Forecasting Operation Metrics for Virtualized Network Functions

Tommaso Cucinotta*, Giacomo Lanciano†*, Antonio Ritacco*, Fabio Brau*, Filippo Galli†*, Vincenzo Iannino*, Marco Vannucci*, Antonino Artale‡, Joao Barata§ and Enrica Sposato‡
**Scuola Superiore Sant'Anna*, Pisa, Italy
†*Scuola Normale Superiore*, Pisa, Italy
‡*Vodafone*, Milan, Italy
§*Vodafone*, Lisbon, Portugal
*firstname.lastname@santannapisa.it †firstname.lastname@sns.it ‡§firstname.lastname@vodafone.com

***Abstract*—Network Function Virtualization (NFV) is the key technology that allows modern network operators to provide flexible and efficient services, by leveraging on general-purpose private cloud infrastructures. In this work, we investigate the performance of a number of metric forecasting techniques based on machine learning and artificial intelligence, and provide insights on how they can support the decisions of NFV operation teams. Our analysis focuses on both infrastructure-level and service-level metrics. The former can be fetched directly from the monitoring system of an NFV infrastructure, whereas the latter are typically provided by the monitoring components of the individual virtualized network functions. Our selected forecasting techniques are experimentally evaluated using real-life data, exported from a production environment deployed within some Vodafone NFV data centers. The results show what the compared techniques can achieve in terms of the forecasting accuracy and computational cost required to train them on production data.**

***Index Terms*—Forecasting, NFV, Operations, Time-series, LSTM**

I. Introduction

In recent years, the landscape of information and communication technologies has been facing an unprecedented turn into distributed computing. The wide-spread availability of high transmission bandwidths—at affordable rates—enabled multiple scenarios where computing can effectively and efficiently be distributed. Coupled with the relentless development of virtualization technologies, this led to the realization of nowadays cloud computing. Cloud technologies enabled the flexible management of pools of shared general-purpose processing, storage and communication resources. Consumers can remotely access them in an on-demand, rapid, completely automated and dynamically adaptable fashion.

At the same time, communication technologies have been evolving towards more sophisticated services, higher capacity and lower latency in both landline and mobile access networks, as well as in backbone transport segments. The new distributed computing scenarios, including big-data processing in the cloud and on-the-fly processing at the edge, exposed the incapability of traditional networks at managing the complex and fast-evolving requirements of new and emerging services.

Thanks to the increasing convergence of networking technologies towards IP-based networks (e.g., LTE), telecommunications are benefiting of well-established principles coming from the cloud computing space. This led to the recent paradigm of *Network Function Virtualization (NFV)* [1], where general-purpose private cloud infrastructures allows for quickly provisioning virtualized resources (i.e., *network slices*) in which to instantiate flexible *virtualized network functions* (VNFs), seconding the instantaneous workload conditions and their requirements. This way, a more *intelligent* use of physical resources can be pursued [2].

At the heart of automated management and orchestration operations (including capacity planning, performance monitoring and management), there are *monitoring systems* continuously gathering a plethora of metrics, for each individual element of the infrastructure. For instance, tens of system-level metrics may be gathered from each individual physical host, virtual machine or networking element, with a typical time granularity of one sample every few minutes. At the same time, individual VNFs continuously collect metrics related to the status, performance and failures happening at the application level. This data is normally aggregated and analyzed in real-time by an alerting system that, under precise threshold-based conditions on said metrics, is able to trigger operators' attention (e.g., when failure rates or response latencies exceed certain thresholds), or even activate automated fault management and recovery actions (e.g., exclude a physical host from the fleet and send it to data-center operators for repair and maintenance).

Accurately forecasting how key metrics will evolve over time is an increasingly important problem, both for short-term and mid/long-term forecasts. For this reason, Machine Learning (ML) techniques have been gaining momentum as key technologies accompanying enterprises operating in pretty much any business domain. In networking, these techniques have been successfully applied in NFV infrastructures for anomaly detection [3], [4], behavioral pattern analysis [5], [6] as well as resource demand estimations [7], [8]. In particular, Deep Learning (DL) methods are among the techniques that are receiving increasing attention from both research and industry. Therefore, it is worth investigating the applicability of DL models and the trade-offs that can be achieved in terms of precision and training cost of the available techniques. In this paper, we contribute with a useful investigation along this

line, comparing various methods based on DL and advanced statistics in forecasting NFV infrastructure metrics.

II. Related Work

Adoption of predictive techniques in NFV and Software Defined Networks (SDN) schemes is a long-standing approach for adapting available virtualized resources to varying loads [3], [4], [9]. In this way, *service-chains* deployed on cloud infrastructures are able to offer quick proactive measures to ensure quality of service (QoS) and reduction of CAPEX and OPEX costs. However, achieving such a goal comes with several challenges like, for instance: (i) employing effective predictive models that do not lead to under- or over-provisioning virtual resources; (ii) assessing which components of a service-chain should be scaled to maximize the overall efficiency; (iii) optimizing the placement of newly created virtual resources on the physical infrastructure [10], [11]. Our work focuses on (i), in particular, by investigating effective time-series forecasting techniques that can provide NFV operation teams with actionable feedbacks to support their decisions (e.g., whether a VNF needs to be scaled to accommodate traffic growth).

A standard solution for real-time forecasting and scaling of resources is *static thresholding* [12]. Despite being a straightforward heuristic, it can provide interesting results when dealing with simple systems. However, in general, different services require different threshold policies, thus a generic approach will lead to over- or under-sizing the infrastructure. On the other hand, *dynamic thresholding* provides an adaptive mechanism to set thresholds, that can be implemented with, e.g., Reinforcement Learning (RL) [13], [14]. However, the burdens related to using RL algorithms often limit their applicability to real infrastructures (e.g., the need for—sophisticated—simulation environments to train the agents). For instance, in [15], the authors do train a Q-learning-based algorithm on a real telco system, but the resulting agent is observed to take several unexpected decisions before converging to the optimal policy, suited for the dynamics of the system. A rather successful RL-based approach to VNF service-chains deployment is presented in [16], where the authors propose *NFVdeep* that jointly minimizes operation costs and maximizes requests throughput, also taking into account different QoS requirements.

When it comes to proactive approaches, that leverage on forecasting techniques to estimate the system dynamics, [17] describes the usage of ML algorithms *ensembles* for auto-scaling of NVI architectures based on VNF features. In [18], instead, a more modern approach, based on Long Short-Term Memory (LSTM) networks, is adopted to forecast VNF requirements. However, it is not clear whether there is an actual improvement, in terms of forecasting accuracy, when compared to other methods, as the authors mainly focus on features selection aspects.

Nowadays, LSTM has been proven to be an extremely effective tool for time-series analysis, both for classification [19], [20] and forecasting [21], [22] tasks. In particular, the *sequence-to-sequence* architectural pattern, when implemented with LSTM-based encoder and decoder components [23], yields surprisingly good results. This kind of architectures are also widely adopted for machine translation and Natural Language Processing (NLP) tasks in general. In NFV context, they can be helpful when translating VNF metrics sequences to infrastructure metrics sequences and vice-versa. Since communication from the encoder to the decoder is limited to the hidden state values, there are no real requirements on the structure and architecture of both, thus allowing even for different types of input and output metrics.

