Enhancing Real-time Network Traffic Scenario Prediction with GRU-based RNN Architecture

Snehil Sharma

School of Computer Engineering,
KIIT Deemed to be University,
Bhubaneswar 751024, India;
2005545@kiit.ac.in

Abstract

Network Traffic Scenario Prediction is a critical aspect of network management. This research paper outlines my methodology for tackling the problem. In response to the challenge of classifying traffic scenarios, I present a novel approach based on recurrent neural networks, as employed in my submission. The proposed model offers predictions for scenario labels at every time step in a continuous sequence of recorded network traffic data. Through rigorous evaluation on an unseen test dataset, the model achieved an impressive accuracy of 75%. This work contributes to the field of network management by providing an effective solution for traffic scenario classification, promising improved network monitoring and control.

Keywords – Machine Learning, Network Traffic, Recurrent Neural Network, Gated Recurrent Unit, Network Resource Allocation

I. Introduction

Network Traffic Scenario Prediction holds a pivotal role in the realm of network management, where the ability to anticipate and classify various traffic scenarios is of utmost importance. This research paper aims to provide a comprehensive methodology for addressing this crucial challenge. The classification of traffic scenarios is integral for optimizing network parameters, facilitating intelligent fault identification, and implementing efficient congestion control measures. The approach detailed herein is a pioneering one, leveraging recurrent neural networks to offer real-time predictions of scenario labels at each time step within a continuous stream of recorded network traffic data [1].

Understanding the diverse range of scenarios is vital, as it enables the network management system to tailor its response, ensuring optimal performance and stability. For instance, in intelligent network operation and maintenance, identifying different fault types through traffic performance analysis is indispensable for rapid problem resolution. Similarly, in the context of congestion control, adjusting rate control parameters based on traffic performance becomes imperative for delivering high throughput, low latency, and minimal packet loss. Nevertheless, the key challenge in this domain lies in the fact that the status parameters of traffic across different scenarios often exhibit subtle variations, and the number of scenarios within a given timeframe can be unpredictable, complicating their identification. Furthermore, in real-time decision-making scenarios, where responses must be delivered within milliseconds or even microseconds, the complexity of the problem increases exponentially [2].

In the course of data transmission and forwarding, network traffic converges at the same port queue from multiple ports. This influx of traffic often exceeds the output capacity, necessitating the allocation of cache space to temporarily store burst data. Consequently, this leads to the formation of cache queues at the egress ports of forwarding devices. The length and dynamics of these cache queues are influenced by the types of forwarding traffic. For example, queue fluctuations induced by varying stream ratios and workloads differ significantly across different service types such as web search and distributed computing. As network applications evolve and shift, the traffic characteristics received by these queues continuously change. Presently, the collection of status parameters, including input rates (v_{in}), output rates (v_{out}), real-time cache queue length (q), and their corresponding timestamps (t), is performed at fixed time intervals. The output rate (v_{out}) is constrained by the port's bottleneck rate (v_{max}), where $v_{out} <= v_{max}$, while the input rate is subject to burst traffic and regulation protocols. This implies that the maximum input rate may surpass the bottleneck rate, posing additional complexity. Input and output rates are quantified by measuring the amount of data transmitted and received within each fixed time interval, forming the basis for analysis and prediction in this research [3].

The remainder of this paper is structured as follows: Section II provides a concise overview of the model architecture and dataset preparation for this study. Section III delves into the essential techniques required for data preprocessing and transformation. Section IV outlines the training and optimization for this study. Section V details the results achieved with the applied models, encompassing quality metrics and algorithm execution times. The final section concludes the paper and outlines future directions for research.

II. Architectural Design of the Model and Dataset Preprocessing

The success of any data-driven endeavor hinges on the careful orchestration of multiple components, including data preparation, model selection, and performance evaluation metrics. In this section, we provide a detailed exposition of our model's architecture, the specific model we employed, and the metrics that underpin the assessment of its performance.

A. Applied Models – Gated Recurrent Unit (GRU)

The selection of the GRU is not arbitrary but strategic. We have chosen the GRU as it provides a powerful framework for capturing sequential dependencies within the data. Notably, we've implemented a bidirectional variant of the GRU. This choice is pivotal because it empowers the model to grasp contextual information from both past and future time steps, significantly enhancing its predictive capabilities. By employing a bidirectional GRU layer, our model can effectively unearth temporal dependencies in both forward and backward directions, ultimately leading to more accurate traffic scenario predictions.

B. Dataset Formatting

To harmonize our chosen model with the dataset, I've meticulously formatted the data. The input to the model consists of feature values representing '*time*,' '*portPktIn*,' '*portPktOut*,' and '*qSize*' for each time step within a specific sequence length of time steps. In parallel, the target values entail the corresponding '*label*' feature for each time step within the sequence. This alignment ensures that the model receives the necessary input to make predictions based on the temporal sequence.

C. Model Architecture

Our final model architecture, meticulously crafted with PyTorch, is constructed with precision. It boasts three GRU layers, each having a $hidden_{size}$ set to 64 and an $input_{size}$ equivalent to the number of features, which in our case amounts to four. The pivotal bidirectional setting is enabled, ensuring that the model's predictive power extends in both temporal directions.

