2018 17th IEEE International Conference on Machine Learning and Applications

Network Traffic Prediction Using Recurrent Neural Networks

Nipun Ramakrishnan and Tarun Soni

***Abstract*—The network traffic prediction problem involves predicting characteristics of future network traffic from observations of past traffic. Network traffic prediction has a variety of applications including network monitoring, resource management, and threat detection. In this paper, we propose several Recurrent Neural Network (RNN) architectures (the standard RNN, Long Short Term Memory (LSTM) networks, and Gated Recurrent Units (GRU)) to solve the network traffic prediction problem. We analyze the performance of these models on three important problems in network traffic prediction: volume prediction, packet protocol prediction, and packet distribution prediction. We achieve state of the art results on the volume prediction problem on public datasets such as the GEANT and Abilene networks. We also believe this is the first work in the domain of protocol prediction and packet distribution prediction using RNN architectures. In this paper, we show that RNN architectures demonstrate promising results in all three of these domains in network traffic prediction, outperforming standard statistical forecasting models significantly.**

I. Introduction

Over the past decade, network traffic has become increasingly complex from a variety of perspectives. From a load management perspective, in the year 2016, global IP traffic reached a record high of 1.2 ZB [1]. From a cybersecurity perspective, network attacks such as Distributed Denial of Service (DDoS) have become increasingly sophisticated. Modeling network traffic has found usage in management as well as cybersecurity for increasingly complex traffic patterns.

Network traffic prediction problems arise in the form of modeling the volume of traffic across nodes in a network in the form of bytes sent across the network. These nodes can represent routers or more generally, any type of Point of Presence in which a communication interface occurs. In this paper, we investigate predicting the number of bytes in future time steps as well as the total number of packets sent. This second metric can be particularly useful in modern applications, in which network traffic packets are often encrypted to hide the number of bytes in the packet. Another characteristic of network traffic prediction is the ability to determine packet level information in future time steps. Specifically, this comes in the form of protocol classification and protocol distribution prediction. The protocol classification problem can be defined as follows: given a series of types of packet protocols (HTTP, TCP, UDP, etc) at particular times, to what extent can future packet protocols be predicted. An extension of this problem involves the packet distribution prediction problem: forecasting the distribution of packet protocols at particular ranges of times. We obtain highly accurate results on both of these problems. These versions of prediction have not been investigated before to our knowledge and have important applications in both a network monitoring context and cybersecurity context. One example is in the presence of DDoS attacks, an influx of TCP, HTTP, or ICMP traffic can be seen, but often disguised to look like normal traffic from a volume perspective [2]. However, in a system that can forecast what type of packets it can expect at particular times as well as the distribution of packets, this type of attack is much harder to disguise.

There are a few factors that contribute to the overall difficulty of the network traffic prediction problem. Predicting network traffic data relies heavily on the statistical nature of this data and the notion that network data is chronologically dependent. Some statistical characteristics that make prediction of network traffic particularly difficult is self-similarity and its highly nonlinear nature. This nonlinear nature has been shown to be insufficiently modeled by Poisson or Gaussian distributions [3]. Furthermore, from a data dependency viewpoint, network traffic is characterized by long-term dependencies that most statistical models have difficulty capturing.

We choose to frame network traffic prediction problem as a time-series prediction problem. Time series prediction problems have had several statistical approaches such as Autoregressive Integrated Moving Average (ARIMA), Exponentional Smoothing (EM), and the Holt-Winters algorithm [4], [5]. Other methods have used probabilistic approaches such as Bayesian Networks or Hidden Markov Models [6], [7]. An approach to time series modeling from the deep learning perspective has proved to be quite successful. Recurrent Neural Networks (RNN's) are particularly powerful models that have demonstrated high accuracy in time series forecasting [8].

The success of RNN's and variations of them such as Long Short Term Memory (LSTM) cells and Gated Recurrent Units (GRU's) in times series forecasting relies in their ability to capture more complex and nonlinear behavior than conventional linear statistical models. It has been shown that RNN's, in theory, can approximate any nonlinear dynamic system to an arbitrary accuracy [9]. Another advantage of these architectures is their ability to capture long range dependencies in time series data [10]. Linear models are also not as accurate when it comes to predicting sudden nonlinearities such as spikes in volume that are common in the network traffic prediction domain. These advantages are especially important in the network traffic prediction domain, since network traffic

N. Ramakrishnan is a undergraduate student in the Department of Electrical Engineering and Computer Sciences,University of California, Berkeley, CA, 94720 USA e-mail: nipun.ramk@berkeley.edu.

