UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2021

Title: Deep Learning for Network Traffic Prediction

Author: Antony Fleischer

Project Abbreviation: DL4NTP

Supervisor(s): Dr Josiah Chavula

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	0
Theoretical Analysis	*0*	*25*	5
Experiment Design and Execution	*0*	*20*	20
System Development and Implementation	*0*	*20*	0
Results, Findings and Conclusions	*10*	*20*	20
Aim Formulation and Background Work	*10*	*15*	15
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	

Deep Learning For Network Traffic Prediction

Using LSTMs as Network Traffic Predictors on SANREN

Antony Fleischer
flsant005@myuct.ac.za
University of Cape Town
Cape Town, South Africa

Abstract

Time series forecasting is a method that uses models fitted on historical data to predict future values of an observation. Network traffic prediction is a form of time series forecasting that allows computer network operators to manage their networks more efficiently. Accurate traffic prediction can improve a network's performance significantly, in areas such as network congestion management, resource distribution and network volume alerts. Most neural network models find it difficult to learn the long-range temporal relationships in a dataset - primarily due to the non-stationary and non-linear qualities that time-series data presents. However, existing literature suggests that Long Short Term Memory (LSTM) models can capture the long and short-term trends in network traffic data. This paper implements three LSTM models for network traffic prediction on the SANReN. Over a range of grid-searched hyperparameters, a stacked LSTM model is the most accurate model in general. However, it is also the most computationally expensive, and the network traffic prediction and resource requirements of a network operator may warrant the implementation of a simple LSTM model.

CCS Concepts: • **Computing methodologies** → **Neural networks**; **Supervised learning by regression**; • **Networks** → *Network performance modeling*.

1 Introduction

Modern computer networks are facing the challenges associated with transferring unprecedented amounts of data. Furthermore, large, confederated computer networks exhibit volatile network traffic flows [27]. Historically, network operators have attempted to use traditional statistical learning methods [22, 25] to optimise their networks. However, as network flows increase dramatically [2], network operators require more efficient and accurate time-series processing methods to overcome network congestion, reduce access times, allocate bandwidth effectively and detect traffic anomalies [16, 18, 22].

Network traffic prediction is relevant for two temporal forecasts: short-term forecasts assist with dynamic resource allocation, while longer-term forecasts capture the network traffic trend more generally, which provides a network operator with an understanding of how to upgrade their network [22]. An accurate and computationally appropriate network traffic prediction model is valuable in two senses. Firstly, resources can be managed through the identification of high traffic volumes, congestion, and burst flows [30] and secondly, a computationally appropriate model - in that it is not overly computationally expensive for the resources of the network it is examining - will be able to provide these services to any network size [10].

There are a few reasons why network traffic prediction requires neural network predictors. Primarily, any prediction model depends on the statistical distribution of a data set [25]. It also depends on the notion that the data is from a time-series dataset [25]. The difficulty in network traffic prediction stems from the statistical nature of the data. Network traffic flows are characterised by self-similarity and non-linearity [20] and are inefficiently modelled by Gaussian or Poisson distribution models [20]. This inefficiency introduces the need for LSTMs and is reinforced by previous literature which has evaluated the performance of traditional statistical learning models on network traffic data [22, 25].

Whereas previous researchers have used publicly available datasets such as the GEANT [15, 19, 29] and Abilene [25, 28] networks, this paper uses network traffic data from the South African National Research and Education Network (SANReN). SANReN is a country-wide network of education and research institutions in South Africa [1]. It is a high-speed network for science, education, research and innovation-based institutions, and has been phased in across the country since 2007 [1]. As a large, federated network, SANReN could benefit from an LSTM model that allows for preemptive network actions.

Existing literature shows LSTMs becoming popular models for approaching network traffic prediction, which is a subset of time-series forecasting. The literature also shows that LSTMs - as well as other deep learning models - outperform traditional statistical prediction and machine learning methods. This suggests that LSTMs are appropriate for implementation as network traffic predictors on SANReN.

This research paper will provide the context of LSTMs for the SANReN network use case. This paper investigates the data processing pipeline of an LSTM - from network traffic flow output to model input and present a preliminary data analysis. Additionally, this paper describes the implementation of a stacked LSTM model and evaluates its

computational cost and predictive accuracy against a simple and bidirectional LSTM across a range of hyperparameters. Penultimately, the process of model optimisation through a hyperparameter grid search will be explained, before the conclusions present an optimal model for the SANReN use case.

2 Background

2.1 Network Traffic Flows

Network traffic flows refer to the groups of bytes being sent – from a node to any other node - across a network [25]. Nodes in this context are any type of device that is connected to the network, not just routers [25]. Given this definition, an LSTM's accuracy is measured by its ability to predict the number of bytes that will be sent in a future time period, while its computational cost is measured by the time it takes to train the model on a dataset.

Simply put, time-series data is a sequence of observed data points - $(y_1, y_2, \ldots, y_t)$ - indexed by time order [31]. Usually, the data points are measured at successive, equidistant points in time. However, the time spacing of observation measurement is not uniform in this case. It is important in this study that network traffic flows are defined as time-series data. These constraints are necessary because they all allow the LSTM to learn short and long-term temporal trends. If the data was not treated as a time-series sequence, then the temporal indexing of the observations would be irrelevant, and the LSTM would be learning to predict the response based on non-temporal values. Additionally, it is important to identify that network traffic data is non-linear. Seasonality and trends are the main reasons that linear predictions models are inaccurate [27, 31], and so it is necessary to use LSTMs to capture the nonlinearity and non-stationarity that network traffic exhibits [27].

Both neural networks and traditional statistical models can use time-series data in different ways. For example, these predictive models can implement different timescales and distinctive forecasting horizons [10]. Predictions for hourly traffic volumes rather than every 5 minutes is an instance of an altered timescale, whereas an adjusted forecasting horizon would change how many time periods ahead a predictor forecasts for [10]. However, because of the non-uniformity of the SANReN data, the LSTM models in this study predict the next byte flow observation at time t_{i+1}. Additionally, network traffic data may be provided to a neural network to predict when a burst in traffic volume will occur, rather than to forecast a pattern in the future of the time-series data [21]. Burst flow prediction can be achieved in parallel with regular neural network prediction [21], and so it is also included in this study.

2.2 Deep Learning

2.2.1 Long Short Term Memory. Time series data are sequential patterns. Essentially, a new data observation is dependent on past values in the time series. The LSTM model is designed to capture these long-term temporal dependencies [25] and can be used to predict sequential patterns in the SANReN data. LSTM models are ideal for this task because of the structure of their memory units. Each unit gives the model the ability to consider its input, and then either keep existing memory or overwrite it with new information [17]. This capacity allows extremely long-term temporal dependencies to be captured by the model, whereas simpler models are unable to capture these trends [25]. Additionally, LSTMs have become the high performing standard for network traffic prediction [17, 18, 25, 29] and so it is sensible to implement a traditional LSTM for this project. Bidirectional and stacked LSTM architectures are more suited to dealing with long-term temporal patterns [7, 8] since they both include more layers than a traditional LSTM. Network traffic data may also be provided to a neural network to predict when a burst in traffic volume will occur, rather than to forecast a sequential pattern in the time-series data [21]. Burst traffic is defined as a prolonged, uninterrupted transfer of data from one device to another. When the sequential pattern of the data is non-linear, and burst traffic is also present, neural network models have shown to be 78% more accurate than traditional statistical prediction methods [29]. Therefore, this study uses a large sample of non-linear SANReN data with burst flow activity. The sample is taken over an entire week and contains 47000 data points.

The mechanics of the LSTM are also important to understand. The LSTM model is effective at capturing long-term temporal dependencies because it has been designed to solve the gradient disappearance or explosion problem [14]. An LSTM model can carry information over time intervals larger than 1000 steps, without diminishing the influence of short-term observations on a prediction [14]. The structure of each neuron - the memory units within the hidden layer [25] - include three gates, the forget gate f_t, the update - or input - gate i_t and the output gate o_t, as shown in Figure 1.

The update gate determines to what extent the unit values new input, the forget gate determines to what extent existing memory is removed, and the output gate determines how much memory is exposed to the next step. The unit's current state, c_t, is determined as a mathematical function of the previous activation function $h_(t-1)$, the previous cell state $c_(t-1)$, the input x_t , and each of the gates [25]. These gates give the memory unit the ability to consider its input, and then either keep existing memory or overwrite it with new information.

