IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 4 July 2023, accepted 12 September 2023, date of publication 25 September 2023, date of current version 5 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3319075

RESEARCH ARTICLE

Sequential Learning for Modeling Video Quality of Delivery Metrics

TAUTVYDAS LISAS AND RUAIRÍ DE FRÉIN
School of Electrical and Electronic Engineering, Technological University Dublin, Dublin 7, D07 XT95 Ireland
Corresponding authors: Tautvydas Lisas (D17125255@mytudublin.ie) and Ruairí de Fréin (ruairi.defrein@tudublin.ie)

This work supported by the Science Foundation Ireland (SFI) under Grant 15/SIRG/3459 and Grant 13/RC/2077_P2.

ABSTRACT Video streaming traffic growth poses a challenge for many video content providers to maintain high video quality on their networks. Modern day networks are highly dynamic due to the adaptation of routing protocols, time-varying loads, and adaptive codecs. We consider how these dynamics should be incorporated in learning algorithms. We ask how can sequential learning be used in order to account for time-varying trends in learning algorithms that perform forecasts in time-varying video delivery systems? We propose two approaches. The first is called ASAPjitter. It uses a Recurrent Network (RN) to forecast gradients in a time series of jitter measurements for a video stream and a Convolutional Neural Network to classify the forecasts. The second approach is called FEATjitter. It maps the jitter time series to a higher dimensional feature domain. This novel feature domain transform has the aim of enhancing the quality of prediction and classification of future video jitter measurements. Jitter captures varying congestion levels present in the network. Three parameters are extracted from the jitter time series: its period, base and rate of decay. These parameters are used to train an artificial RN to perform forecasts. A batch learning approach such as the Neural Network (NN) or otherwise referred to a Multi-Layer Perceptron (MLP) is used for classifying the feature domain forecasts. Experimental results demonstrate how sequential learning can be used effectively to predict and classify jitter using Deep Learning (DL) frameworks. The accuracy achieved by ASAPjitter is 84.5% compared to 91.6% for FEATjitter. Performance gains in terms of increased classification accuracy are due to the learning algorithm operating in this feature-space. Performance gains in terms of forecasts are more effective in the time-domain. Our approach contributes to the development of more effective algorithms for managing video streaming traffic in dynamic network environments.

INDEX TERMS Jitter, sequential learning, supervised learning, video, quality of delivery.

I. INTRODUCTION

In recent years, Deep Learning (DL) has been adopted in various disciplines such as computer vision [1]. However, the application of DL to detect network congestion has only gained attention of late. With video traffic growing substantially due to the increase of video availability, network providers have to adapt and to counter-act the effects of congestion on video traffic [2]. This establishes the need for accurate prediction and classification algorithms.

Traditional routing protocols such as Open Short Path First (OSPF) do not learn from previous experiences regarding network abnormalities such as congestion. Recent Software Defined Network (SDN) frameworks have attempted to re-route traffic around network congestion and link-failures. These approaches come in two forms: reactive and proactive [3]. Proactive path reallocation approaches rely on traffic prediction algorithms which can give advance notice of future congestion events. Batch-learning approaches for video quality prediction typically do not model dependence across time [4], although the approach in [7] estimates parameters for a deterministic function of time. In this paper we investigate the potential gains from using sequential learning that explicitly models time dependence. We name this approach Automated Sequential Adaptive Prediction of jitter (ASAPjitter).

The associate editor coordinating the review of this manuscript and approving it for publication was Wei Liu.

Quality of Delivery (QoD) measurements are network metrics that capture the performance of a service that is deployed on a network [5]. Common video QoD metrics include packet delay from the source to the destination, which includes transport and queuing delays, packet loss and throughput. Modelling packet delay metrics can provide valuable insights into how a network is performing. When a network experiences high levels of congestion it can degrade video quality and cause buffering. When this occurs the playback device aims to estimate how long a player should wait before playing the media so interruptions are not experienced. The buffer size is determined by a QoD metric which is called jitter [6].

Delivering high quality video over dynamic complex networks has traditionally been achieved using static network management solutions. Recently, a novel passive measurement approach was proposed to estimate jitter in the presence of varying traffic [7]. This approach uses the behaviour of a video codec to estimate network congestion. Given that the codec is already estimating the level of compression it needs to apply to the video in relation to network congestion, it can also be used to accurately estimate the congestion changes in the network. The effect congestion has on a jitter time series can be expressed using three parameters: its period, p, its base, b, and its rate of decay, λ. We show that by estimating these parameters we can use classification learning algorithms that would otherwise be unsuited for learning the exponential components observed in jitter time series. We name this approach Feature Estimated Adaptive Training of jitter (FEATjitter).

The recent success of DL approaches in other disciplines motivates the need for an investigation into the applicability of sequential learning algorithms for high accuracy forecasts and classification of network congestion state. Predicting video quality is a complex task. The problem's complexity lies in the nature of the delivery system, which switches between diverse user demands and service types over a finite, and shared network [8]. However, if successful, video quality prediction can improve video quality delivery.

In this paper we make the following contributions.

- We investigate how Recurrent Networks (RN) can be fit to jitter time series, which display the periodically decaying exponential behaviour observed in the time series extracted from a real network test-bed consisting of Cisco routers.
- We determine the issues involved in fitting parameters to the jitter time series which emanate from adaptive video codecs, and propose a solution.
- We consider if fitting RNs to model parameters/features can improve prediction accuracy. The jitter time series is composed of decaying exponentials, which suggests that its dynamic range is large. Model feature time series are not smoother than the underlying time series, but their dynamic range is smaller. This suggests that they can be better modelled by the RN.

TABLE 1. Comparing learning strategies under the following headings: Domain - refers to the domain the learning algorithm is operating; Detection Speed - the amount of samples needed for detecting change in congestion; Modelling Technique - patterns in the data the learning strategy is modelling; Temporal Dependence - if periodicity is explicitly modelled by the learning strategy; Prediction Effectiveness - if the strategy can predict future values effectively; Classification Accuracy - the accuracy results.

Learning Strategy	ASAPjitter	FEATjitter
Domain:	Time	Feature
Detection Speed	1 sample	ω_i
Modelling Technique	Gradients	Tabular Functions
Temporal Dependence	Yes	No
Prediction Effectiveness	High	Low
Classification Accuracy	84.5%	91.6%

- We develop reconstruction methods for jitter time series using the forecasted and the original features.
- The different prediction strategies investigated in this paper are summarized in Table. 1. We evaluate the success of different prediction modalities. We recommend the use of sequential learning methods for jitter prediction, when the jitter time series result from video traffic which is encoded using adaptive codecs.

This paper is organised as follows. In Section II we review the related work on video quality of delivery prediction. Section III describes a network test-bed which is composed of Cisco routers, a VLC media player client and server and Ostinato. It also explains how the jitter time series is measured. Section IV introduces how the jitter features are estimated. Section V contains a review of RNs and the recent success in DL optimisation. Section VI illustrates the results of the experiments and the reconstruction method.

II. RELATED WORK

Video quality prediction has received attention due to the growing quantity of video traffic on modern networks. Machine learning (ML) and DL techniques have previously been used for optimising these networks. The question of how to select, process and predict future network metrics, to successfully improve QoD, is still an open question.

The authors of [7] discuss the problem of using Infinite Impulse Response (IIR) filters, to estimate jitter for video streaming services. These IIR filters do not attempt to estimate the congestion in the network that causes the jitter to change. They demonstrate that if the jitter is estimated during varying congestion levels, it can be used to significantly improve congestion state classification for all congestion levels. To achieve this, they propose a novel algorithm that estimates the jitter parameters using a set of new metrics b, λ, and p, to improve the learning abilities of subsequent classification algorithms. They verify their approach by using the popular ML methods: Decision Trees (DT) and Random Forests (RF). A classification accuracy of 98.3% for low congestion, 41.4% for medium congestion, and 95.8% for high congestion is achieved. A 100-tap moving average filter is used to achieve this classification accuracy, which potentially

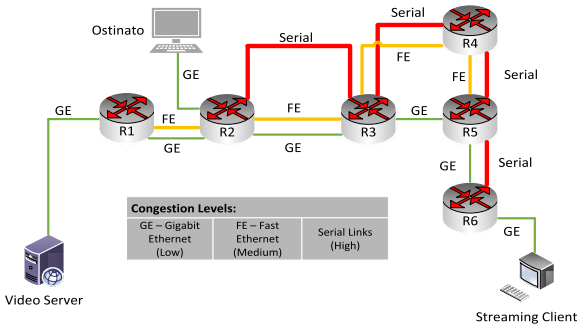

FIGURE 1. The network topology consists of six 2911 ISR routers and a video server which is streaming a H.264 video over RTP. The different link technologies (Gigabit Ethernet, Fast Ethernet and Serial) are used to create routes through the network with different bandwidths, which cause variability in the measured jitter values. This variability is explained by the background congestion level. The level of background traffic is varied using Ostinato.

decreases the responsivity of the system. A second potential shortcoming of the approach is that the temporal dependence between jitter measurements is modelled using a deterministic function, which obtains its parameters from learning rules for b, λ and p. The approach of modelling the system using a deterministic function may involve frequent re-estimation of model parameters when the background traffic varies significantly. We consider a sequential learning approach which takes into account temporal dependencies in the time series: (1) by investigating how jitter time series can be modelled using RNs; and (2) by coupling the expressivity of the feature set b, λ and p, which were proposed in [7], with the sequential learning capability of RNs.

Other learning approaches that do not model temporal dependence in network congestion and traffic classification are described in [9]. The primary focus of this paper is on the use of Support Vector Machines (SVM). In [10], a Multi-layer Perceptron (MLP) is used to classify packet loss to adjust congestion window size effectively. Gradient boosting techniques such as Stochastic Gradient Boosting (SGB) and Extreme Gradient Boosting (EGB) were evaluated in [11] for a traffic classification architecture based on SDN, paving the way for the use of learning algorithms for network control. In [12], progress towards successful SDN traffic classification was made using a DL architecture, called Deep-SDN, which performed a more extensive analysis on the efficiency of the SDN-based architecture by examining the accuracy, precision, recall and F_1-measure. Deep-SDN was found to increase the accuracy of application-type classification.