T. Soni is a professor at the University of Minnesota as well as a Research Scientist at Northrop Grumman in the Communications division

978-1-5386-6805-4/18/$31.00 ©2018 IEEE
DOI 10.1109/ICMLA.2018.00035

data has demonstrated high degrees of non-linearity and long-term dependencies.

In this paper, we propose RNN architectures as an approach towards solving the network volume prediction problem. We also demonstrate the performance of RNN's on the packet prediction and packet distribution problem. Our contributions can be summarized as follows:

- We demonstrate high accuracy in traffic volume prediction for three RNN architectures – the standard RNN, LSTM, and GRU. Our accuracy is comparable to and in some cases better than existing solutions proposed in literature that analyzed smaller samples of public datasets.
- We propose a novel method of using RNN's to solve the protocol classification problem.
- We develop a novel method to approach the application of packet distribution prediction.

The structure of the rest of this paper is as follows: Section II reviews previous work in the network traffic prediction domain. Section III gives an overview of RNN architectures and lays the mathematical foundation of these models. Section IV details data collection and training methods. We present our results and discuss trade-offs in section V. Lastly, we conclude our paper in section VI.

II. Related Work

Several methods have been proposed to solve the network traffic volume prediction problem. [11] attempts to model network traffic using statistical models such as ARMA/ARIMA and the Holt-Winters (HW) algorithm. They find that these methods often have difficulty modeling the nonlinear nature of network traffic volume. Their work finds that Feed Foreward Neural Networks (FFNN's) are much better predictors of network traffic volume than standard linear forecasting models. Another work that confirms this idea is [12], which studied a variety of training algorithms for FFNN's and their performance on volume prediction in network traffic. The analysis concludes that by carefully choosing the right algorithm to train these models, they obtain accurate results in short-term volume prediction. [13] analyzes the Recurrent Multilayer Perceptron to predict large scale traffic matrices. They demonstrate promising results on traffic simulations and parts of the Abilene network traffic dataset. [14] compares the LSTM model for traffic volume prediction to linear forecasting models as well as FFNN's. They find that the LSTM performs significantly better on the GEANT traffic dataset than other models in traffic matrix prediction.

In the packet protocol and distribution prediction aspect of this problem, there has been significantly less work. A majority of Internet traffic application classification such as [15], [16] treats packets independently without considering the time series aspect of the problem. [17] utilizes artificial neural networks to predict protocols of packets. All of these papers attempt to learn features in traffic data that correlate to particular protocol or application classifications. In this paper, we do not collect any extraneous features to make a future packet classification or distribution predictions – we only consider the time series aspect of the problem. With respect to the packet distribution problem, to our knowledge, there is no work that solves this aspect of network traffic prediction.

