Subspace-Based Emulation of the Relationship Between Forecasting Error and Network Performance in Joint Forecasting-Scheduling for the Internet of Things

Mert Nakip
Institute of Theoretical and Applied Informatics
Polish Academy of Sciences (PAN)
Gliwice, Poland
mnakip@iitis.pl

Alperen Helva*, Cüneyt Güzeliş†, Volkan Rodoplu‡
Department of Electrical and Electronics Engineering
Yaşar University
Izmir, Turkey
*alperen.helva@outlook.com,
†{cuneyt.guzelis}, ‡{volkan.rodoplu}@yasar.edu.tr

***Abstract*—We develop a novel methodology that discovers the relationship between the forecasting error and the performance of the application that utilizes the forecasts. In our methodology, an Artificial Neural Network (ANN) learns this relationship while the forecasting error is kept inside a subspace of the entire space of forecasting errors during training. We apply our methodology to the case of Joint Forecasting-Scheduling (JFS) for the Internet of Things (IoT). Our results hold potential to improve the performance of JFS in next-generation networks and can be applied to a much wider range of problems beyond IoT.**

***Index Terms*—Internet of Things (IoT), forecasting, scheduling, Massive Access Problem, Artificial Neural Network (ANN), Machine-to-Machine (M2M) communication**

I. Introduction

The Internet of Things (IoT) is a key technology for the smart cities of the near future [1]. The application areas of IoT widen every day [2][3]. These applications include fleet management, environmental monitoring and control systems, traffic controlling as well as smart buildings [4]. Although IoT is one of the technologies that make the daily life of humans easier, the challenges that must be overcome in order to unleash IoT grow as the number of IoT devices increases. In the near future, it is expected that there will be 30 billion IoT devices on the Internet [5]. In addition, more than half of IoT devices are expected to fall in the Massive IoT segment [6], in which a base station or a gateway will cover a massive number of low-cost IoT devices. The connection requests of the massive number of IoT devices to a single gateway will result in a significant access problem in cellular networks [7][8]. This problem is referred to as the "Massive Access Problem" of IoT.

In order to solve the Massive Access Problem, Reference [9] proposed Joint Forecasting-Scheduling (JFS), which is a machine learning based proactive resource allocation technique. JFS ensures that it allocates the resources in advance based on forecasts of IoT traffic in order to maximize the network throughput. The performance of JFS is evaluated in a number of works [9], [10], [11]. The results of these works show that although JFS performs well in terms of network throughput and energy consumption, its performance depends highly on the performance of the forecasting scheme that it utilizes. Thus, knowledge of the relationship between the forecasting error and network performance would provide crucial information in improving the performance of JFS. One of the major foci of this work is the examination of this relationship.

The focus of the past literature on forecasting has been to minimize the forecasting error, which is captured by any of the forecasting error metrics such as Mean Square Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE) or symmetric Mean Absolute Percentage Error (sMAPE). These traditional forecasting error metrics do not take into account the particular application that will utilize the forecasting metric.[1] In contrast, in this paper, we propose an error metric that is specific to the application that will utilize the results of forecasting. In contrast with the traditional forecasting error metrics in which forecasting is decoupled from the application that utilizes the forecasts, in this paper, we propose an Application-Specific Error Function (ASEF). In the particular example of JFS, which we examine in this paper, the application that utilizes the forecasting results is MAC-layer scheduling of the wireless resources. In this regard, our aim is to discover the relationship between the forecasting error (which is measured at the output of the forecaster) and the network performance (which is measured at the output of the scheduler) for the JFS system. In this context, ASEF measures how the performance of the forecasting algorithm in JFS affects the resulting network performance.

This work has been supported by TUBITAK (Scientific and Technological Research Council of Turkey) under the 1001 program grant no. 118E277.

[1]In many instances, more than one application may utilize the metric. Furthermore, the forecaster might have no knowledge of which application will utilize the forecasts.

