A Long Short-Term Memory Recurrent Neural Network Framework for Network Traffic Matrix Prediction

Abdelhadi Azzouni and Guy Pujolle

LIP6 / UPMC; Paris, France {abdelhadi.azzouni,guy.pujolle}@lip6.fr

***Abstract*—Network Traffic Matrix (TM) prediction is defined as the problem of estimating future network traffic from the previous and achieved network traffic data. It is widely used in network planning, resource management and network security. Long Short-Term Memory (LSTM) is a specific recurrent neural network (RNN) architecture that is well-suited to learn from experience to classify, process and predict time series with time lags of unknown size. LSTMs have been shown to model temporal sequences and their long-range dependencies more accurately than conventional RNNs. In this paper, we propose a LSTM RNN framework for predicting Traffic Matrix (TM) in large networks. By validating our framework on real-world data from GÉANT network, we show that our LSTM models converge quickly and give state of the art TM prediction performance for relatively small sized models.**

keywords - Traffic Matrix, Prediction, Neural Networks, Long Short-Term Mermory

I. Introduction

Most of the decisions that network operators make depend on how the traffic flows in their network. However, although it is very important to accurately estimate traffic parameters, current routers and network devices do not provide the possibility for real-time monitoring, hence network operators cannot react effectively to the traffic changes. To cope with this problem, prediction techniques have been applied to predict network parameters and therefore be able to react to network changes in near real-time.

The predictability of network traffic parameters is mainly determined by their statistical characteristics and the fact that they present a strong correlation between chronologically ordered values. Network traffic is characterized by: self-similarity, multiscalarity, long-range dependence and a highly nonlinear nature (insufficiently modeled by Poisson and Gaussian models) [2].

A network TM presents the traffic volume between all pairs of origin and destination (OD) nodes of the network at a certain time t. The nodes in a traffic matrix can be Points-of-Presence (PoPs), routers or links.

Having an accurate and timely network TM is essential for most network operation/management tasks such as traffic accounting, short-time traffic scheduling or re-routing, network design, long-term capacity planning, and network anomaly detection. For example, to detect DDoS attacks in their early stage, it is necessary to be able to detect high-volume traffic clusters in near real-time. Another example is, upon congestion occurrence in the network, traditional routing protocols cannot react immediately to adjust traffic distribution, resulting in high delay, packet loss and jitter. Thanks to the early warning, a proactive prediction-based approach will be faster, in terms of congestion identification and elimination, than reactive methods which detect congestion through measurements, only after it has significantly influenced the network operation.

Several methods have been proposed in the literature for network traffic forecasting. These can be classified into two categories: linear prediction and nonlinear prediction. The most widely used traditional linear prediction methods are: a) the ARMA/ARIMA model [3], [6], [7] and b) the HoltWinters algorithm [3]. The most common nonlinear forecasting methods involve neural networks (NN) [3], [8], [9]. The experimental results from [13] show that nonlinear traffic prediction based on NNs outperforms linear forecasting models (e.g. ARMA, ARAR, HW) which cannot meet the accuracy requirements. Choosing a specific forecasting technique is based on a compromise between the complexity of the solution, characteristics of the data and the desired prediction accuracy. [13] suggests if we take into account both precision and complexity, the best results are obtained by the Feed Forward NN predictor with multiresolution learning approach.

Unlike feed forward neural networks (FFNN), Recurrent Neural Network (RNNs) have cyclic connections over time. The activations from each time step are stored in the internal state of the network to provide a temporal memory. This capability makes RNNs better suited for sequence modeling tasks such as time series prediction and sequence labeling tasks.

Long Short-Term Memory (LSTM) is a RNN architecture that was designed by Hochreiter and Schmidhuber [15] to address the vanishing and exploding gradient problems of conventional RNNs. RNNs and LSTMs have been successfully used for handwriting recognition [1], language modeling, phonetic labeling of acoustic frames [10].