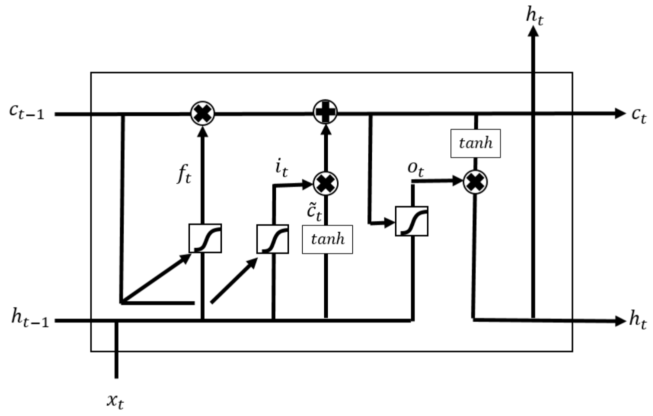

Fig. 1: LSTM Cell Structure

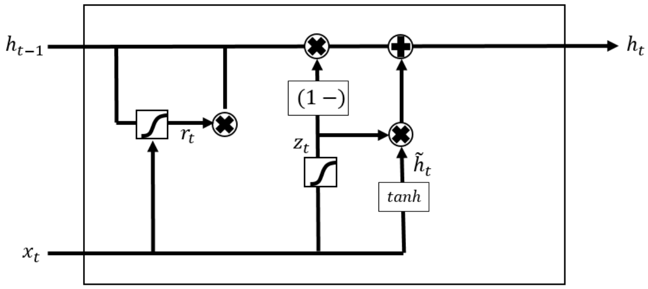

Fig. 2: GRU Structure

III. Recurrent Neural Networks

In traditional FFNN's, we are given an input vector of features $x = (x_1, \ldots, x_n)$, from which we can compute the output vector of the next hidden layer $h = (h_1, \ldots, h_n)$. An activation function is applied to each output of the hidden layer and this result is then passed to subsequent hidden layers. The output $y = (y_1, \ldots, y_n)$ is computed from the output of the last hidden layer. This relationship is captured by the following equations

$$h = f(Wx + b_h)$$

$$y = g(Wh + b_y)$$

where W represents the weight matrices between consecutive layers of the neural network, b represents the bias vector for each layer (b_h for hidden layer bias vector and b_y for output layer bias vector), f represents the activation function for the hidden layer (traditionally the sigmoid function), and g represents that activation function for the output layer (usually linear for regression problems or softmax for classification problems). The network is trained to minimize a loss function using numerical optimization methods such as stochastic gradient descent, calculating gradients through the backpropagation algorithm [18] [19].

The RNN model is quite similar to the fundamental FFNN model, with a few key differences. The most fundamental difference is that RNN's allows outputs from layers to cycle back into the network. This architecture gives the RNN an ability to encode the notion of dependencies among the data, a factor that the tradition FFNN cannot model. In the RNN model, the network takes an input sequence $x = (x_1, \ldots, x_T)$ and computes the hidden vector sequence $h = (h_1, \ldots, h_T)$ and output vector sequence $y = (y_1, \ldots, y_n)$ by iterating through the following equations from $t = 1$ to T:

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$
$$y_t = W_{hy}h_t + b_y$$

where W_{ij} represents the weight matrix between layer i to layer j. To train a RNN, similar optimization methods are used as in FFNN's; however, to compute gradients, the backpropagation through time algorithm is used [20].

One issue with training standard RNN's is the vanishing gradient problem [21], which can cause the network to lose it's ability to weight long term dependencies. Two notable architectures that have been proposed to solve this problem are LSTM's [22] and GRU's [23]. The LSTM architecture modifies the RNN model by adding a *memory unit*. The structure of this memory unit is shown in Figure 1. Each LSTM unit contains a memory cell c_t and the output of the LSTM unit h_t is

$$h_t = o_t \tanh(c_t)$$

where o_t, the *output gate*, determines the amount of memory exposure. The output is calculated by

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_0)$$

where σ represents the sigmoid activation function. The memory content c_t is updated by partially forgetting current memory and adding new memory $\tilde{c}_t$. Formally,

$$c_t = f_t c_t + i_t \tilde{c}_t$$
$$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

where f_t is the extent to which existing memory is forgotten, often called the *forget gate*. The quantity i_t represents the *input gate* or the degree to which new memory content is added to the cell. These gates are determined by the following equations:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_t + b_f)$$
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_t + b_i)$$

The use of these gates allows the LSTM to determine whether to keep existing memory or weigh new information over existing memory, which is a feature that is not present in the RNN model that overwrites its memory at each time step. From an intuitive perspective, this gives an LSTM a greater potential to capture long-term dependencies, since it can easily carry over information if it detects an important feature in the early stages of training.

