Spatiotemporal Modeling and Prediction in Cellular Networks: A Big Data Enabled Deep Learning Approach

Jing Wang, Jian Tang, Zhiyuan Xu, Yanzhi Wang, Guoliang Xue, Xing Zhang and Dejun Yang

***Abstract*—In this paper, we propose to leverage the emerging deep learning techniques for spatiotemporal modeling and prediction in cellular networks, based on big system data. First, we perform a preliminary analysis for a big dataset from China Mobile, and use traffic load as an example to show non-zero temporal autocorrelation and non-zero spatial correlation among neighboring Base Stations (BSs), which motivate us to discover both temporal and spatial dependencies in our study. Then we present a hybrid deep learning model for spatiotemporal prediction, which includes a novel autoencoder-based deep model for spatial modeling and Long Short-Term Memory units (LSTMs) for temporal modeling. The autoencoder-based model consists of a Global Stacked AutoEncoder (GSAE) and multiple Local SAEs (LSAEs), which can offer good representations for input data, reduced model size, and support for parallel and application-aware training. Moreover, we present a new algorithm for training the proposed spatial model. We conducted extensive experiments to evaluate the performance of the proposed model using the China Mobile dataset. The results show that the proposed deep model significantly improves prediction accuracy compared to two commonly used baseline methods, ARIMA and SVR. We also present some results to justify effectiveness of the autoencoder-based spatial model.**

Index Items: Cellular Network; Big Data; Spatiotemporal Modeling, Deep Learning; Autoencoder; Recurrent Neural Network

I. Introduction

There is no doubt that we are living in the big data era [5]. Big data can refer to two different things in the context of wireless networks. First, last decade has seen an exponential growth on mobile devices and Internet of Things (IoT) globally. Beyond communication, these devices have been playing a key role in many aspects of people's daily life, including computing, entertainment, sensing, etc. As a result, these activities have generated enormous mobile data for wireless networks, which we can call *big user data*. In addition, wireless networks have become more and more advanced and complicated, which are generating a large amount of runtime system statistics (such as traffic load, resource usages, etc) every second. For example, In [8], Ding *et al.* showed the volume of spectrum state data could be in the order of zettabytes (ZBs, 1 ZB $= 10^{21}$ Bytes) in a $100 * 100$ km^2 area, during one week, on a spectrum ranging from from 0 to 5 GHz. We can call such data *big system data*.

Tremendous research efforts (e.g., [28], [29]) have been made to develop algorithms and protocols for wireless networks to utilize their resources efficiently and effectively. However, most of them aimed at optimizing resource allocation, assuming that some key factors (such as traffic load, spectrum usages, computing resource usages, etc) are given as input. Limited work has been done to model and predict the pattern of these key factors, which are highly time and location varying. Instead of treating big system data as an unwanted burden, we should leverage them as a great opportunity for better understanding user demands and system capabilities such that we can optimize resource allocation to better serve mobile users.

Wireless system data are basically time series data. Quite a few models and methods [6] have been proposed for time series analysis. AutoRegression Integrated Moving Average (ARIMA) and Support Vector Regression (SVR) are two most widely used methods, which have been applied to wireless networks. For example, ARIMA has been used in [26], [38] to predict the future traffic load. However, the limitation of ARIMA lies in their natural tendency to concentrate on the mean values of the past series data, which makes it unable to capture the rapid variational process underlying traffic load [17]. SVR model is also limited for the reason that the users need to determine some key parameters for the model, and it lacks a structured way for determining best values for these parameters [17]. More importantly, these methods use only historical data of the target for prediction without taking into account spatial dependency (i.e., neighboring BSs), which, however, is very important in a wireless network.

In this paper, we propose a novel deep learning approach for spatiotemporal modeling and prediction in cellular networks, using big system data. Deep learning is a multi-layer representation learning method [34], which aims to automatically discover a simple but proper representation for the given raw data. Each layer is a non-linear module that transforms the representation of the previous layer into a more compact representation. Deep learning has been shown to dramatically improve the state-of-art on many application domains, including image/video processing, natural language processing, etc [34]. It is particularly suitable to infer information from

Jing Wang, Jian Tang, Zhiyuan Xu and Yanzhi Wang are with Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244. Guoliang Xue is with Ira A. Fulton Schools of Engineering, Arizona State University, Tempe, AZ 85287. Xing Zhang is with the Key Lab of Universal Wireless Communications, Beijing University of Posts and Telecommunications (BUPT) and Beijing Advanced Innovation Center for Future Internet Technology, Beijing University of Technology (BJUT). Dejun Yang is with Department of Electrical Engineering and Computer Science, Colorado School of Mines, Golden, CO 80401. This research was supported by NSF grants 1443966, 1525920, 1461886 and 1457262. The information reported here does not reflect the position or the policy of the federal government. Xing Zhang is funded by China NSFC grants 61372114 and 61631005, and the Beijing Nova Program: Z151100000315077.

978-1-5090-5336-0/17/$31.00 ©2017 IEEE

large datasets and requires very little domain knowledge and engineering by hand. This work aims to show how deep learning can be utilized to model time series data collected from a cellular network and make accurate prediction.

First, we perform a preliminary analysis for a big dataset from the largest wireless carrier in China, China Mobile, and use traffic load as an example to show non-zero temporal autocorrelation and non-zero spatial correlation among neighboring Base Stations (BSs), which motivate us to discover both temporal and spatial dependencies in our study. We then present a hybrid deep learning model for time series prediction, which includes a novel autoencoder-based deep model for spatial modeling and Long Short-Term Memory units (LSTMs) for temporal modeling. The autoencoder-based model consists of a Global Stacked AutoEncoder (GSAE) and multiple Local SAEs (LSAEs), which can offer good representations for input data, reduced model size, and support for parallel and application-aware training. Moreover, we present a new algorithm for training this autoencoder-based spatial model. In addition, we conducted extensive experiments to evaluate the performance of the proposed model using the China Mobile dataset. The results show that our model significantly improves prediction accuracy compared to two commonly used baseline methods, ARIMA and SVR. We also show some results to justify effectiveness of the autoencoder-based spatial model. To the best of our knowledge, we are the first to leverage the emerging deep learning techniques for spatiotemporal modeling and prediction in wireless networks by developing a new hybrid deep model, and showing its effectiveness and superiority with real data from a major wireless carrier.

II. Preliminary Data Analysis

In this section, we first describe the dataset used for analysis and evaluation, and then we perform a preliminary analysis for the data, which motivates our design.