Other neural networks do not have this ability and will overwrite memory by default [25]. This singular difference allows LSTM to capture long-term dependencies that other

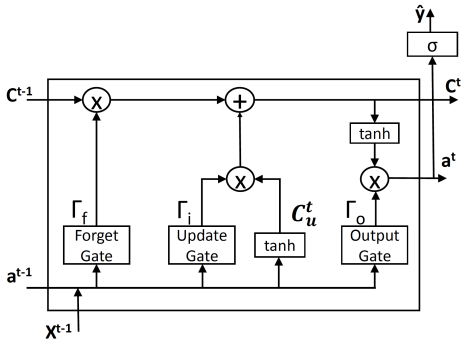

Figure 1. The memory unit of an LSTM [17]

neural networks may not [25]. As mentioned earlier, the LSTM is more accurate than other neural networks and some statistical learning methods on the well-known Abilene and GEANT datasets [25]. Crucially, the LSTM is never less accurate than memory-less neural networks [17, 18, 29], making it the recommended neural network to use for the network traffic prediction needs of SANReN.

3 Problem Statement

It is important to consider the constraints and resources of a network when evaluating a candidate model for network traffic prediction. Both computational complexity and run time can be a limiting factor for less-resourced networks such as SANReN, which may result in different network traffic prediction models being better suited for networks of this class. Existing literature on network traffic predictors has shown that neural networks - particularly Long Short Term Models (LSTM) - provide improvements in accuracy and performance over traditional statistical learning methods. Furthermore, LSTM derivative models, such as the stacked LSTM and bilateral LSTM, have out-performed baseline LSTMs [11, 19]. The computational feasibility of LSTM and LSTM-derivative prediction models will be investigated. These models will be replicated and trained on new SANReN data sets, to further assess their performance against traditional statistical models and conventional LSTM models. This project is a departure from previous work done in the field since this is primarily geared towards creating an optimal model for predicting traffic on the SANReN - whilst the second objective is to assess the effect of different hyperparameters on an LSTMs performance in general.

3.1 Research Question

1. How does the SANReN traffic data vary with time and day in relation to the South African university calendar?
2. Which of the LSTM architectures, baseline, bilateral or stacked, provides the highest accuracy when predicting future SANReN traffic data, subject to network constraints?
3. What is the computational cost of different LSTM architectures, given a required level of accuracy in predicting future traffic flows on SANReN?

4 Related Work

Network traffic prediction approaches have been formalized in past studies. Furthermore, the growth of the internet and its networks have accelerated research, with deep learning techniques emerging as the prevalent tool for network traffic prediction. Historically, researchers used statistical prediction techniques such as ARIMA and Holt-Winters models, but deep learning models - particularly the LSTM - have been shown to out-perform those. A Recurrent Neural Network can suffer from the vanishing gradient problem, which occurs when the network is unable to send back useful gradient information from the output layers to hidden layers. If this occurs, the neural network loses its ability to consider long term dependencies in calculations [6]. It is for this reason that Krishnaswamy et al. [19] propose that using a neural network is unsuitable for network traffic prediction, suggesting that an LSTM should be used for time-series predictions, to eliminate the vanishing gradient problem.

Krishnaswamy et al. [19] discuss the differences between using a simple and stacked LSTM learning approach. Stacked LSTMs were more accurate in predicting future traffic flows compared to traditional LSTM architectures, but at the cost of higher computational complexity as a result of added LSTM layers. There is a trade-off, but the choice between accuracy and computational efficiency allows network operators to decide what is more practical for their needs. In their training, Krishnaswamy et al. [19] noted that adding extra LSTM layers did not cause a noticeable change in accuracy. They observed that a baseline LSTM had the lowest Mean Squared Error overall. One limitation that will be investigated further in this project is whether these LSTM algorithms perform as well for smaller traffic volumes that one may see on an education network, as the results above were for links of a capacity of over 100GB/s.

Cui et al. [11] investigated the use of a bidirectional LSTM for forecasting network traffic. A bidirectional LSTM runs the input in two directions through the model: from both past to future and future to past [26]. This approach allows the model to preserve information from the future and fit the data accordingly. Cui et al. [11] found that a stacked LSTM achieved a more accurate prediction. Importantly, the training times observed indicate that a bidirectional LSTM is nearly double the training time of a regular LSTM. There is no mention of prediction times for a stacked LSTM, so this study will investigate whether this increased prediction

Date	First Seen	Duration	Proto
2020-07-04	20:10:06.480	1.223	TCP
SrcIP	**DstIP**	**Flags**	**Tos**
146.231.4.0	155.232.240.0	.A....	0
Packets	**Bytes**	**pps**	**bps**
4500	234000	3679	1.5 M
Bpp	**Flows**		
52	1		

Table 1. A single observation from a SANReN dataset sample.

accuracy comes at the cost of a higher computational time compared to a stacked LSTM and a bidirectional LSTM.

5 Design and Implementation

The system was designed in Python using the Keras API on top of the TensorFlow machine learning package. A data processing pipeline was programmed to automate the preparation of data for the models. The system also includes a programmed preliminary data analysis, as described in Section 5.2.2. This was also done in Python - using NumPy, scikit-learn and the matplotlib packages. The preliminary statistical analysis outputs graphs and tables that contextualize the data and its trends. Given inputs from the data pipeline, each model can predict network traffic data to a varying degree.

5.1 Obtaining SANREN Data

The first step in conducting our research was to obtain timesseries network traffic data from SANReN. This data was accessed through the University of Cape Town. The data is stored as tab-separated .csv files, and each data set consists of a multitude of variables, each describing the quality of the network traffic flows on SANReN over a measured period of time. The SANReN data has been placed on a data store at the University of Cape Town - which is available to be accessed remotely. However, for this research project, a smaller 47000-observation sample was used to train, validate and test the LSTMs that this paper evaluates. An example of the format of the SANReN data is provided in Table 1.

5.2 Preprocessing

The raw SANReN data has to be extracted and cleaned for a neural network model [3]. This preprocessing step will transform raw data into input for each of the neural network models. This section describes the preprocessing methods that will be applied.

5.2.1 Feature Extraction and Transformation. The network traffic received from SANReN is already stored in tab-separated .csv format, with labels. The sample contained 47000 data points of 14 features. 13 of these features are explanatory variables, whilst the number of bytes that occurred in a flow - stored as Bytes - is the response for this study. However, the data is not stored in a format that is immediately interpretable by a Python program. For example, large byte values are indicated by using an 'M' or 'G' unit of measurement to represent megabyte and gigabyte flows. This additional string in some rows of the file breaks the uniformity of the file, and so each line is processed individually to convert non-byte values into byte representations, as well as removing variables that were mostly homogeneous across the entire data set, such as transfer protocol type, number of flows and flags. Initially, the categorical features were encoded using one-hot encoding - which creates a new binary column for each instance of a categorical variable - however, this caused the DataFrame's dimensions to increase dramatically and had little impact on the predictive power of the LSTM models - so the categorical variables were subsequently dropped from the DataFrame.

Additionally, to capture the relationship between South African university schedules and network traffic data, two new variables were engineered. The first is a simple day variable - which numerically encodes the day of the week - and the second is a binary holiday variable - which asserts whether the network flow occurred during a university holiday or not. The holiday dates were defined by the University of Cape Town's academic schedule for 2020. Interestingly, the date of a traffic flow and the time it was first seen on the network were stored independently, but these were combined to create a TimeStamp value for each observation. Furthermore, each of the SANReN features was transformed into an LSTM-recognised data type. Byte values were encoded to 64-bit float representations, strings were removed - as strings are not acceptable input for a TensorFlow LSTM - and temporal values were encoded to 64-bit integer nanosecond representations.

At this point in the processing pipeline, the SANReN dataset has been condensed into a 47000 x 9 DataFrame. The distribution of the remaining features in the dataset are shown in Figure 2. Lastly, the data were examined for outliers. However, since we want the LSTM models to be able to predict volatile burst flows - flows that do not follow the long-term temporal pattern - it was decided that outliers will not be removed.

5.2.2 Preliminary statistical analysis. To provide insight into the underlying patterns, correlations and composition of the SANReN data, a preliminary statistical analysis was performed.