In this paper, we present a LSTM based RNN framework which makes more effective use of model parameters to train prediction models for large scale TM prediction. We train and compare our LSTM models at various numbers of parameters and configurations. We show that LSTM models converge quickly and give state of the art TM prediction performance

for relatively small sized models. Note that we do not address the problem of TM estimation in this paper and we suppose that historical TM data is already accurately obtained.

The remainder of this paper is organized as follows: Section II summarizes time-series prediction techniques. LSTM RNN architecture and equations are detailed in section III. We detail the process of feeding our LSTM architecture and predicting TM in section IV. The prediction evaluation and results are presented in section V. Related work is presented in section VI and the paper is concluded by section VII.

II. Time Series Prediction

In this section, we give a brief summary of various linear predictors based on traditional statistical techniques, such as ARMA (Autoregressive Moving Average), ARIMA (Autoregressive Integrated Moving Average), ARAR (Autoregressive Autoregressive) and HW (HoltWinters) algorithm. And nonlinear time series prediction with neural networks.

1) Linear Prediction:

a) ARMA model: The time series $\{X_t\}$ is called an ARMA(p, q) process if $\{X_t\}$ is stationary (i.e. its statistical properties do not change over time) and

$$X_t - \phi_1 X_{t-1} - ... - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + ... + \theta_q Z_{t-q} \quad (1)$$

where $\{Z_t\} \approx WN(0, \sigma^2)$ is white noise with zero mean and variance σ^2 and the polynomials $\phi(z) = 1 - \phi_1 z - ... - \phi_p z^p$ and $\theta(z) = 1 + \theta_1 z + ... + \theta_q z^q$ have no common factors.

The identification of a zero-mean ARMA model which describes a specific dataset involves the following steps [20]: a) order selection (p, q); b) estimation of the mean value of the series in order to subtract it from the data; c) determination of the coefficients $\{\phi_i, i = \overline{1,p}\}$ and $\{\theta_i, i = \overline{1,q}\}$; d) estimation of the noise variance σ^2. Predictions can be made recursively using:

$$\widehat{X}_{n+1} = \begin{cases} \sum_{j=1}^{n} \theta_{nj}(X_{n+1-j} - \widehat{X}_{n+1-j}) & if 1 \leq n \leq m) \\ \sum_{j=1}^{q} \theta_{nj}(X_{n+1-j} - \widehat{X}_{n+1-j}) & \\ +\phi_1 X_n + .. + \phi_p X_{n+1-p} & if n \geq m \end{cases}$$

where $m = max(p, q)$ and θ_{nj} is determined using the innovations algorithm.

b) ARIMA model: A ARIMA(p, q, d) process is described by:

$$\phi(B)(1 - B)^d X_t = \theta(B) Z_t \quad (2)$$

where ϕ and θ are polynomials of degree p and q respectively, $(1-B)$ represents the differencing operator, d indicates the level of differencing and B is the backward-shift operator, i.e. $B^j X_t = X_{t-j}$

c) ARAR algorithm: The ARAR algorithm applies memory-shortening transformations, followed by modeling the dataset as an AR(p) process: $X_t = \phi_1 X_{t-1} + .. + \phi_p X_{t-p} + Z_t$

The time series $\{Y_t\}$ of long-memory or moderately long-memory is processed until the transformed series can be declared to be short-memory and stationary:

$$S_t = \psi(B) Y_t = Y_t + \psi_1 Y_{t-1} + ... + \psi_k Y_{t-k} \quad (3)$$

The autoregressive model fitted to the mean-corrected series $X_t = S_t - \overline{S}$, $t = \overline{k+1, n}$, where $\overline{S}$ represents the sample mean for $S_{k+1}, ..., S_n$, is given by $\phi(B) X_t = Z_t$, where $\phi(B) = 1 - \phi_1 B - \phi_{l_1} B^{l_1} - \phi_{l_2} B^{l_2} - \phi_{l_3} B^{l_3}, \{Z_t\} \approx WN(0, \sigma^2)$, while the coefficients ϕ_j and the variance σ^2 are calculated using the YuleWalker equations described in [20]. We obtain the relationship:

$$\xi(B) Y_t = \phi(1) \overline{S} + Z_t \quad (4)$$

where $\xi(B) Y_t = \psi(B) \varphi(B) = 1 + \xi_1 B + ... + \xi_{k+l_3} B^{k+l_3}$ From the following recursion relation we can determine the linear predictors

$$P_n Y_{n+h} = -\sum_{j=1}^{k+l_3} \xi P_n Y_{n+h-j} + \phi(1) \overline{S} \quad h \geq 1 \quad (5)$$

with the initial condition $P_n Y_{n+h} = Y_{n+h}$ for $h \leq 0$.

d) HoltWinters algorithm: The HoltWinters forecasting algorithm is an exponential smoothing method that uses recursions to predict the future value of series containing a trend. If the time series has a trend, then the forecast function is:

$$\widehat{Y}_{n+h} = P_n Y_{n+h} = \widehat{a}_n + \widehat{b}_n h \quad (6)$$

where $\widehat{a}_n$ and $\widehat{b}_n$ are the estimates of the level of the trend function and the slope respectively. These are calculated using the following recursive equations:

$$\begin{cases} \widehat{a}_{n+1} = \alpha Y_{n+1} + (1 - \alpha)(\widehat{a}_n + \widehat{b}_n) \\ \widehat{b}_{n+1} = \beta(\widehat{a}_{n+1} - \widehat{a}_n) + (1 - \beta)\widehat{b}_n \end{cases} \quad (7)$$

Where $\widehat{Y}_{n+1} = P_n Y_{n+1} = \widehat{a}_n + \widehat{b}_n$ represents the one-step forecast. The initial conditions are: $\widehat{a}_2 = Y_2$ and $\widehat{b}_2 = Y_2 - Y_1$. The smoothing parameters α and β can be chosen either randomly (between 0 and 1), or by minimizing the sum of squared one-step errors $\sum_{i=3}^{n} (Y_i - P_{i-1} Y_i)^2$ [20].

2) Neural Networks for Time Series Prediction: Neural Networks (NN) are widely used for modeling and predicting network traffic because they can learn complex non-linear patterns thanks to their strong self-learning and self- adaptive capabilities. NNs are able to estimate almost any linear or non-linear function in an efficient and stable manner, when the underlying data relationships are unknown. The NN model is a nonlinear, adaptive modeling approach which, unlike the techniques presented above, relies on the observed data rather than on an analytical model. The architecture and the parameters of the NN are determined solely by the dataset. NNs are characterized by their generalization ability, robustness, fault tolerance, adaptability, parallel processing ability, etc [14].

A neural network consists of interconnected nodes, called neurons. The interconnections are weighted and the weights are also called parameters. Neurons are organized in layers: a) an input layer, b) one or more hidden layers and c) an output layer. The most popular NN architecture is feed-forward in which the information goes through the network only in the

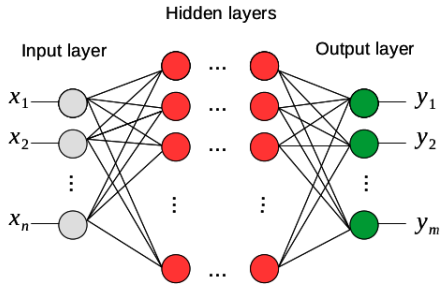

Fig. 1: Feed Forward Deep Neural Network

forward direction, i.e. from the input layer towards the output layer, as illustrated in figure 1.

Prediction using a NN involves two phases: a) the training phase and b) the test (prediction) phase. During the training phase, the NN is supervised to learn from the data by presenting the training data at the input layer and dynamically adjusting the parameters of the NN to achieve the desired output value for the input set. The most commonly used learning algorithm to train NNs is called the backpropagation algorithm. The idea of the backpropagation is to propagate of the error backward, from the output to the input, where the weights are changed continuously until the output error falls below a preset value. In this way, the NN learns correlated patterns between input sets and the corresponding target values. The prediction phase represents the testing of the NN. A new unseen input is presented to the NN and the output is calculated, thereby predicting the outcome of new input data.