In addition to the above conceptual contribution, the major methodological contribution of this paper to the literature is the emulation of the relationship between the forecasting error and the network performance by an Artificial Neural Network (ANN). In our investigations, we have found that it is extremely difficult for an ANN to learn this relationship if the space of forecasting errors is taken to be the original space of all possible forecasting errors produced by the forecaster. One of our key contributions is the idea that if the forecasting errors are constrained to lie on a subspace of the original space of forecasting errors, then it becomes possible for the ANN to learn the relationship between the forecasting error and the network performance. In this work, we design an ANN that learns this relationship successfully on a subspace of the entire space of forecasting errors. Furthermore, we evaluate the performance of this ANN, which we call "Emulator for ASEF", or E-ASEF, for short, via simulations that show that E-ASEF achieves high performance with very fast execution time.

The rest of this paper is organized as follows: In Section II, we present the relationship of this work to the state of the art. In Section III, we state our assumptions. In Section IV, we present ASEF for the JFS system. In Section V, we describe the design of an ANN, called E-ASEF, that emulates ASEF on a subspace. In Section VI, we evaluate the performance of E-ASEF. In Section VII, we present our conclusions.

II. Related Work

We now describe the difference between our work and the current literature. To this end, we categorize the related literature into three categories: (1) The works that aim to improve the performance of JFS. (2) The articles that utilize traditional forecasting error metrics for IoT traffic at layers beyond the MAC layer. (3) The papers that perform subspace learning in machine learning models in contexts outside of IoT.

First, Reference [9] proposed the basic Joint Forecasting-Scheduling system. Reference [10] presented the multi-scale algorithm for JFS, which improved the basic scheme by its ability to extend the window over which accurate forecasts are available. Reference [11] extended the JFS system to the multi-channel case. Even though these works have improved the performance of JFS, they have not investigated the relationship between the forecasting error and the network performance. In contrast, the examination of this relationship is one of the main foci of this work.

Second, Reference [12] forecasts the traffic of individual IoT devices in order to minimize MSE and measures the performance via sMAPE. Reference [13] proposes a neural network model to forecast the traffic volume in an IoT network and measures the performance via various forecasting error metrics (including R^2, MSE, and MAE). In addition, Reference [14] forecasts the type of the IoT traffic based on gradient boosting neural networks and measures the performance via classification metrics (such as accuracy, F1 score, precision). Whereas all of the works that focus on the forecasting of IoT traffic use well-known error metrics to measure the forecasting error, we propose ASEF as a novel error metric that measures the effects of the forecasting error on network performance.

Third, Reference [15] performs subspace learning by using a hebbian/anti-hebbian neural network for which the input data are projected onto the principal subspace. Furthermore, subspace learning has been used for pattern recognition in [16], for spectral regression in [17] and for computer vision via robust Principal Component Analysis (PCA) in [18], [19]. In [20] and [21], the authors perform feature selection and subspace learning jointly. Whereas all of these works project the inputs onto a subspace in order to reduce the input dimension, we train our model by using samples that reside on a *given* subspace such that ASEF becomes learnable.

III. Assumptions

We assume[2] that there is a set of N IoT devices, denoted by $\mathcal{N}$, in the coverage area of Gateway G. In addition, each device i in $\mathcal{N}$ has a direct wireless link to G.

We let m denote the index of a MAC-layer slot (in short, "slot"). Furthermore, we let $x_i[m]$ denote the total number of bits generated by device i in slot m. We assume that the generation of traffic by each IoT device occurs in bursts. Moreover, we assume that every time that an IoT device i transmits to G, it compresses its traffic pattern since its last transmission and sends this along with its actual data. For each IoT device, Gateway G pieces together these traffic patterns and thus has access to the entire past traffic pattern of that device. Gateway G then performs the k-step ahead forecast of the future traffic generation pattern for all values of $k \in \{1, \ldots, K\}$ for each IoT device. Based on these forecasts, G schedules the uplink transmissions from all of these IoT devices in advance.[3]

For a JFS system, the throughput is defined as the ratio of the total number of bits in successfully transmitted bursts to the total number of bits in offered traffic over a scheduling window of duration T_{sch}.[4] While the framework that we develop for ASEF is general and works for any network performance metric, throughput will be the main metric by which we measure network performance in this paper.

IV. Application Specific Error Function (ASEF) for JFS

In this section, we describe the ASEF for the JFS system. After we have defined ASEF, we shall emulate this function via an ANN in Section V.

In Fig. 1, we present the block diagram through which we define ASEF for the JFS system. For the definition of ASEF, we use two JFS systems that run in parallel. The upper

[2]The assumptions of this work are identical to those in [9], which describes the JFS system.