Fadlullah et al. discussed state-of-the-art ML and new DL research for network systems in [13]. The authors highlighted the emerging popularity of DL in various computer science fields due to recent optimisation advances [14]. However, they stated that the application of DL has only started to gain traction in network systems. They suggested that it could be used to stimulate the area of network traffic control. They prepared an overview of state-of-the-art DL architectures relevant to traffic control, suggesting that DL should be used for jitter prediction. The work in [15] illustrated how the choice of a suitable DL algorithm improved Quality of Experience (QoE) using video streaming as a use case. The authors reflected on how a suitable time series representations could improve the achievable accuracy of DL algorithms, by comparing three algorithms, SVMs, RFs and Long-Short-Term-Memory (LSTM), for throughput prediction. The results showed that the LSTM had the highest prediction accuracy and that DL methods could be used to increase the prediction accuracy of a traffic control variable such as throughput. In this paper, we show that a low prediction error does not necessarily mean that we will have a good prediction for the next value. In light of this finding, there is a need for a further clarification on the authors' definition of prediction in [15]. Alternative approaches to implementing LSTM frameworks for network traffic prediction can be seen in [16], where the author stated that the LSTM framework was superior to a Least Square Support Vector Machine (LSSVM), a MLP or an Elman neural network. The work in [17], investigates the use of LSTMs for resource allocation schemes and shows that by using this prediction method, latency could be reduced by 23.2%.

The focus of this contribution is on demonstrating how network congestion can be passively detected. To achieve this we describe the difficulty jitter estimates pose for sequential learning algorithms, which are used to provide predictions. To address this challenge, we develop and evaluate a sequential learning algorithm that can be leveraged to improve the accuracy of QoD classification.

III. TESTBED: VIDEO QoD TIME SERIES

To observe video jitter measurements we used the test-bed which is illustrated in Fig. 1. A VLC server was set up to stream a video which uses the H.264 compression standard, to a VLC client. We used the H.264 codec because it is used more frequently than codecs such as H.265 and AV1, according to the authors of [18]. In the view of the authors of [18], not all browsers and device makers support other codecs such as H.265, DASH and AV1. It was reported in 2020 that 91% of browsers and device manufacturers supported H.264. Only 42% supported H.265 and 65% of participants in the survey supported DASH. The authors of [18] emphasize the continued importance of H.264, stating that its market share was 83 % in 2021 and 78/85 % in 2022. In 2022 the authors considered live encoding video systems in production and VoD systems in production and reported the percentage of systems in each category that supported H.264, e.g 78 % and 85 % respectively. These market share statistics indicate that the H.264 codec warrants continued consideration by the research community as it has not been superseded by rival codecs.

The Real-Time Transport Protocol (RTP) was used to stream the video over a topology which consists of six 2911 ISR routers. During the streaming session, congestion was introduced by injecting bursts of low, medium and high levels of interfering UDP traffic. Delivering RTP traffic, using UDP in the transport layer, is used as a protocol stack for streaming by Microsoft Teams [19]. Delivering RTP over UDP is seeing a

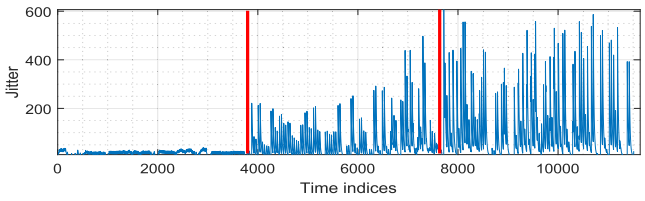

FIGURE 2. The adaptive behaviour of the H.264 codec in response to congestion is observed in this jitter time series. The low congestion indices are between $1 < n < 3939$, medium congestion is between $3939 < n < 7677$ and high congestion is between $7678 < n < 11514$ where n is the measurement index.

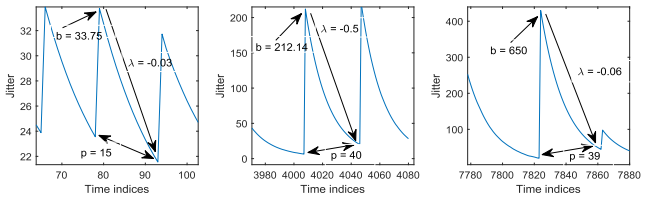

FIGURE 3. The difference in jitter features (base, b, rate of decay, λ, and the period, p), which allows us to identify each of the congestion levels, are illustrated by plotting a sequence of jitter values from low, medium and high congestion time intervals.

resurgence [20]. For example, in April 2020 Microsoft Teams supported 75 million daily active users, which represented a significant increase from the 44 million users it supported in March 2020, which underlines the significance of its market-share. Three different routing changes were invoked so that the video stream was forced to switch between paths offering different bandwidth. The connections between intermediate devices were made using Gigabit Ethernet (GE) links (1000Mbps), Fast Ethernet links (100Mbps) and serial links (64kbps). During the streaming session three congestion levels were injected using the Ostinato traffic generator and jitter measurements, which are denoted as $x[n]$, were taken from the VLC client using Wireshark. The counter n is the jitter measurement index.

We demonstrate that congestion levels can be identified by examining differences in features of the jitter time series. We hypothesize that jitter time series consist of sequences of decaying exponentials which occur periodically in the measured time series. In Fig. 2 we illustrate a jitter time series which is captured from a 10-minute long video. The base of these exponentials that occur in this time series, b, is the first value of each exponential. The rate of decay of each exponential is denoted, λ. And finally, the period of each decaying exponential is denoted as p. Fig. 3 illustrates three zoomed-in segments of the observed jitter time series. During low congestion the base and the period was much lower than during medium and high congestion. Similarly, during medium congestion the base was lower than during high congestion. In Fig. 3 the period and the rate of decay parameters for medium and high congestion are quite similar, however, the amplitude of the base differs significantly.

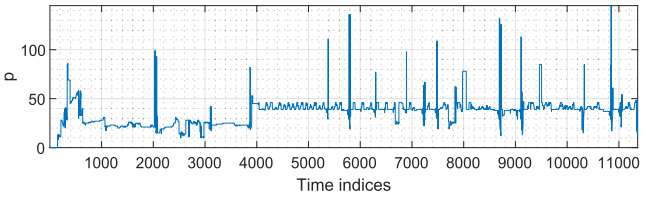

FIGURE 4. Estimated periodicity of the jitter: the low congestion occurs in the range of indices $1 < n < 3939$; medium congestion occurs between the range of indices $3839 < n < 7677$; and finally, high congestion is between $7678 < n < 11514$, where n is the measurement index.

IV. PARAMETER ESTIMATION: BATCH LEARNING

Three parameters can be extracted from the jitter signal: its period, its base and its rate of decay. First, the trend is removed by applying a Low-Pass Filter (LPF) to $x[n]$ and subtracting the low-pass filtered signal from the original signal, $x[n]$,

$$x[n] \leftarrow x[n] - \text{LPF}(x[n]). \tag{1}$$

The filter configuration consists of a 7th order Butter-worth filter with a normalised cut-off frequency of $\delta_c = 0.1$, a pass-band ripple of $\delta_r \leq 3$ dB, and a stop-band attenuation $\delta_s \geq 40$ dB. A sample window of $\omega = 150$ samples, which is approximately $2 \leq \omega_i \leq 3$ periods, is applied to $x[n]$ and using the Auto-Correlation Function (ACF) [21], the *lags* are found. The lag corresponding to the position of the first peak, on the right hand side of the peak at lag 0, is chosen as the period. Fig. 4 illustrates that the period for low congestion is approximately $p[n] = 20$ time indices, whereas during medium and high congestion, the period corresponds to a lag of approximately $40 \leq p[n] \leq 60$ time indices. We estimate the base, b, and rate-of-decay, λ, using a segment of the exponential function, $x[n] = be^{\lambda \hat{n}[n]}$. This approximately comprises the jitter measurements from the last three periods. The segment time indices are, $\omega_i[n]$. The index $\hat{n}[j]$, denotes shifted time indices, so that they start at zero, $\hat{n}[n] = n - \omega_{i-1}[n]$. Taking the natural logarithm of $x[n]$, yields a linear system, $\hat{y}[n] = \hat{b} + \lambda \hat{n}[n]$, where $\hat{y}[n] = \log x[n]$ and $\hat{b} = \log b$. A flow chart in Fig. 5 is used to demonstrate the procedure, and the resulting time series for the estimated, b and λ parameters are illustrated in Fig. 6.

V. SEQUENTIAL: TEMPORAL DEPENDENCE LEARNING

In this section we describe how Recurrent Neural Networks (RNN) can be used to model jitter time series. We then describe how parameter optimisation is achieved for these networks, by considering the characteristics of jitter time series. RNNs are one of the most powerful tools in DL. Popular applications of these systems can be seen in classification and forecasting problems. RNNs were first introduced in 1985 by Rumelhart and L.McClelland [22]. These dynamic systems were designed to process sequential data more efficiently than MLPs, which motivates their use in the context of jitter time series modelling.