Figure 2 demonstrates that correlation between variables is a possibility in the SANReN dataset - as demonstrated by the similar trends between Duration and Packets. To further investigate the possibility of multicollinearity, a correlation matrix 3 was plotted. As expected, Duration and Packets have

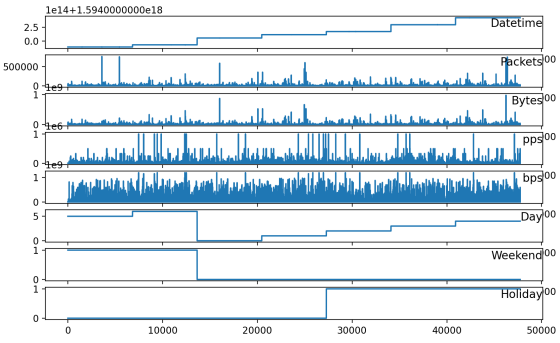

Figure 2. Distribution of the numerical features of the SANReN data.

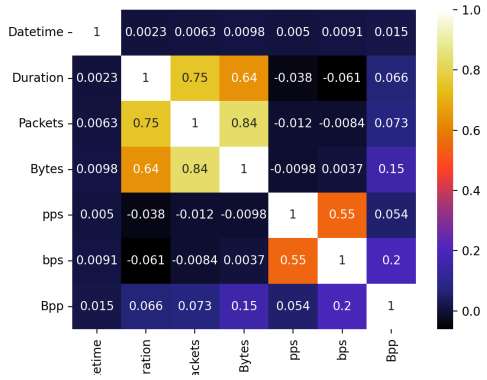

Figure 3. A correlation heatmap of the continuous variables.

a strong positive linear correlation. The existence of multicollinearity in a feature set suggests that a pair of correlated variables are providing redundant information about the response and should be removed [4]. However, this requires further investigation into multicollinearity and is beyond the scope of this study. Furthermore, the option of removing all of the variables apart from the three that make up the stronger, orange correlations in the upper left of the matrix was considered, although due to the non-linearity of the data, this design decision was rejected. Other correlative relationships of interest are the negative relationship between Bytes and Datetime and the strong positive relationship between Duration and the response. In this sample, the weak correlation between Bytes and Datetime suggest that the byte volume per flow is decreasing as time increases. Importantly, this does not imply that there is less network traffic in general because network traffic volume is a function of byte per flow as well as the number of flows. The relationship between Bytes and Duration implies that longer flows generally have more data being transferred - which is a sensible assumption.

The preliminary data analysis provides an opportunity to garner insight into the answer to our first research question: how does the SANReN traffic data vary with time and day in relation to the South African university calendar? This was initially examined using a box-and-whisker plot to examine the differences in the percentiles. However, the number of outliers caused this to be ineffective. To extract the relationship between time of day, university holidays and byte volumes, a pair of scatter plots were fitted. Figure 4 shows the total byte flows over each day of the week. Since seven equal slices (1000000 bytes) were taken from each day's sample, the number of observations from each day should be roughly the same. Surprisingly, there is no clear distinction between bytes flows during the week compared to the weekend. Similarly, the total byte flow per day is shown in Figure 5. In the sample, there are 4 non-holiday dates, and 3 holiday dates, so the totals in Figure 5 are $totalbyteflows/n[whereholiday = 0]$. Again, it is surprising to see that the total byte flows are higher on average during a holiday. This is unexpected, as the SANReN links research units and other university-related institutions.

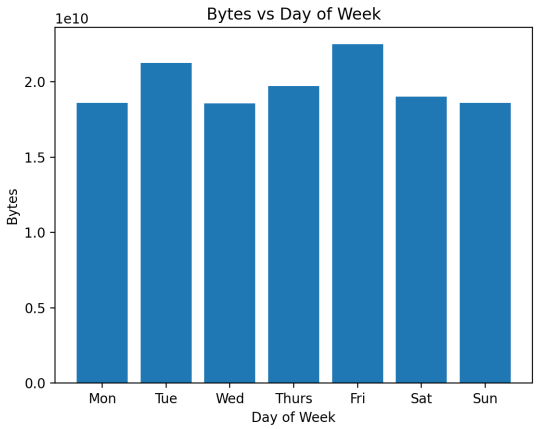

Figure 4. Scatter plot of Day vs Bytes

Lastly, the five-number quartile summary of the response was plotted - shown in Figure6. In the context of a distribution, skewness is a measure of how asymmetric the data is. It is clear that the data is extremely skewed to the right, indicated by the density of low byte value observations to the left of the curve, and the massive byte flows shown as one moves along the tail to the right. The outlying values have the effect of increasing the mean of the data. It is important to note that the LSTM models in this project are designed to capture outlying burst flows, although the magnitude of some of the outliers in the SANReN data mean that the models may under-predict these observations under testing conditions. This is elaborated in Section 6. The exact values of the quartiles of the response distribution in megabytes are also given in Table 2.

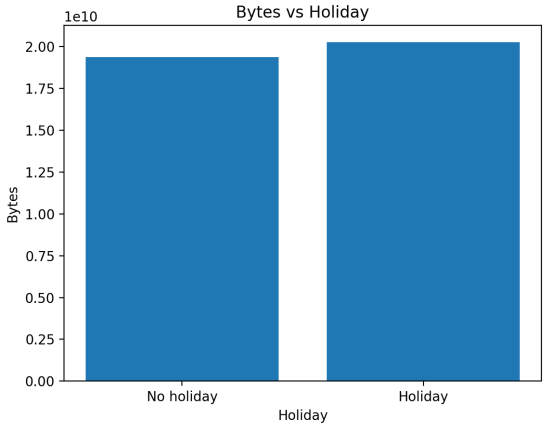

Figure 5. Scatter plot of Holiday vs Bytes

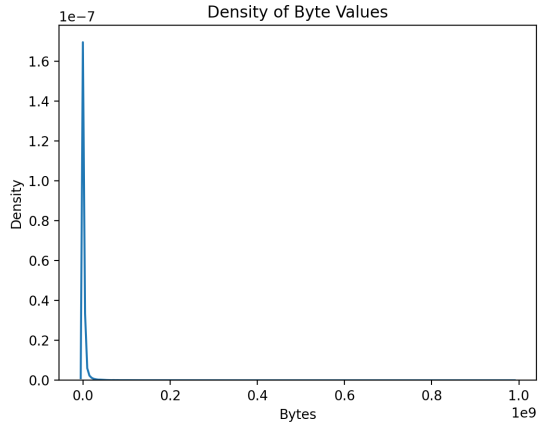

Figure 6. Distribution of the Byte reponse variable

Min	Q1	Median	Q2	Max	Mean
0.02	0.1	0.75	2.2	177	3.15

Table 2. The quartiles of the Byte response

5.2.3 LSTM Data Preparation. All machine learning algorithms require that the input data is split into training and testing samples. The training data is the sample of the SANReN data that is used to adjust the weights of the LSTM. For our implementation of the LSTMs, the training sample is 64% of the full input sample. Additionally, the training sample is split to produce the validation set. An interesting observation is identified when examining how the validation set is constructed. To ensure the sequential nature of the data, the data is not shuffled when it is split into training and test sets, and it is not shuffled when being split into training and validation sets. This essentially means that for 100 fictional observations, the training set would be made of $y_1, y_2, ..., y_64$, the validation set would be $y_{65}, ..., y_{80}$ and the test set would be $y_{81}, ..., y_{100}$. The information provided by the validation set - primarily the out-of-sample MSE of the fitted model - allows the hyperparameters of the model to be adjusted. The model's evaluation on the validation set becomes more biased as the hyperparameters are adjusted. This is because the model starts to fit the validation set better as the information it provides is incorporated into the model configuration. In our implementation, the validation set is 20% of the training set. Lastly, the test set is the sample of data that is used to provide a truly unbiased evaluation of the final model fit. The test set is used as input to the final, trained model.

In the context of time-series prediction, it is important that the validation set's role is well defined. Usually, the validation set is used to provide an evaluation of the model's fit on the dataset. It is effectively a proxy for unseen test data. However, because of the ordered split of the data, the validation set is now an actual measure of the short-term performance of the model. Therefore, this study will propose a model that has been optimised on this unique type of validation set. The optimised model will be assessed on unseen data that lies beyond the indexing of the validation set, and therefore, the final model will be optimised to predict $t_{i+validationsetsize}$ steps ahead of the current time, t_i. It is important to clarify that this does not mean that the model cannot be used by SANReN for short-term network traffic prediction.