III. Long Short Term Memory Neural Networks

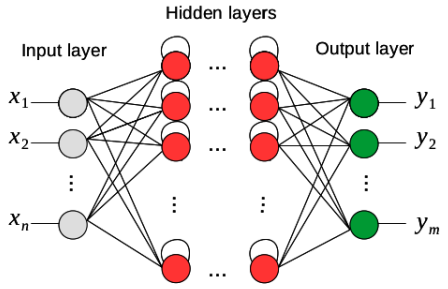

Fig. 2: Deep Recurrent Neural Network

FFNNs can provide only limited temporal modeling by operating on a fixed-size window of TM sequence. They can only model the data within the window and are unsuited to handle historical dependencies. By contrast, recurrent neural networks or deep recurrent neural networks (figure 2) contain cycles that feed back the network activations from a previous time step as inputs to influence predictions at the current time step (figure 3). These activations are stored in the internal states of the network as temporal contextual information [10].

However, training conventional RNNs with the gradient-based back-propagation through time (BPTT) technique is

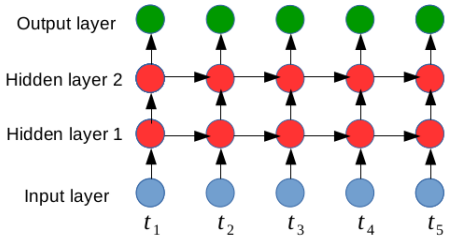

Fig. 3: DRNN learning over time

difficult due to the vanishing gradient and exploding gradient problems. The influence of a given input on the hidden layers, and therefore on the network output, either decays or blows up exponentially when cycling around the networks recurrent connections. These problems limit the capability of RNNs to model the long range context dependencies to 5-10 discrete time steps between relevant input signals and output [11].

To address these problems, an elegant RNN architecture Long Short-Term Memory (LSTM) has been designed [15]. LSTMs and conventional RNNs have been successfully applied to sequence prediction and sequence labeling tasks. LSTM models have been shown to perform better than RNNs on learning context- free and context-sensitive languages for example [5].

A. LSTM Architecture

The architecture of LSTMs is composed of units called memory blocks. Memory block contains memory cells with self-connections storing (remembering) the temporal state of the network in addition to special multiplicative units called gates to control the flow of information. Each memory block contains an input gate to control the flow of input activations into the memory cell, an output gate to control the output flow of cell activations into the rest of the network and a forget gate (figure 4).

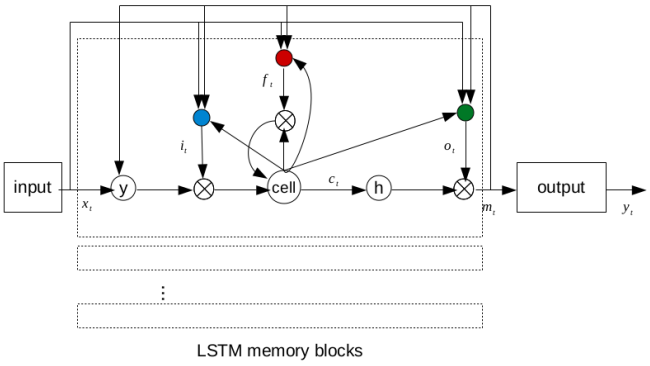

Fig. 4: LSTM node

The forget gate scales the internal state of the cell before adding it back to the cell as input through self recurrent connection, therefore adaptively forgetting or resetting the cells memory. The modern LSTM architecture also contains

peephole connections from its internal cells to the gates in the same cell to learn precise timing of the outputs [4].