[3]For simplicity, we do not model mobile devices that change coverage areas in this work.

[4]We note that this differs from the traditional definition of throughput. In this definition, the throughput serves as a measure of efficiency of the JFS system; it measures the efficiency with which the system is able to deliver bits successfully to the Gateway at the MAC layer.

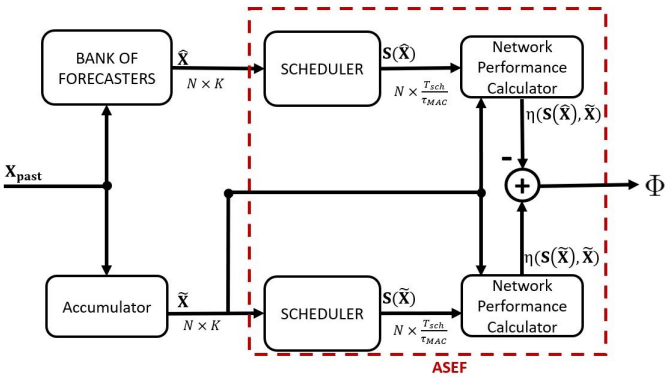

Fig. 1. ASEF is defined as a "black box" by the set of components that appear in the dashed box in this figure. Our goal will be to emulate this ASEF via an ANN.

branch of this figure is a JFS system that consists of a Bank of Forecasters (BoF) and a Scheduler[5]. The lower branch of this figure is a JFS system under the actual traffic generation patterns (a.k.a. "perfect forecasts") of the devices. The ASEF is defined as the function that computes the difference between the performance of the application under forecasts versus the one under perfect forecasts.[6]

In this figure, at each discrete time m, $\mathbf{X_{past}}$ denotes the input matrix of the BoF whose entry (i, s) is $x_i[m-s]$, where s can take values in the range of integers in $[0, m]$. The $\hat{\mathbf{X}}$ is the output matrix of the BoF whose entry (i, k) is the forecast of the number of bits at the kth step in the traffic generation pattern of device i, denoted by $\hat{x}_i[m+k]$, where k can take values in the range of integers $[1, K]$. In addition, the matrix $\tilde{\mathbf{X}}$ denotes the actual past traffic generation pattern that has been accumulated, whose entry (i, k) is $x_i[m+k]$.

Furthermore, the red dashed box in Fig. 1 shows the ASEF for the JFS system, where the inputs of ASEF are $\tilde{\mathbf{X}}$ and $\hat{\mathbf{X}}$. In this figure, ASEF is comprised of three parts that appear in this red dashed box: the upper branch of the figure in the box; the lower branch of the figure in the box; and the difference between the outputs of the Network Performance Calculators that appear in this figure.

First, on the upper branch of ASEF in Fig. 1, based on the forecast future traffic generation matrix $\hat{\mathbf{X}}$, the Scheduler produces a schedule matrix, denoted by $\mathbf{S}(\hat{\mathbf{X}})$, whose entry (i, m) is the binary variable that equals 1 if MAC-slot m has been allocated to device i, and equals 0 otherwise. Then, the Network Performance Calculator in this figure calculates the network performance $\eta(\mathbf{S}(\hat{\mathbf{X}}), \tilde{\mathbf{X}})$ based on $\mathbf{S}(\hat{\mathbf{X}})$ and $\tilde{\mathbf{X}}$. Throughout this paper, the performance metric will be the network throughput (as defined in Section III).

Second, on the lower branch of ASEF in Fig. 1, we compute the network performance of JFS under perfect forecasts, which is denoted by $\eta(\mathbf{S}(\tilde{\mathbf{X}}), \tilde{\mathbf{X}})$. Note that perfect forecasts are available only after the actual traffic has been realized in practice. In our simulations, we use these perfect forecasts in order to measure the difference in the network performance attained under forecasting versus the one under the actual traffic realizations.

