|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | INTERNATIONAL TELECOMMUNICATION UNION  **TELECOMMUNICATION STANDARDIZATION SECTOR**  STUDY PERIOD 2017-2020 | | **GNN CHALLENGE** | |
|  | |
| **Original: English** | |
| **Question(s):** | | N/A | Virtual, TBD, 2023 | |
| **INPUT DOCUMENT** | | | | |
| **Source:** | | Team Yanbiyu | | |
| **Title:** | | Graph Neural Networking Challenge 2023 - Creating a Network Digital Twin with Real Network Data | | |
| **Contact:** | | Blessed Guda,  Carnegie Mellon University, Africa | | Email:[**blessedg@andrew.cmu.edu**](mailto:blessedg@andrew.cmu.edu) |

|  |  |  |
| --- | --- | --- |
| **Contact:** | Carlee Joe-Wong  Carnegie Mellon University (Team Mentor) | Email: **cjoewong@andrew.cmu.edu** |

**1.0 PROBLEM STATEMENT**

Recent advances in networking have led to the development of powerful Graph Neural Networks (GNNs) that can mimic complex network environments effectively. These innovations have enabled the creation of lightweight and real-time Network Digital Twins. However, the absence of real-world data has hindered the understanding of how these models perform in authentic network scenarios. The 2023 edition of the Graph Neural Networking challenge aims to address this gap by tasking participants with building a GNN-based Network Digital Twin using real network data. The challenge involves modeling traffic, topology, and configuration to estimate network performance accurately, specifically focusing on delay. Participants are required to predict mean delay for each flow based on network topology, flow packet traces, and routing configuration. Therefore, the requirements of participating solutions are to predict the mean delay for each given flow given the network's topological layout (L2 and L3), a set of input flow packet traces, and the routing configuration.

**2.0 APPROACH TAKEN BY THE TEAM**

1. The features extracted from the data are the same as the default features specified in the baseline GitHub repository[1].
2. The team only made slight modifications to the Routenet fermi architecture to reach the performance it got during the challenge. The changes made are reported in the table below:

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Activation Functions | GELU (all relu activations were replaced) |
| Max\_Epochs | 50 |
| Path State Dim | 32 |
| Hidden State Dim | 32 |
| Regularization [ Link Update RNN] | L2-regularization with alpha = 0.01 |

1. For the top submission made by the team TensorFlow was used.

**OTHER KEY LEARNINGS BY THE TEAM FOR OTHER EXPERIMENTS CARRIED OUT.**

1. Minimizing the MSE instead of MAPE did not help out.
2. Adding more layers to the GRUcell blocks, readouts or embedding layers does not help out.
3. Using an LSTMcell instead of GRUcell did not help out.
4. The team is also currently trying out experiments with transformer encoders.

The motivation of using transformers is for the model to be able to identify the links/nodes in the path that contribute more to the cumulative delay. To this, two experiments were carried out:

1. Formulating the problem as a sequence “classification” problem.

For an input flow that traverses k links, we get the link embeddings, and an input flow embedding, we concatenate the flow embeddings to the link embeddings to form a sequence. The sequence was passed through an encoder to get the hidden state that is passed to the readout to make predictions. The model started at a validation MAPE of 56.8 after epoch 1, but takes a very long time to train. We were only able to train for 5 epochs and the loss was decreasing but quite very slowly.

1. To replace only the path state GRUcell in the Routenet architecture with the transformer decoder, while re-using the link GRUcell. << In progress>>
2. It only took the model 24 epochs to get to the reported performance level [validation MAPE = 41.23%]
3. The team has also provided an implementation of RouteNet fermi in Pytorch. This can be a helpful resource to the research community [2]. Also, our implementation in Pytorch has the advantage of allowing back-propagating the loss in mini-batches instead of just using batch size =1. However, we couldn’t find an equivalent TensorflowRaggedTensor data type in Pytorch made us to use several for loops to achieve the same functions implemented in torch.

The procedures to make inference or train the model is the same as those in the baseline GitHub repository.

[1] <https://github.com/BNN-UPC/GNNetworkingChallenge/tree/2023_RealNetworkDT>

[2] <https://github.com/gblessed/gnn_challenge/tree/main>