As described in [3], [4], forecasting accuracy can also be improved by leveraging on information about the topology of the deployed VNFs like, e.g., graphs characterizing interactions among the VMs belonging to the same VNFs. In [7], such *topology-aware* time-series forecasts were achieved through Graph Neural Networks (GNNs) [24], [25].

III. Background Concepts

In this section, we recall some fundamental time-series forecasting algorithms used for this work. We included standard system identification models from statistical sciences and signal processing, e.g., SARIMA and Holt-Winters (HW), as well as more advanced ML methods, e.g., Non-linear Auto-Regressive (NAR) neural networks and Long Short-Term Memory (LSTM).

In what follows, $\{x_t\}_{t\in\mathbb{Z}}$ with $x_t \in \mathbb{R}^n$ denotes a generic discrete—possibly multi-variate—time-series whose historical evolution is known up to the current time. Its future evolution $\{\hat{x}_t\}$ is to be predicted with one of the mentioned techniques.

A. SARIMA

The Auto-Regressive Moving Average (ARMA) model provides a flexible tool for forecasting. Given the samples $\{x_t\}$ up to time $t-1$, the forecasted sample $\hat{x}_t$ is computed as:

$$\hat{x}_t = \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{1}$$

where: ϕ_0 is a constant; $\{\phi_i\}_{i=1}^{p}$ are the model parameters controlling the linear dependency of $\hat{x}_t$ from the p last samples of the signal; $\{\theta_j\}_{j=1}^{q}$ are the parameters defining the linear dependency of the output from the q errors $\{\epsilon_{t-j} \triangleq \hat{x}_{t-j} - x_{t-j}\}_{j=1}^{q}$ performed by the model on the last q predictions.

From ARMA, it is possible to derive another technique, known as ARIMA, that can be used for non-stationary time-series. ARIMA is actually an ARMA model applied to the $d-$order differenced signal (defined as $x_t^{(1)} \triangleq x_t - x_{t-1}$, for $d=1$, and $x_t^{(d)} \triangleq x_t^{(d-1)} - x_{t-1}^{(d-1)}$, for $d>1$), for some $d \in \mathbb{N}^+$, to obtain $d-$order differenced forecasts $x_t^{(d)}$, that need to be integrated to reconstruct the final forecast $\hat{x}_t$:

$$\hat{x}_t^{(d)} = \phi_0 + \sum_{i=1}^{p} \phi_i x_{t-i}^{(d)} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}^{(d)} \tag{2}$$

Equation (2) is referred to as an ARIMA(p, d, q) model. From such a general definition, it is possible to derive simpler models by tuning the meta-parameters (p, d, q). For instance:

- When $d = q = 0$, an Auto-Regressive model AR$(p) \equiv$ ARIMA$(p, 0, 0)$ is obtained, where x_t is a linear combination of its lagged values up to $t - p$.
- When $p = d = 0$, a Moving Average model MA$(q) \equiv$ ARIMA$(0, 0, q)$ is obtained, where x_t is a linear combination of the errors at previous timestamps up to $t - q$, not to be confused with moving average filtering.

The meta-parameter d is typically chosen to obtain a $d-$order differenced time-series $x_t^{(d)}$ that is *stationary*, i.e., whose mean, variance and auto-correlation are independent of t.

ARIMA can be further extended to deal with seasonal patterns by introducing additional terms in Equation (2). Such a variant is known as *Seasonal ARIMA (SARIMA)*. Given a seasonality period of m samples and meta-parameters (P, D, Q), that are seasonal equivalents of (p, d, q), we get an additional component that corresponds to an ARIMA where the signal values (and errors) at $t - m, t - 2m, t - 3m, \ldots$ are used to compute the output at t. The parameters of the model are usually optimized via least-square optimization or likelihood maximization with Kalman filters [26].

B. Holt-Winters

Holt-Winters (HW)—often referred to as *triple exponential smoothing* [27]—is a commonly adopted method for forecasting and signal processing. Its peculiarity consists in explicitly separating predictive components into *level* (or expected value, l_t), *trend* (b_t) and *seasonality* (s_t). In its *additive* form, given knowledge of the samples $\{x_t\}$ up to the current time t, HW forecasts future samples $\hat{x}_{t+h}$ for $h \in \mathbb{N}^+$ as

$$\begin{aligned}
\hat{x}_{t+h} &= l_t + (\phi + \phi^2 + \cdots + \phi^h) b_t + s_{t+h-m} \\
l_t &= \alpha(x_t - s_{t-m}) + (1 - \alpha)(l_{t-1} + \phi b_{t-1}) \\
b_t &= \beta(l_t - l_{t-1}) + (1 - \beta)\phi b_{t-1} \\
s_t &= \gamma(x_t - l_{t-1} - \phi b_{t-1}) + (1 - \gamma) s_{t-m}
\end{aligned} \tag{3}$$

where: m is the seasonality period; $0 < \alpha, \beta, \gamma < 1$ are the smoothing factors for the level, trend and seasonality forecasts; $0 < \phi \leq 1$ is the damping factor. Alternatively, HW can be also formalized in a *multiplicative* form as

$$\begin{aligned}
\hat{x}_{t+h} &= [l_t + (\phi + \phi^2 + \cdots + \phi^h) b_t] \cdot s_{t+h-m} \\
l_t &= \alpha \frac{x_t}{s_{t-m}} + (1 - \alpha)(l_{t-1} + \phi b_{t-1}) \\
b_t &= \beta(l_t - l_{t-1}) + (1 - \beta)\phi b_{t-1} \\
s_t &= \gamma \frac{x_t}{l_{t-1} + \phi b_{t-1}} + (1 - \gamma) s_{t-m}
\end{aligned} \tag{4}$$

HW parameters are optimized via *Sequential Least Squares Programming*, a classical optimization technique for constrained minimization problems.

C. Non-linear Auto-Regressive Neural Networks

A NAR neural network is an auto-regressive model where the forecast $\hat{x}_t$ at time t is a non-linear combination of the last p observations of the input signal. In the case of the so-called *tunable-basis*, the estimated sample is obtained as a linear combination of r non-linear functions, where each function f_i processes the same input using a set of q_i specialized parameters $\beta_{i,1}, \ldots, \beta_{i,q_i} \in \mathbb{R}$:

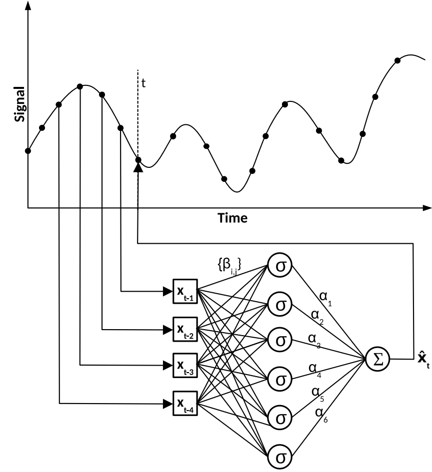


Fig. 1: NAR neural network for time-series forecasting.

$$\hat{x}_t = \sum_{i=1}^{r} \alpha_i f_i(x_{t-1}, \ldots, x_{t-p}, \beta_{i,1}, \ldots, \beta_{i,q_i}) \tag{5}$$

The one-hidden layer sigmoidal neural network is a notable instance of this type of functions. In this case, the parameters are optimized via back-propagation [28], [29], using input-target pairs consisting of past and current observations. Figure 1 represents a NAR forecasting model with a single hidden layer in the case of $p = 4$ and $r = 6$.