Third, after the operations in each of the upper and the lower branches are completed, we calculate the throughput difference, denoted by Φ, which is the output of the ASEF. Thus, the output of ASEF is

$$\Phi = \eta(\mathbf{S}(\tilde{\mathbf{X}}), \tilde{\mathbf{X}}) - \eta(\mathbf{S}(\hat{\mathbf{X}}), \tilde{\mathbf{X}}) \tag{1}$$

V. Emulation of ASEF (E-ASEF) via an ANN

Recall that the main objective in computing the ASEF is to pave the way to algorithms that will significantly improve the JFS system by taking the advantage of the knowledge of Φ. However, since ASEF requires computing the schedule matrices as well as the throughput metrics, the execution time of ASEF in its original form is too high for its use in practical algorithms that can improve the performance of the JFS system. In order to solve this problem, we now aim to emulate ASEF by an ANN. By using an ANN in the place of ASEF, we will generate Φ such that none of the blocks that appear under ASEF, such as the Scheduler or the Network Performance Calculator will be needed. Thus, the ANN that replaces ASEF shall calculate the value of Φ based on the inputs to the ASEF.

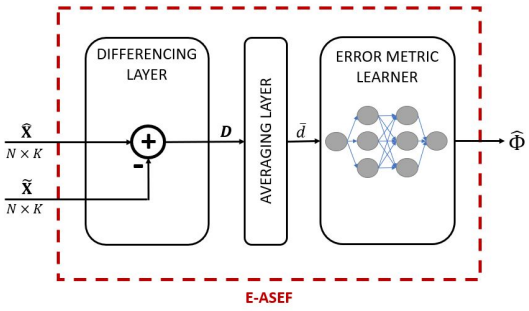

Fig. 2. E-ASEF, which is the ANN-based emulator for ASEF

In Fig. 2, in the design of the emulator, the inputs are the matrices $\tilde{\mathbf{X}}$ and $\hat{\mathbf{X}}$, and the output is the value of Φ that will be estimated by the ANN. We shall denote this estimate by $\hat{\Phi}$. In our design, the E-ASEF consists of three layers: the Differencing Layer, the Averaging Layer and the Error Metric Learner (EML).

At the first layer, the emulator computes $\mathbf{D} = \hat{\mathbf{X}} - \tilde{\mathbf{X}}$ and passes the matrix $\mathbf{D}$ of the forecasting error to the next layer.

[5]For the Scheduler of JFS, we use the Priority based on Average Load (PAL) algorithm, which was developed in [9]. Although any scheduling algorithm can be used in our methodology, in this article, the reason that we use PAL is that the previous studies [9], [10] showed that PAL is a fast heuristic with relatively high performance.

[6]Note that in an actual system, the performance under perfect forecasts can be measured only after the traffic of each IoT device has been realized. The system diagram that appears in Fig. 1 is used for network simulation during which the inputs to the system as well as the performance difference at the output are collected as data that will be used later in order to train the ANN that will emulate the ASEF block shown in this figure.

At the second layer, we take the average over all of the elements of $\mathbf{D}$, which generates $\bar{d}$. The reason that we use averaging is two-fold: (1) During our experimental work, we observed that the value of Φ is highly learnable for a subspace of ASEF on which the forecasting error is kept constant across all of the devices. That is, in this subspace, the value of $\bar{d}$ equals to the constant value of forecasting error; that is $\bar{d} = \mathbf{D}[i,k], \forall i,k$. (2) Instead of using $\tilde{\mathbf{X}}$ and $\hat{\mathbf{X}}$ to form the estimate $\hat{\Phi}$, using $\bar{d}$ significantly decreases the input dimension and hence the computational complexity of the model.

At the third layer, we use ANN as the EML in order to form the estimate $\hat{\Phi}$ based on $\bar{d}$. Since E-ASEF is a model that can be used in other algorithms to improve performance of JFS, the computational complexity of E-ASEF is as important as its learning performance. Thus, we aim to use a relatively simple ANN architecture in the EML layer. To this end, we compare the Linear Perceptron (namely, Adaline), the Nonlinear Perceptron, and the Multi-Layer Perceptron (MLP). For each of these models, we select the architecture and the activation functions as follows: For the Linear Perceptron, note that there is no activation function or any other hyper-parameter. The Nonlinear Perceptron is comprised of a single neuron with an activation function that is selected as the tangent hyperbolic $\tanh$[7]. For the MLP, we use a single hidden layer and an output layer, where the hidden layer contains 5 neurons. The activation function of each neuron in the MLP is selected as $\tanh$.