A. Dataset

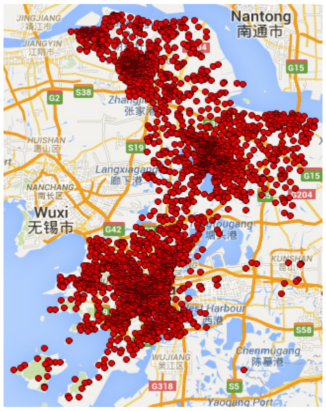

Fig. 1. Locations of BSs in our dataset

The dataset consists of data collected from a large LTE network of China Mobile at Suzhou, a major city located in the southeastern part of China. The data was collected from $2,844$ BSs, roughly covering an area of $6,500$ km^2. Locations of all the BSs are shown in the map given by Fig. 1. Here, our analysis is performed based on the downlink and uplink traffic load. However, the proposed model (Section III) can be applied to other features. The dataset includes average traffic load of each BS in every hour during the period from 00:00 05/01/2015 to 23:00 09/30/2015. To facilitate data analysis, we divide the target area into a grid, with each cell covering a square of 500×500 m^2. Then every BS can be mapped into a cell in the grid. If a cell includes more than one BS, then its traffic load is the aggregated load. Note that unlike traditional cellular networks, current dense small cell networks do not have a hexagon-based layout. A tuple (m, n) is used to uniquely identify each cell. We denote $\mathbf{D} = \{d_{m,n,t}\}, \forall m, n, t$, which is the downlink/uplink traffic load of cell (m, n) at timeslot t. Since uplink and downlink can be considered separately, without abusing the notation, we use this to denote both of them. In addition, we denote $\mathbf{d}_{m,n} = \{d_{m,n,t}\}, \forall t$.

For each cell (m, n), we normalize the data into the range $[0, 1]$. We adopt the *tanh estimator* method, a robust and efficient method for normalizing time series data [14], which calculates the normalized values as follows:

$$\hat{\mathbf{d}}_{m,n} = 0.5(tanh(\frac{0.01(\mathbf{d}_{m,n} - \bar{d}_{m,n})}{\sigma_{\mathbf{d}_{m,n}}}) + 1), \tag{1}$$

where $\bar{d}_{m,n}$ and $\sigma_{\mathbf{d}_{m,n}}$ are the average and standard deviation of $\mathbf{d}_{m,n}$ respectively.

B. Data Analysis

In our preliminary analysis, we try to explore data dependency in both the temporal and spatial domains. $\mathbf{d}_{m,n}$ can be treated as a collection of a random process samples at cell (m, n). So we can examine data dependency in terms of temporal autocorrelation and spatial correlation in the temporal and spatial domains respectively. We summarize our main findings in the following:

Observation 1: Dataset $\mathbf{D}$ *exhibits non-zero autocorrelation in the temporal domain.*

The *sample AutoCorrelation Function* (sample ACF) [6] is a widely used method for discovering data dependency in the temporal domain, which describes the dependency between the values of a sample process as a function of time lag h. The definition of the sample ACF at cell (m, n) can be given as follows (for the sake of readability, we omit the notations of m, n in this definition):

$$\rho(h) = \frac{\sum_{t=1}^{T-|h|} (d_{t+|h|} - \bar{d})(d_t - \bar{d})}{\sum_{t=1}^{T} (d_t - \bar{d})^2}, -T < h < T; \tag{2}$$

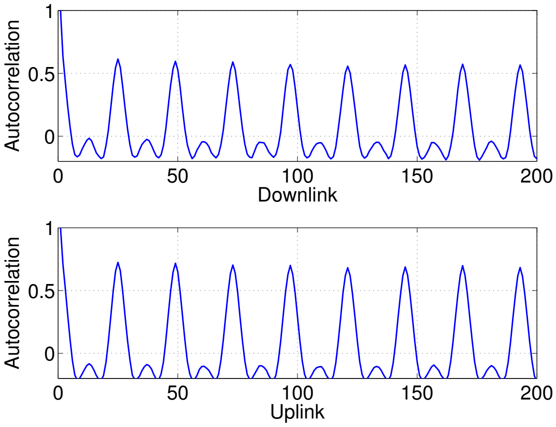

Fig. 2. Temporal autocorelation

where T and $\bar{d}$ are the total count and mean value of data in the temporal dimension, respectively. The autocorrelation value lies in the range $[-1, 1]$. $\rho(h) = 1$ indicates total positive autocorrelation between data with a time lag of h; while $\rho(h) = -1$ means total negative autocorrelation. Note that $\rho(h) = 0$ denotes no autocorrelation.

Fig. 2 shows a sample ACF at time lag $h = 0, 1, \cdots, 200$ for both downlink and uplink data. We can see that when the time lag equals one or multiple of 24 (hours), the autocorrelation is relatively high. This shows that the traffic load at a cell follows a clear daily pattern. For example, the traffic load peak and off-peak hours are similar on each day. Therefore, dataset $\mathbf{D}$ exhibits non-zero autocorrelation in the temporal domain.

Observation 2: Dataset $\mathbf{D}$ *reveals non-zero correlation in the spatial domain.*

We examine the data correlation in the spatial domain by calculating a widely used metric [7] for a pair of cells (m, n) and (m', n'):

$$\rho = \frac{cov(\mathbf{d}_{m,n}, \mathbf{d}_{m',n'})}{\sigma_{\mathbf{d}_{m,n}} \sigma_{\mathbf{d}_{m',n'}}}, \tag{3}$$

where $cov(\cdot)$ is the covariance operator, and σ is the standard deviation. Similarly, this correlation coefficient ranges in $[-1, 1]$ as well.

TABLE I
SPATIAL CORRELATION

	Cell 1	Cell 2	Cell 3	Cell 4	Cell 5	Cell 6	Cell 7
Cell 1	1.000	0.167	0.435	0.130	0.040	0.341	0.307
Cell 2	0.396	1.000	0.338	0.129	0.084	0.310	0.222
Cell 3	0.345	0.541	1.000	0.159	0.162	0.697	0.536
Cell 4	0.437	0.439	0.458	1.000	0.104	0.131	0.114
Cell 5	0.360	0.471	0.492	0.508	1.000	0.163	0.080
Cell 6	0.286	0.491	0.550	0.432	0.535	1.000	0.603
Cell 7	0.284	0.506	0.526	0.459	0.535	0.577	1.000

We examine the correlation among cells for both downlink and uplink data, and present the results among 7 closely located cells in Table I. Each cell is subsequently located on the east side of the previous one. Note that the upper triangular part of Table I shows the correlation for uplink data, while the lower triangular part is for downlink data. We can clearly observe none-zero correlation among these cells from the table. Actually, more than 50% of the correlation values are greater than 0.300. In addition, we can see that the correlation values among cells vary a lot. For instance, downlink data in Cell 1 and Cell 2 have a correlation value of 0.396; while Cell 5 and Cell 6 have a correlation value of 0.535, even though Cell 2 and Cell 6 are of the same spatial relationship to Cell 1 and Cell 5, respectively. This property indicates that the spatial correlation is highly location-dependent.