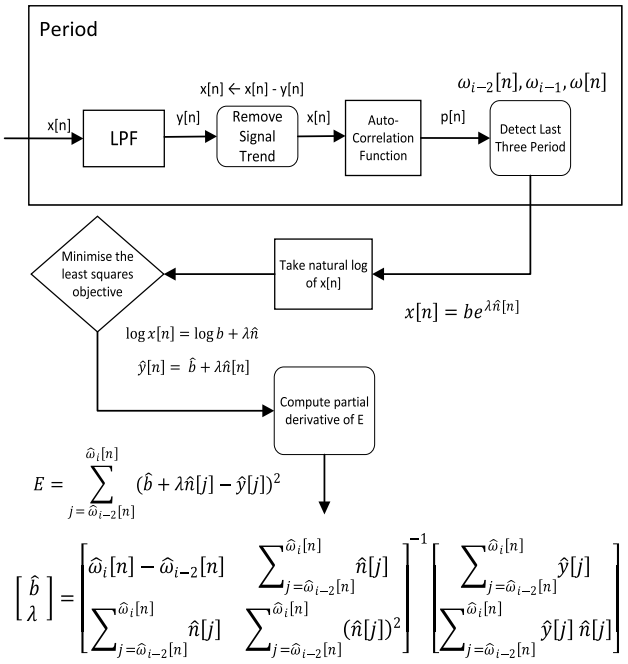

FIGURE 5. Flow-chart illustrating the feature estimation algorithm: initially the trend is removed by a LPF; the period is detected via the ACF; the base, b, and decay, λ, are estimated via least squares estimation.

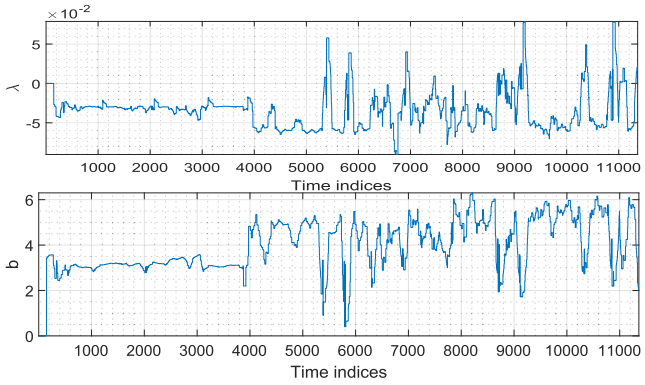

FIGURE 6. Estimated base (top) and rate of decay (bottom) of the jitter time series: low congestion occurs in the range of indices $1 < n < 3939$; medium congestion occurs between the range of indices $3839 < n < 7677$; and finally, high congestion is between $7678 < n < 11514$, where n is the measurement index.

A. RECURRENT NETWORKS

When using a traditional MLP, all the elements of the sequence are fed into the network in one go. This approach ignores temporal dependencies present in the jitter time series we observe in the test-bed [23]. We have discussed how jitter time series can be represented using a base, decay and period parameter, which suggests that temporal dependence is a crucial component. When using RNNs, each element of the sequence is fed into the recurrent unit one by one which can be seen in Fig. 7. This is achieved by applying a time-shifted rectangular window to x_n which produces a truncated time series s_t at time index t. The length of this window, which has a duration of T, is motivated by the fact that it corresponds

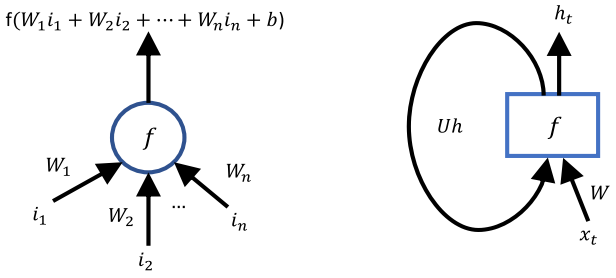

FIGURE 7. The perceptron diagram (LHS) demonstrates that the input is fed into the network all at once where, i_n, is each sample of the x_n jitter time series starting from the beginning and finishing at the end. The simple RNN diagram (RHS) demonstrates that the time series is divided into sequences, x_t, and are fed into the network independently.

to approximately two or three periods of the jitter time series, similar to the approach in [7], and by the fact that it works well in practice. The truncation procedure is defined as follows:

$$s_t = \begin{cases} x_n & \text{for } t \le i \le t + T - 1 \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The rectangular window is successively advanced by one time index, which yields successive time series segments for jitter. Each new sequence is then truncated into a set of input, x_t, and target, y_t, time series as demonstrated in Fig. 8 for the case where $T = 100$, for ease of illustration. The length of the input sequence, $T_x = 60$, and output sequence $T_y = 40$, can be modified accordingly.

$$x_t = \begin{cases} s_t, & \text{for } t \le i \le t + T_x \\ 0, & \text{otherwise, and} \end{cases} \tag{3}$$

$$y_t = \begin{cases} s_t, & \text{for } t + T_x + 1 \le i \le t + T \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

This allows the network to process the time series sequentially by finding the best fit through each sequence using the sliding window approach. The simple RNN has a recurrent hidden state which is

$$h_t = f(Wx_t + Uh_{t-1} + b). \tag{5}$$

The input vector is x_t, at time t, h_t is the hidden state, and f is the activation function. A common choice for this function is the hyberbolic tangent or the Rectified Linear Unit. The input and hidden weights are W and U_h and b is the bias. The common learning algorithms for RNNs in temporal supervised learning tasks are Backpropagation Through Time (BPTT). During the procedure the network is unfolded in time to construct a feed-forward neural network. The generalised delta rule is then applied to the unrolled network to calculate the error and update the weights [24]. Given the simple nature of back-propagation, RNNs can only bridge up to $5 \le t \le 10$ time steps. Any time step that is greater than this causes the error to grow or shrink, which induces the vanishing or exploding error problem [25].

The exploding error problem leads to oscillating weights, whereas the vanishing error problem causes the learning

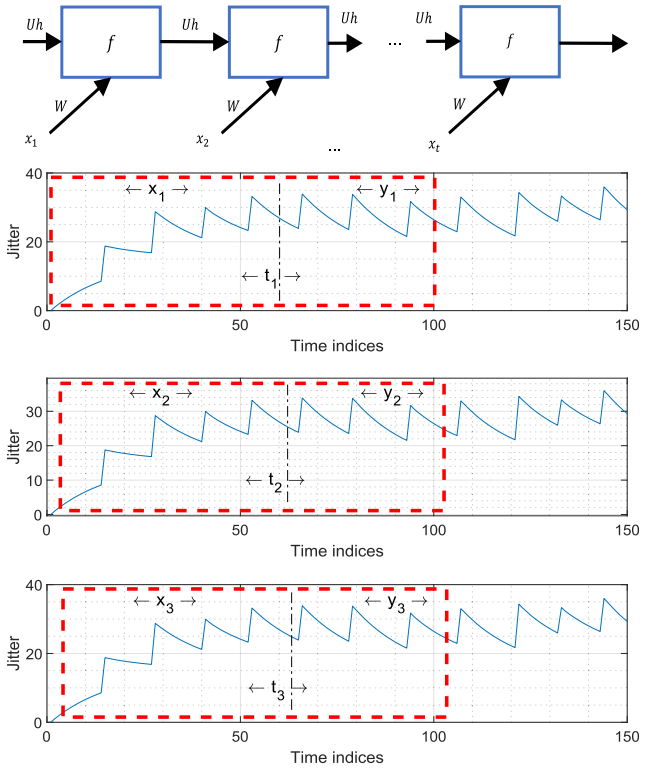

FIGURE 8. The unrolled RNN demonstrates how the jitter time series (in blue) is divided into sequences using a window which is indicated using dashed red lines. Each sequence is fed through the network independently with each window shifted to the right in steps of t+1 and the inputs: x_t and the targets: y_t, are truncated.

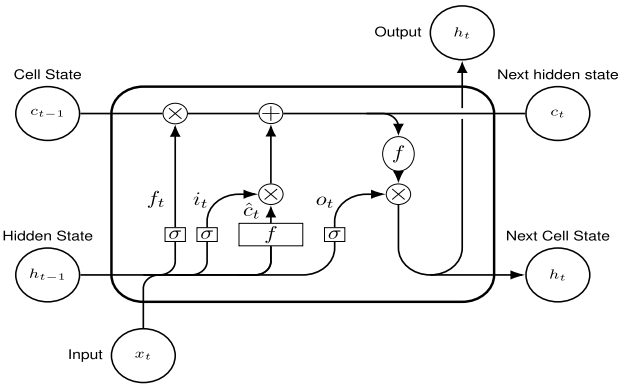

FIGURE 9. LSTM cell architecture.

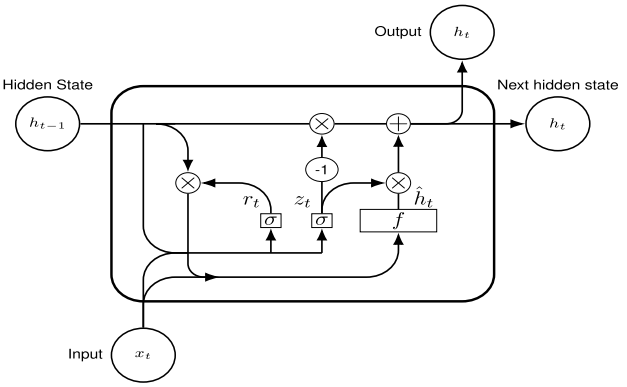

FIGURE 10. GRU cell architecture.

procedure to take an unacceptable amount of time, for the jitter application we have considered, or it does not work at all. One solution that addresses this problem is a gradient-based method called an LSTM. This network can learn how to bridge time lags of more than $t \leq 1000$ discrete time steps. LSTMs use a solution called Constant Error Carousels (CECs), which enforce constant error flow within special cells. The LSTM RNN architecture uses the simple RNN in Eqn. 5 as an intermediate candidate for the internal memory cell (state), $\hat{c}$, and adds it in an element-wise weighted-sum to the previous value of the internal memory state, c_{t-1}, to produce the current state, c_t,

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t, \tag{6}$$
$$\hat{c}_t = f(W_c x_t + U_c h_{t-1} + b_c), \tag{7}$$
$$h_t = o_t \odot f(c_t). \tag{8}$$

We pay particular attention to the activation non-linearity, f, used in Eqns. 6, 7 and 8 in this paper. We modify it based on an empirical investigation to improve its performance on the jitter time series we observe. The weighted sum is implemented in Eqn. 6 in [26], and the gate units, i_t, f_t, and o_t are represented using the o-dot operator, $\odot$. The purpose of the input, forget and output gating units is to control the access to the hidden and memory state. They are represented as

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{9}$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{10}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{11}$$

where σ is the recurrent activation, which can also be modified depending on the input sequence. The representation of the single cell is illustrated in Fig. 9 for easy comparison with the GRU approach in Fig. 10.