VI. Results

A. IoT Data

We use the dataset in [22], which is comprised of the traffic generation patterns of 10000 bootstrapped IoT devices. According to the classification of Reference [12], there are four distinct IoT classes: Fixed Bit Periodic (FBP), Variable Bit Periodic (VBP), Fixed Bit Aperiodic (FBA) and Variable Bit Aperiodic (VBA). In this classification, "Fixed Bit" states that the IoT device generates a fixed number of bits in each burst while "Variable Bit" states otherwise. In addition, "Periodic" states that the inter-arrival time between the bursts of a device is constant while "Aperiodic" states otherwise. Since both number of bits and the generation time of each burst of each device in the FBP class are known in advance, forecasting is needed only for the VBP, FBA, and the VBA classes. Thus, in our simulations, we include only these three classes in order to obtain a system in which we evaluate the performance of E-ASEF.

B. Performance Evaluation of E-ASEF

We now evaluate the performance of E-ASEF under each of the Linear Perceptron, Nonlinear Perceptron and the MLP forecasting models on the following four distinct network set-ups: $\mathcal{N}$ is comprised of (1) equal percentages of VBP, FBA and VBA classes; (2) only devices in the VBP class; (3) only devices in the FBA class; (4) only devices in the VBA class.

[7]We selected the activation function as $\tanh$ because the value of Φ is in the range $[-1, 1]$ for our application.

For each network set-up, we aim to evaluate the performance in a subspace in which the forecasting error is constant across all of the forecasters. To this end, we replace each forecaster in the BoF by adding a constant number of bits, which we denote by a, to the actual traffic generation of each of the devices.[8] Thus, whenever $\hat{x}_i[m] = x_i[m] + a$, we say that the forecaster overestimates the value of the traffic generation pattern for device i if $a > 0$, and underestimates it if $a < 0$. (The estimate matches the actual whenever $a = 0$.) In our experiments, we increment a in multiples of 10 from the lowest to the highest value in the range $[-100, 100]$. In addition, we set the number of IoT devices in the coverage area of the Gateway G as $N = 2000$. We set the duration of a MAC-layer slot, denoted by τ_{MAC}, to 0.1 s.

In our experiments, we observed that the behavior of ASEF is distinct for each of the negative and positive values of a. For convenience, we separated the values of a into two categories as follows: We hold the negative values of a in the vector $\mathbf{A}^-$ and the positive values in the vector $\mathbf{A}^+$. Then, we let Φ^- denote the outputs of E-ASEF for the corresponding $\mathbf{A}^-$, and let Φ^+ denote those for $\mathbf{A}^+$. A single E-ASEF structure should not be expected to learn both the Φ^- and Φ^+ functions due to the structural differences that we observed between these functions in our experiments.[9] In this paper, E-ASEF is trained separately for the Φ^- and the Φ^+ functions.

1) Training of E-ASEF: Based on the above observations, we shall now explain how we train E-ASEF. First, for each value of a, we simulate the JFS system for 1000 distinct scheduling windows, each of whose duration equals 15 minutes. For each scheduling window, we randomly select the IoT devices from each device class for the following four distinct experimental set-ups: (1) We select an equal number of devices from each of the VBP, FBA and the VBA classes. (2) All of the devices are in the VBP class. (3) All of the devices are in the FBA class. (4) All of the devices are in the VBA class. Let s be the index of a scheduling window. We shall append a superscript s to each of the variables that appear in Fig. 2 in order to indicate to which scheduling window that variable belongs. Now, for each experimental set-up, we extract the traffic generation patterns $\tilde{\mathbf{X}}^{(s)}$ from the dataset and calculate $\hat{\mathbf{X}}^{(s)}$ for a. Based on $\tilde{\mathbf{X}}^{(s)}$ and $\hat{\mathbf{X}}^{(s)}$, we compute the schedule via the PAL scheduling algorithm and calculate the network performance difference $\Phi^{-(s)}$ if $a < 0$ and calculate $\Phi^{+(s)}$ otherwise.

Then, for each s, the inputs of E-ASEF are $\hat{\mathbf{X}}^{(s)}$ and $\tilde{\mathbf{X}}^{(s)}$. The $\Phi^{-(s)}$ is the output in the case of underestimation, and $\Phi^{+(s)}$ is the output in the case of overestimation. The connection weights and the biases of EML are updated via backpropagation using the gradient descent algorithm in pattern mode.