Another variant of the RNN is the GRU, which has quite a similar architecture to a LSTM. The GRU also has gated units that control the flow of information inside the unit; however, unlike the LSTM, the GRU does not have separate memory cells. The gated unit structure is depicted in Figure 2. The activation h_t at time t of this unit is a linear combination of the activation at the previous time step h_{t-1} and the candidate activation $\tilde{h}_t$:

$$h_t = (1 - z_t)h_{t-1} + z_t\tilde{h}_t$$

where the the *update gate* z_t decides the extent to which the unit updates its activation. It is computed by

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1})$$

and the candidate activation $\tilde{h}_t$ is computed by

$$\tilde{h}_t = \tanh(W_{x\tilde{h}}x_t + W_{h\tilde{h}}(r_t \odot h_{t-1}))$$

where r_t represents a reset gate and $\odot$ is element wise multiplication of two vectors. The reset gate is defined as the extent to which the unit should remember it's previous state (e.g $r_t \approx 0$ makes the unit essentially forget and not factor in the previous state). The reset gate is computed by the following equation:

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1})$$

Notice the similarity of the equation for computing the reset gate with the equation for computing update gates.

IV. Data Collection and Training

Our goal is to be able to predict traffic volume in networks as well as future packet classifications and distributions. To test the performance of various architectures of RNN's in this problem domain, we collected network traffic data between two virtual machines. These virtual machines created real world network traffic through internal HTTP, TCP, and ICMP requests from the local network. We used the tool Wireshark to collect packet level data of this network traffic [24]. We collected data for a period of two hours from the network. In the rest of this paper, we refer to this dataset as the internal network dataset. We also evaluated the volume prediction aspect of our model on two publicly available traffic datasets: the Abilene network dataset [25] and the GEANT network dataset [26].

A. Volume Prediction

To perform volume predictions using our models, we accumulated packet data sampled every 5 seconds from the internal network dataset. In this 5 second window, we counted the number of packets and bytes sent across the network. The goal of the volume prediction model is to be able to predict the number of packets and bytes we can expect to see in a future 5 second time window.

To formally state the volume prediction problem, given a time series $((p_1, b_1), \dots, (p_T, b_T))$ of sampled volume of packets p and bytes b obtained from each 5 second time window, we want to be able to predict $((p_{T+1}, b_{T+1}), \dots, (p_{T+n}, b_{T+n}))$ for n steps into the future. To train our RNN architectures, we use a *sliding window* method as shown in Figure 8. This method uses packet/byte volumes at fixed previous t time steps $(x_1, \dots, x_t)$ as features to make a prediction for the number of packets or bytes at the next time step x_{t+1}. With this configuration, for N data points in the training set, our training data matrix takes the shape of a $(N-t)$-by-t matrix. Each data point represents one sampled volume of packets of bytes within a 5 second window.

We also perform byte predictions on both the Abilene and GEANT network datasets. The Abilene network consists of 12 nodes and 144 origin-destination flows. The GEANT network is a larger network that is composed of 23 nodes with 529 origin-destination flows. The generalized method to make volume predictions on a traffic matrix of N nodes is to process each N-by-N matrix at each time step as a row vector of N^2 dimension. With T time steps, our data matrix has dimension T-by-N^2. We split this data matrix into training and test sets for evaluation (discussed in detail in section V). Each column of our matrix corresponds to a particular flow in our matrix, and we train our model with the sliding window method on these columns and test them on future time steps. In our experiments with both the Abilene and GEANT network datasets, we look at a training sequence of the first $T = 1000$ time steps in each dataset.

B. Future Packet Classification and Distribution Prediction

While the volume prediction problem is treated more as a regression problem, this problem is a pure classification problem. To formally state this problem, given packet classifications $(k_1, \dots, k_T)$, we want to predict the packet classifications $(k_{T+1}, \dots, k_{T+n})$ for n steps into the future. Each k represents a possible protocol classification. In our internal dataset, we observed the following protocols: TCP, HTTP, ARP, and ICMP. For this problem, since each data point is a classification, we do not perform the sampling outlined in the volume prediction problem; we simply take every packet's classification as is and feed it into our model using the same sliding window technique. In a model with k classes, the classifier will predict a real number and we round this number to the nearest classification in the range of integers $[0, k-1]$. We evaluate our model's performance using the standard classification accuracy metric.

