DeepFlow

Flow data prediction using deep learning techniques

Dr. Sebastian Fischer
seb1988@stanford.ed
Github

Predicting network traffic is of high relevance for Internet Service Providers, because it enables them to prepare for upcoming peak network utilizations better. This report illustrates how recurrent neural networks, specifically Long Short-Term Memory models, outperform classical autoregressive models (arima) in forecasting IP flow time series data. The analysis was based on a proprietary data set from Deutsche Telekom in Germany covering a time period from 2016 until 2019. It will become clear that even a very simple univariate analysis without extensive data preparation, hyperparameter search, or model architecture enhancements achieved moderately high performance measures.

INTRODUCTION

Internet Service Providers (ISP) face increasing traffic volumes, especially due to services like Netflix, Google (i.e. Youtube), Amazon (i.e. Amazon Prime, Amazon Web Services), and Facebook. The usage patterns of these services significantly influence the overall available capacity of ISP networks. Therefore, the ability to understand current and predict potential future capacity demands is of utmost importance. The project report at hand aims to address this need by applying deep learning techniques to predict future network traffic (i.e. IP flow data). More specifically, the report will show how three different Long Short-Term Memory (LSTM) architectures perform compared to a baseline auto-regressive integrated moving average (ARIMA) model. Extensive experiments on all four services (Google, Amazon, Facebook, Netflix) and all four models (one ARIMA, three LSTM) have shown that the deep learning techniques outperform the baseline ARIMA model, both in predicting the next 24 hours as well as the next 7 days. All models were trained on flow data which were measured from 2016 till 2019. This propiritory data set covers all traffic produced by the four observed services which was piped through the German IP backbone network of Deutsche Telekom AG (DT, largest telecommunications provider in Europe). It became clear, that especially training on longer past periods, which the LSTM approach was especially suitable for, lead to higher predicting performance. All models where univariate, hence only took the data traffic at a past time of day or hour as independent variable into account. Even this very basic approach lead to surprisingly accurate results in predicting data flows.

RELATED WORK

The world wide web is structured in so called “autonomous systems”. According to Bates et al (1995) an AS “is a group of IP networks operated by one or more network operator(s) [...]”. All ASs have an unique identification number - for instance DT is identified by AS3320. Furthermore, according to Zseby et al (2004) a “data flow” as depicted in Figure 1 “is defined as a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties”. An extensive overview of how analysing IP flow data can be used to detect network intrusion was given by Sperotto et al (2010). The authors claim that this kind of analysis is useful especially when analysing packet payloads is not feasible, i.e. due to privacy issues. Predicting flow data using machine learning is a challenge with high practical relevance. In this context, other authors have applied classical methods such as ARIMA (Papagiannaki et al, 2004) for forecasting but also PCA for better understanding the inner structure of flow data (Lakhina et al, 2004). Nowadays, the state of the art in time series data analysis are in fact Recurrent Neural Networks (RNN). Especially when analysing larger unstructured data, the use of RNNs lead to significant enhancements in the field of time series modeling (Schmidhuber, 2015). Especially when longer time periods are under observation Long Short-Term Memory cells (Hochreiter & Schmidhuber, 1997) are used, because they keep a "memory" about information that lay further back in a time series. Although LSTM architectures have been applied to structured time series data (see e.g. Gers et al, 2001) for prediction purposes, the application of this model family to IP flow data has not been widely conducted yet. This is a clear research gab that motivates the paper at hand.

DATA

By using specific measurement software, DT's subsidiary company *Benocs* collects flow data from within the DT network. Figure 2 shows that for each data flow Benocs keeps track of the source AS (i.e. Netflix, Facebook), the handover AS (the last AS that has the actual interface with the DT AS), as well as the NextHop AS (the AS via which the data flow leaves the DT AS) and the destination AS (i.e. another ISP). In case a data flow is consumed by a customer of DT, it will be terminated inside of the DT's AS. Nevertheless, what we cannot see is the *full* chain of ASs from source to destination.