B. LSTM Equations

In this subsection we provide the equations for the activation (forward pass) and gradient calculation (backward pass) of an LSTM hidden layer within a recurrent neural network. The backpropagation through time algorithm with the exact error gradient is used to train the network. The LSTM equations are given for a single memory block only. For multiple blocks the calculations are simply repeated for each block, in any order [12].

a) Notations:

- w_{ij} the weight of the connection from unit i to unit j
- a_i^t the network input to some unit j at time t
- b_i^t the value of the same unit after the activation function has been applied
- ι input gate, ϕ forget gate, ω output gate
- C set of memory cells of the block
- s_c^t state of cell c at time t (i.e. the activation of the linear cell unit)
- f the activation function of the gates, g cell input activation functions, h cell output activation functions
- I the number of inputs, K the number of outputs, H number of cells in the hidden layer

Note that only the cell outputs b_c^t are connected to the other blocks in the layer. The other LSTM activations, such as the states, the cell inputs, or the gate activations, are only visible within the block.

We use the index h to refer to cell outputs from other blocks in the hidden layer.

As with standard RNNs the forward pass is calculated for a length T input sequence x by starting at t = 1 and recursively applying the update equations while incrementing t, and the BPTT backward pass is calculated by starting at t = T , and recursively calculating the unit derivatives while decrementing t (see Section 3.2 for details). The final weight derivatives are found by summing over the derivatives at each timestep, as expressed in Eqn. (3.34). Recall that

$$\delta_j^t = \frac{\partial O}{\partial a_j^t} \tag{8}$$

Where O is the objective function used for training.

The order in which the equations are calculated during the forward and backward passes is important, and should proceed as specified below. As with standard RNNs, all states and activations are set to zero at $t = 0$, and all δ terms are zero at $t = T + 1$.

b) Forward Pass: **Input Gates**

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{9}$$

$$b_\iota^t = f(a_\iota^t) \tag{10}$$

Forget Gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{11}$$

$$b_\phi^t = f(a_\phi^t) \tag{12}$$

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{13}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \tag{14}$$

Output Gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^{t-1} \tag{15}$$

$$b_\omega^t = f(a_\omega^t) \tag{16}$$

Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t) \tag{17}$$

c) Backward Pass:

$$\epsilon_c^t = \frac{\partial O}{\partial b_c^t} \tag{18}$$

$$\epsilon_s^t = \frac{\partial O}{\partial s_c^t} \tag{19}$$

Cell Outputs

$$\delta_\iota^t = f'(a_\iota^t) \sum_{c=1}^{C} g(a_c^t) \epsilon_s^t \tag{20}$$

Output Gates

$$\delta_\phi^t = f'(a_\phi^t) \sum_{c=1}^{C} s_c^{t-1} \epsilon_s^t \tag{21}$$

States

$$\delta_c^t = b_\iota^t g'(a_c^t) \epsilon_s^t \tag{22}$$

Cells

$$\epsilon_s^t = b_\omega^t h'(s_c^t) \epsilon_c^t + b_\phi^{t+1} \epsilon_s^{t+1} + w_{c\iota} \delta_\iota^{t+1} + w_{c\phi} \delta_\phi^{t+1} + w_{c\omega} \delta_\omega^{t+1} \tag{23}$$

Forget Gates

$$\delta_\omega^t = f'(a_\omega^t) \sum_{c=1}^{C} h(s_c^t) \epsilon_c^t \tag{24}$$

Input Gates

$$\epsilon_s^t = \sum_{k=1}^{K} w_{ck} \delta_k^t + \sum_{h=1}^{H} w_{ch} \delta_h^{t+1} \tag{25}$$

where f() (frequently noted as $\sigma(.)$) is the standard logistic sigmoid function defined in Eq.(8), g() and h() are the transformations of function () whose range are [-2,2] and [-1,1] respectively: $\sigma(x) = \frac{1}{1+e^{-x}}$, $g(x) = \frac{4}{1+e^{-x}} - 2$ and $h(x) = \frac{2}{1+e^{-x}} - 1$

IV. Traffic Matrix Prediction Using LSTM RNN

In this section we describe the use of a deep LSTM architecture with a deep learning method to extract the dynamic features of network traffic and predict the future TM. This architecture can deeply excavate mutual dependence among the traffic entries in various timeslots.