The packet distribution problem is defined as follows: given a distribution of packets $((p_1^1, \dots, p_1^k) \dots (p_T^1 \dots p_T^k))$ with k distinct protocols, where p_t^j represents the distribution of protocol j at time step t and $\sum_{n=1}^k p_t^n = 1$. We want to predict the packet distribution $((p_{T+1}^1, \dots, p_{T+1}^k) \dots ((p_{T+N}^1, \dots, p_{T+N}^k))$ for N time steps into the future. We can think of this problem as a histogram prediction problem in which each protocol has a specific frequency that we are trying to estimate at future ranges of times. To solve this problem, we treat it as a more general version of the volume prediction problem. After obtaining the sampled packet distributions for T time steps, each data point is a row vector of dimension k. Our data matrix is then of dimension T-by-k, and we apply the sliding window method outlined earlier to train our model.

Dataset	Internal Network				Abilene	GEANT
Model	MSE (Packets/s)	MSE (Bytes/s)	MSE Distribution	Accuracy (%)	MSE (GB/s)	MSE (KB/s)
NM	0.1152	0.2022	0.0288	58.4	0.0260	0.0099
MA	0.0377	0.0598	0.0164	77.9	0.0216	0.0141
AIRMA	0.0127	0.0099	0.0246	48.3	0.0153	0.0090
GRU	0.0030	0.0022	0.0083	83.8	0.0089	0.0083
LSTM	0.0028	0.0023	0.0083	92.9	0.0086	0.0081
RNN	0.0026	0.0029	0.0092	91.1	0.0088	0.0080

TABLE I: Summary of Results

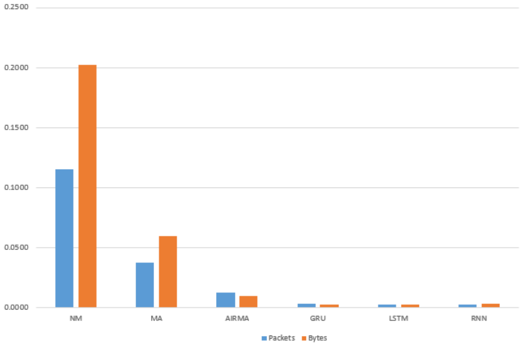

Fig. 3: Comparison of MSE for Internal Network

C. Model Parameters and Architecture

We built the RNN, LSTM, and GRU models using the Python Keras library. We found that the sliding window was a particularly important parameter in our network. To obtain the optimal sliding window parameter, we evaluated the model on several sliding window lengths and picked the sliding window that allowed for optimal performance. We present the results of these tests in section V. Each model was build with 400 nodes, two dense hidden layers as well as an output layer ReLU activation function. These hyperparameters were determined using random search [27].

D. Evaluating Performance of Models

To quantitatively evaluate the predictions of our models for future time steps, we used the Mean Squared Error (MSE) metric. This metric is defined as follows for N data points:

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

where y is the correct value and $\hat{y}$ is our model's prediction. It should be noted that while training our RNN architecture, we used standard training practices in which all input data was normalized to a range of between 0 and 1 by dividing by the maximum value in the dataset. Therefore, the MSE for our models will exclusively be between the 0 and 1, with values closer to 0 meaning higher accuracy in the model's predictions. For traffic matrix volume prediction, our MSE is the same definition, except averaged over every column of volume data.

To lay a groundwork for interpreting the MSE of our models, we compare our model's performance with three other models. The first model for prediction is simply to carry over the last observed value. Formally, the prediction at time step x_t is the value at time step x_{t-1}.

$$x_t = x_{t-1}$$

We refer to this model as the Naive Method (NM). The second model is to make a future prediction based on the average of past values or more formally

$$x_t = \frac{x_{t-1} + x_{t-2} + \ldots + x_{t-n}}{n}$$

where n represents the sliding window length. This model is formally known as a Moving Average (MA) model [28]. Lastly, we also compare the performance of our RNN architectures with the performance of the ARIMA linear forecasting model on our datasets. For a detailed discussion on the ARIMA model, refer to [4].