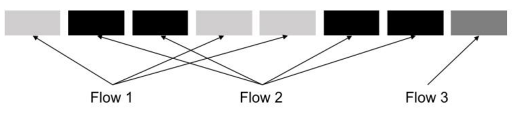

Figure 1: Data flows and packages

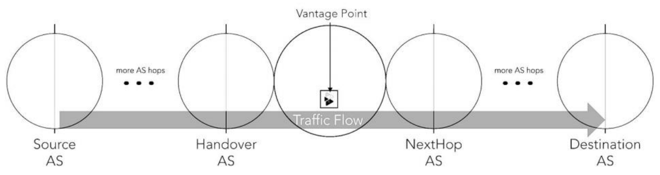

Figure 2: Autonomous systems (AS) from the view of Deutsche Telekom as vantage point

The following table 1 shows in each row an hourly aggregate of all flow data in GByte for each AS that is under observation including the total traffic. The Table has 24002 rows and ranges from 2016-05-31 till 2019-02-25.

time	Netflix	Facebook	Google	Amazon	Total
2019-02-25 19:00:00	1580.449	483.028	1949.896	917.490	4930.863
2019-02-25 20:00:00	1789.146	476.777	1892.020	1035.780	5193.723
2019-02-25 21:00:00	1596.300	394.469	1668.030	851.168	4509.967
2019-02-25 22:00:00	1210.791	242.347	1247.707	568.717	3269.562
2019-02-25 23:00:00	759.347	128.303	841.293	356.381	2085.324

Table 1: Hourly traffic for each service in GByte

Basically, table 1 and its daily aggregate are the single data input for training the four evaluated models. The following figure 3 shows a simple line chart that represents the daily traffic that flows through DTs network separated by the four observed services. It becomes clear that Google is responsible for most of the traffic, followed by Netflix, Amazon, and Facebook. Furthermore, one can clearly see an uprising trend for all services as well as some common drops that are probably caused by internal system issues. Additionally, Facebook significantly increased it capacity demand at the beginning of 2018.

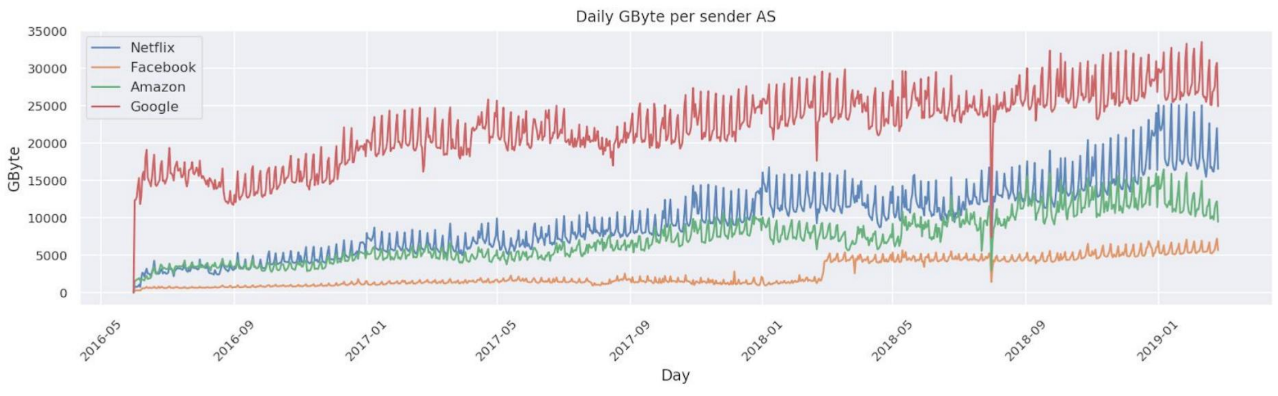

Figure 3: Traffic in GByte for each of the four observed ASs

MODELING

The following four models take the time series data as univariate input for training. Univariate means, that besides the time, no other additional independent variables are taken into account. Basically, all models try to predict the next x units (Days or Hours) given a set of previous observations (e.g. the past 7 days or the past 48 hours). To compare model performance, the author chose to use the RMSPE metric. RMSPE stands for root mean square percentage error and is calculated as follows:

$$RMSPE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \widehat{y}_i}{y_i}\right)^2}$$

In the formula y_i denotes the actual traffic volumes and y_i the predicted values. The RMSPE basically turns the commonly used RMSE metric into a percentage value in terms of the actual true values (and their scales). Because the traffic volumes of the different ASs are of different scales, using the RMSPE helps to compare the performance metrics among all models and ASs, without the need to normalize the input data. A note on hyperparameter tuning: in models 2, 3 and 4 the hyperparameters were hand picked by the author and not systematically explored using grid-search or other approaches. However, even with the hand picked hyperparameters a goal of this project, to show how deep learning techniques outperform a statistical baseline model, was achieved.