A. Problem Statement

Let N be the number of nodes in the network. The N-by-N traffic matrix is denoted by Y such as an entry y_{ij} represents the traffic volume flowing from node i to node j. We add the time dimension to obtain a structure of N-by-N-by-T tensor (vector of matrices) S such as an entry s_{ij}^t represents the volume of traffic flowing from node i to node j at time t, and T is the total number of time-slots. The traffic matrix prediction problem is defined as solving the predictor of Y^t (denoted by $\widehat{Y}^t$) via a series of historical and measured traffic data set (Y^{t-1}, Y^{t-2}, Y^{t-3}, ..., Y^{t-T}). The main challenge here is how to model the inherent relationships among the traffic data set so that one can exactly predict Y^t.

B. Feeding The LSTM RNN

To effectively feed the LSTM RNN, we transform each matrix Y^t to a vector X^t (of size $N \times N$) by concatenating its N rows from top to bottom. X^t is called traffic vector (TV). Note that x_n entries can be mapped to the original y_{ij} using the relation $n = i \times N + j$. Now the traffic matrix prediction problem is defined as solving the predictor of X^t (denoted by $\widehat{X}^t$) via a series of historical measured traffic vectors (X^{t-1}, X^{t-2}, X^{t-3}, ..., X^{t-T}).

One possible way to predict the traffic vector X^t is to predict one component x_n^t at a time by feeding the LSTM RNN one vector $(x_0^t, x_1^t, ..., x_{N^2}^t)$ at a time. This is based on the assumption that each OD traffic is independent from all other ODs which was shown to be wrong by [21]. Hence, considering the previous traffic of all ODs is necessary to obtain a more accurate prediction of the traffic vector.

Continuous Prediction Over Time: Real-time prediction of traffic matrix requires continuous feeding and learning. Over time, the total number of time-slots become too big resulting in high computational complexity. To cope with this problem, we introduce the notion of learning window (denoted by W) which indicates a fixed number of previous time-slots to learn from in order to predict the current traffic vector X^t (Fig. 5).

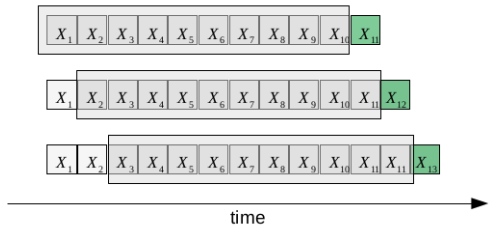

Fig. 5: Sliding learning window

We construct the W-by-N^2 traffic-over-time matrix (that we denote by M) by putting together W vectors (X^{t-1}, X^{t-2}, X^{t-3}, ..., X^{t-W}) ordered by time. Note that $T \geq W$ (T being the total number of historical matrices) and the number of matrices M is equal to T/W.

C. Performance Metric

To quantitatively assess the overall performance of our LSTM model, Mean Square Error (MSE) is used to estimate the prediction accuracy. MSE is a scale dependent metric which quantifies the difference between the forecasted values and the actual values of the quantity being predicted by computing the average sum of squared errors:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y}_i)^2 \tag{26}$$

where y_i is the observed value, $\widehat{y}_i$ is the predicted value and N represents the total number of predictions.

V. Experiments and Evaluation

In this section, we will evaluate the prediction accuracy of our method using real traffic data from the GÉANT backbone networks [16]. GÉANT is the pan-European research network. GÉANT has a PoP in each European country and it carries research traffic from the European National Research and Education Networks (NRENs) connecting universities and research institutions. As of 2005, the GÉANT network was made up of 23 peer nodes interconnected using 38 links. In addition, GÉANT has 53 links with other domains.