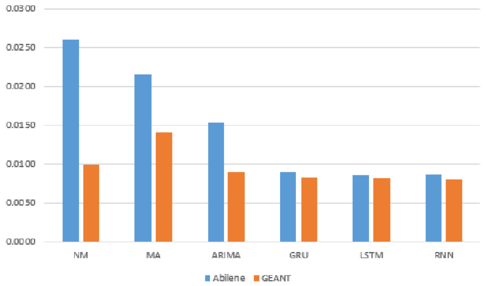

Fig. 4: Comparison of MSE for Abilene and GEANT Datasets

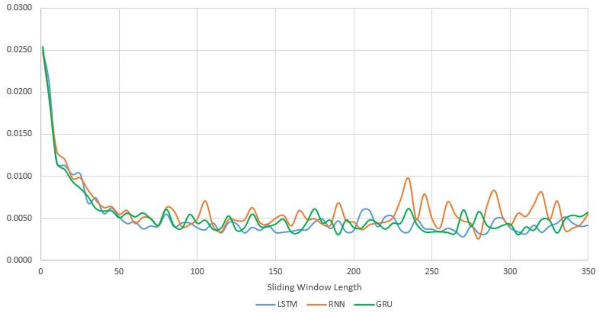

Fig. 5: Effect of Window Length on Packet Volume MSE

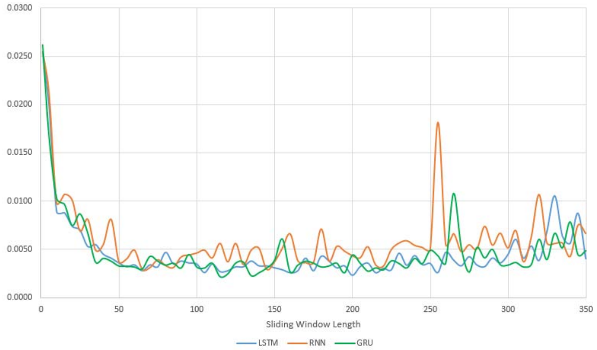

Fig. 6: Effect of Window Length on Byte Volume MSE

V. Results and Discussion

We tested our model on the data that we collected from the internal network over a period of two hours. This data consisted of 25,881 packets and after sampling packets every 5 s, we obtained 1,160 data points from this dataset. 67% of this data was used as training data for the model. We used the remaining data as test data to determine performance of each model. On the Abilene and GEANT dataset, we took a subset of 1000 flow measurements at each node and similarly split this dataset into training and test datasets. For the Abilene dataset, these 1000 data points corresponded to volume measurements from March 1, 2004 at 12:00 A.M to March 4, 2004 at 11:15 A.M. In the GEANT dataset, this subset of data corresponds to traffic matrices from January 1, 2005 at 12:30 A.M to January 11, 2005 at 10:15 A.M.

We present a summary of the results on test datasets for all our experiments in Table I. The first two columns of the internal network section of the table represent the volume prediction results. The MSE Distribution column presents results of protocol distribution prediction and the accuracy column shows results for the protocol classification problem. We found that in all of our experiments, the RNN architectures consistently outperformed other models that we tested. Figure 4 shows the results of our models on predicting the volume of future traffic on our internal network dataset. As mentioned earlier, we found that an incredibly important parameter in the accuracy of our model on the internal network datsets was the sliding window length. This factor in the model intuitively represents the amount of information the model uses to generate its next prediction. As the model gains more information, it generally performs better, up to a certain point where it starts to overfit. Figures 6 and 7 show the effect of this parameter on the MSE on the internal network datasets. To train our models, we used the sliding window length corresponding to the global minimum of these curves. The MSE calculations in all experiments are from each model's optimal sliding model value as determined by this curve. One interesting feature in this curve was the the simple RNN architecture was more prone to overfitting than the LSTM and GRU architecture. Therefore, we believe that LSTM and GRU architectures are more suitable networks for network volume prediction problems. It is worth noting that in these experiments, while sliding window lengths linearly increased training time, they negligibly effected prediction times of models, making these models highly efficient for near real-time prediction. The linear increase in training time is explained by the increase in dimension of the inputs to the recurrent neural network architectures, resulting in more weights the network has learn during backpropagation. In applications of these models, pre-trained models are used to make predictions; therefore, it is reasonable to optimize entirely on appropriate sliding window lengths. Figure 8 demonstrates how the GRU predictions compared with the actual volume of network traffic in the internal network dataset with the optimal sliding window. For figures 8 and 9, we chose the optimal sliding window for each model based on Figure 6.