Once the train, test and validation sets have been created, LSTM models require that all data is normalised to an equal range of values. The MinMaxScaler from the scikit-learn package [24] scales all variables to a range of 0 to 1. Usually, categorical variables would have to be encoded separately, however, the categorical variables of the SANReN dataset have not been used as predictors in the LSTM models. After the LSTM models have been implemented, the response can be predicted. However, due to the scaled nature of the inputs, the output is also a scaled value between 0 and 1. To convert the LSTM predicted response back to a byte value, a MinMaxScaler is used. The data is then fully prepared, and ready to be given to the LSTM models.

5.3 Implementation of Models

The simple, bidirectional and stacked LSTM models were all developed in a Python environment using the Keras API on top of the TensorFlow package. The system was initially built on Google Colaboratory, to use its GPU capabilities to train the models. However, the system was then transferred into a Python executable - as it offered the ability to develop functions in a more modular fashion, and because SANReN data sizes became difficult to manage in a cloud environment. As mentioned in Section 5, the data is preprocessed, formatted and then used to provide preliminary insight into the dataset. It is then transformed and given to the LSTM models as input.

5.3.1 Simple LSTM. A simple LSTM, also known as a vanilla LSTM, is an LSTM with a single hidden layer. As it is the simplest model, the expectation was that it would be the least accurate and computationally cheapest, allowing it to function as the baseline for the performance for all of the project's models. A Keras LSTM requires a 3D input - in the form (samples, time steps, number of features). The data from SANReN is naturally two-dimensional, with the observed features spanning the horizontal dimension in the DataFrame, and a time-related index descending vertically through the data. As an LSTM designer, the product of the 2D SANReN input must be the same as the product of the reshaped 3D LSTM input. One can either break the input into smaller samples, decreasing the 'sample' dimension of the input, or default to a time step of 1, which leaves the product of the dimensions as it was. This project implemented a time-step of one - which allowed the LSTM to access the entire sample input at once.

The simple LSTM accepts the SANReN data and adjusts its neurons - by changing the weights of the internal input, update and output gates - to best fit the model to the true non-linear pattern of the sample. Although the model is fitted on the training data - given both the explanatory variables and the response - it still makes a response prediction whilst it is being fitted. This allows the model to return training accuracy metrics - such as MSE loss. Once the model has been optimally fitted, it is ready to be fed unseen test data to make predictions.

5.3.2 Bidirectional LSTM. A bidirectional LSTM involves duplicating the first recurrent layer in the network so that there are now two layers side-by-side. It then provides a reversed copy of the input sequence to the second, reversed layer [21]. Other LSTMs, and recurrent neural networks in general, only consider information from the past, $x_1, ..., x_t - 1$, and the current inputs x_t to determine the value of a prediction [12]. This approach is generally used in machine learning tasks where context is necessary - such as speech recognition - but it has been implemented as a solution to time-series forecasting by Althelaya et al. [5] in their work. In their study, it outperformed a simple LSTM for short-term and long-term predictions.

A bidirectional LSTM allows the whole input sequence to be considered when predicting y_t [12]. A bidirectional LSTM is implemented in Keras by wrapping a simple LSTM in a Bidirectional layer. The Keras API then ensures that the LSTM processes data forward and backwards through time. Furthermore, to ensure that the difference in performance between the bidirectional and baseline LSTM is due to the bidirectional flow of data, the two encapsulated LSTMs are defined identically to the baseline LSTM.

5.3.3 Stacked LSTM. A stacked LSTM is a simple LSTM with multiple LSTM layers. Rather than a hidden layer immediately producing a scalar output, the LSTM layer will provide a sequence output to the LSTM layer below it. This implies that the data remains in a three-dimensional format whilst it passes through each of the hidden LSTM layers, before being converted to the required scalar value [10]. The motivation behind a stacked LSTM for time-series prediction can be attributed to Graves et al. [13], who found that the depth - the number of layers of the network - had more of an effect on predictive power than the number of neurons in a neural network layer.

Our implementation of a stacked LSTM in Keras uses three LSTM layers, each with the same number of neurons. As discussed earlier, a Keras LSTM layer requires input in a three-dimensional format. For this reason, the first and second LSTM layers return three-dimensional output for the LSTM layers below them. Essentially, an LSTM layer returns a sequential output rather than a single scalar [7]. The output layer, the Dense layer in Keras, outputs a scalar prediction based on the three-dimensional input it receives from the LSTM layer above it.

5.3.4 Prediction Metrics. Before searching for the optimal models across the hyperparameter space, it is important to define how each model will be evaluated. Mean Absolute Error (Equation 1) and Mean Squared Error (Equation 2 are both measures of how close a model's predicted values are to the actual observed value. MAE averages this difference across the entire model, with all scores being equally weighted. The primary difference with MSE is that it squares the errors before averaging them - resulting in a larger penalty being given to outlying values. For both prediction metrics, a lower result is better, because it shows a better model fit. Additionally, due to the seemingly stochastic nature of the SANReN time-series data, we are more concerned with how MSE responds to the data. The model should learn the short and long-term trends well enough that there are few outliers to penalise. However, MAE is also used as a preliminary metric to assess the fit of the LSTM models. Both of these metrics are implemented from the scikit-learn package in Python [24].

$$MAE = (\frac{1}{n}) \sum_{i=1}^{n} (y_i - \widehat{y_i}| \tag{1}$$

$$MSE = (\frac{1}{n}) \sum_{i=1}^{n} (y_i - \widehat{y_i})^2 \tag{2}$$

Furthermore, the Coefficient of Determination - R^2 - is used to evaluate the models. It shows the proportion of the variance in the response that can be explained by the independent features. In this case, it shows the proportion of the variation in Bytes that can be explained by the numerical features in the dataset. In statistics, the range of the metric is from 0-1, with a low score showing that the independent features do not explain the response well, and a high score

illustrating that the independent features explain the variation in the response well. However, the scikit-learn package implementation of R^2 allows the value to be negative [24]. This is common when the data being evaluated is not from the same subset as the training data - as would be the case with a test set - or with non-linear data [9]. When the R^2 value is negative, this means that a model predicting the mean of the data would provide a better fit than the fitted model [9].

$$R^2 = 1 - \frac{SS_{regression}}{SS_{total}} \tag{3}$$

5.4 Hyper-parameter Tuning

All three of the LSTM models are parameterised functions, meaning that there is an optimal combination of parameters that will minimise the chosen cost function and produce the best performing model [29]. This process is known as hyperparameter tuning. The simple, bidirectional and stacked LSTM models were fitted over a range of hyperparameters. In the context of LSTMs, these parameters are epochs - the number of times the data is passed through the model - and neurons - the number of memory units in each LSTM layer. In this study, each LSTM model was trained using a range of epochs from 25 to 150, with 25 epoch intervals, and with 50 or 100 neurons in each LSTM layer of the model. This resulted in 12 permutations of hyperparameter combinations for each model. It is of paramount importance that the hyperparameters are adjusted to optimise the fit of the models on the training and validation data. If the models' test prediction performance is used to retrospectively adjust the hyperparameters of the models, then the test data is not truly unseen and the results of the study may not apply to the SANReN use-case. It is for this reason that a model's potential predictive power is assessed on the training and validation sets. Additionally, because predictive performance is not the only measure of an optimal model for the SANReN use case, the training time of the models will also be assessed across the hyperparameters.

When using an LSTM for network traffic prediction, the most expensive computational task is training each model. The time it takes for a model to output a set of predictions is minimal compared to the training time. For example, training the most basic LSTM - a 50 neuron baseline LSTM over 25 epochs - took 77 seconds, while the prediction on the same model was 0.68 seconds. Therefore, the training time will be used to assess the applicability of a model to the SANReN use case. The models were trained on a 2Ghz Dual-Core Intel Core i5, with 8 GB of RAM, and the hardware on the training machine must be considered by anyone who applies the conclusions of this study to their research.

Figure 7 shows the training time for each of the three models as epochs and neurons are adjusted. The general linear trend is positive, with the models' training times reflecting their depth. Generally, the simple LSTM has the smallest training time, with the bidirectional LSTM being slightly more complex, and the stacked LSTM taking far longer to train. There seem to be training time anomalies during the 100 and 125 epoch iterations across all of the models. Without further testing, this cannot be attributed to anything other than noise, and the overall trend is maintained. It is noticeable that by having two more layers than the simple LSTM and an additional layer over the bidirectional LSTM, the stacked LSTM takes 228% and 187% longer to train on average. Based on training complexity alone, a model cannot be recommended for the SANReN study, but this information will be considered as the hyperparameter tuning process is assessed.