2004-timeslot traffic matrix data is sampled from the GÉANT network by 15-min interval [17] for several months. In our simulation, we also compare our prediction and estimation methods with a state-of-the-art method, that is, the PCA method introduced in the above section.

To evaluate our method on short term traffic matrix prediction, we consider a set of 309 traffic matrices measured between 01-01-2005 00am and 04-01-2005 5:15am. As detailed in section IV-B, we transform the matrices to vectors of size 529 each and we concatenate the vectors to obtain the traffic-over-time matrix M of size 309×529. We split M into two matrices, training matrix M_{train} and validation matrix M_{test} of sizes 263 and 46 consecutively. M_{train} is used to train the LSTM RNN model and M_{test} is used to evaluate and validate its accuracy. Finally, We normalize the data by dividing by the maximum value.

We use Keras library [18] to build and train our model. The training is done on a Intel core i7 machine with 16GB memory. Figures 6 and 7 show the variation of the prediction error when using different numbers of hidden units and hidden layers respectively. Finally, figure 8 compares the prediction error of the different prediction methods presented in this paper and shows the superiority of LSTM. Note that, the prediction results of the linear predictors and FFNN are obtained from [13] and they represent the error of predicting only one traffic value which is obviously an easier task than predicting the whole traffic matrix.

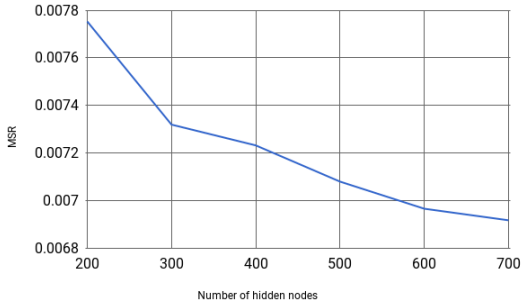

Fig. 6: MSE over size of hidden layer

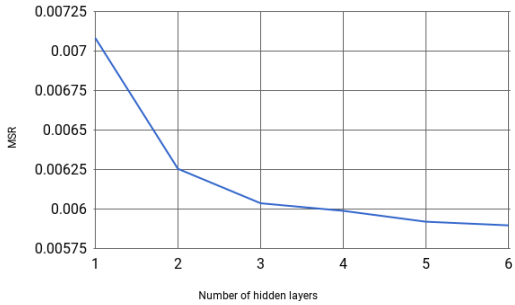

Fig. 7: MSE over number of hidden layers (500 nodes each)

VI. Related Work

Various methods have been proposed to predict traffic matrix. [13] evaluates and compares traditional linear prediction models (ARMA, ARAR, HW) and neural network based prediction with multi-resolution learning. The results show that NNs outperform traditional linear prediction methods which cannot meet the accuracy requirements. [21] proposes a FARIMA predictor based on an α-stable non-Gaussian self-similar traffic model. [19] compares three prediction methods: Independent Node Prediction (INP), Total Matrix Prediction with Key Element Correction (TMP-KEC) and Principle Component Prediction with Fluctuation Component Correction (PCP-FCC). INP method does not consider the correlations among the nodes, resulting in unsatisfying prediction error. TMP-KEC method reduces the forecasting error of key elements as well as that of the total matrix. PCP-FCC method improves the overall prediction error for most of the OD flows.

VII. Conclusion

In this work, we have shown that LSTM RNN architectures are well suited for traffic matrix prediction. We have proposed a data pre-processing and RNN feeding technique that achieves high prediction accuracy in a few seconds of computation (approximately 60 seconds for one hidden layer of 300 nodes). The results of our evaluations show that LSTM RNNs outperforms traditional linear methods and feed forward neural networks by many orders of magnitude.