III. SPATIOTEMPORAL MODELING AND PREDICTION

A. Overview

As mentioned above, simple temporal modeling that uses only historical data of the target may not work well here due to strong spatial correlation observed from the data. Motivated by the observations described above, we design a novel hybrid deep learning model to perform spatiotemporal modeling and prediction for each cell (m, n), which leverages historical data collected from both the target cell and its neighboring cells surrounding it. The proposed model consists of three major components: Local Stacked AutoEncoders (LSAEs), a Global Stacked AutoEncoder (GSAE) and Long Short-Term Memory units (LSTMs). As illustrated by Fig. 3, the proposed model works as follows:

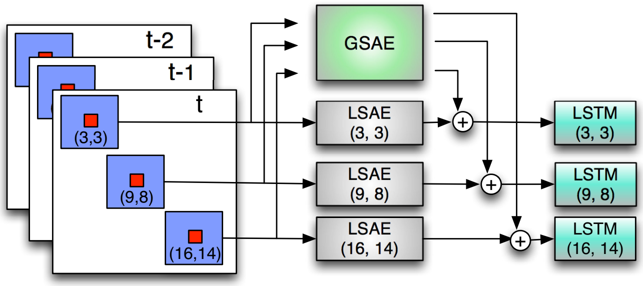

Fig. 3. The proposed deep learning model

1) Data of the cell of interest (marked red) and its neighboring cells form a data patch (marked blue), which can include values of one or multiple features of interest (such as downlink/uplink traffic load). The GSAE takes such a data patch as input, producing an encoded representation (called global representation). Note that there is only one GSAE, which is applied to all patches.

2) After being encoded by the GSAE, each patch will be fed to the corresponding LSAE to generate another representation (called local representation). The global representation and local representation will then be concatenated ($\oplus$) to represent each patch.

3) The concatenated representations will then be passed to LSTMs for prediction.

For spatial modeling, we choose autoencoder [3] as a starting point in our design because it has been shown to be a simple and effective model for providing a good representation of input data with much smaller size. We come up with a new hybrid structure based on autoencoders by introducing GSAE and LSAE, whose benefits are explained in Section III-B. However, existing training methods do not work for the proposed hybrid model. Hence, we also present a new training algorithm in Section III-B. Note that one way to select neighboring cells for a target cell is to choose all those surrounding it and falling into a square box as shown in Fig. 3. However, the proposed model is not restricted to this method. This can be determined according to actual networks and applications.

In addition, we choose an RNN, particularly LSTM, for temporal modeling and prediction because gated RNNs (such as LSTM), use gates to control how to update hidden states and specify how much past information should be let through, which have been shown to be effective on modeling long-term dependencies [11].

TABLE II
MAJOR NOTATIONS

Notation	Description
(m, n)	Index of cell and the corresponding data patch
t and T	Index and total number of data points in the temporal domain
i and I	Index and total number of GSAE layers
j and J	Index and total number of LSAE layers
$\mathbf{W}_{g_i}$ and $\mathbf{W}'_{g_i}$	Weights of encoder and decoder in layer i of GSAE
$\mathbf{W}_{l_j}$ and $\mathbf{W}'_{l_j}$	Weights of encoder and decoder in layer j of LSAE
$\mathbf{b}_{g_i}$ and $\mathbf{b}'_{g_i}$	Biases of encoder and decoder in layer i of GSAE
$\mathbf{b}_{l_j}$ and $\mathbf{b}'_{l_j}$	Biases of encoder and decoder in layer j of LSAE
$\mathbf{h}_{g_i}$ and $\mathbf{h}_{l_j}$	Hidden units in GSAE and LSAE

B. Spatial Modeling

Here, we describe the proposed model for spatial modeling, which is a combination of a GSAE and multiple LSAEs.

An autoencoder is a model (usually a one-hidden-layer neural network) trained to reconstruct its input, which can be used to obtain a different representation (i.e., hidden layer) of the input with a much smaller size [4], [20]. The process to obtain a different representation is referred to as *encoding*, while the process to reconstruct its input is referred to *decoding*. In our implementation, we adopt the denoising autoencoder, which is an extension of a classic autoencoder [32]. It was designed to make the learned representation robust by reconstructing partially corrupted input. Autoencoders can be stacked to form a deep network [32]. Stacked autoencoders have been shown to be able to effectively extract further non-linear representation [3], [32].

A global representation (i.e., hidden layer of an autoencoder) can be obtained, given the data patch of a cell and a trained GSAE. However, as discussed above, there exists location-dependent spatial correlation for a data patch. Therefore, it is necessary to obtain a better representation with less reconstruction loss. To achieve this goal, we propose to use an LSAE together with the trained GSAE to capture the local location-dependent spatial correlation and yield a better representation.

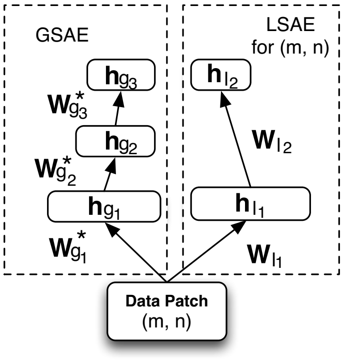

Fig. 4. The proposed autoencoder-based spatial model

An example of the LSAE for cell (m, n) with the GSAE is given in Fig. 4. Note that superscript $(*)$ indicates they are trained variables. So $\mathbf{W}^*_{g_i}$ are trained weights of layer i of GSAE. For the sake of readability, notations for bias variables $\mathbf{b}_{l_j}$, $\mathbf{b}'_{l_j}$, $\mathbf{b}^*_{g_i}$ and $\mathbf{b}'^*_{g_i}$ are omitted in both figures. Note that for either GSAE or LSAEs, the layer number can be 1, resulting in a single-layered autoencoder.

Given a trained GSAE, we use an LSAE to further reduce the reconstruction loss of a data patch. The layer 1 weights of the LSAE can be trained to reduce reconstruction loss of layer 1 in the GSAE. Then higher layers of the LSAE are trained to learn a different representation of the lower layers. Finally, the highest representation of the GSAE concatenated by the highest representation of the LSAE generates a better representation of a local data patch.