D. Long Short-Term Memory

As to more advanced ML techniques, LSTM networks [30] represent a common architecture for time-series classification and forecasting [23], [31]. They consist of an improved version of Recurrent Neural Networks (RNNs), NNs with feedback connections used to—theoretically—predict arbitrarily long sequences. If $\{x_t\}$, for $x_t \in \mathbb{R}^n$ and $t \in [0, \tau]$, is a discrete

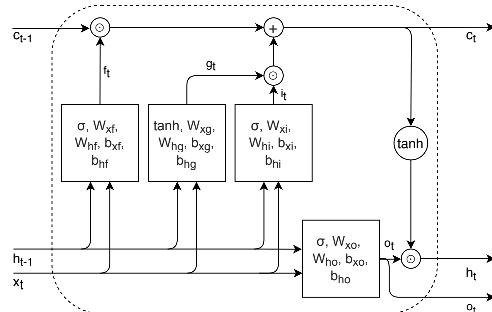


Fig. 2: LSTM cell processing a time-series together with the additional internal signals c_t and h_t.

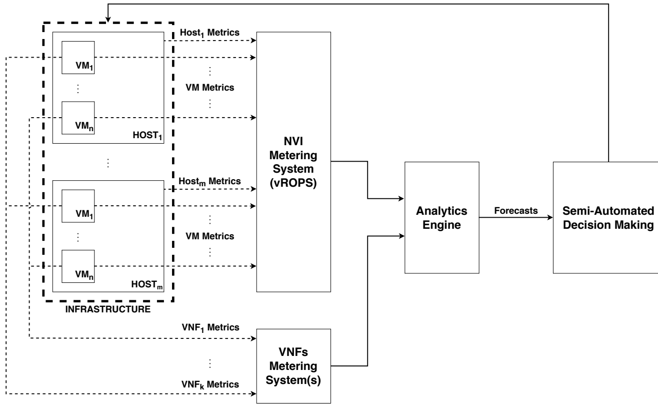


Fig. 3: Integration of the realized Analytics Engine within Vodafone NFV infrastructure operations.

multi-variate input sequence, then we can formalize how an LSTM cell with d units processes its inputs at time t as

$$\begin{aligned} i_t &= \sigma\left(W_{xi}x_t + b_{xi} + W_{hi}h_{t-1} + b_{hi}\right) \\ f_t &= \sigma\left(W_{xf}x_t + b_{xf} + W_{hf}h_{t-1} + b_{hf}\right) \\ g_t &= \tanh\left(W_{xg}x_t + b_{xg} + W_{hg}h_{t-1} + b_{hg}\right) \\ o_t &= \sigma\left(W_{xo}x_t + b_{xo} + W_{ho}h_{t-1} + b_{ho}\right) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \tanh\left(c_t\right) \end{aligned} \tag{6}$$

where each W is a (learnable) weight matrix, $\odot$ is the element-wise (or Hadamard) product and $\sigma(y) = 1/(1 + e^{-y})$ is the (element-wise) sigmoidal function. We refer to $\tilde{h}_t = (h_t, c_t) \in \mathbb{R}^{2d}$ as the *latent state* at time t.

Figure 2 visualizes how the *gates* that compose an LSTM cell are connected with each other. In practical terms, LSTM leverages on the interactions between the *cell state* c_t, the *hidden state* h_t and the current value of the signal x_t to formulate an internal representation of the input variables, that captures long- and short-term dependencies among them. The main reason why LSTMs show an advantage when compared to standard RNNs is that the cell state can traverse the cell freely without being altered—other than linear interactions as summation and element-wise product—in backward and forward passes of the learning algorithm, thus reducing the risk of computational problems affecting regular RNNs, such as the *vanishing gradient* [32]. The exact way $\hat{x}_t$ is computed depends on the architecture where the LSTM cell is plugged into. Section IV reports some practical examples.

IV. Compared Approaches

NVI metric forecasting, both at INFRA and VNF level, aims at providing a snapshot of the system dynamics in the future. This information is useful to operation teams to support their decisions, e.g., for capacity planning purposes. Figure 3 shows how forecasting capabilities enhance Vodafone NFV operations. The NVI time-series produced by each component of the infrastructure are ingested by the VMWare vRealize Operations Manager (vROPS) and the monitoring subsystems of the individual VNFs. The Analytics Engine sits in front of these monitoring components and computes NVI forecasts. Such outputs are used in a semi-automated decision-making process, where humans consume them to get insights and possibly uncover early symptoms of system outages. The acquired knowledge provides decision-makers with actionable feedbacks that may trigger capacity planning and infrastructure management actions.

The data provided by Vodafone consists of a set of relevant indicators (metrics), each one coming in the form of a time-series that describes the evolution in time of a specific NVI metric. Dealing with time-series—especially in multi-variate settings—is an inherently complex problem. An effective model must take into account the dynamic and sequential nature of the information. Furthermore, possible high variance in the data poses an additional challenge. To mitigate the impact of such quality issues, we applied a pre-processing pipeline that includes scaling and normalizing the data using a *min-max* strategy.

We tested three classes of metric forecasting methods: (i) SARIMA, (ii) Holt-Winters and (iii) neural architectures. In order to train the latter on NVI metrics, we reshaped the data-set in the form of training samples. Namely, we built a set of pairs of input and output sub-sequences, fixing the length of input and output time periods upfront.

Note that supervised models rely on the data seen during training, to discover patterns and correlations among the defined variables. Although several techniques have been applied to reduce the generalization error, such models perform the best when test data and training data distributions are similar.

A. Neural Architectures

From a general standpoint, we aim at forecasting the future dynamics of a discrete uni-variate time-series $\{x_t\}$, for $x_t \in \mathbb{R}$, leveraging on the knowledge of its values for $t \in [0, T_{\texttt{train}}]$. In other words, we want to compute $\{\hat{x}_t\}$ for $t \in [T_{\texttt{train}} + 1, T_{\texttt{test}}]$. If available, we can exploit additional *auxiliary* metrics $\{y_t\}$, for $y_t \in \mathbb{R}^m$ whose historical dynamics is known in the same time range as the prediction target $t \in [0, T_{\texttt{train}}]$. In the context of NFV operations, for instance, $\{x_t\}$ could represent the average cpu utilization of a given subset of hosts in the infrastructure, while $\{y_t\}$ could represent a set of service-level indicators that are particularly relevant for the VNF deployed on such hosts.