2) Learning Performance of E-ASEF: Fig. 3 shows how each of the Linear Perceptron, the Nonlinear Perceptron and

[8]The fact that this constant a does not have an i-index indicates that it is constant across all of the devices.

[9]The structural differences between Φ^- and Φ^+ will be shown in Fig. 3-6.

the MLP fits to each function Φ^- and Φ^+. In this figure, we see that MLP fits both functions almost perfectly. In addition, both the Linear Perceptron and the Nonlinear Perceptron can fit only the average of each of the underestimation and overestimation segments of the function. In this figure, the performance of E-ASEF under MLP shows that ASEF for JFS is highly predictable.

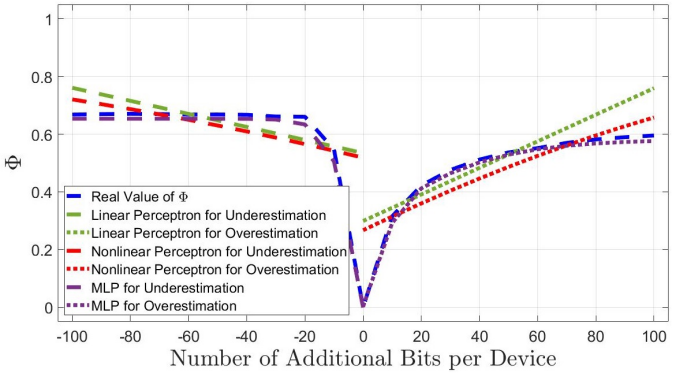

Fig. 3. Learning performance of E-ASEF under each of the Linear Perceptron, Nonlinear Perceptron, and MLP for the throughput difference Φ under the network set-up that has 1/3 VBP, 1/3 FBA and 1/3 VBA as the fractions in each device class.

In Fig. 4, for the simulation that is comprised of IoT devices, each of which falls in the VBP class, we see that the MLP is the best predictor for the majority of the values of a. However, for E-ASEF under MLP, the difference between $\hat{\Phi}$ and Φ slowly increases as the $|a|$ increases. In addition, we see that since the Linear Perceptron fits well to the flat segment of the function which corresponds to Φ^-, it is not able capture the sharp decrease at $a = -10$. On the other hand, the Nonlinear Perceptron is not able to capture Φ^-, but it successfully captures the sharp decrease at $a = -10$.

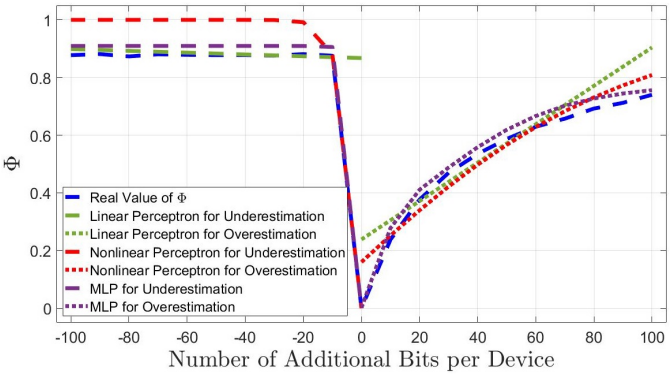

Fig. 4. The learning performance of E-ASEF under each of the Linear Perceptron, Nonlinear Perceptron, and MLP for the throughput difference Φ for a network that is constituted by only VBP devices.

In Fig. 5, for the simulation that is comprised of the IoT devices, each of which falls in the FBA class, we see that MLP predicts Φ almost perfectly and achieves the most accurate result compared with those for the Linear Perceptron and the Nonlinear Perceptron models for the majority of the values of a.