We performed similar experiments on the Abilene and GEANT network datasets for volume prediction. Figure 5 shows the performance of the models on the Abilene network and GEANT network test data respectively. The MSE measurements in these two figures are over the entire traffic matrix rather than a single connection as in the internal network dataset. Our model performs better than the other models we tested on this dataset. It is worth noting, however, though that the best RNN model on the GEANT dataset only outperformed the ARIMA model by 11% on the GEANT dataset, while it outperformed the ARIMA model by 44% and 78% on the Abilene and internal network datasets. We believe this is due to the larger presence of spikes in the internal network dataset and the Abilene datasets, which the ARIMA models had a harder time predicting. The underlying reason why the RNN architectures are much better at predicting the spikes is the traffic are due to it's ability to approximate nonlinear and complex relationships, relationships that a standard linear forecasting tool such as ARIMA cannot model. The difference between the the volume curve in Figure 8 and 9 aptly demonstrate this characteristic, where Figure 9 depicts the time series prediction ability of our model on the link of the GEANT network with the worst MSE. It is worth noting that our error values on both these public datasets were comparable and in some cases better than all other literature that used these datasets. An important addition to this observation is that we also predicted traffic volume over larger subsets of the traffic matrix data than any other group. For example, in [14], their LSTM MSE values were comparable to our results, but they only used a subset of 309 traffic matrices, while we used 1000 traffic matrices.

For the protocol classification problem, we performed experiments on our internal network dataset since to our knowledge, there are no publicly available datasets with information about protocol data. Figure 10 outlines the results of each model on this problem. Figure 11 shows the results of packet distribution prediction experiments. In both these experiments, RNN architectures significantly outperformed other models. This is once again due to the inherently nonlinear structure of this time-series prediction problem, which RNN's do a much better job of modeling than any of the other models.