The proposed hybrid (global + local) model leads to the following benefits:

- *Better Representation*: Different cells share some common characteristics, which are captured by the GSAE. However, as discussed above, each cell also has its specific location-dependent characteristics, which are captured by the corresponding LSAE. Hence, compared to the GSAE-only model, the proposed hybrid model can provide a better presentation for the given data, which has been validated by results presented later.
- *Reduced Model Size*: An SAE with H_i hidden units in layer i has $\sum_{i=1}^{I} H_{i-1} * H_i$ weight variables (where H_0 is the input dataset size), and $(H_0 + \sum_{i=1}^{I-1} 2H_i + H_I)$ bias variables. The number of variables will get very large, when the dataset size is big. A large model is usually difficult to train. With the proposed hybrid structure, we have one global, and multiple local SAEs, which both have moderate sizes. Training such models is much easier and faster.
- *Support for Parallel Training*: Given a trained GSAE, training LSAEs is independent of each other. Therefore,

they can be trained in parallel.

- *Support for Application-aware Training*: LSAEs can be trained according to the needs of applications. For some applications, we may not be interested in all the cells in the cellular network. If so, we can only train those LSAEs corresponding to cells of interest.

Training the hybrid model in Fig. 4 is not straightforward. A well-known work [32] introduced a greedy layer-wise algorithm for effectively training SAE. GSAE can be trained using this algorithm. However, the next step is to train an LSAE with a trained GSAE, for which the existing algorithm [32] cannot be directly applied.

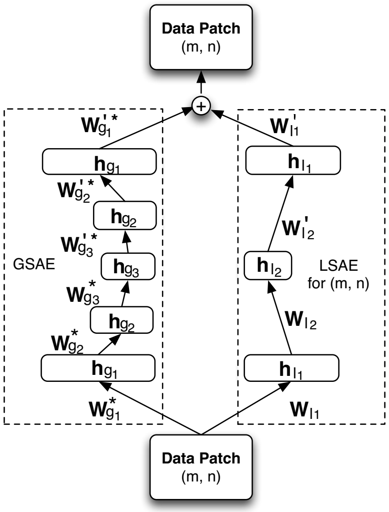

Fig. 5. The unrolled GSAE and LSAE

For training and fine-tuning, we need to unroll the GSAE and LSAE, which are shown in Fig. 5. Next, we formally define the encoding function $q_{l_j}(\cdot)$ and decoding function $q_{l_j}(\cdot)$ for each layer i of an LSAE.

$$p_{l_j}(\mathbf{X}_{l_j}) = \mathbf{Y}_{l_j} = \delta(\mathbf{W}_{l_j}\mathbf{X}_{l_j} + \mathbf{b}_{l_j})$$

$$q_{l_j}(\mathbf{Y}_{l_j}) = \begin{cases} \delta(\mathbf{W}'^*_{g_1}(\mathbf{W}^*_{g_1}\mathbf{X}_{l_1} + \mathbf{b}^*_{g_1}) + \mathbf{b}'^*_{g_1} \\ \qquad + \mathbf{W}'_{l_1}\mathbf{Y}_{l_1} + \mathbf{b}'_{l_1}), & j = 1 \\ \delta(\mathbf{W}'_{l_j}\mathbf{Y}_{l_j} + \mathbf{b}'_{l_j}), & \text{otherwise.} \end{cases}$$

Here, $\delta(\cdot)$ is the activation function (we used the sigmoid function in our implementation). $\mathbf{X}_{l_j}$ is the input of layer j, which will be encoded. $\mathbf{Y}_{l_j}$ is the encoded result of layer j, which can be decoded for reconstruction. However, $\mathbf{Y}_{l_j}$ can also be encoded by upper layer to obtain a more abstract representation. That is to say, $\mathbf{X}_{l_{j+1}} = \mathbf{Y}_{l_j}$. $\mathbf{W}_{l_j}$, $\mathbf{W}'_{l_j}$, $\mathbf{b}_{l_j}$ and $\mathbf{b}'_{l_j}$ are the weights for encoding, weights for decoding, bias for encoding and bias for decoding, respectively, in the LSAE. $\mathbf{W}^*_{g_1}$, $\mathbf{W}'^*_{g_1}$, $\mathbf{b}^*_{g_1}$ and $\mathbf{b}^*_{g_1}$ are the trained weights for encoding, trained weights for decoding, trained bias for encoding and trained bias for decoding, respectively, in the GSAE. Note that $q_{l_1}(\cdot)$ establishes the connection between the GSAE and LSAE.

Algorithm 1: Training the LSAE with J layers for cell (m, n)

```
Input  : Dataset $\mathbf{D}$, the trained GSAE with weights
         $\mathbf{W}^*_{g_i}$, bias $\mathbf{b}^*_{g_i}$, $\mathbf{b}'^*_{g_i}$
Output: $\mathbf{W}^*_{l_j}$, $\mathbf{b}^*_{l_j}$, $\mathbf{b}'^*_{l_j}$
1  $\mathbf{X}_{l_1} := \emptyset$;
2  forall $t$ do
3      $\mathbf{X}_{l_1} := \mathbf{X}_{l_1} \cup patch(d_{m,n,t})$;
4  $\mathbf{b}'_{l_1} = \mathbf{0}$;
5  $\mathbf{W}_{l_j} := \mathbf{0}$, $\forall j$;
6  $\mathbf{W}_{l_1}, \mathbf{b}_{l_1}, \mathbf{b}'_{l_1} :=$
     $\arg\min_{\mathbf{W}_{l_1}, \mathbf{b}_{l_1}, \mathbf{b}'_{l_1}} L(\mathbf{X}_{l_1}, q_{l_1}(p_{l_1}(\widetilde{\mathbf{X}}_{l_1})))$;
7  $\mathbf{X}_{l_2} := p_{l_1}(\mathbf{X}_{l_1})$, $j := 2$;
8  while layer $j \leq J$ do
9      $\mathbf{W}_{l_j}, \mathbf{b}_{l_j}, \mathbf{b}'_{l_j} :=$
         $\arg\min_{\mathbf{W}_{l_j}, \mathbf{b}_{l_j}, \mathbf{b}'_{l_j}} L(\mathbf{X}_{l_j}, q_{l_j}(p_{l_j}(\widetilde{\mathbf{X}}_{l_j})))$;
10     $\mathbf{X}_{l_{j+1}} := p_{l_j}(\mathbf{X}_{l_j}))$;
11     $j := j + 1$;
12 unroll the GSAE and LSAE as in Fig. 5;
13 $\mathbf{W}^*_{l_j}, \mathbf{b}^*_{l_j}, \mathbf{b}'^*_{l_j} := \arg\min_{\mathbf{W}_{l_j}, \mathbf{b}_{l_j}, \mathbf{b}'_{l_j}} L(\widetilde{\mathbf{X}}_{l_1}, \mathbf{X}')$,
     $\forall j \in \{1, 2, ..., J\}$;
14 return $\mathbf{W}^*_{l_j}$, $\mathbf{b}^*_{l_j}$, $\mathbf{b}'^*_{l_j}$;
```