Model 1: “ARIMA” (baseline)

ARIMA stands for “autoregressive integrated moving average” and is a statistical model to forecast time series data. ARIMA does not build upon neural-network approaches that were covered in the class of cs230. Thereby this model shall not be explained in detail here, because it only serves as a baseline model to compare with the LSTM based models. However, it is worth to mention that the author used the Python library PMDARIMA to run the auto_arima function that finds the optimal hyperparameters (i.e. p, the lag order; d, the degree of differencing; q, the order of moving average) that leads to the best possible forecasting performance. This auto_arima function also takes care of the issue of stationarity. A time series is called “stationary” if the observations do not depend on time. For instance a randomly created time series is stationary, whereas in contrast a time series like the one we are dealing with in our case has trends and seasonal features is therefore not stationary because time is an influencing factor. A further comprehensible explanation of the ARIMA modeling approach can be either obtained from the pmdarima library documentation or alternatively from Duke University (Nau, n.d.). Furthermore, Brownlee (2018) has published an extensive overview of related classical time series forecasting methods in Python. The work of Portilla (2018) provides additional suggestions how to implement ARIMA in Python.

Model 2: “Basic LSTM”

In time-series data analysis one cannot simply split (or even randomly sample) the data set into train, development and test set because this would omit the underlying time dependency. Thereby, model 2, 3, and 4 are trained by gradually taking period after period into account and predicting the respective next period. Furthermore, a period is actually not a fixed time window, but a sliding window instead. So for example when predicting 7 days based on the previous 7 days, as training data input one would take a list of [day 1 to day 7] into account to predict [day 2 to day 8], then [day 2 to day 8] to predict [day 3 to day 9] etc. This approach is called “walk-forward validation” (see i.e. Brownlee (2016), Perera (2016)). The basic LSTM model architecture is illustrated by the table below. It is the output of the model.summary() function in Keras - in this case of a model that predicts the next 7 days. One can see, that the basic LSTM model consists of a single hidden LSTM layer with 200 units. The amount of units was taken from the suggestions given in Brownlee (2018b). After the hidden layer follows a dense layer of 100 units, which further condenses the learned features. To actually predict the next 7 days, the model’s last layer outputs a vector of 7 elements. All hidden layers take a ReLU activation. The optimizer used was ADAM. The model was trained with a batch size of 16 in 70 epochs. As mentioned above this configuration was hand picked in order to obtain a train-test loss figure that looks like Figure 4.

```
Architecture: Basic LSTM
Layer (type)                Output Shape      Param#
================================================
lstm_239 (LSTM)            (None, 200)       161600
dense_367 (Dense)           (None, 100)       20100
dense_368 (Dense)           (None, 7)         707
================================================
Total params: 182,407, Trainable params: 182,407
```

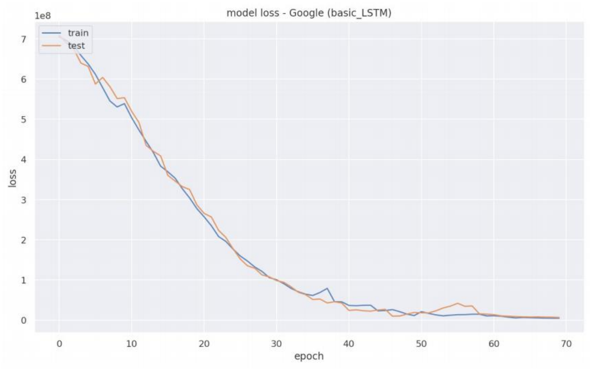

Figure 4: Decreasing model loss after 70 epochs

Model 3: “Encoder-Decoder LSTM”