D. Metrics Utilized

To gauge the model's performance effectively, a range of metrics have been meticulously selected. These metrics encompass essential evaluation criteria to assess the model's efficacy in traffic scenario prediction. They include but are not limited to accuracy, learning rate, loss and possibly additional domain-specific metrics tailored to the specific nuances of network traffic scenario prediction.

In sum, this holistic approach unifies the dataset, model architecture, and evaluation metrics into a coherent framework designed to yield accurate and reliable network traffic scenario predictions.

III. Data Preprocessing and Transformation

In the realm of data-driven research, the journey begins with the careful preparation and transformation of the dataset. This pivotal phase, often underappreciated, lays the foundation for robust and accurate model training. In this section, we delve into the intricate world of data preprocessing and transformation, employing a methodical approach to enhance the effectiveness and efficiency of our model training.

A. Feature Scaling for Robustness

To ensure that our model is resilient in the face of outliers and variations in data distribution, we employ feature scaling with RobustScaler. This meticulous step is not to be overlooked, as it guards against data irregularities that may plague real-world datasets. RobustScaler is our method of choice due to its adept handling of data points that deviate from a normal distribution. By leveraging robust statistical measures such as the median and interquartile range, RobustScaler standardizes our features, safeguarding the model against undue influence from extreme values.

B. Sequence Padding for Consistency

In the dynamic landscape of batch processing, where computational efficiency is paramount, uniformity in data handling is a non-negotiable requirement. We prepare our dataset for this challenging task by strategically padding sequences to a fixed length. This astute maneuver not only streamlines the processing pipeline but also ensures consistency in handling sequences of varying lengths, a common occurrence in real-world datasets. The ability to seamlessly manage this diversity is a hallmark of a well-crafted model.

C. Introducing an Additional Class

A critical juncture in the preprocessing journey is the introduction of an additional class. This seemingly minor but pivotal step distinguishes our approach. The purpose of this added class is to demarcate padded values, thus endowing the model with the intelligence to discern between real data and padding. The ramifications of this decision are profound, as it enables the model to effectively ignore the padded values during both training and prediction phases. This deliberate exclusion prevents any inadvertent interference with the learning process, safeguarding the integrity of the model's predictions.

D. Optimizing Memory Utilization and Computation

In the era of modern parallel hardware, particularly Graphics Processing Units (GPUs), optimizing memory utilization and computational efficiency is imperative. Our chosen approach not only enhances the model's effectiveness but also streamlines resource utilization. By preventing undue computational burden and memory consumption during batch processing, our strategy aligns seamlessly with the capabilities of advanced hardware, ensuring that the model operates at peak efficiency.

IV. Model Training and Fine-Tuning

In the realm of neural network research and development, the choice of framework plays a pivotal role. In our pursuit of training and optimizing a model for our specific task, we have turned to the robust PyTorch framework. This section provides a detailed insight into the strategies and techniques employed for the effective training and optimization of our model.

A. Data Splitting and Validation

A fundamental aspect of model development is the division of data into training and validation sets. To ensure the integrity of our model, we adopted a unique approach. Data splitting was conducted based on log IDs, where a randomized selection of IDs from the dataset was allocated to a validation split of 15%. This methodological decision guarantees the model's exposure to a representative cross-section of data for validation, enhancing its generalization capabilities.

B. Batch Training

Efficiency in model training hinges on effective batch processing. Our model was trained by feeding it data in batches of 4. Each batch consisted of sequences with a fixed length of 5000, representing a contiguous series of input features, namely '*time*,' '*portPktIn*,' '*portPktOut*,' and '*qSize*.' This approach optimizes memory utilization and computational efficiency, paving the way for a streamlined training process.

C. Loss Function and Optimization Algorithm

The choice of loss function and optimization algorithm is instrumental in achieving stable convergence and optimal results. For our multi-class classification task, we harnessed the power of CrossEntropyLoss. This loss function is a well-suited choice, enabling the model to effectively measure the discrepancy between predicted and actual labels. In the pursuit of convergence, the Adam optimizer was employed, equipped with a learning rate of 0.001. This dynamic optimizer adapts to the data's characteristics, facilitating the model's journey towards optimal accuracy.

D. Learning Rate Adaptation

In the dynamic landscape of model training, adaptability is key. To further enhance training efficiency and precision, we introduced a learning rate scheduler, more specifically the ReduceLROnPlateau. This adaptive strategy dynamically regulates the learning rate during training based on validation accuracy. By reducing the learning rate as the model progresses, faster convergence is achieved in the initial phases, while ensuring fine-tuning in the later stages. This adaptive approach aligns seamlessly with the model's evolving needs.

E. Model Checkpointing and Early Stopping

To safeguard against training stagnation and to retain the best model configuration, we implemented a model checkpointing mechanism. During training, the model's progress was closely monitored, and checkpoints were created based on the best validation accuracy achieved. Additionally, an early stopping criterion was introduced. If the model's validation accuracy failed to improve over a span of 10 epochs, the training process was gracefully halted. This strategic decision prevents unnecessary computational overhead and aligns with the principle of efficiency in model development.