We use tied weights [32] for the GSAE and LSAE: the weight matrix in a decoding function is the transpose of the weight matrix in the encoding function, i.e.,$\mathbf{W}'_{g_i} = \mathbf{W}^T_{g_i}$, $\mathbf{W}'_{l_j} = \mathbf{W}^T_{l_j}$. Note that if $\mathbf{W}_{l_1} = \mathbf{0}$ and $\mathbf{b}'_{l_1} = \mathbf{0}$, the proposed model degenerates into a GSAE, because in this case, $q_{l_1}(\mathbf{Y}_{l_1}) = \delta(\mathbf{W}'^*_{g_1}(\mathbf{W}^*_{g_1}\mathbf{X}_{l_1} + \mathbf{b}^*_{g_1}) + \mathbf{b}'^*_{g_1})$ is actually the reconstructed result of the GSAE. We can initialize $\mathbf{W}_{l_j} = \mathbf{0}$ and $\mathbf{b}'_{l_1} = \mathbf{0}$ as the starting point for training an LSAE. We formally present the LSAE training algorithm as Algorithm 1, which consists of two phases: pre-training and fine-tuning.

In this algorithm, $patch(d_{m,n,t})$ gives input data corresponding to Cell (m, n) and its neighboring cells (surrounding Cell (m, n)) at timeslot t. $\widetilde{\mathbf{X}}_i$ is the corrupted version of $\mathbf{X}_i$. Lines 1–3 generate the input data for the first layer in the LSAE. As discussed above, Line 4 initializes $\mathbf{b}'_{l_1} = \mathbf{0}$ and line 5 initializes the weights $\mathbf{W}_{l_j} = \mathbf{0}$. Line 6 pre-trains the first layer with a partially corrupted input. The reconstruction loss is defined to be the cross entropy as in [3], [20]:

$$L(\mathbf{X}, \mathbf{Z}) = \sum xlog(z) + (1 - x)log(1 - z). \tag{4}$$

In our implementation, we applied the commonly-used Stochastic Gradient Descent (SGD) [19] algorithm to minimize the reconstruction loss. Other methods, such as RMSProp and AdaGrad [19], can also be applied here to train the model. Line 7 generates the input $\mathbf{X}_{l_2}$ for the second layer with uncorrupted $\mathbf{X}_{l_1}$. Lines 8–11 show the pre-training process for layer 2 up to layer J. After layer j is pre-trained, the input $\mathbf{X}_{l_{j+1}}$ for $(j + 1)$-th layer can be obtained from $\mathbf{X}_{l_{j+1}}$ = $p_{l_j}(\mathbf{X}_{l_j})$. Note that the uncorrupted input $\mathbf{X}_{l_j}$ is fed to the encoder. After all the layers have been pre-trained, we unroll the trained GSAE and LSAE as shown in Fig. 5 for fine-tuning (Line 13), where all weight matrices and bias variables are updated. $\mathbf{X}'$ is the reconstructed input.

An LSAE cannot be trained without the GSAE because the decoding function of LSAE relies on the trained GSAE. The first layer of the LSAE is pre-trained and fine-tuned differently from other (upper) layers, which takes the trained layer 1 of GSAE as input. Given the pre-trained and fine-tuned layer 1, layers 2 to J of the LSAE are pre-trained and fine-tuned independently from the GSAE. Note that, given a trained GSAE, all LSAEs can be trained in parallel. In addition, it is not required to have the same structure for the GSAE and LSAE: GSAE and LSAEs can have different numbers of layers; and the number of hidden units in each layer can also be set differently. Moreover, the structures of LSAEs do not have to be the same.

C. Temporal Modeling and Prediction

As mentioned above, we propose to use an RNN for temporal modeling and prediction, which takes the representations learned from the hybrid spatial model as input.

An RNN is a generalization of the feed forward neural network for modeling sequence (time series) data [22]. However, a well-known problem with standard RNNs is that it can be difficult to model long-term dependencies [16]. Long Short-Term Memory (LSTM) was proposed in [15], which is known to be able to capture long-term temporal dependencies [11], [12]. LSTM incorporates gates, which allow the model to learn how to forget previous hidden states and how to update the current states. A diagram of the LSTM unit from [36] is shown in Fig. 6, which is a slight simplification of [13].

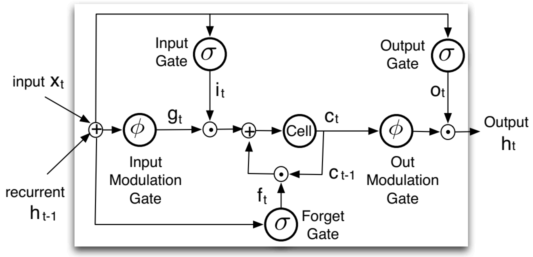

Fig. 6. LSTM unit [36]

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i)\\
\mathbf{f}_t &= \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)\\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)\\
\mathbf{g}_t &= \phi(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c)\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t\\
\mathbf{h}_t &= \mathbf{o}_t \odot \phi(\mathbf{c}_t)
\end{aligned}
$$

The LSTM unit consists of a single memory cell $\mathbf{c}_t$, an input and output modulation gate ($\mathbf{g}_t$ and $\mathbf{h}_t$) and three gates (input $\mathbf{i}_t$, output $\mathbf{o}_t$ and forget $\mathbf{f}_t$). $\sigma(\cdot)$ is the sigmoid function; and $\phi(x)$ is the hyperbolic tangent function $\phi(x) = 2\sigma(2x) - 1$. $\odot$ and $\oplus$ denote the dot product and sum of two vectors, respectively. The $\mathbf{W}$ terms denote the weight matrices. For example, $\mathbf{W}_{hf}$ is the hidden-forget weight matrix; while the $\mathbf{b}$ terms are the biases.

The memory cell combines the previous cell states, current input and previous output, to update hidden states. The forget gate determines if the information should forgotten or remembered. The output gate learns how the memory cell should affect the hidden states.

To predict the future value $d_{m,n,t'+1}$ for a cell (m,n), the data patches corresponding to the past T timeslots are taken as the input. They will be encoded by the GSAE and the LSAE. For each timeslot $t \leq t'$, the following three values will be concatenated as a vector: $d_{m,n,t}$, and GSAE and LSAE representations of $patch(d_{m,n,t})$. In this way, we obtain a temporal sequence of vectors, as shown in Fig. 7. Then the LSTM unit processes this sequence as described above and predict $d_{m,n,t'+1}$.