The model consists of two steps, first an encoder reads the input sequence and encodes it so that a decoder can read this encoding to make a predictions for each of the output vector elements. As illustrated by the model.summary() output below, the model architecture starts with a 200 unit LSTM layer which functions as an encoder. This layer outputs one value for each unit (200 in total). These 200 output values represent learned features from the given input sequence. In an example of predicting the next 7 days, the encoder-decoder LSTM then repeats the encoding for all 7 days. This encoding is then fed into another 200 units LSTM layer that functions as a decoder, which outputs for each of the 7 days 200 values. Finally, these values are each separately further condensed by a fully connected layer of 100 units and then a final output layer of 1 unit (leading to 7 values, one for each day in this example). The Keras TimeDistributed wrapper function allows this combined approach for each of the 7 predicted values. This approach actually allows the model to learn the context of a day in the prediction period (e.g. one week), because the weights for each day in a week are shared throughout the learning process. This addresses the issue of weekend traffic volumes appearing to be different from days within the week.

```
Architecture: Encoder-Decoder LSTM
Layer (type)                 Output Shape      Param#
=====================================================
lstm_211 (LSTM)              (None, 200)       161600
repeat_vector_84 (RepeatVect (None, 7, 200)    0
lstm_212 (LSTM)              (None, 7, 200)    320800
time_distributed_167 (TimeDi (None, 7, 100)    20100
time_distributed_168 (TimeDi (None, 7, 1)      101
=====================================================
Total params: 502,601, Trainable params: 502,601
```

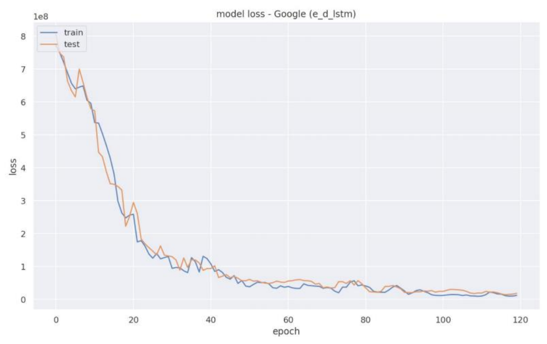

Figure 5: Decreasing model loss after 120 epochs

Model 4: “CNN Encoder-Decoder LSTM”

Model 4 is yet another model extension and replaces the encoder of model 3 with a 1-dimensional convolutional neural network unit. This alteration starts with a first 1-dimensional convolutional layer that directly reads in the input sequence and projects that onto a feature map. A second 1-dimensional convolutional layer repeats this mapping in the hope to even better extract latent features out of the input sequence. The output of the second convolutional layer is then fed into a max pooling layer of pool size 2, which means that a quarter of all values (the ones with the highest values) will be kept to further feed into the decoder part, which is the same as in model 3. For both convolutional layers 64 feature maps and a kernel size of three were used. Alternatives were tried by hand in a hyperparameter optimization process but did not significantly change the overall performance. As stated before, a more systematic hyperparameter search could lead to even better model performances - although it is already reasonably high as it will be shown in the results part of this report.

```
Architecture: CNN-Encoder-Decoder LSTM
Layer (type)                 Output Shape      Param#
=====================================================
conv1d_93 (Conv1D)           (None, 22, 64)    256
conv1d_94 (Conv1D)           (None, 20, 64)    12352
max_pooling1d_47 (MaxPooling (None, 10, 64)    0
flatten_47 (Flatten)         (None, 640)       0
repeat_vector_96 (RepeatVect (None, 24, 640)   0
lstm_231 (LSTM)              (None, 24, 200)   672800
time_distributed_191 (TimeDi (None, 24, 100)   20100
time_distributed_192 (TimeDi (None, 24, 1)     101
=====================================================
Total params: 705,609, Trainable params: 705,609
```