The Gated Recurrent Unit (GRU) represented in Fig. 10 was proposed in 2014 by Cho et al. in [27] to make each recurrent unit adaptively capture the dependencies of different time scales, which suggests that it may perform well for jitter modelling, especially when its characteristics change on different time scales due to changing background congestion. It is one of the most recently contributed gated RNNs that addresses the common issues of the vanishing and exploding gradient behaviour. Similar to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, it does not have separate memory cells [28]. This allows the network to only have two gates: the update gate, z_t, that decides how much the unit updates its activation or content, and the reset gate, r_t, which makes the unit act as if it was reading the first symbol of an input sequence, allowing it to forget the previously computed state. The GRU model can be represented in the form of

$$h_t = (1 - z) \odot h_{t-1} + z_t \odot \hat{h}_t, \tag{12}$$
$$\hat{h}_t = f(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h). \tag{13}$$

Its two gates are presented as

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z), \tag{14}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r). \tag{15}$$

One of the benefits the LSTM and GRU architectures hold over the traditional RNN, is the presence of the forget gate (LSTM) or the update gate (GRU). Traditionally, the content of the unit is replaced by the new value computed from the current input and the previous hidden state. This limits its ability to learn specific features of a sequence over long series of steps. Because we are modelling large sample sequences, we do not consider the traditional RNN to be a viable approach for jitter modelling. On the contrary, both the GRU and the LSTM keep the existing content and add the new content on top of it. In addition, the increase of the number of non-linearities used in these models, and our ability to modify them, reduces vanishing gradients. This is important for jitter modelling, because these non-linearities are what enables these DL networks to accurately represent sequences such as the those present in jitter time series.

Although, it is easy to distinguish between the advantages of the newer architectures, such as LSTMs and GRUs, over the traditional RNN qualitatively, it is more difficult to perform a qualitative comparison of the most recent learning approaches. A quantitative comparison was conducted in [29], to observe the benefits of the LSTMs and GRUs. The authors concluded that the GRU was faster than the LSTM, however, the accuracy of the GRU started to diminish compared to the LSTM for larger time series. This can be explained by the structure of the gating units. The control of the amount of new memory content that is added to the memory cell for the LSTM is done by two gating units that are independent to each other such as the forget and the input gate. On the other hand, the GRU does not control the amount of new memory content added to the memory cell with independent gates but rather a single gating unit such as the reset gate which controls the information flow from previous activation to the new one. In addition, the LSTM output gate controls the amount of memory content that is seen by other units in the network, whereas the GRU exposes its full content without any control. Consequently, these differences allow the GRU to achieve faster training times because of the lower number of multiplications, however, because of the better memory content control, LSTMs can model larger time series. In the context of jitter prediction, we did not observe any reduction in performance accuracy by using the GRU instead of the LSTM. We did, however, observe a speed up in training times using the GRU.

In summary, given that the GRU could be trained faster with higher accuracy for the jitter time series, we explored the GRU architecture for modelling jitter. The speed up can be explained by the simpler structure of the GRU, which has one less gate, which results in a lower number of matrix multiplications.

B. OPTIMISATION LITERATURE

We review the parameter optimisation literature because parameter optimisation impacts the practical aspects of training models to predict jitter. In addition we review the literature that motivates the choice of the correct activation function.

1) PARAMETER OPTIMISATION

DL optimisation algorithms attempt to minimise the value of the cost function between the output of the model and a set of given targets by updating the model parameters, such as the weights, thus enabling learning. However, this procedure usually requires computations, which have a large computational complexity. As the networks become deeper, conventional Stochastic Gradient Descent (SGD) optimisation objective functions exhibit non-convexity. These objectives potentially contain multiple local minima, critical points, and saddle points [30]. Inputs to each layer are affected by the preceding layers, causing any small change in parameters to be significantly amplified in deeper layers. Deciding on an appropriate optimisation algorithm can be a difficult task as they are often used as part of a black-box approach. Ruder et al. [31] explains the various stochastic descent optimisation methods and proposes that the Adaptive Moment Estimation (ADAM) algorithm [32] can be used as a first choice if one cannot decide on what algorithm to use. We found empirically, that for the jitter time series, the NADAM (Nestrov-accelerated Adaptive Moment Estimation) optimisation method performed better. The NADAM algorithm enables faster convergence than ADAM as it uses the Nesterov Accelerated Gradient (NAG) algorithm [33] to take steps, which are more accurate, in the direction that reduces the error. It does so by updating the parameters with a modified momentum step before computing the gradient.

2) WEIGHT INITIALISATION AND ACTIVATION FUNCTIONS

Proper weight initialisation for DL networks, especially the recurrent type, is crucial for its convergence. In early deep networks the weights of each layer were initialised in an independent and identically distributed way, using a Gaussian distribution [34]. Xavier et al. [35] discusses how the Sigmoid activation function

$$\text{sigmoid}(x) = (1 + e^{-x})^{-1}, \tag{16}$$

which is applied here to the variable x, is not suitable for feed-forward neural networks with random initialisation, as it leads to neuron saturation that damages both the information capacity and its learning capabilities. This phenomenon refers to a state during training, where a neuron output value is close to the value of the asymptotic ends of the bounded activation function [36]. This is where proper weight initialisation has the largest effect. Xavier discusses how random initialisation during backward propagation, especially for deep networks, leads to divergence in the gradients resulting in larger gradients in the lower layers and smaller gradients in the larger layers.

Recall that the jitter time series model is composed of a periodic sequence of decaying exponentials. The derivative of an exponential is an exponential. This is a concern for training both the LSTM and the GRU on jitter time series because the change in the weight values depend on the gradients, thus if the gradients in the lower layers are much larger, the growth of the weights will be correlated resulting in neuron saturation which will cause a binary output from the specific neurons and distort learning.

Xavier proposes a method that causes the back-propagated gradients to be dependent on each individual layer, which reduces the variance between individual layer weights and gives rise to the Hyperbolic Tangent

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{17}$$

which is a more suitable alternative to use with the normalised distribution. This approach is called Xavier initialisation. However, this method's derivations are based on the assumption that the activation functions are linear, which is not the case for the Rectified Linear Unit activation,

$$\mathrm{ReLU}(x) = \max(0, x). \tag{18}$$

In 2015, He et al. [37] discussed how Xavier initialisation was less suitable for the ReLU activation and showed how his method, which promises to take into account the non-linearity of the ReLU, would have a higher preference for deeper networks.

Alleviating the vanishing gradient problem has made the ReLU activation one of the most popular functions for deep neural networks. Despite its success, the ReLU activation has a non-zero mean that introduces a bias into the consecutive layers. If the units do not cancel each other out, learning causes a bias shift for units in the next layer. The bias shift is larger if the units are correlated, which slows down the training procedure. Furthermore, the ReLU does not take into account the statistical nature of the input time series, which in the case of jitter is typically a times series which is composed of periodic, decaying exponentials. The consequence of this is a problem called Internal Covariate Shift [38]. This phenomenon refers to the saturating affect the change in distribution of the inputs has on the distribution of the non-linear inputs. These saturating effects are amplified according to the network deepness which leads to poor learning. Batch normalisation can be used to significantly decrease these effects by normalising the distribution of the non-linearities relative to the input time series distribution. This allows the DL network to adapt its parameters to the distribution of the input, consequently reducing saturation. Layer normalisation also addresses the covariate shift problem, however, it does so by fixing the mean and the variance of the summed inputs within each layer, rather than the distribution of the time series. Batch normalisation was found to reduce the training time of deep neural networks in [39] and layer normalisation for RNs by Ba et al. in [40]. The work conducted by Ba et al. used the RN to predict text for Natural Language Processing (NLP). We found that, in fact, these methods did reduce the training time of the LSTM. However, the consequence of this reduction was a decreased prediction accuracy. These contrasting results to Ba et al. can be attributed to the encoding methods employed by NLP, where each sequence represented by binary values of 0 and 1. Due to the reduced sensitivity of these methods to the adverse effects of normalization, we do not regard the ReLU activation function as a suitable choice for the observed jitter time series presented in this paper. A promising new activation function was introduced by Clevert et al. [41] which has a zero mean, and is named the Exponential Linear Unit

$$\mathrm{ELU}(x) = \begin{cases} \alpha(e^{x}\text{-}1), & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases}, \text{ where } \alpha > 0. \tag{19}$$

This activation contains negative values, which allows the network to push the mean activation closer to zero which can be seen in Fig. 11. This speeds up learning by reducing the difference between the normal gradient and the natural gradient. Additionally, Self-Normalizing neural networks were introduced in 2017, they are based on the Scaled Exponential Linear Units,

$$\mathrm{selu}(x) = \lambda \begin{cases} \alpha\,(e^{x}\text{-}\alpha), & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases}, \tag{20}$$

which induce self-normalizing properties such as variance stabilisation which avoids exploding and vanishing gradients [42]. It is advised to use the Lecun initialisation, and $\lambda > 1$ to ensure a slope larger than one for positive net input. This method attempts to replace previous normalisation methods such as Batch and Layer normalisation by pushing the neuron activations to zero mean and unit variance.