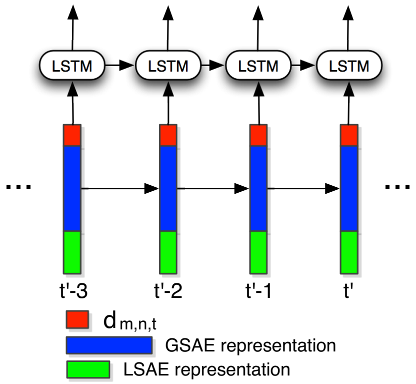

Fig. 7. Temporal modeling and prediction

IV. Performance Evaluation

A. Settings

We compared our approach with two widely used methods for time series analysis. The first approach is ARIMA [18], which is one of the most popular linear models for time series forecasting and has been applied to wireless networks [33], [38]. The second baseline approach is SVR, which is a variant of Support Vector Machine (SVM) proposed for regression [9], [27]. It has been also applied for time series analysis in many applications [21], [23]. In the experiments, we used the implementation of ARIMA and SVR in two libraries [19] and [24], respectively. These two baselines were compared with the proposed model in terms of three commonly used performance metrics [10]: Mean Squared Error (MSE), Mean Absolute Error (MAE), and Log Loss (also known as binary cross-entropy).

For neighboring cell selection in spatial modeling, we chose to use data from all cells located within a 11×11 square box that is centered at the location of a target cell. That is to say, we considered data from 120 neighboring cells for modeling Cell (m, n).

We chose the commonly used sigmoid function as the activation function in each layer of both the GSAE and LSAEs. Regarding the corruption process in autoencoders, we adopted a stochastic method proposed in [32]. In our implementation, the corruption level was set to 0.1. The GSAE has two layers (unrolled), and the lower layer has 20 hidden units, and second layer has 2 hidden units. All the LSAEs has a single layer with 2 hidden units.

We randomly chose 15 cells for testing. For each cell (m, n), we split the data into training set and test set. We presented the corresponding results in the following.

B. Prediction Results

First, we present the experimental results to show the overall prediction performance of the proposed model.

Fig. 8 shows a comparison between prediction results and the actual values (from the dataset) for both downlink and uplink traffic load at a randomly chosen cell. We can see that the prediction results well match the trend of actual values. Specifically, the MSE, MAE and Log Loss are 0.042, 0.165 and 0.583, respectively for downlink traffic load; while, they become 0.031, 0.137, 0.556 respectively for uplink. Moreover, prediction results are very close to the actual values around the major transition points, when the traffic load falls below or rises above 0.4.

Fig. 9 shows a comparison among ARIMA, SVR and the proposed model for both downlink and uplink traffic load in terms of MSE, MAE and Log Loss for one of the chosen locations, while Fig. 10 presents the average errors over all the chosen locations. From these two figures, we can see that the proposed model consistently outperforms ARIMA and SVR in terms of all the metrics. Specifically, in Fig. 10, the proposed model offers about 30.8%, 20.5%, 33.1% less error than SVR on average in terms of MSE, MAE and Log Loss, respectively. Moreover, it leads to around 40.4%, 28.4%, 18.5% less error than ARIMA on average in terms of MSE, MAE and Log Loss, respectively. These results well justify effectiveness of the emerging deep learning models on cellular network data analysis and more importantly, the superiority of our design that takes into account data dependencies in both the temporal and spatial domains.

C. Spatial Modeling

In this subsection, we present the results to justify the effectiveness of the proposed hybrid model for spatial modeling approach.

Data patches are encoded by both the GSAE and LSAEs. The decoders can reconstruct them so that we can take an in-depth look to make sure the encoded results are indeed good representations of the original data. Fig. 11 (downlink) and Fig. 12 (uplink) show the reconstructed results of data patches corresponding to 9 cells. Each image corresponds to a data patch; and each tiny block (i.e., pixel) in an image corresponds to a cell. Hence, each image has $11 * 11$ blocks. Brightness of a pixel indicates how heavy the traffic load of the corresponding cell is (the brighter, the heavier).

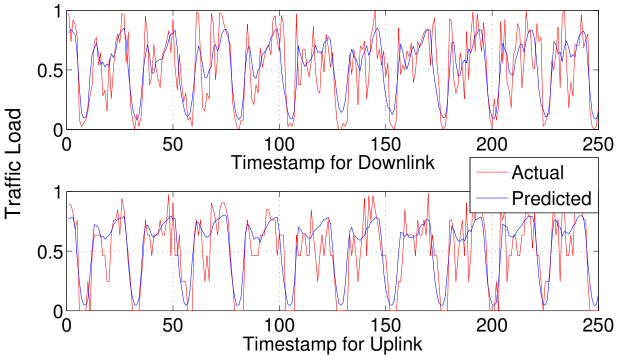

Fig. 8. Prediction results VS. actual values

In both figures, the first rows are the original data patches of the 9 randomly chosen cells; The second rows are the corresponding patches reconstructed by the GSAE. The last rows are data patches reconstructed by The proposed hybrid model (GSAE+LSAE). Note that multiple LSAEs were trained since the original data patches came from different cells.

From these two figures, we can see that the reconstructed results given by the GSAE is relatively “blurry” but somehow still captures the patterns of the original data; while the final reconstructed results given by the proposed hybrid model are very close to the original data. These results confirm that the proposed hybrid model does offer good representations for the original data.

Now we show how LSAEs can help improve the prediction performance. In this experiment, the number of hidden units of our single-layered LSAEs, k, was changed from 0 to 4, with 0 corresponding to the case without LSAEs. The second layer of GSAE was then set to have $(4 - k)$ hidden units. All other settings remain the same. In this way, even though the value of k is changed, the GSAE and a LSAE together had a representation with a constant length of 4, which ensures a fair comparison. Fig. 13 shows how the performance improvement ratio changes with k, which is defined as follows:

$$\frac{M(0) - M(k)}{M(0)} * 100\%,$$

where $M(k)$ denotes the prediction error (MSE, MAE or Log Loss) corresponding to k (M(0) then corresponds to the case without LSAEs).