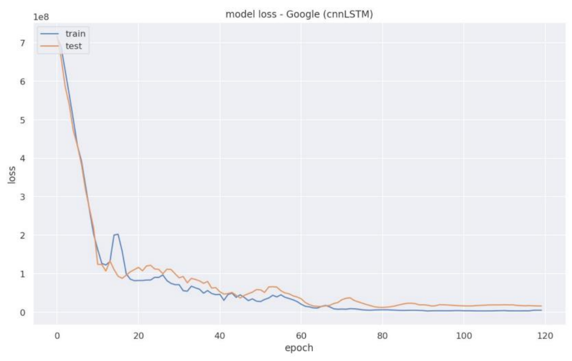

Figure 6: Decreasing model loss after 120 epochs

EXPERIMENTAL RESULTS

AS	Unit	best RMSPE	Model	History	Forecast
Amazon	Days	8.10%	basic_LSTM	49	7
Amazon	Days	9.61%	Arima	7	7
Amazon	Hours	25.59%	basic_LSTM	168	24
Amazon	Hours	54.68%	Arima	24	24
Facebook	Days	9.20%	cnnLSTM	49	7
Facebook	Days	9.96%	Arima	28	7
Facebook	Hours	28.18%	basic_LSTM	168	24
Facebook	Hours	46.54%	Arima	24	24
Google	Days	6.38%	e_d_lstm	35	7
Google	Days	7.88%	Arima	7	7
Google	Hours	34.99%	basic_LSTM	168	24
Google	Hours	45.26%	Arima	48	24
Netflix	Days	8.94%	basic_LSTM	49	7
Netflix	Days	11.52%	Arima	7	7
Netflix	Hours	46.90%	cnnLSTM	120	24
Netflix	Hours	152.09%	Arima	48	24
Total	Days	7.50%	basic_LSTM	49	7
Total	Days	9.20%	Arima	7	7
Total	Hours	35.41%	cnnLSTM	168	24
Total	Hours	59.91%	Arima	72	24

Table 2: Best Deep Learning Models vs. ARIMA baseline for each AS (daily & hourly)

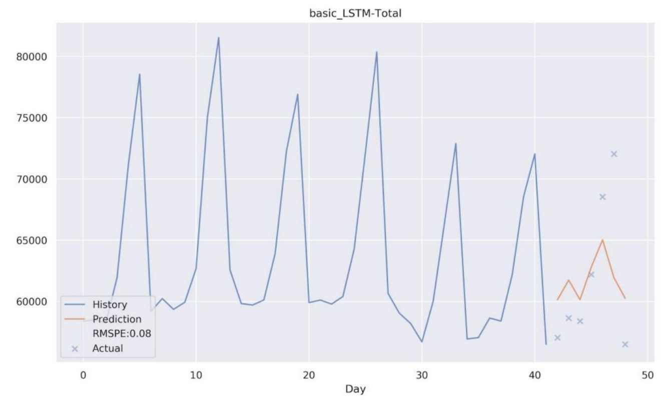

Figure 7: Performance of best model (Total)

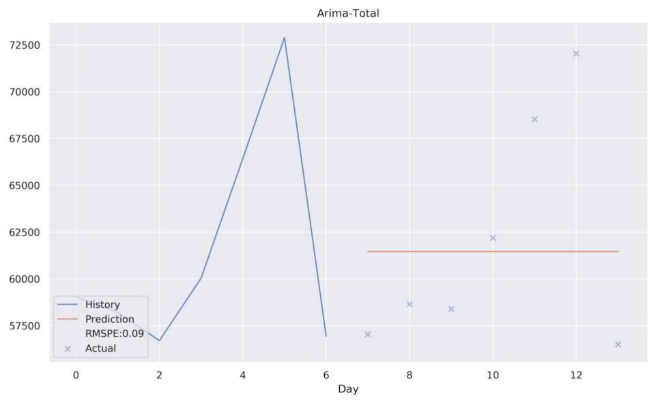

Figure 8: Performance of ARIMA model (Total)

Table 2 shows the *quantitative* assessment of a subset of the the trained models. For each AS, for both days and hours the best LSTM based model and the best ARIMA model is displayed. As mentioned before, the performance metric RMSPE is used to compare the model performances. One can see that in all cases the LSTM approaches perform much better than the ARIMA model. Additionally, it becomes clear that the LSTM models took longer past periods into account to come up with better predictions. Furthermore, one can see that a day-based prediction is significantly better among all models compared to the hour-based prediction. The interpretation of this, is that the relative differences of hours is much more volatile than the day-based measurements. Looking at Figures 7 and 8, one can see a *qualitative* assessment of the models. For illustration purposes, we can look at the Total traffic and compare the LSTM and ARIMA approach directly. Furthermore, in this example the ARIMA model clearly fails to predict values that follow the actual cyclic behaviour of the data. The LSTM models are much more capable of addressing this pattern and instead of the ARIMA they do not output a linear flat line as prediction.

CONCLUSION & FUTURE WORK

The report has shown that deep learning approaches outperform a statical model baseline. It became clear that even with a very simple univariate input and without a systematic hyperparameter search nor any data preprocessing or feature engineering reasonably high performance measures were possible. Although the RMSPE values suggest a good prediction performance, one would argue that the prediction performance should still be enhanced further when looking (qualitatively) at the forecasting graphs. This shall be achieved with a more systematic hyperparameter search and the exploration of further model architectures. However, the conducted experiments serve as a solid basis for these more explorative next steps. From a practical point of view, it is worth mentioning that since DT needs to adhere to European Data Privacy regulations, the data flows are not on an IP-level, but aggregated instead and thereby fully comply with the current legal framework in Europe.