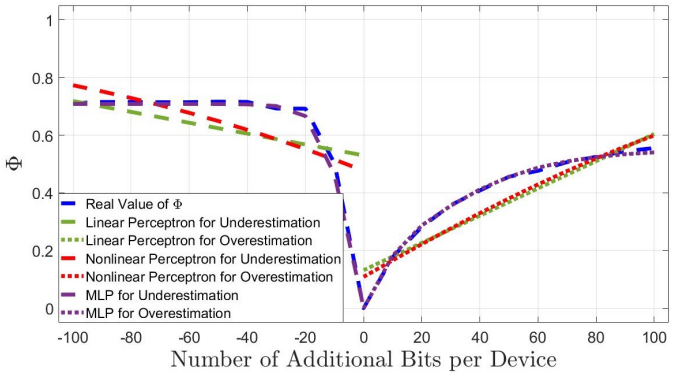

Fig. 5. Learning performance of E-ASEF under each of the Linear Perceptron, Nonlinear Perceptron, and the MLP for the throughput difference Φ under the network set-up that has 100% of the IoT devices in the FBA class.

In Fig. 6, we present the performance of E-ASEF under each of the Linear Perceptron, Nonlinear Perceptron, and the MLP for the simulation that is comprised of the IoT devices each of which falls in VBA class. As seen in this figure, $\hat{\Phi}$ under MLP is almost equal to Φ, where MLP is able to predict the sharp decrease between $a = -10$ and $a = 0$. In addition, while the Linear Perceptron fits to the mean of each of Φ^- and Φ^+ and the Nonlinear Perceptron also fits close to the mean of each of those, the MLP is the best-performing model as it is for all of the other simulation set-ups.

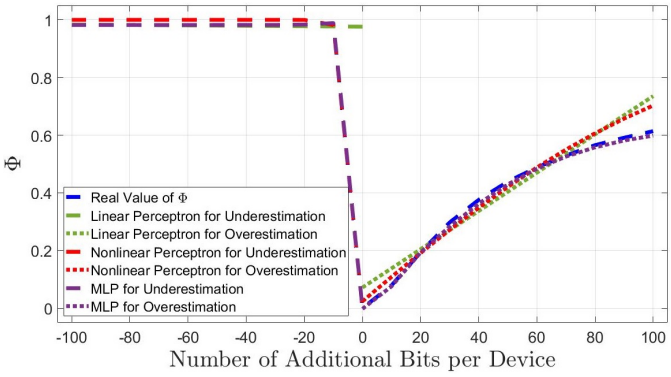

Fig. 6. Learning performance of E-ASEF under each of the Linear Perceptron, Nonlinear Perceptron, and MLP for the throughput difference Φ under the network set-up that has devices only in the VBA class.

C. Execution Time of E-ASEF

In Table I, we present the execution time of E-ASEF. We calculate the mean and the standard deviation of the execution time over 100 simulation runs on the Google Colab platform with no accelarator. In this table, we see that the execution time of E-ASEF is under 20 μs for each of the Linear Perceptron, Nonlinear Perceptron and the MLP models. This shows that E-ASEF is a practical emulation methodology in this setting. In addition, we see that the execution time difference between

E-ASEF under MLP and that under the other models is less than 1 μs. Thus, MLP is the best selection among the models that we have examined for the EML block in E-ASEF, since it achieves the best performance at an execution time that is comparable to those of the other models.

TABLE I
EXECUTION TIME OF E-ASEF [μS]

EML Model in E-ASEF	Mean	Standard Deviation
Linear Perceptron	18.68	0.63
Nonlinear Perceptron	19.16	0.85
MLP	19.42	0.22

VII. CONCLUSION

In this paper, we have developed a novel methodology in order to investigate the relationship between the forecasting error and network performance in Joint Forecasting-Scheduling (JFS). Our methodology has been to build an Artificial Neural Network (ANN) whose input is the forecasting error and whose output is the difference between the network performance obtained under forecasting versus that obtained under the actual network traffic realization. The key novel aspect of our work is the idea of a subspace-based emulation where only a subspace of the entire space of forecasting errors is employed. We have demonstrated that a Multi-Layer Perceptron (MLP) can successfully learn the relationship between the forecasting error and the network performance difference on this subspace.

In our future work, we plan to apply the techniques developed in this paper to the task of improving the performance of JFS, which will enable an IoT Gateway to accommodate a massive number of IoT devices. Furthermore, the subspace-based emulation technique developed in this paper can potentially be applied in diverse contexts beyond IoT.

REFERENCES

[1] M. Hasan, E. Hossain, and D. Niyato, "Random access for machine-to-machine communication in LTE-advanced networks: issues and approaches," *IEEE communications Magazine*, vol. 51, no. 6, pp. 86–93, 2013.