In 2017, Ramachandran et al. conducted a study in [43] which concluded that the Swish activation could consistently improve the accuracy of conventional DL networks such as *ImageNet* and *Cifar*. They compared Softplus, ELU, SELU, GELU, RELU, LReLU, PReLU and Swish,

$$\mathrm{swish}(x) = x\ \mathrm{sigmoid}(\beta x), \tag{21}$$

activations and found that the Swish activation function outperformed previously discussed activation functions. They suggested that this improvement could be due to the non-monotonic "bump" when $x < 0$, that can be seen in Fig. 11. They showed that the distribution of the pre-activations falls inside the domain of the bump, $-5 \leq x \leq 0$, for when $\beta = 1$. The selection of activation functions depends significantly on the type of time series used. We take an empirical approach to determine the appropriate activation for jitter, and start this process by implementing the recently contributed functions.

VI. ANALYSIS OF SEQUENTIAL LEARNING FOR JITTER

We examine how sequential learning can be used to predict jitter. We adopt the following analysis strategy. As a first step, given that jitter can be modelled using a periodic exponentially decaying time series, our investigation is driven by an empirical analysis on how sequential learning algorithms can be fit to a jitter time series which is constructed of a single

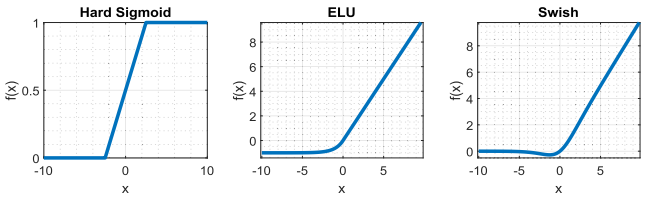

FIGURE 11. Activation functions that were considered for the parameter optimisation procedure: Hard Sigmoid, ELU and Swish.

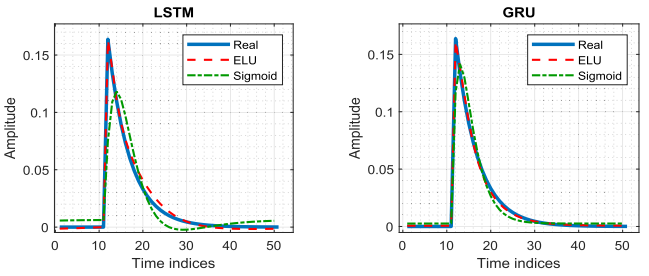

FIGURE 12. Comparison of the prediction accuracy achieved by the LSTM and GRU architectures for one synthetically created exponential. We list the Root Mean Square Error (RMSE) of the solution in brackets after the learning-activation function pair: LSTM ELU (0.16717), LSTM Sigmoid (0.03426), GRU ELU (0.01721), GRU Sigmoid (0.09270).

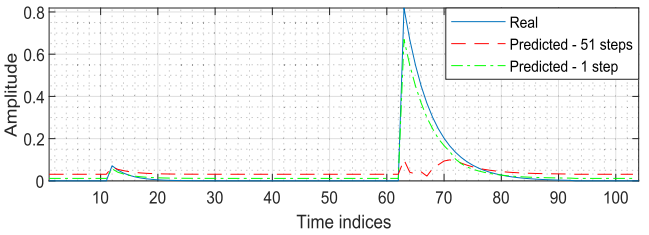

FIGURE 13. Comparing 1 and 51 steps ahead prediction accuracy of two consecutive exponentials that contain different values for the base, rate of decay and period.

exponential function. An important part of this analysis is to evaluate how effective different activation functions are as part of a prediction algorithm and to determine if the LSTM or the GRU architecture performs better. We then construct increasingly more complex time series in order to analyse when the learning architecture fails, before fitting parameters to a real-world jitter time series. In the second step, we consider a two-parameter synthetic test times series, which consists of the parameters b and λ, and in the third step of this evaluation, we consider a three-parameter test time series, which consists of the parameters, b, λ and p. We then determine if it is possible to fit the algorithm to the jitter time series and show what parameter modifications can be made to achieve the best performance. Finally, we determine how to fit and predict the features b, λ and p, to predict the jitter time series by reconstructing the future time series values.

A. RNs FOR A SINGLE DECAYING EXPONENTIAL

To begin the training procedure, a time series which has known features is used to determine if the RN can reproduce it. The reasoning behind this is that it is not always known what and how these statistical models learn. Therefore, tuning models on a simple time series, that has some of the features of the jitter time series we want to model will help to determine if the learning algorithm can be fit to this type of time series. This approach also allows us to determine which hyper-parameters have the largest impact on model accuracy. A synthetic time series that resembles the jitter time series which was observed on the live network, was created. It consisted of the decaying exponential in Fig. 12. Both of the RNs were trained on a $t = 200$-sample time series which contained four $t = 50$-sample exponentials. The configuration consisted of $T_x = 50$-sample input and $T_y = 50$-sample prediction. Initially, due to the constraints of the activation functions, the networks could not capture the base or the correct decay of the exponential. We modified the recurrent activation from Hard Sigmoid to Exponential Linear Unit and the non-linearity activation to Swish. These functions are illustrated in Fig. 11.

The results in Fig. 12 show the improvements that were made by changing the activation functions. We found that the recurrent activation had a much larger impact on capturing the base of the exponential than the non-linear activation. Additionally, if a conventional activation, such as the ReLU was used, we began to observe vanishing gradients. The resulting time series was inconsistent with what we expected; it contained abnormalities such as zero-value outputs. We modified the other parameters such as the learning rate over the range, $10^{-5} \leq l_r \leq 10^{-1}$, and the loss function to be one of the following functions: Mean Squared Error (MSE), Mean Squared Logarithmic Error (MSLE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Huber Loss, Cosine Similarity or Log Cosh loss. We varied the number of recurrent layers over the range, $1 \leq l_n \leq 10$, the number of dense layers over the range, $1 \leq l_d \leq 8$, and number of cells over the range, $1 \leq l_c \leq 1024$. The results showed that for this simple time series, these parameters did not have a significant impact on the accuracy of the fit. Finally, we compared the LSTM and the GRU architectures and determined that for this application the GRU was more accurate. Therefore, the experiments that we include in the remainder of this paper were conducted using the GRU architecture.

B. RNs FOR SEQUENCES OF DECAYING EXPONENTIALS

Having trained the RNs on one decaying exponential, we considered the problem of training the RNs on two consecutive exponentials, e.g. time series that were generated using three different parameters, b, λ and p. These time series posed greater challenges for the RNs. Fig. 13 illustrates a sequence of two exponentials which were generated using different parameters, e.g. values of b, λ and p. To demonstrate the challenge, we trained the GRU on two consecutive exponentials which had a different base, rate of decay and periodicity. The experiment was conducted with $T_x = 102$ step input and $T_y = 51$ step prediction. Fig. 13 illustrates the inaccurate predictions obtained. Prediction

Algorithm 1 ASAPjitter: Prediction Model

1: **Input** : Split training and testing sequences of x_n, using a window size T, with input size T_x, and output size T_y.
2: **Parameters** : Number of recurrent, l_n, and dense layers l_d. Number of cells, l_c in the recurrent layers and the number of neurons in the dense layers.
3: Learning rate, recurrent activation, non-linear activation, optimisation, initialisation, epoch, batch, and the loss parameters are selected.
4: **Function** *TrainRN*(T_x, T_y)
5: *return* : *predictionModel*
6: **Function** *Predict*(*predictionModel*)
7: *return* : *predictions*

Algorithm 2 ASAPjitter: Classification Model

1: **Input** : Split *predictions* into training and testing.
2: **Parameters** : Number of parallel, depth, dense, and the number of neurons in the dense layers. Select number of filters, the kernel size, convolution layers, pooling size and dropout.
3: Learning rate, non-linear activation, optimisation, initialisation, epoch, batch, and the loss parameters are selected.
4: **Function** *TrainCNN*(*predictions*)
5: *return* : *ASAPjitter*
6: **Function** *Classify*(*ASAPjitter*)
7: *return* : *congestionState*

Algorithm 3 FEATjitter: Classification Model

1: **Input** : Estimate b, λ and p, parameters from jitter x[n]. Split into training, T_x, and testing, T_s, with one hot encoded targets, T_y.
2: **Parameters** : Number of dense layers, and the number of neurons in the dense layers.
3: Learning rate, non-linear activation, optimisation, initialisation, epoch, batch, and the loss parameters are selected.
4: **Function** *TrainNN*(T_x, T_y)
5: *return* : *FEATjitter*
6: **Function** *Classify*(*FEATjitter*)
7: *return* : *congestionState*

quality was improved if the prediction was changed to be 1 step ahead. In the test-bed scenario described in this paper, this corresponded to being able to predict jitter values 1 second in the future. The reason for the inaccuracies when we increased the prediction step to be greater than 1, is that the RN was unable to model the large jumps in the base value of the exponentials, especially when a sequence contained multiple jumps. The auto-regression based gates performed better when they were modelling time series with slopes that were shallower than the exponentials observed in jitter times series.

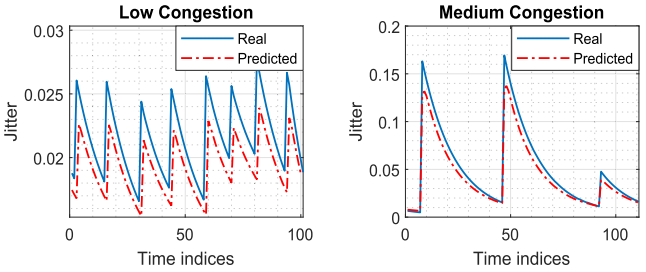

FIGURE 14. Low and medium congestion prediction plot of $t = 40$ samples (in the future) when the models are trained on sequences in the range of $0 \leq s_t \leq 20$ samples. Obtaining a good RMSE is not always a good indicator of performance. For example, the RMSE obtained for the entire 11514 samples (selected low and medium samples are plotted above) is 29.9. This is a good RMSE but the predictions are not accurate for the following reasons. Predictions during low congestion have an offset, whereas, predictions during medium congestion cannot fully predict the base value.