REFERENCES

Bates, T., Gerich, E., Joncheray, L., J-M. Jouanigot, Karrenberg, D., Terpstra, M., & Yu, J. (1995). *Representation of IP Routing Policies in a Routing Registry (ripe-81++)* (No. RFC1786). RFC Editor. https://doi.org/10.17487/rfc1786

Brownlee, J. (2016, December 18). How To Backtest Machine Learning Models for Time Series Forecasting. Retrieved March 17, 2019, from https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/

Brownlee, J. (2018a, August 5). 11 Classical Time Series Forecasting Methods in Python (Cheat Sheet). Retrieved March 17, 2019, from https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

Brownlee, J. (2018b, October 10). How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption. (n.d.). Retrieved March 17, 2019, from https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/

Gers, F. A., Eck, D., & Schmidhuber, J. (2001). Applying LSTM to Time Series Predictable through Time-Window Approaches. In G. Dorffner, H. Bischof, & K. Hornik (Eds.), *Artificial Neural Networks — ICANN 2001* (pp. 669–676). Springer Berlin Heidelberg.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Lakhina, A., Papagiannaki, K., Crovella, M., Diot, C., Kolaczyk, E. D., & Taft, N. (2004). Structural analysis of network traffic flows. In *Proceedings of the joint international conference on Measurement and modeling of computer systems - SIGMETRICS 2004/PERFORMANCE 2004* (p. 61). New York, NY, USA: ACM Press. https://doi.org/10.1145/1005686.1005697

Nau, Robert. (n.d.). Introduction to ARIMA. Retrieved from https://people.duke.edu/~rnau/411arim.htm

Papagiannaki, K., Taft, N., Zhang, Z.-L., & Diot, C. (2005). Long-Term Forecasting of Internet Backbone Traffic. *IEEE Transactions on Neural Networks*, *16*(5), 1110–1124. https://doi.org/10.1109/TNN.2005.853437

Perera, S. (2016, June 3). Rolling Window Regression: a Simple Approach for Time Series Next value Predictions. Retrieved March 17, 2019, from https://medium.com/making-sense-of-data/time-series-next-value-prediction-using-regression-over-a-rolling-window-228f0acae363

Portilla, J. M. (2018, March 26). Using Python and Auto ARIMA to Forecast Seasonal Time Series. Retrieved March 17, 2019, from https://medium.com/@josemarcialportilla/using-python-and-auto-arima-to-forecast-seasonal-time-series-90877adff03c

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., & Stiller, B. (2010). An Overview of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys & Tutorials*, *12*(3), 343–356. https://doi.org/10.1109/SURV.2010.032210.00054

Zseby, T., Quittek, J., Claise, B., & Zander, S. (2004). Requirements for IP Flow Information Export (IPFIX). Retrieved March 17, 2019, from https://tools.ietf.org/html/rfc3917

KERAS - Keras Documentation. (n.d.). Retrieved March 17, 2019, from https://keras.io/

MATPLOTLIB: Python plotting — Matplotlib 3.0.3 documentation. (n.d.). Retrieved March 17, 2019, from https://matplotlib.org/

NUMPY - NumPy. (n.d.). Retrieved March 17, 2019, from http://www.numpy.org/

PANDAS - Python Data Analysis Library — pandas: Python Data Analysis Library. (n.d.). Retrieved March 17, 2019, from https://pandas.pydata.org/

SCIKIT-LEARN: machine learning in Python — scikit-learn 0.20.3 documentation. (n.d.). Retrieved March 17, 2019, from https://scikit-learn.org/stable/

SCIPY - SciPy.org. (n.d.). Retrieved March 17, 2019, from https://www.scipy.org/

SEABORN: statistical data visualization — seaborn 0.9.0 documentation. (n.d.). Retrieved March 17, 2019, from https://seaborn.pydata.org/

PMDARIMA - Smith, T. (n.d.). pmdarima: ARIMA estimators for Python — pmdarima 1.1.0 documentation. Retrieved March 17, 2019, from https://www.alkaline-ml.com/pmdarima/index.html

STATSMODELS: Statistics in Python — statsmodels 0.9.0 documentation. (n.d.). Retrieved March 17, 2019, from https://www.statsmodels.org/stable/index.html

TENSORFLOW. (n.d.). Retrieved March 17, 2019, from https://www.tensorflow.org/