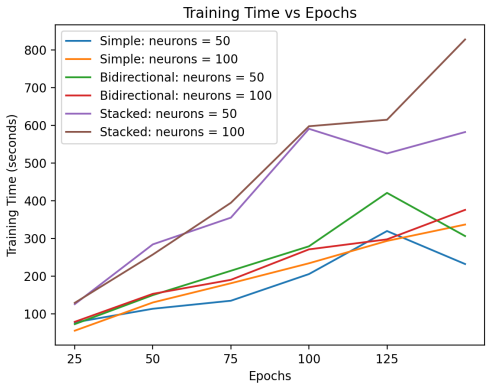

Figure 7. Model training time as a function of epochs and neurons.

Although training and prediction were done on the full sample, the effect of dataset size on training time was investigated. As shown in Figure 8 the gradients of the linear training time functions of each model are ordered as expected, with the stacked LSTM's training time increasing faster than the bidirectional and simple models. The bidirectional model's training time also increases faster than the simple model.

Lastly, it is worth noting how the models will be referenced in this section. A model with 50 neurons in its hidden LSTM layer will be referred to as 'the 50-neuron model', whilst additional context will be provided to clarify which epoch parameter is being discussed.

5.4.1 Simple LSTM Hyperparameters. One of the benefits of searching for an optimal model across a range of hyperparameters is that each model can be optimised independently. Figure 9 shows the near-parabolic shape of MAE as epochs increase. Interestingly, the more complex models - the 100 neuron models - are outperformed by their simpler counterparts on both the training data and the validation set. The red curve is higher than the orange until 100 epochs,

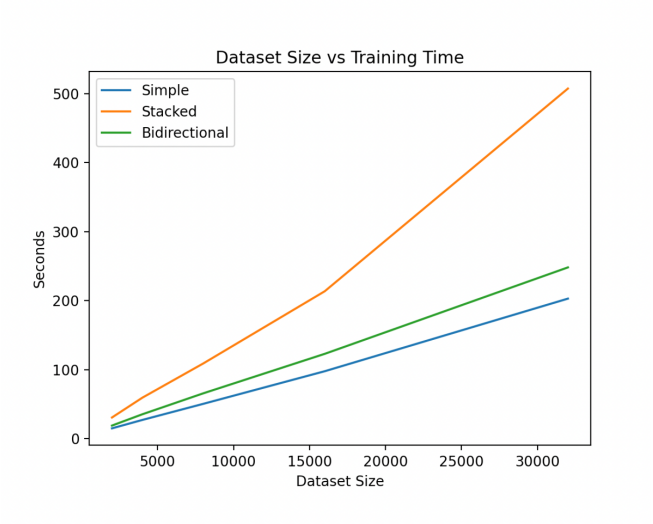

Figure 8. Model training time as a function of dataset size.

and the green curve is higher than the blue. Ideally, the MAE of the validation set would not be significantly higher than the MAE of the training set, however, this is not the case here. This suggests that the model is overfitting the training data, and is demonstrating too much bias. One point in the plot where the validation and training set MAE are similar is at 125 epochs. Therefore, to allow the simple LSTM model to pick up some of the variations in the future test sample, and to avoid overfitting, 125 epochs and 100 neurons is the optimal configuration for the simple LSTM when considering MAE as a performance metric.

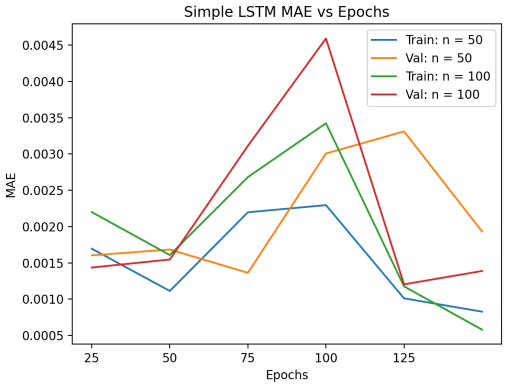

Figure 9. Model MAE as a function of epochs and neurons.

By looking at Figure 10, we are able to refine the hyperparameter suggestions from Figure 9. It is evident that the general linear trend is decreasing, with MSE still decreasing when the upper epoch limit is reached. Surprisingly, the validation MSE is lower than the training set MSE in both the 50 and 100 neuron models. The sequential nature of the data means that the data is not uniform, and so the distributions of data within the train, validation and test sets may differ.

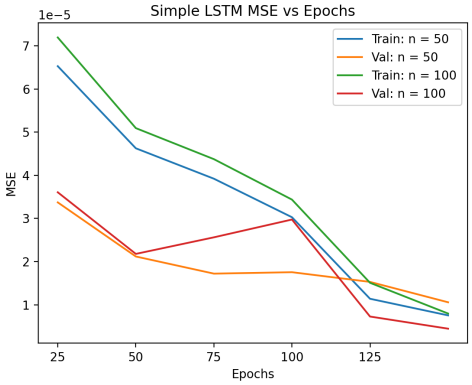

Figure 10. Model MSE as a function of epochs and neurons.

All of the MSE values reach their global minimum after 150 epochs. A low training MSE shows that the simple LSTM is fitting the data with more bias, and a low validation MSE shows that the model is still able to capture the variation in other samples without overfitting. Additionally, as a proxy for testing, the MSE of the validation set suggests the range of MSE values we can expect when the simple LSTM model performs predictions is between 0.00000727 and 0.00000445. These values can be compared with the MSE values of the bidirectional and stacked LSTM model to suggest which will perform best on the unseen test data.

Considering the information provided from the MAE, MSE and R^2 metrics: that the simple LSTM is overfitting after 125 epochs; that the marginal improvement in MSE decreases for all sets after 125 epochs; and that R^2 shows the 125-epoch model explaining the variation in Bytes well for both models, the optimal epoch hyperparameter for the simple LSTM model is 125 epochs. Additionally, at 125 training epochs, the 100 neuron model consistently performs better. Therefore, the optimal hyperparameters for the baseline LSTM model are 125 epochs and 100 neurons.

5.4.2 Bidirectional LSTM. The process by which the bidirectional LSTM's hyperparameters were selected is similar to the simple LSTM. By assessing the MAE, MSE and R^2 metrics, the optimal hyperparameters for the bidirectional model are 150 epochs and 50 neurons. Interestingly, the bidirectional model also experienced a decrease in all performance metrics when trained over 100 epochs - especially when run with 100 neurons. This trend was observed in the simple LSTM model too - and can be attributed to noise in the data and variation in the metric. This is best shown by the sharp increase in the MSE trend of the 100 neuron model on the training and validation sets, as shown in Figure 12.

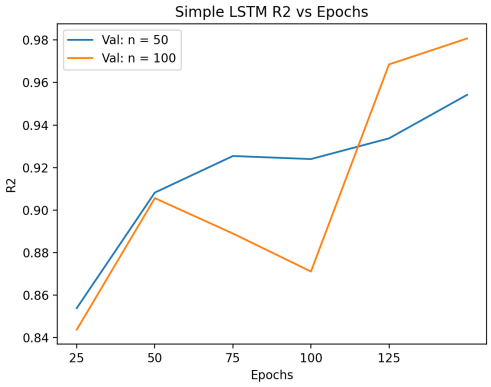

Figure 11. Model R2 as a function of epochs and neurons.

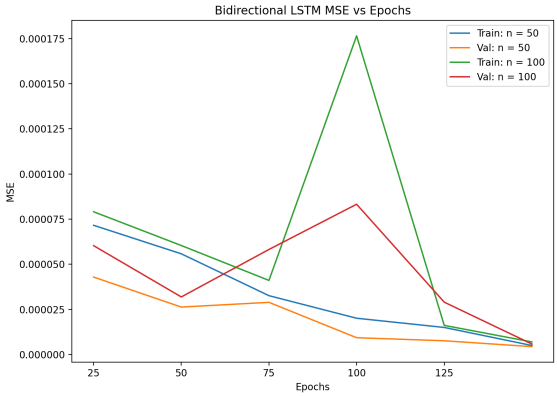

Figure 12. Model MSE as a function of epochs and neurons.