All the models described in this section are examples of *encoder-decoder* architectures, a common architectural pattern for sequential data processing. An *encoder-decoder* architecture is composed of two distinct layers connected in series: (i) the *encoder* $\mathcal{E}_\theta$, whose job is to accept a—possibly variable-length—sequence in input and compute a *state* with fixed shape, and (ii) the *decoder* $\mathcal{D}_\theta$, that maps the state to an output sequence.

The parameters of the neural models can be trained by comparing their outputs with the ground truth values, provided by the training data-set, and minimizing the *Mean Squared Error (MSE)*. To this aim, in our implementation we used the

Adam optimization algorithm, setting its hyper-parameters as suggested in the seminal paper [33].

1) Baseline: Our baseline architecture is rather simple and consists of an LSTM layer followed by a fully-connected layer, jointly trained by back-propagation. An input time-series is fed to the LSTM that is set to output only the last value of its recurrent process, for each internal unit. The resulting vector is then provided to the fully-connected layer that outputs an estimation of the target time-series in a *one-shot* fashion (i.e., the entire output is generated by a single inference pass). More formally, if φ is the desired length of the output sequence, d the number of neurons of the LSTM, m the number of auxiliary variables, the model is characterized by:

$$\begin{aligned} &\mathcal{E}_\theta : \mathbb{R}^{m+1} \times \mathbb{R}^{2d} \to \mathbb{R}^{d} \times \mathbb{R}^{2d} \\ &\mathcal{D}_\theta : \mathbb{R}^{d} \to \mathbb{R}^{1\times\varphi} \end{aligned} \tag{7}$$

Note that $\mathcal{E}_\theta$ is designed to process each element of the input sequence separately, not the whole sequence at once. Let $(x_{t-\lambda+1}, \ldots, x_t)$ be the input sequence and $(y_{t-\lambda+1}, \ldots, y_t)$ be the auxiliary sequence, both with length equal to λ. The output $\hat{x} = (\hat{x}_{t+1}, \ldots, \hat{x}_{t+\varphi})$ is generated as follows:

$$\begin{aligned} &\tilde{h}_{t-\lambda} = 0, && \tilde{h}_{t-\lambda} \in \mathbb{R}^{2d} \\ &z_s = \mathrm{vec}\,(x_s, y_s) && t-\lambda < s < t \\ &(o_s, \tilde{h}_s) = \mathcal{E}_\theta(z_s, \tilde{h}_{s-1}), && t-\lambda < s \leq t \\ &\hat{x} = \mathcal{D}_\theta\,(o_t) && \end{aligned} \tag{8}$$

where $\tilde{h}_s$ are the latent states.

Due to the way the fully-connected layer is configured, this architecture requires the output length to be fixed upfront. Such requirement entails that if we need to increase the forecasting horizon, then we have to re-train a different model from scratch. These settings do not allow for fully leveraging on the recurrent nature of LSTMs that, in general, can be trained and perform inference on sequences of variable length. On top of that, the complexity of the model—in terms of learnable parameters—grows proportionally to the output length. This aspect is crucial not only from a scalability perspective but also for the quality of the inference. The more the parameters, the higher the risk of over-fitting the training data-set and performing poorly on unforeseen inputs. For these reasons, we decided to build upon this first attempt and devise other architectures that overcome in part the aforementioned limitations.

2) Sequence-to-dense and Dense-to-dense: The model described in Section IV-A1 imposes the limitation of deciding the length of the output sequence (i.e., the number of time steps to forecast in a test scenario) upfront. To address such issue, we have introduced a change to the inference process that consists of using—smaller—forecasted sequences as inputs to the model. This way, it is possible to forecast an unlimited number of time-steps while using a model with a fixed output size. However, due to its *closed-loop* nature, the revised inference process has an important drawback: it is possible to use only the historical values of the target variable to estimate its evolution. In other words, the architectures leveraging on this method can only be used in uni-variate mode (i.e., without auxiliary variables).

In particular, we devised two architectures that employ this method at their core. The first one—referred to as *sequence-to-dense* (seq2den)—has a structure similar to the baseline described in Section IV-A1. It consists of an LSTM layer followed by a fully-connected layer. Since the LSTM layer weights are shared between time-steps, the input length does not affect the number of model parameters. The second one—referred to as *dense-to-dense* (den2den)—consists of two fully-connected layers in series. When feeding the input time-series to the input layer, each input time-step has an associated weight that is independently optimized during back-propagation. In this case, the length of the input sequence contributes to the growth of the learnable parameters. Moreover, due to the output layer being fully-connected, for both these variants the number of learnable parameters is proportional to the output sequence length.

3) Sequence-to-sequence with Time Embedding: The models described in Section IV-A2 improve our baseline by introducing a *closed-loop* inference strategy. However, they are not able to work in multivariate settings and, thus, we cannot leverage on contextual information to get more accurate forecasts. With the aim of taking the best from both worlds, we developed an additional model—referred to as *sequence-to-sequence* (seq2seq)—that is able to accept auxiliary variables while not being constrained in the length of the output. Additionally, as the time-series under consideration turn out to be greatly affected by timing information such as the hour of the day and the week day, including non-periodic changes occurring on holidays, we added to this model the capability to use additional *time embedding* metrics, as described below.

a) Temporal Embedding: Each sample of both input and target signals, at each time-step t, is associated with a unique date and time. This information can be encoded using additional time-series: (i) $\{\alpha_t\}$, for $\alpha_t \in \{0, 1, \ldots, 23\}$, encoding the hours; (ii) $\{\beta_t\}$, for $\beta_t \in \{0, 1, \ldots, 6\}$, encoding the week-days; (iii) $\{\gamma_t\}$, for $\gamma_t \in \{0, 1, \ldots, 11\}$, encoding the months; (iv) $\{\delta_t\}$, for $\delta_t \in \{0, 1\}$, encoding whether the time-steps correspond to holidays. Note that such time-series are completely known a-priori. However, $\{\alpha_t\}$, $\{\beta_t\}$ and $\{\gamma_t\}$ cannot be fed directly to a neural network, because their values fail to encode the cyclic nature of the series (e.g., that 23:00 and 01:00 are at the same distance from 00:00). Therefore, we need to embed them properly into a vector space. A simple *one-hot* encoding (i.e., one Boolean per possible value) would produce an excessive number of variables, and it would lose again the relative temporal distance among the values. To this aim, we propose a *circular 2D* embedding that can preserve such information by mapping each original k-values sequence to the 2D coordinates $\pi_k(i)$ of the vertices of a regular k-sides normalized polygon:

$$\pi_k(i) = \left(\sin\left(\frac{2\pi i}{k}\right), \cos\left(\frac{2\pi i}{k}\right)\right)^T \tag{9}$$

Using such an embedding we are able to produce an additional—*pilot*—time-series by concatenating the desired embedded temporal features. For instance, for hourly timestamps, consider $p_t = (\pi_{24}(\alpha_t),\, \pi_7(\beta_t),\, \pi_{12}(\gamma_t),\, \delta_t)^T \in \mathbb{R}^p$ ($p = 7$).