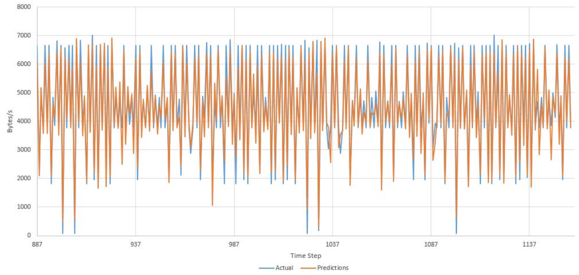

Fig. 7: GRU Predictions on Internal Network

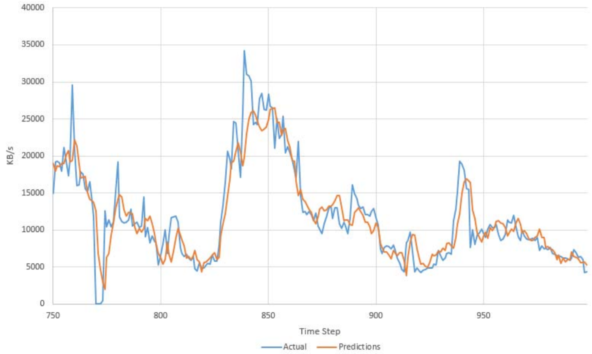

Fig. 8: LSTM Predictions GEANT dataset

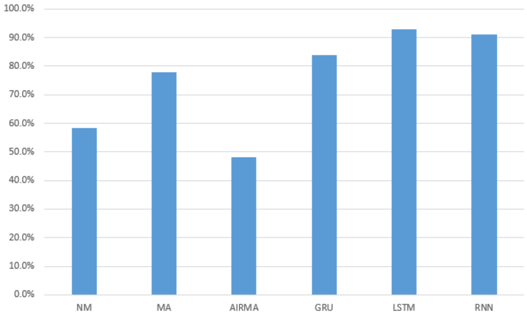

Fig. 9: Comparison of Protocol Classification Accuracy on Internal Network

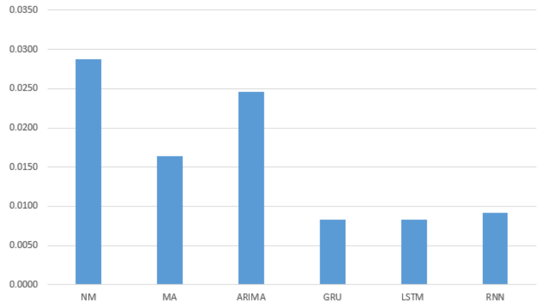

Fig. 10: Comparison of Packet Distribution MSE on Internal Network

VI. Conclusion

In this work, we demonstrate the strong performance of three RNN architectures on several problems in the network traffic prediction domain – volume prediction, protocol classification, and protocol distribution prediction. From our extensive testing on several datasets, we show that the standard RNN, LSTM, and GRU outperform other standard forecasting models, while also predicting future traffic efficiently. We demonstrate the feasibility of RNN models in modeling complex and nonlinear time-series data within the network traffic prediction domain. These predictive models have the potential to help in various applications, including network traffic load management, network monitoring, and cybersecurity.

Acknowledgment

The authors would like to thank the developers of Wireshark and the providers of the network datasets used in this research. We would also like to thank our affiliated organizations for funding this work.

References

[1] The zettabyte era: Trends and analysis, Jun 2017.

[2] Keunsoo Lee, Juhyun Kim, Ki Hoon Kwon, Younggoo Han, and Sehun Kim. Ddos attack detection method using cluster analysis. *Expert Systems with Applications*, 34(3):1659–1665, 2008.

[3] Will E Leland, Murad S Taqqu, Walter Willinger, and Daniel V Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on networking*, 2(1):1–15, 1994.

[4] Billy Williams, Priya Durvasula, and Donald Brown. Urban freeway traffic flow prediction: application of seasonal autoregressive integrated moving average and exponential smoothing models. *Transportation Research Record: Journal of the Transportation Research Board*, (1644):132–141, 1998.

[5] Jake D Brutlag. Aberrant behavior detection in time series for network monitoring. In *LISA*, volume 14, pages 139–146, 2000.

[6] Shiliang Sun, Changshui Zhang, and Guoqiang Yu. A bayesian network approach to traffic flow forecasting. *IEEE Transactions on intelligent transportation systems*, 7(1):124–132, 2006.

[7] Alberto Dainotti, Antonio Pescapé, Pierluigi Salvo Rossi, Francesco Palmieri, and Giorgio Ventre. Internet traffic modeling by means of hidden markov models. *Computer Networks*, 52(14):2645–2662, 2008.

[8] Sharat C. Prasad and Piyush Prasad. Deep recurrent neural networks for time series prediction. *CoRR*, abs/1407.5949, 2014.

[9] Ken-ichi Funahashi and Yuichi Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

[10] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[11] Melinda Barabas, Georgeta Boanea, Andrei B Rus, Virgil Dobrota, and Jordi Domingo-Pascual. Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition. In *Intelligent Computer Communication and Processing (ICCP), 2011 IEEE International Conference on*, pages 95–102. IEEE, 2011.

[12] Samira Chabaa, Abdelouhab Zeroual, and Jilali Antari. Identification and prediction of internet traffic using artificial neural networks. *Journal of Intelligent Learning Systems and Applications*, 2(03):147, 2010.

[13] Dingde Jiang and Guangmin Hu. Large-scale ip traffic matrix estimation based on the recurrent multilayer perceptron network. In *Communications, 2008. ICC'08. IEEE International Conference on*, pages 366–370. IEEE, 2008.

[14] Abdelhadi Azzouni and Guy Pujolle. A long short-term memory recurrent neural network framework for network traffic matrix prediction. *CoRR*, abs/1705.05690, 2017.

[15] Wei Li and Andrew W Moore. A machine learning approach for efficient traffic classification. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International Symposium on*, pages 310–317. IEEE, 2007.

[16] Dario Rossi and Silvio Valenti. Fine-grained traffic classification with netflow data. In *Proceedings of the 6th international wireless communications and mobile computing conference*, pages 479–483. ACM, 2010.

[17] Zhanyi Wang. The applications of deep learning on traffic identification. *BlackHat USA*, 2015.

[18] Robert Hecht-Nielsen et al. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[20] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[21] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[23] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[24] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Syngress, 2006.

[25] Yin Zhang. Abilene dataset. *URL http://www. cs. utexas. edu/yzhang/research/AbileneTM*.

[26] S. Uhlig, B. Quoitin, S. Balon, and J. Lepropre. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM Computer Communication Review*, 36(1), January 2006.

[27] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[28] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012.