp. 1513–1527, 2011.

[18] X. Jia, A. Khandelwal, G. Nayak, J. Gerber, K. Carlson, P. West, and V. Kumar, "Incremental dual-memory lstm in land cover prediction," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: ACM, 2017, pp. 867–876. [Online]. Available: http://doi.acm.org/10.1145/3097983.3098112

[19] Ú. Erlingsson, M. Peinado, S. Peter, M. Budiu, and G. Mainar-Ruiz, "Fay: extensible distributed tracing from kernels to clusters," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 4, p. 13, 2012.

[20] B.-C. Tak, C. Tang, C. Zhang, S. Govindan, B. Urgaonkar, and R. N. Chang, "vpath: Precise discovery of request processing paths from black-box observations of thread and network activities." in *USENIX Annual technical conference*, 2009.

[21] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using magpie for request extraction and workload modelling." in *OSDI*, vol. 4, 2004, pp. 18–18.

[22] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed systems of black boxes," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 74–89, 2003.

[23] P. Bahl, R. Chandra, A. Greenberg, S. Kandula, D. A. Maltz, and M. Zhang, "Towards highly reliable enterprise network services via inference of multi-level dependencies," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 13–24.

[24] R. Apte, L. Hu, K. Schwan, and A. Ghosh, "Look who's talking: Discovering dependencies between virtual machines using cpu utilization." in *HotCloud*, 2010.

[25] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, "Modeling and synthesizing task placement constraints in google compute clusters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 3.

[26] N. Burling, " Introducing Service Map for dependency-aware monitoring, troubleshooting and workload migrations," https://blogs.technet.microsoft.com/hybridcloud/2016/11/21/introducing-service-map-for-dependency-aware-monitoring-troubleshooting-and-workload-migrations/, 2016.

[27] C. W. Granger, "Some recent development in a concept of causality," *Journal of econometrics*, vol. 39, no. 1-2, pp. 199–211, 1988.

[28] M. Dhamala, G. Rangarajan, and M. Ding, "Estimating granger causality from fourier and wavelet transforms of time series data," *Physical review letters*, vol. 100, no. 1, p. 018701, 2008.

[29] A. Arnold, Y. Liu, and N. Abe, "Temporal causal modeling with graphical granger methods," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 66–75.

[30] K. Kalpakis, D. Gada, and V. Puttagunta, "Distance measures for effective clustering of arima time-series," in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE, 2001, pp. 273–280.

[31] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[32] 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.

[33] J. T. Connor, R. D. Martin, and L. E. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 240–254, 1994.

[34] C. Smith and Y. Jin, "Evolutionary multi-objective generation of recurrent neural network ensembles for time series prediction," *Neurocomputing*, vol. 143, pp. 302–311, 2014.