b) Piloted Sequence-to-sequence Model: The neural network model discussed in this section could be thought of as a modified version of a sequence-to-sequence model (e.g., [23]) adapted to accept an external variable, the *pilot* sequence, whose dynamics is known a-priori. It can be described as

$$\begin{aligned} &\mathcal{E}_\theta : \mathbb{R}^{(m+p+1)} \times \mathbb{R}^{2d} \to \mathbb{R}^{2d} \\ &\mathcal{D}_\theta : \mathbb{R}^{p} \times \mathbb{R}^{2d} \to \mathbb{R}^{d} \times \mathbb{R}^{2d} \\ &\mathcal{R}_\theta : \mathbb{R}^{d} \to \mathbb{R} \end{aligned} \tag{10}$$

where $\mathcal{E}_\theta$ and $\mathcal{D}_\theta$ are two LSTM layers with d units and $\mathcal{R}_\theta$ is a fully-connected layer acting as a *rectifier* (that reshape the result to the desired dimensions).

Let $(x_{t-\lambda+1}, \ldots, x_t)$ be the input sequence, $(y_{t-\lambda+1}, \ldots, y_t)$ the auxiliary sequence, and $(p_{t-\lambda+1}, \ldots, p_t, \ldots, p_{t+\varphi})$ the pilot sequence. The output $(\hat{x}_{t+1}, \ldots, \hat{x}_{t+\varphi})$ is generated as follows:

$$\begin{aligned} &\tilde{h}_{t-\lambda} = 0, && \tilde{h}_{t-\lambda} \in \mathbb{R}^{2d} \\ &z_s = \mathrm{vec}(x_s, y_s, p_s), && t-\lambda < s \le t \\ &\tilde{h}_s = \mathcal{E}_\theta(z_s, \tilde{h}_{s-1}), && t-\lambda < s \le t \\ &(o_s,\, \tilde{h}_s) = \mathcal{D}_\theta(p_s, \tilde{h}_{s-1}), && t < s \le t+\varphi \\ &\hat{x}_s = \mathcal{R}_\theta(o_s), && t < s \le t+\varphi \end{aligned} \tag{11}$$

where $\tilde{h}_s$ are the latent states. Note that the number of learnable parameters depends only on the feature dimensions, not on the length of input and output sequences (λ and φ).

c) Trend-Seasonality Decomposition: Time-series decomposition techniques can be effectively used to improve forecasting accuracy [34]. In this case, we opted for *Additive Decomposition*, by decomposing time-series $\{x_t\}$ in its *trend* and *seasonality*, such that $x_t = b_t + s_t$. By assuming such a decomposition, our aim is to devise a decomposable model that produces forecasts by aggregating the contributions of two different models for trend and seasonal components. Without loss of generality, let us consider a simplified case in which there are no auxiliary variables. The workflow to train a decomposable model, and to ultimately forecast the target time-series, consists of:

a) **Fitting the trend model.** In our implementation, we fit a logistic function $a + \frac{b}{1+e^{-ct}}$ over $\{x_t\}$, using the Levenberg–Marquardt optimization technique, to obtain $\{b_t\}$ for $t \in [0, T_{\mathtt{train}}]$.
b) **Data detrendization.** Remove the trend component from the data: $s_t := x_t - b_t$.
c) **Fitting seasonality model.** Train the neural architecture over $\{s_t\}$, for $t \in [0, T_{\mathtt{train}}]$.
d) **Seasonality forecasting.** Use the neural model to generate a forecast $\{\hat{s}_t\}$ for $t \in [T_{\mathtt{train}} + 1, T_{\mathtt{test}}]$.
e) **Trend forecasting.** Use the fitted trend function to generate the forecasted trend $\hat{b}_t$ for $t \in [T_{\mathtt{train}} + 1, T_{\mathtt{test}}]$.
f) **Aggregate contributions.** Compute $\hat{x}_t := \hat{b}_t + \hat{s}_t$d for $t \in [T_{\mathtt{train}} + 1, T_{\mathtt{test}}]$.

If needed, in steps d) and e), temporal embedding can be added to train the seasonality and to forecast the target time-series.

V. Experiments

In this section, we report experimental results from the application of the techniques described in Sections III and IV on data provided by Vodafone. We compared the different techniques according to three main aspects: *accuracy*, *training time* and *stability*. Stability refers to the capability of a model to produce similar results under different weights initial values and/or small variations of the hyper-parameters.

A. Experimental Set-up

We focused our experimental evaluation on data coming from two distinct VNFs, namely *CSCF* and *DRA*. Both data-sets report samples recorded with hourly granularity, but we also re-sampled the DRA data-set to get a daily-aggregated (average) version. The metrics under analysis was chosen by Vodafone due to their relevance in the monthly monitoring and reporting activities performed by the NFV capacity team. Due to confidentiality reasons, such data-sets will not be made available to the public.

The CSCF data-set spans a time range of 16 months, with hourly frequency. It reports the dynamics of the `cpu|usage_average` infrastructure-level metric, averaged among the VMs composing the VNF, and of four application-level metrics: `SCSCF_REGISTERED_USERS`, `SCSCF_SUCCESSFUL_INIT_REGIST`, `SCSCF_RE_REGISTRATION_ATTEMPTS`, `UNREGISTERED_IMPI_ON_SCSCF`. Such metrics exhibit a low Pearson correlation. We used the last 30 days as test set, to evaluate the trained models. The reminder is divided in training and validation splits, by taking the first 90% and the last 10%, respectively. The goal is to forecast `cpu|usage_average`.

The DRA data-set includes hourly- (DRAh) and daily-aggregated (DRAd) time-series spanning a time range of 16 months. The data-sets report the dynamics of the `cpu|usagemhz_average` infrastructure-level metric, averaged among the VMs composing the VNF, and of two application-level metrics: `DRA-DIAM-MSG.0.Max_TPS`, `DRA-DIAM-INT.0-S6a/S6d.Res_Sent`. Such metrics exhibit a low Pearson correlation. We used the last 6 months as test set, to evaluate the trained models. The remainder is divided in training and validation splits, by taking the first 90% and the last 10%, respectively. The goal is to forecast `cpu|usagemhz_average`.

Experiments involving neural architectures were carried out on a Google Cloud Platform VM, equipped with: an Intel Xeon processor (24 virtual CPU cores, 2 GHz); 120 GB of RAM; an NVidia Tesla V100 GPU (16 GB of dedicated memory, CUDA 10.0); Debian 9.9 operating system. Experiments involving classical forecasting techniques were carried out on an on-premise test-bed, equipped with: an AMD Ryzen 7 2700x processor (16 virtual CPU cores, 3.7 GHz); 64 GB of RAM;

TABLE I: Neural architecture configurations for each data-set.