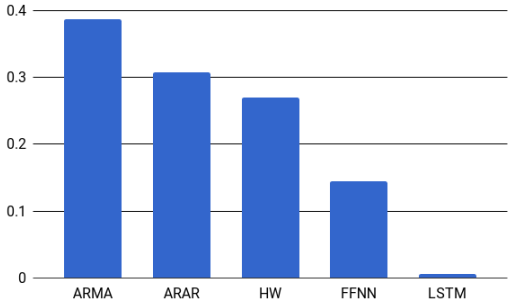

Fig. 8: Comparison of prediction methods

References

[1] Liwicki, Marcus, et al. "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks." Proc. 9th Int. Conf. on Document Analysis and Recognition. Vol. 1. 2007.

[2] W. Leland, M. Taqqu, W. Willinger and D. Wilson, On the self-similar nature of Ethernet traffic, In Proc. SIGCOMM 93, pp.183193, 1993.

[3] P. Cortez, M. Rio, M. Rocha, P. Sousa, Internet Traffic Forecasting using Neural Networks, International Joint Conference on Neural Networks, pp. 26352642. Vancouver, Canada, 2006.

[4] Felix A. Gers, Nicol N. Schraudolph, and Jurgen Schmidhuber, Learning precise timing with LSTM recurrent networks, Journal of Machine Learning Research , vol. 3, pp. 115143, Mar. 2003

[5] Felix A. Gers and Jurgen Schmidhuber, LSTM recurrent networks learn simple context free and context sensitive lan- guages, IEEE Transactions on Neural Networks , vol. 12, no. 6, pp. 13331340, 2001

[6] H. Feng, Y. Shu, Study on Network Traffic Prediction Techniques, International Conference on Wireless Communications, Networking and Mobile Computing, pp. 10411044. Wuhan, China, 2005.

[7] J. Dai, J. Li, VBR MPEG Video Traffic Dynamic Prediction Based on the Modeling and Forecast of Time Series, Fifth International Joint Conference on INC, IMS and IDC, pp. 17521757. Seoul, Korea, 2009.

[8] V. B. Dharmadhikari, J. D. Gavade, An NN Approach for MPEG Video Traffic Prediction, 2nd International Conference on Software Technology and Engineering, pp. V1-57V1-61. San Juan, USA, 2010.

[9] A. Abdennour, Evaluation of neural network architectures for MPEG-4 video traffic prediction, IEEE Transactions on Broadcasting, Volume 52, No. 2, pp. 184192. ISSN 0018-9316, 2006.

[10] Sak, Hasim, Andrew W. Senior, and Franoise Beaufays. "Long short-term memory recurrent neural network architectures for large scale acoustic modeling." Interspeech. 2014.

[11] Sak et al. Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling. https://arxiv.org/pdf/1402.1128.pdf

[12] Alex Graves. Supervised Sequence Labelling with Recurrent Neural Networks. http://www.cs.toronto.edu/∼graves/phd.pdf

[13] Barabas, Melinda, et al. "Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition." Intelligent Computer Communication and Processing (ICCP), 2011 IEEE International Conference on. IEEE, 2011.

[14] H. Feng, Y. Shu, Study on Network Traffic Prediction Techniques, International Conference on Wireless Communications, Networking and Mobile Computing, pp. 10411044. Wuhan, China, 2005.

[15] Hochreiter, Sepp, and Jrgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[16] https://www.geant.org/Projects/GEANT_Project_GN4

[17] Uhlig, Steve, et al. "Providing public intradomain traffic matrices to the research community." ACM SIGCOMM Computer Communication Review 36.1 (2006): 83-86.

[18] https://keras.io/

[19] Liu, Wei, et al. "Prediction and correction of traffic matrix in an IP backbone network." Performance Computing and Communications Conference (IPCCC), 2014 IEEE International. IEEE, 2014.

[20] P. J. Brockwell, R. A. Davis, Introduction to Time Series and Forecasting, Second Edition. Springer-Verlag,ISBN 0-387-95351-5, 2002.

[21] Wen, Yong, and Guangxi Zhu. "Prediction for non-gaussian self-similar traffic with neural network." Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on. Vol. 1. IEEE, 2006.