VII. RESULTS: RN ALGORITHMS FOR JITTER

Armed with the insights obtained from applying RNs to successively more complex synthetic time series, we then trained the RNs on the jitter time series which were observed in the network in Fig. 1. We increased the complexity of the RN by adding a larger number of cells, recurrent layers and dense layers. We started by examining the performance of the ELU recurrent activation, and then modified the function used. The non-linearity activation selected was the Swish function. We decreased the learning rate to $l_r = 10^{-4}$. We used the Huber loss function and the NADAM optimiser for all experiments. Samples were selected in the range $0 \leq n \leq 300$ from each congestion levels and we fit the RN separately to determine if it was possible to fit the model to the jitter time series and to obtain good performance.

One interesting point is that when the RN was fit to a the sequence in the range, $0 \leq s_t \leq 20$, from the low congestion levels, the trained model produced a relatively good prediction of the period for the rest of the jitter time series (that had not been seen by the network). The accuracy achieved can be determined visually from Fig. 14. This performance may be explained by the fact that the network is learning a temporal dependency at a very early stage of the jitter time series, which remains the same through-out each of the congestion levels. Although, if we examine the full prediction, we can see that the output does not capture the gradients in the time series, so even though we are estimating the next period, the benefit of the predictions are limited. For this reason, we modified the RN so it could learn the exponential decay in the jitter time series. We found that using the Huber loss function improved performance. We hypothesize that this could be due to it being less sensitive to outliers.

To complete our study of the jitter time series, we fit the network on a longer jitter time series, which comprised of 11514 samples. We found that during medium and high congestion levels, due to the large energy of the sequences that contained large base value exponentials, the network exhibited exploding gradients. To investigate this, we summed the values

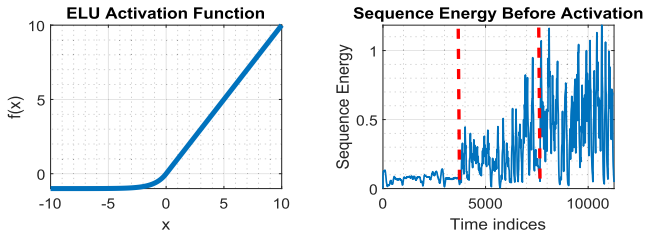

FIGURE 15. ELU activation function (LHS) and sequence energy (RHS) which shows when using the ELU activation the sequence energy remains the same before and after the activation function. The low congestion indices are between $1 < n < 3939$, medium congestion is between $3939 < n < 7677$, and high congestion is between $7678 < n < 11514$, where n is the measurement index.

of each input sequence, x_t, and called this the sequence energy, and defined it as

$$s_e(i) = \sum_{i=t}^{t+T_x-1} x_t(i). \tag{22}$$

In Fig. 15, it can be seen that the energy of the medium and high congestion sequences were much larger than in the low congestion states. Furthermore, when these large sequences were fed through the activation functions they caused exploding gradients which were inherited by the predictions, and appeared as large pulses. The experiments that are presented here were performed on time series that were normalised so that they had a range of 0 to 0.5. Further investigation on the effect of normalization indicates that it have a positive effect on reducing the exploding gradient problem, however, this may come at the cost of increased training time. To counter the affect of exploding gradients, we reverted back to the hard Sigmoid recurrent activation function, as it has boundaries which clip the high energy sequences. Consequently, the normalisation ratio should be modified accordingly so there is no loss of information.

A. ASAPjitter: SEQUENTIAL LEARNING FOR MODELLING GRADIENTS

By modifying these parameters, we were able to fit the RN to the full jitter time series, however, because of the unpredictability of the base value we could not satisfactory predict the jitter exponential before it occurred. This can be seen in Fig. 16, where the prediction was approximately linear until the exponential occurred. Then the RN attempted to predict the decay. This suggests that most of the predictions are unreliable at best, and not useful at worst, when it comes to detecting congestion changes. An additional recommendation is that we should only model the jitter time series when the exponentials decay, e.g. after the start the start of an exponential, in the jitter time series rather than the full time series, which is composed of decaying exponentials and sudden jumps due to the start of a new exponential or a change in the congestion level. Furthermore, when we trained the classifier on every prediction the classifier exhibited a lack of sensitivity to the true class and incorrectly classified ranges of

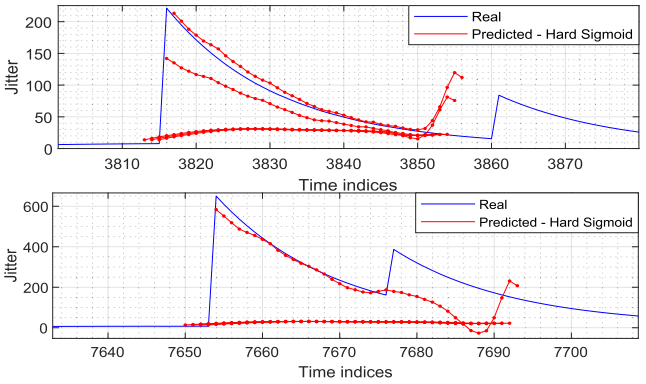

FIGURE 16. Predicting the exponential using the hard Sigmoid activation function, for low to medium congestion change (top), and medium to high (bottom). The predictions, p_t show that the GRU can predict the rate of decay once the initial base value is captured by the GRU, we calculate the total RMSE as 43.6.

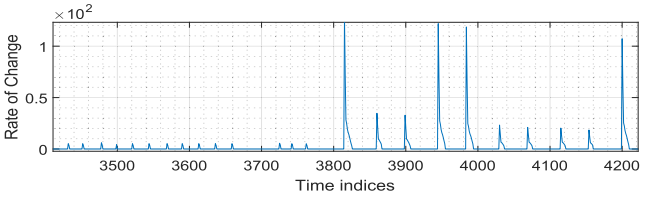

FIGURE 17. The rate of change of jitter which is used to detect when the exponentials occur. The pulses correspond to the high values in the base parameter.

jitter to the wrong class. This can be explained by the linear predictions used during each of the congestion states. It means that the only relevant points for acceptable predictions are the first couple of samples of the exponential. Therefore, we introduced a pre-processing step, which detected each sequence that contained the first value of an exponential, and we then segmented those sequences for training.

To detect these sequences, we calculated the Rate Of Change (ROC) in the jitter time series. A time series for this new metric is illustrated in Fig. 17. To calculate the ROC, we took the difference between the current sample, p_t, and previous sample, p_{t-1},

$$\text{ROC}(t) = (p_t - p_{t-1}). \tag{23}$$

The results show that the ROC for the low congestion state had a much lower amplitude than the two other states. There was a significant difference in amplitude between the ROC in the low and then the medium and high congestion states. This was due to the difference in the base values in these states. When these changes occurred, we could detect the start of the exponential, which contained information about the congestion state and we used that sequence for training and prediction.

This preprocessing step has several benefits. The first benefit is that all the irrelevant sequences have been removed, which allows the model to be trained on many more exponentials without over-fitting the model. The second benefit is that by removing all of the linear predictions in each of the congestion states we accommodated the

TABLE 2. GRU parameters.

Parameter	$ASAPjitter\ b$	λ	p	
Number of Cells	256	256	256	256
Number of Layers	2	2	2	2
Dense Layers	3	3	3	3
Dense Cells	2000	2000	2000	2000
Learning Rate	1.0^{-5}	1.0^{-5}	1.0^{-5}	1.0^{-5}
Optimiser	NADAM	NADAM	NADAM	NADAM
Batch size	256	256	256	256
Epoch	50	120	100	100
Kernel Initialiser	He Norm.	He Norm.	He Norm.	He Norm.
Recurrent Activation	Hard Sig.	Swish	Swish	Swish
Dense Activation	Swish	Swish	Swish	Swish
Regression Loss	Huber	Huber	Huber	Huber
Data Normalised	$(0, 0.5)$	$(0, 1)$	$(0, 1)$	$(0, 1)$
RMSE	43.6	0.4359	0.0299	9.8834

TABLE 3. Classifier parameters.

Parameter	$ASAPjitter\ CNN$		$FEATjitter$
Parallel Layers	3	2	
Depth Layers	1	1	
Dense Layers	3	2	7
Dense Cells	4000	4000	256
Filters	16	32	
Kernel Size	$5 \times 8 \times 10$	3×5	
Max Pool	2	2	
Learning Rate	1.0^{-4}	1.0^{-4}	1.0^{-3}
Optimiser	NADAM	NADAM	NADAM
Batch size	256	256	256
Epoch	200	80	150
Kernel Initialiser	He Normal	He Normal	
Dense Activation	Swish	Swish	Swish
CNN Activation	ReLU	ReLU	
Data Normalised	$(0, 1)$	$(0, 1)$	$(0, 1)$
Probabilistic-loss	Categorical-Crossentropy	Categorical-Crossentropy	Categorical-Crossentropy

classifier, because if we do not remove the linear predictions, the classifier will be fitted on the same linear predictions throughout each of the congestion states and will not be able to distinguish between them. To classify the predictions we trained a Convolutional Neural Network (CNN) as a classifier with the selected parameters that are illustrated in Table. 3. The CNN architecture was chosen as it modelled temporal dependencies similar to recurrent networks. The results show that the accuracy of this classification method was 100% for low congestion, 56% for medium congestion and 97.5% for high congestion, with only 1 sample of an exponential. We name this method as ASAPjitter. These results suggest that we can use this method for classification at a very early stage (1 sample of a new exponential).

B. FEATjitter: PREDICTION

To investigate if the forecast accuracy coud be improved, we trained the GRU on the estimated parameters b, λ and p. Once again, the input length of $T_x = 60$ and prediction length of $T_y = 40$ were selected for the GRU. This configuration was chosen because depending on the congestion levels, the period of the original time series varied between $20 \leq p[n] \leq 40$ samples, meaning that if the prediction length was $T_y = 40$ samples into the future, we coud estimate the next $1 \leq \omega_i \leq 2$ periods. To prepare the training data, we used the sliding window approach where we truncated the full time series into $0 \leq T \leq 100$ sample windows with each window shifted by one sample to the right.