	CSCF	DRAh	DRAd
λ	24, 168, 720	24, 168, 720	7, 30
φ	24, 168, 720	24, 168, 720	7, 30, 182
d	25, 50,..., 150	25, 50,..., 150	25, 50,..., 150
b	32, 512	64, 512	16
$\mathcal{T}$	false	true	true

TABLE II: HW configurations space for each data-set.

	CSCF	DRAh	DRAd
m	24, 48,..., 168	24, 48,..., 168	3, 7, 14, 21, 28
α	0.0, 0.1,..., 0.5	0.0, 0.1,..., 0.5	0.0, 0.1,..., 0.5
β	0.0, 0.1,..., 0.5	0.0, 0.1,..., 0.5	0.0, 0.1,..., 0.5
γ	0.0, 0.1,..., 0.5	0.0, 0.1,..., 0.5	0.0, 0.1,..., 0.5
ϕ	0.1, 0.2, 0.3, 0.4, 1	0.1, 0.2, 0.3, 0.4, 1	0.1, 0.2, 0.3, 0.4, 1
b_t	add, mul	add, mul	add, mul
s_t	add, mul	add, mul	add, mul

TABLE III: SARIMA configurations space for each data-set.

	CSCF	DRAh	DRAd
p	1, 2,..., 5	1, 2,..., 5	1, 2,..., 5
d	0	0	0, 1
q	0, 1, 2	0, 1, 2	0, 1, 2
P	0, 1,..., 7	0, 1,..., 7	0, 1,..., 7
D	0	0	0, 1
Q	0, 1, 2	0, 1, 2	0, 1, 2
m	24	24	7

TABLE IV: Average KPIs (among 10 repetitions) for the configuration with the best average MAPE (CSCF data-set).

	MAPE [%]	RMSE	TT [s]
baseline	4.39	1.17	30.23
den2den	4.18	1.22	2.44
seq2den	4.27	1.30	98.11
seq2seq	3.04	0.95	23.39

Ubuntu 18.04 operating system. Both environments were configured with Python 3.7.7, `tensorflow-gpu` 1.14.0, `statsmodels` 0.12.0, `numpy` 1.18.5.

B. Presentation of Results

We compare the performance of the models when provided with the same data. So, we trained each model with each of the 3 data-sets described above. For our evaluation, we considered the following key performance indicators (KPIs):

- **RMSE**: the Root Mean Squared Error of the forecast $\{\hat{x}_t\}$ with respect to the ground truth $\{x_t\}$, for $t \in [T_{\texttt{train}} + 1, T_{\texttt{test}}]$.
- **MAPE**: the Mean Absolute Percentage Error of the forecast $\{\hat{x}_t\}$ with respect to the ground truth $\{x_t\}$, for $t \in [T_{\texttt{train}} + 1, T_{\texttt{test}}]$. MAPE is defined as the percentage of the average forecasting error:

$$\frac{100}{T_{\texttt{test}} - T_{\texttt{train}}} \cdot \sum_t \left| \frac{x_t - \hat{x}_t}{x_t} \right|$$

- **Training Time (TT)**: The time elapsed between the start of the first epoch the end of the last epoch of training, expressed in seconds.

Each model was tested under multiple different configurations of its parameters. If the training algorithm of the model involved a *random* weights initialization, each configuration was repeated for 10 independent runs. The performance indicators were then computed by averaging the independent scores. For neural architectures, a model configuration consists of the following hyper-parameters:

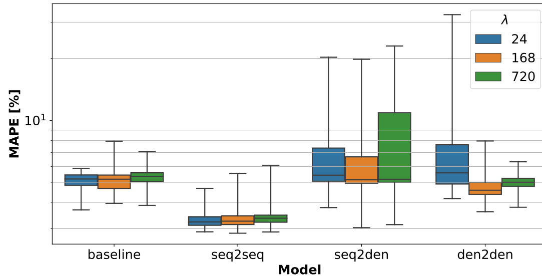
- λ: the length of the input sequence
- φ: the length of the output sequence
- d: the number of units of the LSTM cell
- b: the batch size
- $\mathcal{T}$: whether the trend decomposition was used.

For the *baseline* architecture, φ corresponds to the length of the forecast (i.e., $T_{\texttt{test}} - T_{\texttt{train}}$). Also, note that trend decomposition is only available for the *seq2seq* architecture. The configurations used for the neural architectures are summarized in Table I. Likewise, the configurations used for HW and SARIMA are summarized in Table II and Table III, respectively.

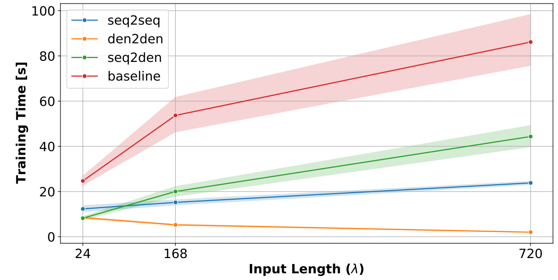
In the remainder of this section, results will be presented by means of tables, box-plots and line-plots. For each model, tables report the mean scores of the most accurate configuration (i.e., the one with the lowest average MAPE). In other words, tables show the *lower-bounds*, in terms of accuracy, for each model. Box-plots allow for visualizing the *average* accuracy of a model: the shorter the box, the lower the variance in the accuracy; the lower the box is positioned along the y-axis, the higher the average accuracy of the model. Combined with tables, box-plots provide insights on models stability. To ease the visualization, box-plots are grouped by λ. For each value of λ, the boxes show how the model accuracy changes when the remaining hyper-parameters (i.e., φ, b and d) are tuned. Note that box-plot whiskers are set such that no value is excluded from the visualization. While running such a huge amount of configurations, we noticed that the choice of λ may deeply impact on the training time of the model. Line-plots allow for visualizing how big the impact of λ is for the proposed architectures. For each value of λ, the solid line is an estimate of the central tendency, computed as φ, b and d change, while the stripe width indicates the confidence interval.

C. Neural Architectures

1) CSCF Dataset: The CSCF data-set contains a stationary time-series with a strong seasonality component. Table IV highlights that the seq2seq architecture outperforms the others in terms of accuracy, but the den2den has an interestingly reduced training time by an order of magnitude, at the cost of raising the accuracy from 3% to 4.2%. Figure 4a also shows that, in general, seq2seq provides more stable results, as 50% of the observations are densely concentrated around the average and the whiskers are relatively short (consider also the log-scale on the Y axis). This entails that the model is very robust with respect to the hyper-parameters variations.

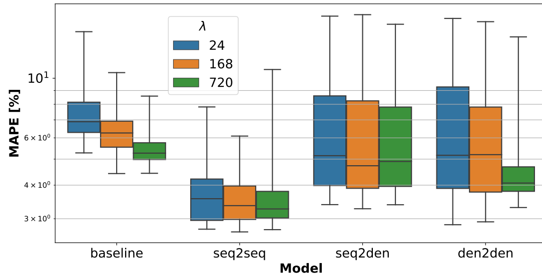


(a) MAPE distributions per input sequence length.