[2] O. Bello and S. Zeadally, "Toward efficient smartification of the Internet of Things (IoT) services," *Future Generation Computer Systems*, vol. 92, pp. 663–673, 2019.

[3] G. Fortino, W. Russo, C. Savaglio, M. Viroli, and M. Zhou, "Modeling opportunistic IoT services in open IoT ecosystems," in *WOA*, 2017, pp. 90–95.

[4] C. Kuhlins, B. Rathonyi, A. Zaidi, and M. Hogan, "White paper: Cellular networks for massive IoT," Jan. 2020. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/white-papers/cellular-networks-for-massive-iot–enabling-low-power-wide-area-applications

[5] Cisco, *Cisco Annual Internet Report (2018–2023)*, Mar. 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[6] Ericsson, *Ericsson Mobility Report*, Nov. 2019. [Online]. Available: https://www.ericsson.com/en/mobility-report

[7] F. Ghavimi and H.-H. Chen, "M2M communications in 3GPP LTE/LTE-A networks: Architectures, service requirements, challenges, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 525–549, 2015.

[8] A. Zanella, M. Zorzi, A. F. dos Santos, P. Popovski, N. Pratas, C. Stefanovic, A. Dekorsy, C. Bockelmann, B. Busropan, and T. A. Norp, "M2M massive wireless access: Challenges, research issues, and ways forward," in *2013 IEEE Globecom Workshops*. IEEE, 2013, pp. 151–156.

[9] M. Nakip, V. Rodoplu, C. Güzeliş, and D. T. Eliiyi, "Joint forecasting-scheduling for the Internet of Things," in *2019 IEEE Global Conference on Internet of Things (GCIoT)*. IEEE, 2019, pp. 1–7.

[10] V. Rodoplu, M. Nakıp, D. T. Eliiyi, and C. Güzelis, "A multi-scale algorithm for joint forecasting-scheduling to solve the massive access problem of IoT," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8572–8589, 2020.

[11] V. Rodoplu, M. Nakip, R. Qorbanian, and D. T. Eliiyi, "Multi-channel joint forecasting-scheduling for the Internet of Things," *IEEE Access*, vol. 8, pp. 217 324–217 354, 2020.

[12] M. Nakip, B. C. Gül, V. Rodoplu, and C. Güzeliş, "Comparative study of forecasting schemes for IoT device traffic in machine-to-machine communication," in *Proceedings of the 2019 4th International Conference on Cloud Computing and Internet of Things*, 2019, pp. 102–109.

[13] M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas, "Neural network architecture based on gradient boosting for IoT traffic prediction," *Future Generation Computer Systems*, vol. 100, pp. 656–673, 2019.

[14] ——, "IoT type-of-traffic forecasting method based on gradient boosting neural networks," *Future Generation Computer Systems*, vol. 105, pp. 331–345, 2020.

[15] C. Pehlevan, T. Hu, and D. B. Chklovskii, "A hebbian/anti-hebbian neural network for linear subspace learning: A derivation from multidimensional scaling of streaming data," *Neural computation*, vol. 27, no. 7, pp. 1461–1495, 2015.

[16] X. Jiang, "Linear subspace learning-based dimensionality reduction," *IEEE Signal Processing Magazine*, vol. 28, no. 2, pp. 16–26, 2011.

[17] D. Cai, X. He, and J. Han, "Spectral regression for efficient regularized subspace learning," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8.

[18] Y. Li, "On incremental and robust subspace learning," *Pattern recognition*, vol. 37, no. 7, pp. 1509–1518, 2004.

[19] F. De La Torre and M. J. Black, "A framework for robust subspace learning," *International Journal of Computer Vision*, vol. 54, no. 1, pp. 117–142, 2003.

[20] Q. Gu, Z. Li, J. Han *et al.*, "Joint feature selection and subspace learning," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1. Citeseer, 2011, p. 1294.

[21] K. Wang, R. He, L. Wang, W. Wang, and T. Tan, "Joint feature selection and subspace learning for cross-modal retrieval," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 2010–2023, 2015.

[22] "IoT Traffic Generation Pattern Dataset," Jan 2021. [Online]. Available: https://www.kaggle.com/tubitak1001118e277/iot-traffic-generation-patterns