The GRU parameter configuration can be seen below in Table. 2. To determine the accuracy we calculated the RMSE. In the results we can see that the calculated percentage error was low, which suggests a high prediction accuracy has been achieved. However, this is accuracy is due to overlapping linear prediction values throughout the time series. In fact, the predictions are ineffective because the estimated parameter trace contains sudden changes that are difficult to predict.

The reason for this is that the feature time series does not contain any recurring slopes that can be modelled, as a consequence, the algorithm models linear functions, so it cannot anticipate the sudden changes in the parameter time series, which are there because of the changes in the jitter parameters. Consequently, the benefit of this prediction method is limited because the changes in congestion are determined by the changes in the jitter parameters. If these parameters cannot be predicted before they occur, the changes in the time series will not be accurately predicted. We would like to highlight the point that a low error does not necessarily mean an effective prediction. Based on these results, we do not recommend the utilization of a prediction layer as it consumes computational resources without yielding substantial benefits.

C. FEATjitter: CLASSIFICATION

To conclude our investigation into sequential learning, we tested the classification abilities of the GRU architecture for the estimated parameters. We used the configurations presented in Table. 2. However, for classification we selected the input length to be $T_x = 100$. Using the jitter time series we were unable to accurately classify between the three states, which resulted in single state classification where we were only identifying high congestion. This is likely due to the low values in each congestion states which caused the linear predictions. However, by mapping the jitter metric into a feature domain we found that we could accommodate learning and increase the classification accuracy. Using a multivariate configuration on the b, λ and p parameters, we achieved a classification accuracy of 97% for low congestion, 20.4% for medium congestion and 94% for high congestion as seen in Table. 4. The challenge was classifying between medium and high congestion, as the GRU classified most of the medium congestion state belong to the high class. In addition, we found that adding a parallel CNN layer improved classification accuracy up to 82% for the medium congestion state. This suggests that for classification it would be better to use a CNN architecture rather than a recurrent one. The results for the multivariate CNN implementation can be seen in Table. 4 with the CNN parameters in Table. 3.

We recognise that in practise training a CNN requires a lot more computational resources than the traditional batch

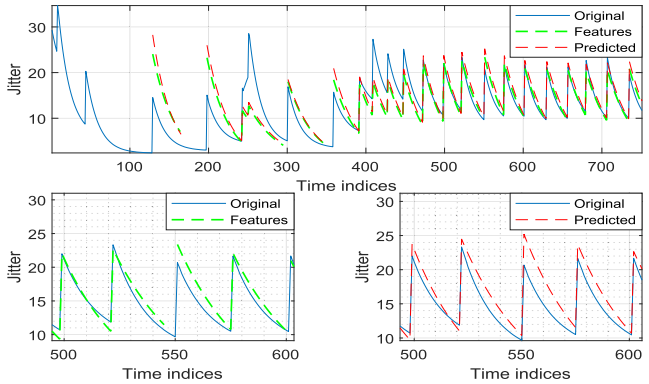

FIGURE 18. Reconstructed low congestion levels (top), reconstructed parameter (bottom LHS) and predicted reconstructed parameter (bottom RHS).

learning approaches such as the MLP. The classification accuracy achieved by the MLP was 100% for low congestion, 83% for medium congestion and 91.9% for high congestion. Therefore, we suggest the use of a MLP for classifying the b, λ and p parameters. We named the resulting approach FEATjitter. This result is achieved for the predicted and original b, λ and p parameters. Due to variability in the effectiveness of the prediction layer, we do not report results obtained using the prediction layer in the FEATjitter algorithm.

D. TIME SERIES RECONSTRUCTION

To evaluate the accuracy of the proposed (b, λ, p)-estimation method and to validate our choice of not including a prediction layer in FEATjitter, the original parameters (b, λ, p) and the predicted parameters were taken and used to reconstruct the jitter time series. Each constituent decaying exponential in the time series is defined using the following model,

$$\hat{x}[n] = be^{-n\lambda}, \text{ where } n \geq 0, \tag{24}$$

and the entire time series is reconstructed by adding delayed and truncated decaying exponentials according to the model in [7]. The results are plotted against the original signal in Fig. 18. The plots show that for the low congestion level, the reconstructed time series was accurate for both the predicted and the original parameters. Although, there was a slight difference in the base of the reconstructed time series for the FEATjitter predictions, the periodicity and rate of decay was reconstructed with high accuracy.

One point to note, is the importance of the transition period, that is, when congestion changed between states. In the original jitter time series, congestion state changes occurred at $n = 3939$ and $n = 7677$ in Fig. 2. We considered this congestion state change from low to medium congestion in Fig. 19. To predict congestion changes we need our model to accurately capture the point of transition. We can see in Fig. 19, that the transition was not captured by either of the methods. As mentioned in Section VII-B, the predicted features do not offer significant advantages over the original features, given that the FEATjitter model failed to anticipate abrupt changes before they occurred.

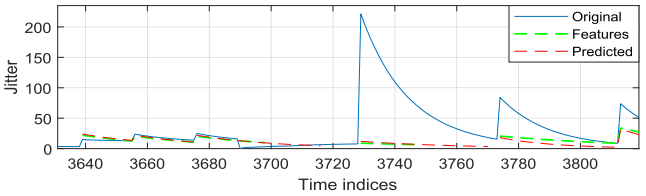

FIGURE 19. Transition from low to medium congestion, for the reconstructed time series using the original features and the predicted features; it can be seen that the transition period cannot be captured by either of the methods.

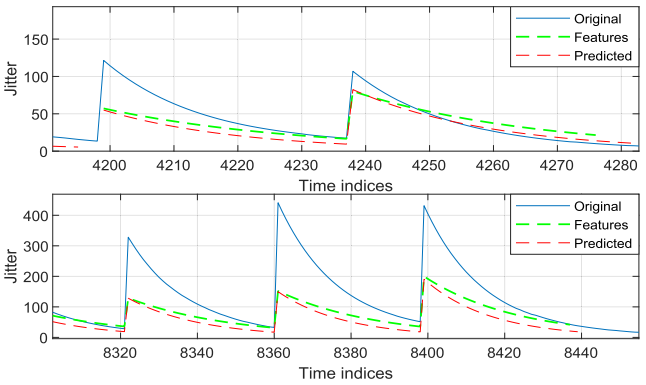

FIGURE 20. Reconstructed medium (top) and high (bottom) congestion levels from original and forecasted parameters.

TABLE 4. Classification accuracy for a GRU, FEATjitter and ASAPjitter methods.

Architecture	*Low*	*Med*	*High*	*Total*
GRU ($b\ \lambda\ p$)	97%	20.4%	94%	70.46%
ASAPjitter	100%	56%	97.5%	84.5%
FEATjitter	100%	83%	91.9%	**91.6%**
CNN ($b\ \lambda\ p$)	97%	82%	94%	91%
Random Forest [7]	98.3%	41.1%	95.8%	78.5%
Decission Trees [7]	96.8%	45.2%	95.1%	79%

Furthermore, we wanted to compare the reconstruction accuracy with medium-high congestion levels. Although, Fig. 20 demonstrates that the base value accuracy decreases for the higher exponentials, we can reproduce the rate of decay and periodicity with high accuracy. This suggests that the parameter estimation method cannot capture the base of the exponential during high congestion. Further improvement of the estimation method suggest that improvements in classification accuracy of the FEATjitter model might be possible.

VIII. CONCLUSION

In this paper we contribute sequential learning solutions for predicting and classifying network congestion states using the adaptive behaviour in jitter, that results from the use of the H.264 video codec. We demonstrate how difficult it is for sequential learning algorithms to classify congestion states when the jitter time series are composed of a series of delayed and truncated exponential functions. We propose that by mapping the jitter time series into a feature domain which contains estimates of the jitter features such as the base, b, the rate of decay, λ, and

period, p, can significantly improve the classification abilities of sequential learning algorithms. We explain how this feature domain representation is unsuitable for prediction, because it contains unpredictable jumps which are difficult to model using RNs.

We then show how the jitter time series can be processed so that they are more suited to prediction by modifying the parameters of the RN. We demonstrate that we can successfully predict the decay of the exponential in the time series. We proposed a rate of change metric to identify the useful segments of the sequences that will yield useful predictions and train a CNN architecture to classify them. By implementing this method, which we name ASAPjitter, we were able to classify congestion states by only using $n = 1$ sample of a new congestion state, with an accuracy of 100% for low congestion, 56% for medium congestion and 97.5% for high congestion. In addition, we proposed a method named FEATjitter, that used a MLP architecture to classify the three parameters: b, λ, and p. We provided evidence that we could detect congestion changes using FEATjitter with an accuracy of 100% for low congestion, 83% for medium congestion and 91.9% for high congestion.

We briefly summarize some of the potential advantages and disadvantages of the two approaches. First of all, running the parameter (b, λ, p)-estimation algorithm is computationally cheap compared to executing a RN, as it only requires several matrix multiplications. This requires the estimator to estimate the whole jitter time series and to produce the three parameters which are all of the same length as the original jitter time series. To train a classifier we need to train it on the three parameters, making it computationally expensive for sequential classifiers. Therefore, we suggest that batch learning approaches such as MLP would be more suitable for classifying the estimated parameters. In comparison, the DL approach segments the jitter so only relevant congestion information is used to train the classifier, which reduces the computational requirements making it an attractive approach. A second important comparison category is the speed which the classifier detects changes in congestion. The RN can detect changes with only one sample of the exponential. The estimation of the parameters only occurs at the end of each period which reduces the response time. We believe that because of these differences, these two algorithms could work in tandem. ASAPjitter could be used to detect any changes early, and FEATjitter could be used to rectify any inaccuracies in detecting medium congestion.