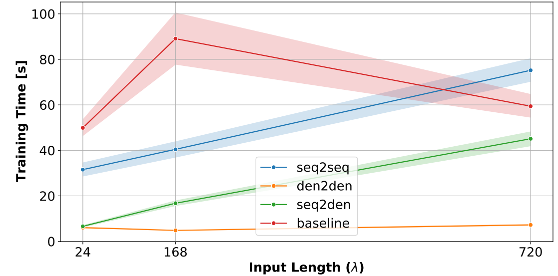


(b) Relation between training time and input sequence length.

Fig. 4: Performance measures for neural architectures applied to the CSCF hourly data-set.



(a) MAPE distributions per input sequence length.



(b) Relation between training time and input sequence length.

Fig. 5: Performance measures for neural architectures applied to the DRAh hourly data-set.

TABLE V: Average KPIs (among 10 repetitions) for the configuration with the best average MAPE (DRAh data-set).

	MAPE [%]	RMSE	TT [s]
baseline	5.01	55.69	40.37
den2den	3.38	41.19	3.12
seq2den	3.72	42.37	4.93
seq2seq	2.87	38.43	35.60

TABLE VI: Average KPIs (among 10 repetitions) for the configuration with the best average MAPE (DRAd data-set).

	MAPE [%]	RMSE	TT [s]
baseline	3.54	38.80	6.12
den2den	1.94	23.29	1.35
seq2den	2.12	24.72	1.67
seq2seq	2.06	26.26	3.29

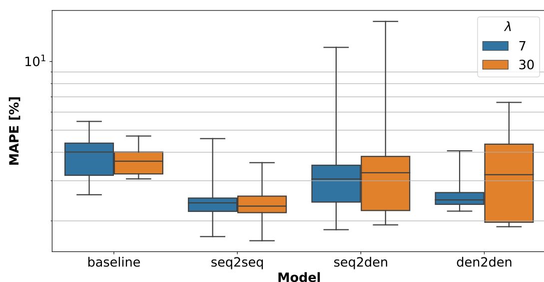
As expected, Figure 4b shows that the training time grows as the sequence gets longer for all models but den2den.

2) DRAh Dataset: Table V shows that the seq2seq architecture achieves the best performance for the DRAh data-set. From Figure 5a, we can see that, in general, its accuracy is more stable with respect to variations of the hyper-parameters, compared to the other models. As expected, Figure 5b shows that the training time grows as the sequence gets longer for all models but den2den. When applied on this data-set, with such a huge forecasting range (i.e., $T_{\texttt{test}} - T_{\texttt{train}} = 4368$), the baseline model turns out to have a decoder (i.e., a dense layer) composed by a 4368-rows weight matrix, as it is designed to provide the output sequence in one-shot at the decoding stage. In this case, we were expecting an explosion of the training time, which does not happen. This is probably due to the nice way the computations of the decoder can be parallelized on the underlying GPU. Indeed, the other architectures do not fully exploit the GPU acceleration, either because they are made entirely of recurrent units or because of their *closed-loop* nature. However, defining the root-cause of such an unexpected performance boost requires a deeper investigation.

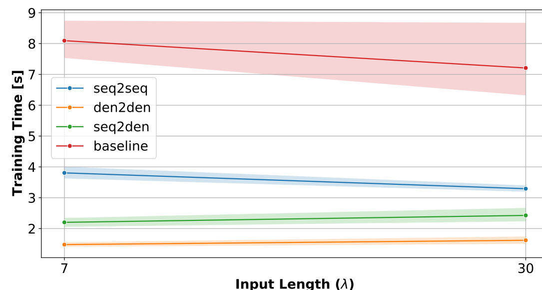
3) DRAd Dataset: Table VI shows that the den2den model achieves the best performance for the DRAd data-set. However, from Figure 6a, we can see that the variability of the accuracy for this architecture strongly depends on the choice of the hyper-parameters. As can be seen in Figure 6b, due to the small number of timestamps, the differences in terms of training time are not relevant for this data-set.

D. Classical Forecasting Techniques

Table VII summarizes the best results achieved by the classical forecasting techniques, described in Section III. Regarding the accuracy, Figure 7a shows that HW outperforms SARIMA, for all the tested data-set. Looking at Figure 7b, We can draw a similar conclusion for what concerns the training time. In particular, for the CSCF and DRAh data-sets, SARIMA generally requires more than 100 seconds per run, while HW less than a second.

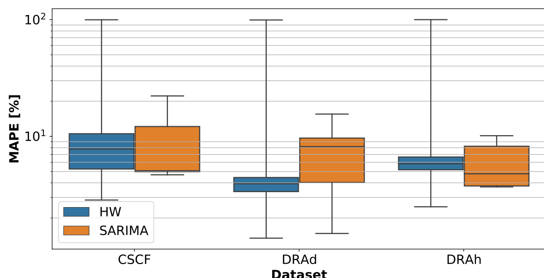


(a) MAPE distributions per input sequence length.

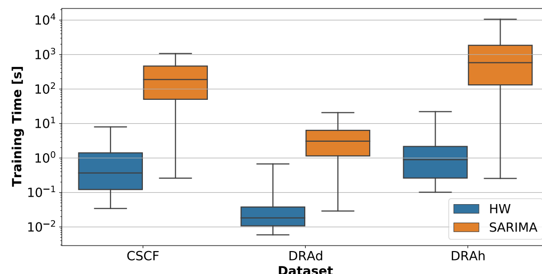


(b) Relation between training time and input sequence length.

Fig. 6: Performance measures for neural architectures applied to the DRAd daily data-set.



(a) MAPE distributions.



(b) Training time distributions.

Fig. 7: Performance measures for classical forecasting techniques.

TABLE VII: KPIs for the configuration with the best MAPE.

Dataset	Model	MAPE [%]	RMSE	TT [s]
CSCF	HW	2.85	1.00	2.14
	SARIMA	4.69	1.41	832.46
DRAd	HW	1.34	17.09	0.01
	SARIMA	1.47	18.34	12.86
DRAh	HW	2.49	35.07	0.22
	SARIMA	3.69	42.08	154.43

E. Comparative Analysis

The presented results highlight what trade-offs can be achieved between accuracy and training time with the various techniques. Among the neural architectures, in general, the *seq2seq* outperforms the others in terms of accuracy and stability (see Figures 4a, 5a and 6a). Such level of accuracy is achieved thanks to the time embedding and the additive decomposition. In particular, the highest stability is due to using recurrent layers in both encoding and decoding phases, that reduces the impact of the hyper-parameters λ and φ. However, in general, seq2seq training time grows proportionally to the length of the input and output sequences, as the recurrent layers process samples sequentially (see Figures 4b and 5b). Note that this process cannot be improved even using GPU acceleration, as it is not possible to parallelize the computation. On the contrary, the *den2den* model makes a much better use of GPU acceleration and results to be the fastest neural model.