V. Result Analysis

In this study, the modeling was conducted within the following environments: Python 3.11 programming language and the VScode software were utilized for building the prediction algorithm. Additionally, various libraries for data analysis, including Numpy, Pandas, Scikit-learn, PyTorch and Keras were employed.

The problem is framed within the domain of machine learning, and the outcomes of the applied algorithms are presented in the table. The problem is characterized by the following:

1) Problem type: Regression
2) Target Variable: Network Traffic
3) Features used for prediction:
   - Time, Port Packet In, Port Packet Out, Cache Size
   - Binary features representing Labels for network communication cells ($label$)
4) Evaluation Metrics: Accuracy, Learning Rate and Loss

The ultimate measure of the success of any machine learning model lies in its results. In this section, we delve into the detailed analysis of the outcomes of our modeling approach, shedding light on the promise it holds for real-time traffic classification.

Our model's performance is underscored by the validation accuracy score of 0.74172 (74%) and an even more impressive accuracy score of 0.75381 (75%) on the unseen testset. These scores serve as a testament to the robustness and reliability of the model in effectively classifying traffic scenarios. The model's ability to deliver consistently high accuracy on both validation and unseen datasets is indicative of its potential for practical applications.

Efficiency, in terms of both training and inference times, is a crucial factor in real-time applications. The training process on Google's 15 GB Tesla K80 GPU was completed in approximately 17 minutes, a commendably swift timeline. Dataset preparation, a vital precursor to training, took an additional 15 minutes, a testament to the streamlined data handling pipeline.

Remarkably, when confronted with the substantial $23,38,000$-row unseen testset, dataset preparation was achieved in just one minute. The model's inference time proved to be remarkably swift, requiring a mere second with a batch size of 16. This remarkable efficiency speaks to the model's potential for real-time deployment, offering rapid decision-making capabilities in dynamic network management scenarios.

It's worth noting that even with the same model architecture, we have explored an alternative approach to sequence lengths. By opting for input sequence lengths that align with the maximum sequence length in the dataset (149000), rather than an arbitrary 5000, not only do we obtain very similar accuracy scores, but we also significantly reduce dataset preparation time. This adjustment leads to dataset preparation times of approximately 1 minute and 30 seconds for the training dataset and only a few seconds for the inference dataset. This alternative approach showcases the flexibility of the model and its adaptability to various data preprocessing strategies.

VI. Conclusion

In the dynamic landscape of network traffic management, the demand for precision, efficiency, and real-time decision-making is insatiable. It is in this context that our exploration into a GRU-based Recurrent Neural Network (RNN) architecture takes center stage. As we draw the curtains on our study, we are presented with a compelling narrative of effectiveness and efficiency, with promises of far-reaching real-world applications.

The cornerstone of our research journey is the GRU-based RNN architecture, meticulously designed to grapple with the complexities of continuous-time traffic data. Through rigorous experimentation and analysis, we have unearthed its remarkable effectiveness in the realm of scenario prediction at each time step. The model's ability to make accurate predictions, informed by the temporal dynamics of network traffic, is a testament to its robustness and reliability.

While our study showcases the immense potential of the GRU-based RNN architecture, it is crucial to acknowledge that the journey is far from over. Further refinements and in-depth analyses are the stepping stones towards validating the model's real-world application. The intricacies of network management are often far more convoluted than what can be simulated in controlled environments, demanding a rigorous and methodical approach to ensure applicability.

In the landscape of network traffic management, where every second counts, our model emerges as a beacon of hope. Its potential for enhancing real-time decision-making and optimization is profound. By enabling network administrators to anticipate and respond to traffic scenarios in real-time, our model holds the promise of reducing downtime, enhancing resource allocation, and ultimately delivering a superior network management experience. The implications of our work extend to a plethora of practical scenarios, from intelligent fault detection to congestion control, making it a versatile tool in the hands of network professionals.

In the grand scheme of things, our research on the GRU-based RNN architecture is but a stepping stone in the larger journey of optimizing network traffic management. As we close this chapter, we do so with a sense of optimism, knowing that the possibilities for refinement and expansion are boundless. It is through the tireless efforts of researchers and practitioners in the field that the vision of seamless, efficient, and real-time network traffic management will continue to evolve, with our model as a valuable contribution to this ongoing narrative.

References

[1] Agarap, Abien Fred M. "A neural network architecture combining gated recurrent unit (GRU) and support vector machine (SVM) for intrusion detection in network traffic data." *Proceedings of the 2018 10th international conference on machine learning and computing*. 2018.

[2] R. Dey and F. M. Salem, "Gate-variants of Gated Recurrent Unit (GRU) neural networks," 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, USA, 2017, pp. 1597-1600, doi: 10.1109/MWSCAS.2017.8053243.

[3] Fu, Rui, Zuo Zhang, and Li Li. "Using LSTM and GRU neural network methods for traffic flow prediction." *2016 31st Youth academic annual conference of Chinese association of automation (YAC)*. IEEE, 2016.