5.4.3 Stacked LSTM. Optimisation of the stacked LSTM starts with the investigation of the MAE metrics across the hyperparameter space - as seen in Figure 13. The trend of MAE for the stacked model is different for the 50-neuron and 100-neuron models. For the 50-neuron model, training MAE initially increases, before decreasing as epochs reach 150. The MAE of the 50-neuron model's validation sample increased sharply after 100 epochs, suggesting that the model strongly overfits after this point. The 100-neuron model's trend is the inverse: it initially decreases before increasing and again finding a minimum at 150 epochs. The 100-neuron model is overfitting after 75 epochs, but the average of the errors decreases again when the model is trained on 125 and 150 epochs. Additionally, both the models exhibit too much bias on the training data when the model is given 150 epochs to train. This is shown by the divergence in the blue and orange, and red and green pairings in Figure 13. Therefore, the models of interest are the 100-epoch and 125-epoch models. The MAE values for the 100 and 125-epoch models are shown in Table 3. The difference between the less-complex and more-complex models is much more evident at 100 epochs, with the 50-neuron model having a better MAE by $7.1e-7$ compared to the 100-neuron model having a better MAE by $4e-8$ at 125-epochs.

Epochs	Neurons	MAE
100	50	0.00000261
100	100	0.0000019
125	50	0.00000307
125	100	0.00000311

Table 3. Stacked LSTM MAE on validation data for 100 and 125-epoch models

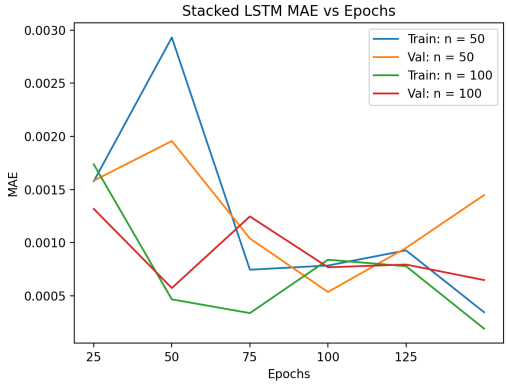

Figure 13. Model MAE as a function of epochs and neurons.

The more complex model generally outperform the less complex model in this case. This is aligned with what we would expect, as adding neurons to the hidden LSTM layer should increase the capability of the model to fit the dataset. Due to the minimal differences between the MAE values of the 100 and 125 epoch models, MSE will be used to refine an optimal hyperparameter pair for the stacked LSTM model.

Figure 14 shows that the 100-neuron model's MSE mirrors the MAE's sharp initial decrease. However, the similarity stops there. After 50 epochs, the validation set MSE for the 100-neuron model hits a minimum. In the 50-epoch model, the validation set MSE is 0.0000014, while the 150-epoch model has a validation set MSE of 0.00000125. Both the validation and training MSE only exhibit small fluctuations from then on, suggesting that exposing the LSTM model to the data more than 50 times has little effect on the model's fit of the data. The training MSE's minimum occurs in the 150-epoch model, but the separation between the red and green line shows that, at this point, the model is starting to overfit slightly.

Nevertheless, the 100 and 125-epoch models have already been presented as optimal models, and so the information

the MSE provided can be used to distinguish between these two models. It is immediately evident that at 125 epochs, the model is attempting to correct for overfitting. The increased MSE shows that the model's fit to the training data has worsened. However, the validation MSE has increased far less. An increase in training MSE with a smaller increase in the MSE of a proxy test set is encouraging, as it shows that the model is generalizing well on unseen data.

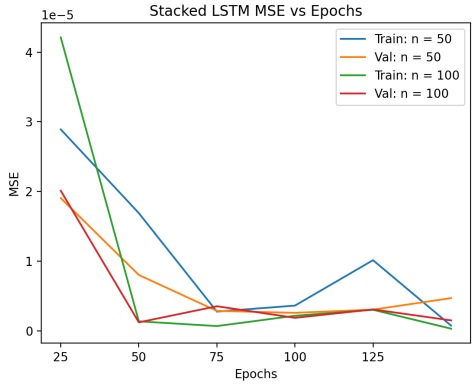

Figure 14. Model MSE as a function of epochs and neurons.

The R^2 values for the stacked LSTM follow a similar trend to the MSE values - as shown in Figure 15. As training epochs increase, the general trend of the validation set R^2 increases. The 100 neuron model's R^2 is generally above or similar to its less complex partner and increases dramatically at 50 epochs. Interestingly, according to the formula, the estimated R^2 is not expected to change as the sample size decreases. However, the variation of its value increases as the sample size decreases. Since the validation set is 20% of the training data, the fluctuations in an R^2 value of a validation set cannot be attributed to a worse model fit alone. However, the R^2 value of both models begin to decrease beyond 100 epochs, and so the 100-epoch model will be used as the optimal candidate based on what we can infer from the model's R^2 scores. The R^2 values of the 100-epoch model show that 98.8% and 99.1% of the variation in Bytes are explained by the independent features of the 50-neuron and 100-neuron model respectively.

The metrics show that the stacked LSTM is overfitting after 125 epochs, that the marginal improvement in MSE decreases for all sets after 50 epochs but is comparable at 100 or 125 epochs, and that the R^2 metric shows the 100-epoch model explaining the variation in Bytes well for both models. Therefore, the optimal epoch hyperparameter for the stacked LSTM model is 100 epochs. Although the R^2 values for the 100-neuron model continue to increase past 125 epochs, existing concerns about overfitting cause this improvement to be of negligible importance. Additionally, at 100 training epochs, the 50 and 100 neuron models have

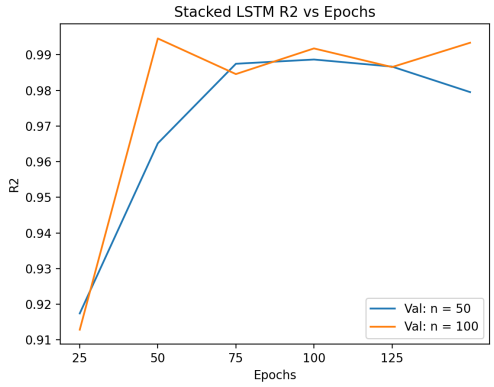

Figure 15. Model R2 as a function of epochs and neurons.

comparable performance. The comparison between the two models fitted over 100 epochs is shown in Table 4. The 50-neuron model performs better in MAE and R^2, and due to the additional constraint of complexity, the minuscule difference in MSE is an acceptable allowance. Therefore, the optimal hyperparameters for the stacked LSTM model are 100 epochs and 50 neurons.

	Metric	50 Neurons	100 Neurons
1	MAE	0.00053	0.00077
2	MSE	2.61E-06	1.90E-06
3	R^2	0.992	0.987

Table 4. Table of model evaluation metrics for Stacked LSTM over 100 epochs on a validation set.

6 Results

The hyperparameter search allowed each model to be optimised independently. According to the information the training and validation sets provide, the optimised models are a 100-neuron simple LSTM over 125 epochs, a 50-neuron bilateral LSTM over 150 epochs, and a 50-neuron stacked LSTM over 100 epochs. Based on these hyperparameters, the prediction results on an unseen test set are provided in Table 5. The prediction metrics were used to evaluate the second of the research questions from Section 3.1: which of the LSTM models provides the highest accuracy?

Model	MAE	MSE	R^2	Training Time
Simple	0.0063	0.00010	0.65	**293.67**
Bidirectional	0.0078	0.00013	0.55	306.60
Stacked	**0.0029**	**0.00006**	**0.80**	591.20

Table 5. Table of model evaluation metrics on test set using optimal models.

The stacked LSTM is the best performing model on the test data. It has the lowest MAE and MSE, and the highest R^2. However, it also takes extremely long to train compared to the other models. This trade-off will be assessed in Section 7. Although it is more important to assess the models' predictive power, the graphical results are worth examining too. In all of the plots of predicted bytes vs observed bytes, the left-hand plot shows if any predictions are under-predicting, while the right-hand plot shows if any of the predictions are over-predicting. The results also show that the bidirectional model's additional complexity offers no improvement in accuracy or training time over the simple LSTM.

The simple LSTM's predictions are shown in Figure 16. From the two plots, it is evident that the simple LSTM is over-predicting more frequently than it is under-predicting. The fitted simple LSTM only accounts for 65% of the variation in the Bytes, so, understandably, there is some unexplained noise in the predictions. Additionally, the general trend is impressive. The model is capturing the non-linearity and non-stationarity in the model, although it has under-predicted a large burst flow by 25%. This large burst flow has affected the next observation too, and it has over predicted by a large amount too. This suggests that the influence of data point, t_{i-1}, is affecting the prediction of the data point at t_i.