For what concerns classical forecasting techniques, there are cases such that they match neural architectures in terms of accuracy. For instance, comparing Tables IV to VI with Table VII, we can see HW *best* runs consistently scoring a—slightly—lower MAPE than seq2seq, for CSCF (2.85% vs 3.04%), DRAh (1.34% vs 2.06%) and DRAd (2.49% vs 2.87%) datasets. However, such level of performance is most likely due to a particularly *lucky shot*, in terms of hyper-parameters tuning, rather than an evidence of its superior capacity at modeling complex time-series.

The low stability shown by HW seems to support this hypothesis. Without any other statistical assumption (e.g., similarity of the distribution with other times-series or different time-ranges), the results reported in Section V-D suggest that HW is less likely to achieve the same accuracy of the seq2seq architecture for the tested data-sets. Figure 7a shows that the choice of the hyper-parameters has a strong impact on the performance of HW. In fact, for all the datasets, the worst HW runs score a MAPE that is very close to 100%. This is not the case for seq2seq, whose overall worst performance is around 10% (see Figure 5a). On hourly data-sets (i.e., CSCF and DRAh), despite the best HW run achieves a MAPE lower than 3%, the first quartile is greater than 5% (see Figure 7a). Instead, for the seq2seq architecture, the third quartile is lower than 4% (see Figures 4a and 5a). In other words, 75% of HW runs score a MAPE greater than 5%, while 75% of seq2seq runs score a MAPE lower than 4%. On the DRAd data-set, despite the best HW run achieves a MAPE lower than 2%, the first quartile is greater than 3% (see Figure 7a). In this case,

the seq2seq architecture exhibits a third quartile strictly lower than 3% (see Figure 6a). In other words, 75% of HW runs score a MAPE greater than 3%, while 75% of seq2seq runs score a MAPE lower than 3%.

VI. Conclusions and Future Works

In this paper, the problem of forecasting the future evolution of metrics in an NFV infrastructure was tackled. A number of techniques for time-series forecasting was compared experimentally, using a real data-set from the production NFV infrastructure of the Vodafone network operator.

Regarding possible future works on the topic, we plan to extend our analysis by: (i) comparing additional data-sets among the many available within the Vodafone data centers, possibly extending the set of considered metrics; (ii) considering techniques that can process additional information—besides the operational metrics to be forecasted—like, for instance, rough traffic volume forecasts manually produced by internal analysts quarterly; (iii) investigating the applicability of topology-aware prediction techniques [7] that look quite promising in the context of NFV metrics forecasting.

References

[1] NFV Industry Specif. Group, "Network Functions Virtualisation," Introductory White Paper, 2012. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[2] I. P. Chochliouros, A. S. Spiliopoulou, A. Kostopoulos, M. Belesioti, E. Sfakianakis, P. Georgantas, E. Vasilaki, I. Neokosmidis, T. Rokkas, and A. Dardamanis, "Putting intelligence in the network edge through nfv and cloud computing: The sesame approach," in *Engineering Applications of Neural Networks*. Springer, 2017, pp. 704–715.

[3] M. Miyazawa, M. Hayashi, and R. Stadler, "vNMF: Distributed fault detection using clustering approach for network function virtualization," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 640–645.

[4] T. Niwa, M. Miyazawa, M. Hayashi, and R. Stadler, "Universal fault detection for nfv using som-based clustering," in *17th Asia-Pacific Network Operations and Management Symposium*, 2015, pp. 315–320.

[5] T. Cucinotta, G. Lanciano, A. Ritacco, M. Vannucci, A. Artale, J. Barata, E. Sposato, and L. Basili, "Behavioral analysis for virtualized network functions: A som-based approach," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, INSTICC. SciTePress, 2020, pp. 150–160.

[6] G. Lanciano, A. Ritacco, T. Cucinotta, M. Vannucci, A. Artale, L. Basili, E. Sposato, and J. Barata, "Som-based behavioral analysis for virtualized network functions," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, ser. SAC '20, 2020, p. 1204–1206.

[7] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 106–120, 2017.

[8] N. Jalodia, S. Henna, and A. Davy, "Deep Reinforcement Learning for Topology-Aware VNF Resource Prediction in NFV Environments," in *IEEE Conference on Network Function Virtualization and Software Defined Networks*, 2019, pp. 1–5.

[9] C. Makaya, D. Freimuth, D. Wood, and S. Calo, "Policy-based nfv management and orchestration," in *IEEE Conference on Network Function Virtualization and Software Defined Network*, 2015, pp. 128–134.

[10] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction," in *IEEE Conference on Computer Communications*, vol. 2018-April, 2018, pp. 486–494.

[11] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu, "NFVnice: Dynamic backpressure and scheduling for NFV service chains," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, vol. 14, 2017, pp. 71–84.

[12] G. A. Carella, M. Pauls, L. Grebe, and T. Magedanz, "An extensible autoscaling engine (ae) for software-based network functions," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2016, pp. 219–225.

[13] C. H. T. Arteaga, F. Rissoi, and O. M. C. Rendon, "An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an nfv-based epc," in *13th International Conference on Network and Service Management*, 2017, pp. 1–7.

[14] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A Comparison of Reinforcement Learning Techniques for Fuzzy Cloud Auto-Scaling," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 64–73.

[15] P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang, "Efficient auto-scaling approach in the telco cloud using self-learning algorithm," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[16] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proceedings of the International Symposium on Quality of Service*, vol. 19, 2019, pp. 1–10.

[17] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling vnfs using machine learning to improve qos and reduce cost," in *IEEE International Conference on Communications*, 2018.

[18] Z. Zaman, S. Rahman, and M. Naznin, "Novel approaches for vnf requirement prediction using dnn and lstm," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[19] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. A. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, vol. 33, no. 4, pp. 917–963, 2019.

[20] P. Malhotra, T. Vishnu, L. Vig, P. Agarwal, and G. Shroff, "Timenet: Pre-trained deep recurrent neural network for time series classification," in *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2017.

[21] S. S. Rangapuram, M. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, "Deep state space models for time series forecasting," in *Advances in Neural Information Processing Systems*, vol. 2018-Decem, 2018, pp. 7785–7794.

[22] N. Laptev, J. Yosinski, L. Erran Li, and S. Smyl, "Time-series Extreme Event Forecasting with Neural Networks at Uber," *International Conference on Machine Learning - Time Series Workshop*, pp. 1–5, 2017.

[23] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[24] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[25] D. Bacciu, F. Errica, A. Micheli, and M. Podda, "A gentle introduction to deep learning for graphs," *Neural Networks*, vol. 129, 2020.

[26] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[27] C. Chatfield, "The holt-winters forecasting procedure," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 27, no. 3, pp. 264–279, 1978.

[28] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, 1976.

[29] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 6088, no. 33, pp. 533–536, 1986.

[30] J. Schmidhuber and S. Hochreiter, "Long short-term memory," *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, 1997.

[31] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, San Diego, CA, USA, May 7-9*, 2015.

[32] J. F. Kolen and S. C. Kremer, *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*, 2001, pp. 237–243.

[33] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR*, 2015.

[34] S. J. Taylor and B. Letham, "Forecasting at scale," *The American Statistician*, vol. 72, no. 1, pp. 37–45, 2018.