From Fig. 13, we can see that the prediction performance improvement rises monotonically with k. Specifically, in terms of MSE for downlink, the improvement ratio goes up from 1.96% to 5.54%, when n increases from 1 to 4. This observation validates our claim that learning local characteristics is essential and learning more helps improve prediction performance. However, the tradeoff is that more complicated local models may lead to much longer training time. Determining

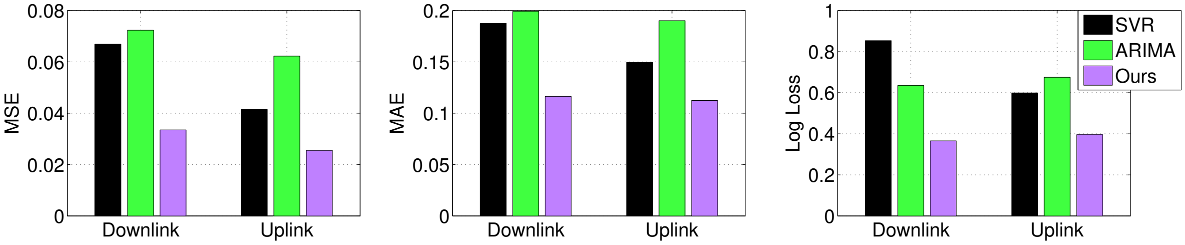

Fig. 9. Prediction errors for a randomly chosen cell

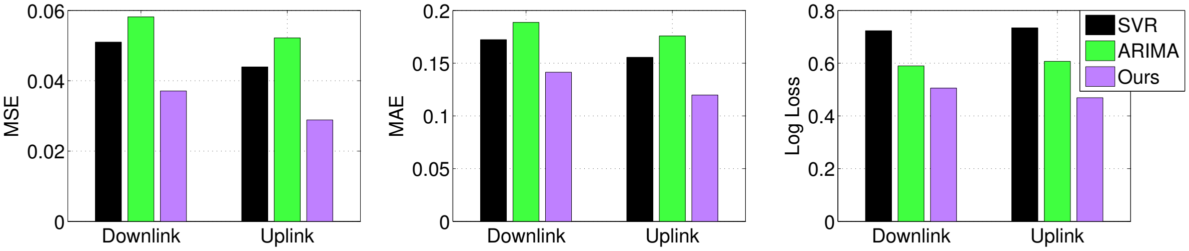

Fig. 10. Average prediction errors

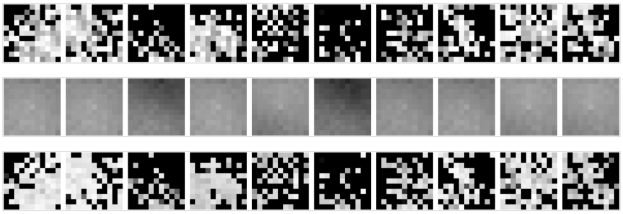

Fig. 11. Reconstructed downlink traffic load

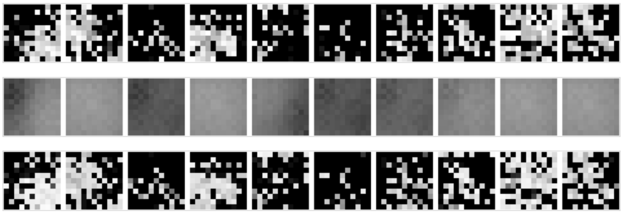

Fig. 12. Reconstructed uplink traffic load

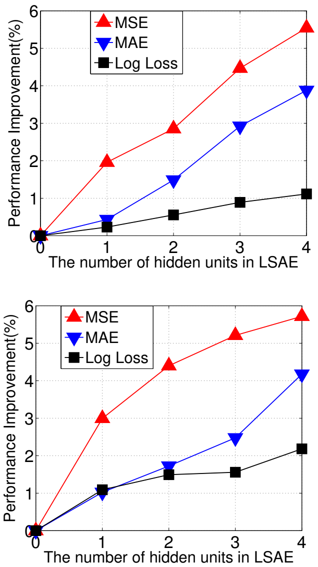

Fig. 13. Prediction performance improvement (Top: Downlink, Bottom: Uplink)

the best configurations for the GSAE and LSAEs is task dependent. It depends on the nature of input data, available computing resources and the number of cells of interest.

V. Related Work

Research efforts have been made for modeling and prediction in communication networks. Specifically, time series analysis methods have also been applied for predicting traffic load. In [26], Shu *et al.* showed that seasonal ARIMA models could be used to model and predict wireless traffic. In [38], Zhou *et al.* proposed a network traffic prediction model, which is a combination of linear time series ARIMA model and non-linear GARCH model. Hong *et al.* applied SVR for short-term traffic load forecasting, and proposed a simulated annealing algorithm and a genetic algorithm to optimize the selection of SVR parameters in [17]. Spatial modeling and estimation methods have been proposed for traffic load in wireless networks [2], [31]. To predict the self-similar network traffic with high burstiness, the authors of [35] proposed a new hybrid method based on the combination of the covariation orthogonal prediction and the artificial neural network. A spatiotemporal compressive sensing framework was proposed for modeling Internet traffic matrices in [37]. Moreover, a very recent work [25] was focused on spatiotemporal analysis for application usages in wireless networks.

In addition, Akbar *et al.* proposed to model and predict the spectrum occupancy of licensed radio bands with Hidden Markov Models (HMMs) [1]. In [30], Tumulus *et al.* designed two adaptive channel status predictor using a neural network based on multilayer perceptron and the hidden Markov model. *A priori* knowledge of the statistics of channel usage is not required in the prediction schemes. Chen *et al.* presented a detailed study [7] with first and second order statistics of collected data, including channel occupancy/vacancy, channel utilization and temporal, spectral and spatial correlation. A 2-dimensional frequent pattern mining algorithm was developed to predict channel availability based on past observations.

Unlike these works, we are the first to propose a deep learning model for spatiotemporal prediction in cellular networks.

VI. Conclusions

In this paper, we first performed a preliminary analysis for a real dataset from China Mobile to show temporal and spatial dependencies. Then we presented a hybrid deep learning model for spatiotemporal prediction, which includes a novel autoencoder-based deep model for spatial modeling and LSTMs for temporal modeling. The autoencoder-based model consists of a GSAE and multiple LSAEs, which can offer better representations for input data (compared to the GSAE-only model), reduced model size, and support for parallel and application-aware training. Moreover, we presented a new algorithm for training the proposed spatial model. The experimental results show that, compared to ARIMA and SVR, the proposed deep model significantly improves prediction accuracy; and the autoencoder-based spatial model is effective and efficient.

References

[1] I. A. Akbar and W. H. Tranter, Dynamic spectrum allocation in cognitive radio using hidden markov models: poisson distributed case, *Proceedings of IEEE SoutheastCon'2007*, pp. 196–201.

