Temporal Link Prediction Using Graph Neural Networks

*Term Project Report for WSDM 2022 Challenge

1st Abdurrahman Kürşat Özkan
Dept. of Computer Science
Akdeniz University
Antalya, Türkiye
20210808047

2nd Alperen Cevahiroğlu
Dept. of Computer Science
Akdeniz University
Antalya, Türkiye
20200808016

***Abstract*—We tackle temporal link prediction on the WSDM 2022 Challenge Dataset A, a sequence of timestamped node interactions. Our pipeline comprises custom preprocessing, a Temporal Graph Network built with PyTorch Geometric Temporal's GConvGRU, and end-to-end training with negative sampling. Inference on held-out query intervals yields an AUC of 0.4639. We discuss key design choices, implementation details, and avenues for improving predictive performance.**

***Index Terms*—temporal link prediction, dynamic graphs, TGN, GConvGRU, PyTorch Geometric Temporal, WSDM Challenge**

I. Introduction

Temporal link prediction aims to forecast future edges in a time-evolving graph. Unlike static link prediction, it must capture both structural context and temporal dynamics. The WSDM 2022 Challenge Dataset A provides a real-world benchmark of timestamped node interactions. We implement a Temporal Graph Network (TGN) that maintains per-node memory, processes edge events via graph convolutions, and scores candidate links using an MLP.

II. Methodology

A. Preprocessing

Our `preprocess.py` script performs:

1) **Column naming.** Assign headers (`src`, `dst`, `etype`, `ts`) to raw CSVs.
2) **ID remapping.** Map original node and edge-type IDs to contiguous zero-based integers; dump `node_id_map.json` and `etype_id_map.json`.
3) **Sampling.** Optionally subsample edges or stratify by timestamp bins.
4) **Feature loading.** Read node features and edge-type features from CSVs, convert to NumPy, then to PyTorch tensors.
5) **Data splitting.** Retain `edges_train_A_mapped.csv` for training; `input_A.csv` defines positive/negative queries with labels.

This work was conducted as part of the CSE 5074 Social Network Analysis course at Akdeniz University.

B. Dataset & Loader

We define `TemporalLinkPredictionDataset`, which:

- Loads remapped edges and unique timestamps.
- Preloads `node_features` $[N \times F]$ and `etype_feats` $[T \times E]$.
- Implements `__len__()` as the number of evaluation intervals.
- In `__getitem__()`, for interval $(t, t_{+1}]$:
  - Builds history graph of edges with $ts \leq t$.
  - Samples positive pairs in the next interval.
  - Generates negative samples uniformly over nodes and etypes (ratio 1.0).
  - Returns a minibatch dict with keys: `x`, `edge_index`, `pairs`, `labels`, `etype_feats`.

A PyTorch `DataLoader` streams this dataset (batch_size=None).

C. Model Architecture

We implement `GConvGRULinkPredictor`:

- **GConvGRU.** A temporal graph convolutional GRU ($K = 2$ hops) that updates node embeddings $h \in \mathbb{R}^{N \times H}$ given event graph $\mathbf{E}$.
- **Link Predictor.** An MLP:

$$\mathrm{MLP}(\,[h_s \,\|\, h_d \,\|\, e_f]\,) \to H \xrightarrow{\mathrm{ReLU}} 1,$$

where $[\,\cdot \| \cdot\,]$ denotes concatenation of source/destination hidden states and the edge-type feature.

Input dims:

$F =$ node_features.shape[1], $\quad E =$ etype_feats.shape[1], $\quad H = 64$.

D. Training Setup

- **Optimizer:** AdamW (lr $= 1 \times 10^{-4}$, weight decay $= 0.01$).
- **Loss:** `BCEWithLogitsLoss` on positive/negative labels.
- **Negative sampling ratio:** 1.0 (one negative per positive).

- **Epochs:** 10.
- **Device:** GPU if available (noted as "Using device: cuda" on V100).
- **Batching:** Streaming per-interval batches via `DataLoader`.
- **Checkpointing:** Save model when epoch avg. loss improves.

III. Experimental Results

After training, we run inference on `input_A.csv` and compute ROC AUC. Results are shown in Table I.

TABLE I
Test AUC on Dataset A

Metric	Value
ROC AUC	0.4639

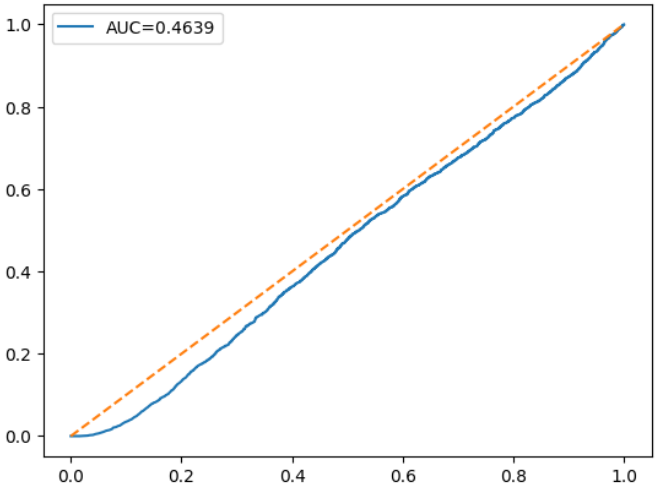

Fig. 1. ROC Curve for Dataset A

The notebook also saves `output/output_A.csv` with predicted probabilities.

IV. Insights and Discussion

- **Moderate performance.** An AUC of 0.4639 indicates the model performs slightly below random on the held-out intervals.
- **Data challenges.** The temporal splits in `input_A.csv` may contain sparse or shifting node activity.
- **Hyperparameters.** Hidden size $H = 64$ and 1:1 negative sampling may underfit; larger H or varied sampling could help.
- **Future tuning.** Learning rate schedules, increased epochs, or advanced message-passing (e.g. attention) may improve results.

V. Conclusion

We present an end-to-end pipeline for temporal link prediction using a GConvGRU-based TGN. Despite a modest AUC of 0.4639 on the WSDM 2022 Challenge Dataset A, our modular design and preprocessing facilitate rapid experimentation. Future work will explore hyperparameter sweeps, richer edge features, and real-time online updates.

References

[1] WSDM 2022 Challenge, "Temporal Link Prediction," https://www.dgl.ai/WSDM2022-Challenge/.
[2] F. Rossi *et al.*, "PyTorch Geometric Temporal," *arXiv:2006.10637*, 2020.