We hypothesise that a joint ASAPjitter-FEATjitter approach could be implemented as a universal algorithm for detecting congestion changes in video streaming. For example, different videos contain different jitter variations, and so if a universal algorithm was trained on a large time series which contained a large number of congestion changes, it would not need to be retrained for different videos. In future work, we will investigate if larger networks might perform better by implementing attention based RNs such as Transformers [44]. We posit that this joint ASAPjitter-FEATjitter approach will pave the way for self-aware networks, whereby self-measurement, and self-observation, together with on-line control mechanism, operate adaptively to attain the required performance and QoE [45].

ACKNOWLEDGMENT

This work supported by the Science Foundation Ireland (SFI) under Grant 15/SIRG/3459 and Grant 13/RC/2077_P2.

REFERENCES

[1] F. Tang, B. Mao, Z. M. Fadlullah, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 154–160, Feb. 2018.

[2] M. Usama, J. Qadir, A. Raza, H. Arif, K. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised machine learning for networking: Techniques, applications and research challenges," *IEEE Access*, vol. 7, pp. 65579–65615, 2019, doi: 10.1109/ACCESS.2019.2916648.

[3] A. Malik, R. de Fréin, and B. Aziz, "Rapid restoration techniques for software-defined networks," *Appl. Sci.*, vol. 10, no. 10, p. 3411, May 2020, doi: 10.3390/app10103411.

[4] R. de Fréin, "Source separation approach to video quality prediction in computer networks," *IEEE Commun. Lett.*, vol. 20, no. 7, pp. 1333–1336, Jul. 2016, doi: 10.1109/LCOMM.2016.2563418.

[5] M. T. Vega, C. Perra, and A. Liotta, "Resilience of video streaming services to network impairments," *IEEE Trans. Broadcast.*, vol. 64, no. 2, pp. 220–234, Jun. 2018.

[6] S. Zhang, W. Lei, W. Zhang, and Y. Guan, "Congestion control for RTP media: A comparison on simulated environment," in *Simulation Tools and Techniques*. Cham, Switzerland: Springer, 2019, pp. 43–52.

[7] R. de Frein, O. Izima, and A. Malik, "Detecting network state in the presence of varying levels of congestion," in *Proc. IEEE 31st Int. Workshop Mach. Learn. Signal Process. (MLSP)*, Oct. 2021, pp. 1–6, doi: 10.1109/MLSP52302.2021.9596271.

[8] R. de Frein, "Effect of system load on video service metrics," in *Proc. 26th Irish Signals Syst. Conf. (ISSC)*, Jun. 2015, pp. 1–6, doi: 10.1109/ISSC.2015.7163768.

[9] J. B. Madalgi and S. A. Kumar, "Development of wireless sensor network congestion detection classifier using support vector machine," in *Proc. 3rd Int. Conf. Comput. Syst. Inf. Technol. Sustain. Solutions (CSITSS)*, Dec. 2018, pp. 187–192, doi: 10.1109/CSITSS.2018.8768738.

[10] K. Han, J. Y. Lee, and B. C. Kim, "Machine-learning based loss discrimination algorithm for wireless TCP congestion control," in *Proc. Int. Conf. Electron., Inf., Commun. (ICEIC)*, Jan. 2019, pp. 1–2, doi: 10.23919/ELINFOCOM.2019.8706382.

[11] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: Data collection and traffic classification," in *Proc. IEEE 24th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2016, pp. 1–5.

[12] A. Malik, R. de Frein, M. Al-Zeyadi, and J. Andreu-Perez, "Intelligent SDN traffic classification using deep learning: Deep-SDN," in *Proc. 2nd Int. Conf. Comput. Commun. Internet (ICCCI)*, Jun. 2020, pp. 184–189, doi: 10.1109/ICCCI49374.2020.9145971.

[13] Z. Md. Fadlullah, F. Tang, B. Mao, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2432–2455, 4th Quart., 2017, doi: 10.1109/COMST.2017.2707140.

[14] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, Vancouver, BC, Canada, May 2013, pp. 8624–8628, doi: 10.1109/ICASSP.2013.6639349.

[15] D. Raca, A. H. Zahran, C. J. Sreenan, R. K. Sinha, E. Halepovic, R. Jana, and V. Gopalakrishnan, "On leveraging machine and deep learning for throughput prediction in cellular networks: Design, performance, and challenges," *IEEE Commun. Mag.*, vol. 58, no. 3, pp. 11–17, Mar. 2020, doi: 10.1109/MCOM.001.1900394.

[16] H. Lu and F. Yang, "A network traffic prediction model based on wavelet transformation and LSTM network," in *Proc. IEEE 9th Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Beijing, China, Nov. 2018, pp. 1–4, doi: 10.1109/ICSESS.2018.8663884.

[17] X. Liang, Q. Tian, F. Wang, W. Yu, and X. Xin, "A dynamic resource allocation based on network traffic prediction for sliced passive optical network," in *Proc. 19th Int. Conf. Opt. Commun. Netw. (ICOCN)*, Qufu, China, Aug. 2021, pp. 1–3, doi: 10.1109/ICOCN53177.2021.9563790.

[18] *Bitmovin's 4th Annual Video Developer Report*. Accessed: Sep. 19, 2022. [Online]. Available: https://go.bitmovin.com/video-developer-report-2020

[19] N. H. Gauthier and M. I. Husain, "Dynamic security analysis of zoom, Google meet and Microsoft teams," in *Proc. Silicon Valley Cybersecurity Conf.*, vol. 1383, Y. Park, D. Jadav, and T. Austin, Eds. Cham, Switzerland: Springer, 2021, pp. 3–24, doi: 10.1007/978-3-030-72725-3_1.

[20] D. Thorp-Lancaster, "Microsoft Teams hits 75 million daily active users, up from 44 million in March," Windows Central, Tech. Rep., 2020. [Online]. Available: https://www.windowscentral.com/microsoft-teams-hits-75-million-daily-active-users

[21] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1975.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Micro-structure of Cognition: Foundations*. Cambridge, MA, USA: MIT Press, 1987, pp. 318–362.

[23] S. Elsworth and S. Guttel, "Time series forecasting using LSTM networks: A symbolic approach," 2020, *arXiv:2003.05672*.

[24] R. C. Staudemeyer and E. R. Morris, "Understanding LSTM—A tutorial into long short-term memory recurrent neural networks," 2019, *arXiv:1909.09586*.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[26] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (GRU) neural networks," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Boston, MA, USA, Aug. 2017, pp. 1597–1600, doi: 10.1109/MWSCAS.2017.8053243.

[27] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proc. SSST-8, 8th Workshop Syntax, Semantics Struct. Stat. Transl.*, Doha, Qatar, 2014, pp. 103–111, doi: 10.3115/v1/W14-4012.

[28] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *Proc. NIPS 2014 Workshop Deep Learn.*, Dec. 2014, pp. 1–9.

[29] S. Yang, X. Yu, and Y. Zhou, "LSTM and GRU neural network performance comparison study: Taking yelp review dataset as an example," in *Proc. Int. Workshop Electron. Commun. Artif. Intell. (IWECAI)*, Shanghai, China, Jun. 2020, pp. 98–101, doi: 10.1109/IWECAI50956.2020.00027.

[30] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287, 3rd Quart., 2019, doi: 10.1109/COMST.2019.2904897.

[31] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

[32] D. P. Kingma and J Ba, "ADAM: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–13.

[33] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence $O(\frac{1}{k^2})$," in *Proc. Dokl. Akad. Nauk. SSSR*, vol. 269, 1983, pp. 543–547.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[35] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.

[36] A. Rakitianskaia and A. Engelbrecht, "Measuring saturation in neural networks," in *Proc. IEEE Symp. Ser. Comput. Intell.*, Dec. 2015, pp. 1423–1430, doi: 10.1109/SSCI.2015.202.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[38] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. 32nd Int. Conf. Mach. Learn.*, Lille, France, vol. 37, Jul. 2015, pp. 448–456. [Online]. Available: https://proceedings.mlr.press/v37/ioffe15.html

[39] M. Hasani and H. Khotanlou, "An empirical study on position of the batch normalization layer in convolutional neural networks," in *Proc. 5th Iranian Conf. Signal Process. Intell. Syst. (ICSPIS)*, Shahrood, Iran, Dec. 2019, pp. 1–4, doi: 10.1109/ICSPIS48872.2019.9066113.

[40] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," in *Proc. NIPS*, 2016, pp. 1–14.

[41] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," 2015, *arXiv:1511.07289*.

[42] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–10.

[43] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017, *arXiv:1710.05941*.

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[45] E. Gelenbe, J. Domanska, P. Fröhlich, M. P. Nowak, and S. Nowak, "Self-aware networks that optimize security, QoS, and energy," *Proc. IEEE*, vol. 108, no. 7, pp. 1150–1167, Jul. 2020, doi: 10.1109/JPROC.2020.2992559.

TAUTVYDAS LISAS received the B.E. degree in electrical and electronic engineering from Technological University Dublin, Ireland, in 2021, where he is currently pursuing the Ph.D. degree with the School of Electrical and Electronic Engineering. His area of research is prediction algorithms for video quality-of-delivery metrics for network management. His research interests include machine learning, deep learning, quantum machine learning, and signal processing.

RUAIRÍ DE FRÉIN received the B.E. degree in electronic engineering and the Ph.D. degree in time-frequency analysis and matrix factorization from University College Dublin (UCD), Ireland, in 2004 and 2010, respectively. He is a CONNECT Funded Investigator and a Lecturer with the School of Electrical and Electronic Engineering, Technological University Dublin, Ireland. He held Marie Sklodowska-Curie Fellowships with the KTH Royal Institute of Technology, Stockholm, and Amadeus SAS, Sophia Antipolis, France. Over the past few years, he has developed algorithms for predicting quality-of-delivery metrics for network management and monitoring strategies for small cell networks and monitoring techniques for internet protocol television (IPTV). His research interests include machine learning for integrating microgrids, sparse signal processing, software-defined networks, and VANETs.