[2] F. Ashtiani, J. Salehi, and M. Aref, Mobility modeling and analytical solution for spatial traffic distribution in wireless multimedia networks, *IEEE Journal on Selected Areas on Communications*, Vol. 21, No. 10, 2003, pp. 1699–1709.

[3] Y. Bengio, P. Lamblin, D. Popovici and H. Larochelle, Greedy layer-wise training of deep networks, *Proceedings of NIPS'2006*, pp. 153–160.

[4] Y. Bengio, Learning deep architectures for AI, *Foundations and Trends in Machine Learning*, Vol. 2, No. 1, 2009, pp. 1–127.

[5] S. Bi, R. Zhang, Z. Ding and S. Cui, Wireless communications in the era of big data, *IEEE Communications Magazine*, Vol. 53, No. 10, 2015, pp. 190–199.

[6] P. J. Brockwell and A. D. Richard, Introduction to time series and forecasting, *Springer Science & Business Media*, 2006.

[7] D. Chen, S. Yin, Q. Zhang, M. Liu and S. Li, Mining spectrum usage data: a large-scale spectrum measurement study, *Proceedings of ACM Mobicom'2009*, pp. 13–24.

[8] G. Ding, Q. Wu, J. Wang and Y. D. Yao, Big spectrum data: the new resource for cognitive wireless networking, *arXiv preprint arXiv'2014*, Vol. 1404.6508.

[9] H. Drucker, C. Burges, L. Kaufman, A. Smola and V. Vapnik, Support vector regression machines, *Proceedings of NIPS'1996*, pp. 155–161.

[10] C. Ferri, J. Hernandez-Orallo and R. Modroiu, An experimental comparison of performance measures for classification, *Pattern Recognition Letters*, Vol. 30, No. 1, 2009, pp. 27–38.

[11] F. A. Gers, J. Schmidhuber and F. Cummins, Learning to forget: continual prediction with LSTM, *Neural Computation*, Vol. 12, No. 10, 2010, pp. 2451–2471.

[12] A. Graves, Supervised sequence labelling with recurrent neural networks, *Springer*, 2012.

[13] A. Graves and N. Jaitly, Towards end-to-end speech recognition with recurrent neural networks, *Proceedings of ICML'2014*, pp. 1764–1772.

[14] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw and W. A. Stahel, Robust statistics: the approach based on influence functions, *John Wiley & Sons*, 2011.

[15] S. Hochreiter and J. Schmidhuber, Long Short-Term Memory, *Neural Computation*, Vol. 9, No. 8, 1997, pp. 1735–1780.

[16] S. Hochreiter, Y. Bengio, P. Frasconi and J. Schmidhuber, Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, *A Field Guide to Dynamical Recurrent Neural Networks*, IEEE Press, 2001.

[17] W. C. Hong, Application of seasonal SVR with chaotic immune algorithm in traffic flow forecasting, *Neural Computing and Applications*, Vol. 21, No. 3, 2012, pp. 583–593.

[18] R. J. Hyndman and G. Athanasopoulos, Forecasting: Principles and Practice, *OTexts*, 2014.

[19] *http://keras.io/*

[20] H. Larochelle, D. Erhan, A. Courville, J. Bergstra and Y. Bengio, An empirical evaluation of deep architectures on problems with many factors of variation, *Proceedings of ICML*, 2007, pp. 473–480.

[21] K. R. Muller, A. Smola, G. Ratsch, B. Scholkopf, J. Kohlmorgen and V. Vapnik, Predicting time series with support vector machines, *Proceedings of ICANN*, 1997, pp. 999–1004.

[22] D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning internal representations by error propagation, *Parallel Distributed Processing*, MIT Press, pp. 318–362.

[23] N. I. Sapankevych and R. Sankar, Time series prediction using support vector machines: a survey, *IEEE Computational Intelligence Magazine*, Vol. 4, No. 2, 2009, pp. 24–38.

[24] *http://scikit-learn.org/stable/*

[25] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang and J. Wang, Geospatial and temporal dynamics of application usage in cellular data networks, *IEEE Transactions Mobile Computing*, Vol. 14, No. 7, 2015, pp. 1369–1381.

[26] Y. Shu, M. Yu, J. Liu and O. Yang, Wireless traffic modeling and prediction using seasonal ARIMA models, *Proceedings of IEEE ICC'2003*, pp. 1675–1679.

[27] J. A. K. Suykens, J. Vandewalle, Least squares support vector machine classifiers, *Neural Processing Letters*, Vol. 9, No. 3, 1999, pp. 293–300.

[28] J. Tang, G. Xue and C. Chandler, Interference-aware routing and bandwidth allocation for QoS provisioning in multihop wireless networks, Wireless Communications and Mobile Computing (WCMC), Vol. 5, No. 8, 2005, pp. 933–944.

[29] J. Tang, G. Xue and W. Zhang, Cross-layer optimization for end-to-end rate allocation in multi-radio wireless mesh networks *Wireless Networks Journal*, Vol. 15, No. 1, 2009, pp. 53–64.

[30] V. K. Tumuluru, P. Wang and D. Niyato, Channel status prediction for cognitive radio networks, *Wireless Communications and Mobile Computing*, Vol. 12, No. 10, 2012, pp. 862–874.

[31] K. Tutschku and P. Tran-Gia, Spatial traffic estimation and characterization for mobile communication network design, *IEEE Journal on Selected Areas on Communications*, Vol. 16, No. 5, 1998, pp. 804–811.

[32] P. Vincent, H. Larochelle, Y. Bengio and P.A. Manzagol, Extracting and composing robust features with denoising autoencoders, *Proceedings of ICML*, 2008, pp. 1096–1103.

[33] Z. Wang and S. Salous, Spectrum occupancy statistics and time series models for cognitive radio, *Journal of Signal Processing Systems*, Vol. 62, No. 2, 2011, pp. 145–155.

[34] Y. LeCun, Y. Bengio and G. Hinton, Deep learning, *Nature*, Vol. 521, No. 7553, 2015, pp. 436–444.

[35] L. Xiang, X. Ge, C. Liu, L. Shu, and C. Wang, A new hybrid network traffic prediction method, *Proceedings of IEEE Globecom'2010*.

[36] W. Zaremba and I. Sutskever, Learning to execute, *arXiv preprint arXiv'2014*, Vol. 1410.4615.

[37] Y. Zhang, M. Roughan, W. Willinger, and L. Qiu, Spatio-temporal compressive sensing and internet traffic matrices, *Proceedings of ACM Sigcomm'2009*, pp. 267–278.

[38] B. Zhou, D. He, Z. Sun and W. H. Ng, Traffic modeling and prediction using ARIMA/GARCH model, *Proceedings of HET-NETs*, 2005.