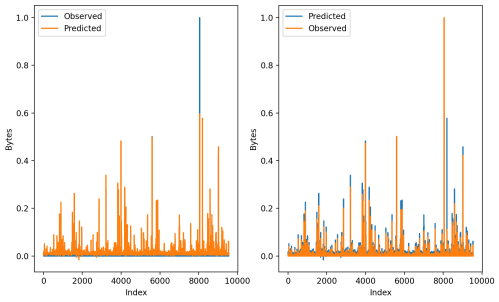

Figure 16. Predicted vs observed test set values for the optimised simple LSTM.

The bidirectional LSTM's predictions are shown in Figure 17. The dual flow of information through the model has had an unusual effect on the shape of the LSTM's predictions. Despite the inputs being bound to a range of 0 - 1, the bidirectional LSTM has produced byte outputs that are less than 0. In general, the bidirectional LSTM over-predicts most values in the test set, and also fails to capture the full extent of the burst flow. Additionally, the over-estimating effect of burst flows on the next predicted value is evident - as it was with the simple LSTM.

Lastly, the graphical results of the optimised stacked LSTM model reinforce that it is the best performing model in terms of accuracy. On the left-hand side of Figure 18, the model

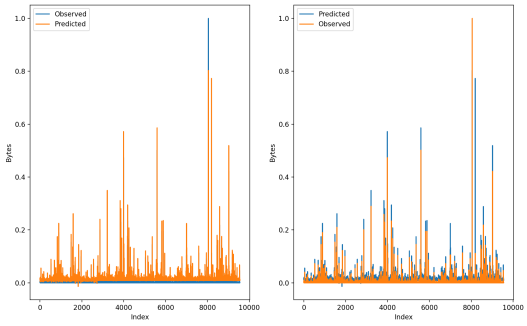

Figure 17. Predicted vs observed test set values for the optimised bidirectional LSTM.

is slightly under-predicting for low byte values and misses the outlying burst flow that the other two models failed to capture. Again, the model is over-predicting in general, but the amount by which it is overpredicting is much less than the other models. In this stacked LSTM, the independent variables account for 23% more variation in the response than the simple model, and 64% more than the bidirectional model.

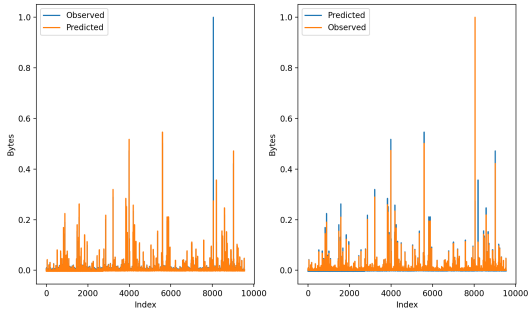

Figure 18. Predicted vs observed test set values for the optimised stacked LSTM.

The answer to the first research question was discussed in Section 5.2.2. The SANReN network traffic data does not show any obvious trend that follows the South African university calendar. The average total daily flow of bytes was higher for holiday days than it was for 'school' days. The daily total flows were highest on Tuesday and Friday, as shown in Figure 4, although there is no valuable conclusion that can be inferred from this. Further insight into how the university calendar affects SANReN traffic and congestion would require further study. The size of the SANReN data meant that only a small sample could be used, whereas the trend between bytes and days would be more evident if hundreds of days could be sampled simultaneously.

$$trainingtime = 100 + 0.0172(samplesize) \quad (4)$$

Additionally, the training times of each model have been assessed in Section 5.4 as a measure of the computational cost of the different models. The stacked LSTM model training process takes 228% and 187% longer than the simple and stacked models, respectively. The training time of each model increases linearly, as shown in Figure 8, and so the training time of any of the models with more observations as input could easily be extrapolated. As the best performing LSTM architecture, the stacked LSTM's linear regression function was calculated. This is shown in Equation 4 and can be used to estimate the training time - in seconds - of a stacked LSTM on *samplesize* number of observations. As this project was done on personal computers, it is likely that SANReN has the computing hardware to implement a stacked model with more observations than were used in this study.

7 Discussion

The stacked LSTM is the best model for prediction, but the value of its superior predictive performance is determined by the implementer of an LSTM model. If the stacked LSTM's training time - which is two times greater than both simpler models - is prohibitive, then a network provider may decide that the predictive performance of the simple LSTM model is sufficient. The stacked LSTM's R^2 value shows that as the depth of an LSTM increases, it is able to better find a non-linear model fit that attributes more of the variation in the response to the independent features. Additionally. the optimal stacked LSTM model was parameterised with the fewest epochs. This suggests that an even deeper model trained over fewer epochs may out-perform a stacked model.

Furthermore, when looking at the discrepancies between training and test metrics, it is immediately evident that all of the selected models were overfitting the training data. The models are too biased - and are capturing too much noise in the training sample. They are unable to capture the variation in the unseen test sample. The reason this wasn't picked up is because the baseline measure - the simple LSTM - is also overfitting. Therefore, the stacked and bidirectional models seem to only be overfitting slightly, but they are actually capturing more bias than a baseline model with a high existing bias. One possibility is that the non-uniformity in the data causes the training set to exhibit higher byte volumes than those of the training set, but this is unlikely.

Additionally, it was mentioned in Section 5.4 that prediction times were not considered as a metric of model complexity. All three models' prediction times are extremely low compared to their training times. This is because the model has already learned to fit the data, and is applying an established non-linear function to a set of inputs, rather than finding a model that minimises an MSE metric.

In determining the non-linear function that best fits the data, the LSTMs were provided with a range of hyperparameters. The effect of changing the two studied hyperparameters is very similar. Generally, if epochs or neurons are increased, then the performance of a model improves. In each of the models' figures of MAE, MSE and R^2, the effect of an increase in epochs appears to be more consequential. The difference in the measured value between the first epoch parameter and the last epoch parameter seems to be larger than the effect of increasing neurons. An increase of 25 epochs does seem to increase accuracy slightly more than an increase of 50 neurons, as illustrated in Figure 10. The vertical distance between the green and blue, and red and orange, lines show the effect of increasing neurons, and the vertical distance between a model's metric at epoch x and epoch $x + 25$ shows the effect of increasing epochs. In general, epochs cause a greater increase in MSE. However, to make these increases genuinely comparable, a researcher would have to find the effect of changing a parameter by the same unit, which was not done in this study.

LSTMs can adjust other hyperparameters too. In the existing literature, the other two adjusted hyperparameters are time steps and batch size. A time step in an LSTM model refers to the number of times a neuron iterates before passing the output to the next neuron [23]. Although an LSTM naturally learns the data's trend using the entire sample, by increasing the time steps parameter, the LSTM has explicit memory of n number of observations, where $timesteps = n$. In this study, increasing time steps presented two difficulties. Firstly, changing the time steps parameter affects the required input shape of the LSTM models. Reshaping the SANReN data to a variable size proved to be out of scope for this study and would be worth examining in another study. Secondly, increasing the number of time steps caused the LSTM models' training set performance to drop significantly. The LSTMs were no longer capturing any of the variations in the model and were producing predictions that were all based on the sample mean. This functionality is not useful at all for network traffic prediction and would be more appropriate in a less granular network analysis study. The batch size determines how many observations the LSTM processes at once. When configuring an LSTM, a researcher can break up a sample into multiple batches or give the entire sample to the LSTM at once. In this study, the entire sample was provided to the models at once. This meant that batch size did not have to be defined, because the batch size was the size of the LSTM input.

Due to the nature of the train, validation and test sets, their natural temporal order is maintained by the LSTMs. This essentially means that the validation set is not a proxy for the test set, but rather an actual measure of how each of the models performs in the short term, whilst the test data is now framed as a measure of how the model performs beyond the range of the validation set. However, the MAE, MSE and

R^2 metrics of the optimal models' validation sets cannot be used to define any results, as they have been used to optimise the hyperparameters of the optimal models themselves. Therefore, this study is assessing an LSTM that is a medium to long-term predictor for network traffic prediction. In the future, cross-validation or walk forward validation would be more appropriate for creating a test set proxy, without losing the ability to assess the short-term performance of a model.

