ITU Network Traffic Scenario Prediction Challenge

Daniel Bruintjies
dmbruintjies@gmail.com

October 2nd, 2023

Abstract – This report details my approach to the Network Traffic Scenario Prediction Challenge by ITU, a competition hosted on the Zindi platform. To address this challenge of traffic scenario classification in network management, I propose, as used in my submission, a model based on recurrent neural networks. The proposed model predicts scenario labels for each time step within a continuous sequence of logged network traffic data. The model was evaluated against an unseen testset to have an accuracy score of 0.75.

1. Introduction

Network traffic scenario classification is a critical aspect of network optimization and management. Accurate classification enables targeted adjustments of network parameters to be made and allows for effective control measures to be carried out. To address this challenge of traffic scenario classification, I propose, as used in my code submission to the Network Traffic Scenario Prediction Challenge by ITU, a deep learning model based on recurrent neural networks, capable of predicing the traffic scenario at each time step.

2. Model Architecture and Dataset Preparation

The dataset comprises of samples of network traffic log data. Each sample contains sequential numeric feature data across timesteps of a continuous period, with the objective to predict a traffic scenario label at each time step within the sequences. To model this dataset, a model architecture based on recurrent neural networks is chosen, namely, the Gated Recurrent Unit (GRU). The choice of a bidirectional architecture is also fundamental to capture both past and future context for each time step. By using a bidirectional GRU layer, the model can effectively learn temporal dependencies in both forward and backward directions, enhancing its ability to make accurate predictions for each timestep. To make use of these architectures effectively, the dataset is prepared in such a way that, for input to the model we have the features values of 'time', 'portPktIn', 'portPktOut' and 'qSize' for each timestep across a specific sequence length of timesteps, and for the target, we have values of corresponding 'label' feature for each timestep in the sequence. The final model architecture, built using Pytorch, is constructed to have 3 GRU layers with hidden_size set to 64, input_size set to the number of features (4) and bidirectional set as True. The last GRU layer connects to the final layer of the model, a linear classiffication layer, where the out_features equal to the number of class labels in the dataset after preprocessing.

3. Preprocessing and Data Transformation

The dataset undergoes meticulous preprocessing to facilitate effective and efficient model training. This includes scaling the features using RobustScaler to ensure robustness against outliers and inconsistencies in data distribution. RobustScaler is chosen due to its appropriateness for data that may contain outliers or deviate from a normal distribution, utilizing the median and interquartile range.

Additionally, the dataset is prepared for uniform handling during batch processing by padding sequences to a fixed length. This ensures computational efficiency and consistency in handling varied sequence lengths, critical in real-world datasets. A key step in this process involves adding an additional class to distinguish padded values, enabling the model to effectively ignore the padded values during training and prediction, thus preventing interference with the learning process. This approach optimizes memory utilization and computation during batch processing, especially on modern parallel hardware like GPUs.

4. Training and Optimization

The Pytorch framework was chosen for building and training the model. The data was split into training and validation sequences by log ids by randomly selecting ids from the dataset for a 0.15 validation split. The data was given to the model in batches of 4 during training, where each batch contains sequence of length of 5000, representing the consecutive input features, 'time', 'portPktIn', 'portPktOut' and 'qSize'. The loss function used is CrossEntropyLoss, a suitable choice for multi-class classification tasks. For stable convergence to an optimal validation accuracy, the Adam optimizer is employed, with a learning rate of 0.001. Furthermore, a learning rate scheduler, specifically ReduceLROnPlateau, is employed to dynamically reduce the learning rate during training based on validation accuracy. This adaptive approach helps in faster convergence during the initial training phases and precise fine-tuning as the training progresses. The model is checkpointed during training based on the best validation accuracy. If the validation accuracy does not improve for 10 epochs, then the model training is stopped.

5. Result Analysis and Discussion

This modelling approach resulted in a validation accuracy score of 0.76937, and an accuracy score of 0.75084 on the unseen testset on Zindi's leaderboard. The training took ~23 minuntes on a 16GB P100 GPU and training dataset preparation took ~15 minuntes. On the 2,338,000 row unseen testset, dataset preparation took 1 minute 20 seconds and inference time took only 1 second with batchsize of 16. With the relatively good accuracy scores, fast training time, and fast inference time, this model holds great promise as a viable solution to the task of real-time traffic scenario classification. It should also be noted that, with the same model architecture, very similar scores are obtainable if we opt for the input sequence lengths to rather be equal to the max sequence length (of 149000) across the dataset, instead of an arbitrary 5000. This choice will also result in a significant boost in dataset preparation time, to around roughy 1 minute for the training dataset, and only a few seconds on inference dataset, as fewer computations need be carried out. To possibly improve on the models accuracy scores, one could attempt to make the network architecture more complex by incorporating more RNN layers and/or possibly add more features to the model by incorporating a good feature engineering strategy.

6. Conclusion

This GRU-based RNN architecture demonstrates effectiveness and efficiency in handling continuous-time traffic data for scenario prediction at each timestep. Further refinements and analyses are needed to validate its real-world application. The model holds potential for enhancing real-time decision-making and optimization in network traffic management.