Lastly, based on the research questions stated in 3.1, it is important to recommend a model for the SANReN use case. Based on the results, a stacked LSTM model provides the highest accuracy when predicting future SANReN traffic data. However, when subject to network constraints, the simple LSTM is a less accurate alternative. The computational cost of the stacked LSTM model is at least two times that of the simple model, and so model selection would ultimately be decided given a required level of accuracy. If a predetermined level of accuracy was required by SANReN, then this study could recommend either the simple or stacked LSTM models based on whether the simple LSTM produced the required accuracy level. Alternatively, if a maximum training time allowance was provided, a model could be recommended based on whether the stacked LSTM's training time fell below the threshold. However, given that SANReN is a country-wide network spanning multiple institutions, this study recommends that they use a stacked LSTM model as a network traffic predictor.

8 Limitations and Future Work

There were a couple of limitations that the study encountered. Firstly, the size of the SANReN data is prohibitive when implementing LSTM models on a personal device. This study has investigated the capabilities of LSTMs to predict network traffic data over a week, using almost 7000 data observations from each day. However, each SANReN data file - of which there is one per day - contain at least 9 000 000 observations. With a better understanding of the resources SANReN has at its disposal, it would be easier to implement more realistic modelling of the SANReN data. However, the project is scalable and can accept an input of any size, as long as there is hardware to support it. Secondly, the study aimed to extrapolate its conclusions to other low-resource networks. In the context of this study, a low-resource network has not been well defined and so it is hard to conclude without a quantitative value of the performance capabilities of a network. In the future, another network could be assessed - provided that a research team had knowledge of the computing resources at their disposal. Lastly, the nature of time-series data presents a few challenges. The primary limitation of sequential data for LSTMs is that the data cannot be shuffled when split into sets or when provided as input. In this study, this had the effect of causing the test-set to be a longer-term forecast, whilst the validation set became a short-term proxy of test performance. In future work, a study could determine if the temporal quality of the SANReN data is a necessary feature for an accurate network traffic predictor.

Other future work could include developing a stateful LSTM model, implementing a grid search over additional hyperparameters, further investigation into the relationship between computer hardware and the effect it has on potential model complexity, and a more thorough, analytical approach to model's validation sets.

9 Conclusions

In this study, LSTM models were applied as network traffic predictors to SANReN data. An examination of the data shows that SANReN data does not have a well-defined relationship with the South African university schedule, and does not have an obvious weekly trend. Although all of the models were able to capture the long-term trends in the data, the positive skew in the sample resulted in none of the models being able to model the volatility of the short-term burst flows.

Using a grid-search for hyperparameter tuning, it was found that the optimal models to predict SANReN network traffic data were a 100-neuron simple LSTM over 125 epochs, a 50-neuron bilateral LSTM over 150 epochs, and a 50-neuron stacked LSTM over 100 epochs. Of these models, a stacked LSTM model is the best performing model in all three measured metrics. Despite taking 228% longer to train than the simple model, the stacked model had an R^2 value of 0.80 compared to values of 0.65 and 0.55 for the simple and bidirectional LSTM models. Therefore, the stacked LSTM is the recommended model for the SANReN network traffic prediction use case.

By using the stacked LSTM as a network traffic predictor, SANReN will be able to implement measures to manage network congestion and network resources. However, given a complexity constraint, the simple LSTM's accuracy may be an adequate network traffic predictor on a low resource network. The bidirectional LSTM was the worst performing model, and its additional complexity offers no improvements in training time over the simple LSTM. Therefore, this study aligns with existing literature in concluding that LSTMs are useful and accurate network traffic predictors, and that deeper, more complex stacked LSTM architecture outperform simple and bidirectional LSTM models.

10 Acknowledgements

The researcher would like to thank SANReN, and Dr Josiah Chavula and Justin Myerson from the University of Cape Town, for their contributions to this study.

References

[1] The south african nren. URL https://sanren.ac.za/south-african-nren/.

[2] Cisco annual internet report - cisco annual internet report (2018–2023) white paper, Mar 2020. URL https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[3] 2021. URL https://cloud.google.com/architecture/data-preprocessing-for-ml-with-tf-transform-pt1.

[4] Basil Alothman. Raw network traffic data preprocessing and preparation for automatic analysis. In *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pages 1–5, 2019. doi: 10.1109/CyberSecPODS.2019.8885333.

[5] Khaled A. Althelaya, El-Sayed M. El-Alfy, and Salahadin Mohammed. Evaluation of bidirectional lstm for short-and long-term stock market prediction, 2018.

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

[7] Jason Brownlee. Stacked long short-term memory networks, Aug 2019. URL https://machinelearningmastery.com/stacked-long-short-term-memory-networks/.

[8] Jason Brownlee. How to develop a bidirectional lstm for sequence classification in python with keras, Jan 2021. URL https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/.

[9] A. Colin Cameron and Frank A.G. Windmeijer. An r-squared measure of goodness of fit for some common nonlinear regression models. *Journal of Econometrics*, 77(2):329–342, 1997. ISSN 0304-4076. doi: https://doi.org/10.1016/S0304-4076(96)01818-0. URL https://www.sciencedirect.com/science/article/pii/S0304407696018180.

[10] Paulo Cortez, Miguel Rio, Miguel Rocha, and Pedro Sousa. Internet traffic forecasting using neural networks. In *The 2006 IEEE international joint conference on neural network proceedings*, pages 2635–2642. IEEE, 2006.

[11] Zhiyong Cui, Ruimin Ke, Ziyuan Pu, and Yinhai Wang. Stacked bidirectional and unidirectional lstm recurrent neural network for forecasting network-wide traffic state with missing values. *Transportation Research Part C: Emerging Technologies*, 118:102674, 2020.

[12] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013. URL http://arxiv.org/abs/1303.5778.

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

[15] Yuxiu Hua, Zhifeng Zhao, Rongpeng Li, Xianfu Chen, Zhiming Liu, and Honggang Zhang. Deep learning with long short-term memory for time series prediction. *IEEE Communications Magazine*, 57(6):114–119, 2019.

[16] Muhammad Faisal Iqbal, Muhammad Zahid, Durdana Habib, and Lizy Kurian John. Efficient prediction of network traffic for real-time applications. *Journal of Computer Networks and Communications*, 2019, 2019.

[17] Shan Jaffry. Cellular traffic prediction with recurrent neural network. *arXiv preprint arXiv:2003.02807*, 2020.

[18] Nan Jiang, Yansha Deng, Osvaldo Simeone, and Arumugam Nallanathan. Online supervised learning for traffic load prediction in framed-aloha networks. *IEEE Communications Letters*, 23(10):1778–1782, 2019.

[19] Nandini Krishnaswamy, Mariam Kiran, Kunal Singh, and Bashir Mohammed. Data-driven learning to predict wan network traffic. In *Proceedings of the 3rd International Workshop on Systems and Network Telemetry and Analytics*, pages 11–18, 2020.

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

[21] Huang Lin, Wang Diangang, Liu Xiao, Zhuo Yongning, and Zeng Yong. A predictor based on parallel lstm for burst network traffic flow. In *Proceedings of the 2020 6th International Conference on Computing and Artificial Intelligence*, pages 476–480, 2020.

[22] Rishabh Madan and Partha Sarathi Mangipudi. Predicting computer network traffic: a time series forecasting approach using dwt, arima and rnn. In *2018 Eleventh International Conference on Contemporary Computing (IC3)*, pages 1–5. IEEE, 2018.

[23] C. Olah. Understanding lstm networks.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[25] Nipun Ramakrishnan and Tarun Soni. Network traffic prediction using recurrent neural networks. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 187–193. IEEE, 2018.

[26] Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681, 12 1997. doi: 10.1109/78.650093.

[27] Sebastian Troia, Gao Sheng, Rodolfo Alvizu, Guido Alberto Maier, and Achille Pattavina. Identification of tidal-traffic patterns in metro-area mobile networks via matrix factorization based model. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 297–301. IEEE, 2017.

[28] Sebastian Troia, Rodolfo Alvizu, Youduo Zhou, Guido Maier, and Achille Pattavina. Deep learning-based traffic prediction for network optimization. In *2018 20th International Conference on Transparent Optical Networks (ICTON)*, pages 1–4. IEEE, 2018.

[29] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. Applying deep learning approaches for network traffic prediction. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2353–2358. IEEE, 2017.

[30] Weitao Wang, Yuebin Bai, Chao Yu, Yuhao Gu, Peng Feng, Xiaojing Wang, and Rui Wang. A network traffic flow prediction with deep learning approach for large-scale metropolitan area network. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[31] Steven Wheelwright, Spyros Makridakis, and Rob J Hyndman. *Forecasting: methods and applications*. John Wiley & Sons, 1998.