Extended Abstract Track

Enhancing the Expressivity of Temporal Graph Networks through Source-Target Identification

Editors: List of editors' names

Abstract

Despite the successful application of Temporal Graph Networks (TGNs) for tasks such as dynamic node classification and link prediction, they still perform poorly on the task of dynamic node affinity prediction – where the goal is to predict 'how much' two nodes will interact in the future. In fact, simple heuristic approaches such as persistent forecasts and moving averages over *ground-truth labels* significantly and consistently outperform TGNs. Building on this observation, we find that computing heuristics *over messages* is an equally competitive approach, outperforming TGN and all current temporal graph (TG) models on dynamic node affinity prediction. In this paper, we prove that no formulation of TGN can represent persistent forecasting or moving averages over messages, and propose to enhance the expressivity of TGNs by adding source-target identification to each interaction event message. We show that this modification is required to represent persistent forecasting, moving averages, and the broader class of autoregressive models over messages. Our proposed method, TGNv2, significantly outperforms TGN and all current TG models on all Temporal Graph Benchmark (TGB) dynamic node affinity prediction datasets.

Keywords: Graph Neural Networks, Temporal Graphs, Dynamic Node Affinity Prediction

1. Introduction

Temporal Graph (TG) models have become increasingly popular in recent years (Xue et al., 2021; Skarding et al., 2021; Feng et al., 2024) due to their suitability for modeling a range of real-world systems as dynamic graphs that evolve through time, e.g. social networks, traffic networks, and physical systems (Deng et al., 2019; Song et al., 2019; Zhao et al., 2020; Guo et al., 2019; Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021). Unlike static graphs, dynamic graphs allow the addition of nodes and edges, and graph features to change over time. Despite the successes of current TG models for dynamic node classification and link prediction, they have been shown to struggle in *dynamic node affinity prediction*, being significantly outperformed by simple heuristics such as persistent forecasting and moving average over ground-truth labels (Huang et al., 2023; Yu, 2023).

In dynamic node affinity prediction, the task is to predict a node's future 'affinity' for other nodes given the temporal evolution of the graph. Informally, the affinity of a node x towards a node y over some time interval $[t, t+\delta]$ refers to how much x has interacted with y over that interval. For example, if node A sends 10 identical messages to node B and 100 of the same messages to node C over some time interval $[t, t+\delta]$, then A has a higher affinity for C over that interval. This formulation is useful in settings such as recommender systems, e.g. predicting a user's future song preferences given their past listening history (Huang et al., 2023). A concrete example from the Temporal Graph Benchmark (TGB) (Huang et al., 2023) is `tgbn-trade`, where nodes represent nations and edges represent the amount of goods exchanged in a single trade. In this case, the goal of dynamic node affinity prediction is to predict the amount of trade one nation would have with another nation in the next year, given the past evolution of global trading patterns.

Contributions. This work is based on the assumption that considering past messages between two nodes is important to predict their affinity at a future time. We start by empirically validating this assumption by demonstrating that a moving average computed over a node's past messages to another node is a powerful heuristic, beating all existing TG models. Armed with this result, we ask whether Temporal Graph Networks (TGNs) (Rossi et al., 2020), a popular TG model, can represent moving averages over messages. Surprisingly, we find that no formulation of TGN can represent moving averages of any order k. This result implies that TGNs are unable to represent persistent forecasting (i.e. the simple heuristic of outputting the most recent message between a pair of nodes), indicating a substantial weakness in its design. To remedy this, we propose to modify TGN by adding source-target identification to each interaction event message. We prove that our method, TGNv2, is strictly more expressive than TGN as it is able to represent persistent forecasting, moving averages, and autoregressive models. Further, we show that TGNv2 significantly outperforms all current TG models on all TGB datasets on dynamic node affinity prediction.

2. The Hidden Limitation of Temporal Graph Networks

This work is motivated by our observation that computing moving averages over past messages, despite still lagging behind moving average over ground-truth labels, is a competitive heuristic that outperforms all current TG models on every node affinity prediction dataset (Table 1). Given an order $k \in \mathbb{N}^+$, the moving average heuristic over past messages for node affinity prediction is defined as:

$$\hat{\mathbf{y}}_t[u, v] = \frac{1}{k} \sum_{t' \in M(u,v,t,k)} e_{uv}(t')$$

where $M(u, v, t, k)$ returns k ordered timestamps that constitute the k most-recent messages sent from node u to node v up to time t and $e_{uv}(t')$ is the scalar event message passed from node u to node v during their interaction at time t'. Given this observation, we focus on TGNs and study if there exists a formulation of TGN that can *exactly represent a moving average of order* k. Our first important result is proving that this cannot be the case:

Theorem 1 *No formulation of TGN can represent a moving average of order $k \in \mathbb{N}^+$ for any temporal graph with a bounded number of vertices.*

We prove Theorem 1 in Appendix B.1. In short, the proof constructs a minimal example of two nodes in two graphs sending different messages. We show that TGNs cannot distinguish the two nodes, leading them to compute the same moving average for both nodes. This is a direct consequence of the permutation-invariance of TGNs, which renders them unable to discriminate between senders and receivers of messages, and in turn incapable of capturing important functions.

Since the above theorem holds for any k and persistent forecasting is equivalent to a moving average when $k = 1$, it follows that TGNs cannot represent persistent forecasting. Proceeding similarly to our proof for Theorem 1, we can show that, more generally, TGNs cannot represent the class of autoregressive functions (proof in Appendix B.2):

Corollary 2 *No formulation of TGN can represent an autoregressive model of order $k \in \mathbb{N}^+$ for any temporal graph with a bounded number of vertices.*

2.1. TGNv2: Increasing the expressive power of TGNs

The main problem with TGN lies in the construction of the messages when an event occurs. In TGN, for every interaction between nodes i and j, two messages are constructed:

$$\mathbf{m}_i(t) = \text{msg}_s(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \phi(\Delta t), e_{ij}(t)); \quad \mathbf{m}_j(t) = \text{msg}_d(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \phi(\Delta t), e_{ij}(t))$$

If we look closely, however, we can see that each message does not contain the source or the destination of the message. Not only does this make it impossible for the memory vectors to have an imprint of past interactions, but this also renders TGNs to be *invariant* to the identities of the senders and receivers of messages–a property that is undesirable for dynamic node affinity prediction. To address this issue, we introduce TGNv2, where we modify the message construction of TGNs to include source-target identification:

$$\mathbf{m}_i(t) = \text{msg}_s(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \phi_t(\Delta t), e_{ij}(t), \phi_n(i), \phi_n(j))$$
$$\mathbf{m}_j(t) = \text{msg}_d(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \phi_t(\Delta t), e_{ij}(t), \phi_n(j), \phi_n(i))$$

Here, we map all nodes to an arbitrary, but fixed node index, and $\phi_n \in \mathbb{R} \to \mathbb{R}^d$ is an encoder function for node indices, similar to ϕ_t. Incoming nodes that have not been encountered before are assigned to fresh, unused node indices as the graph evolves. This modification is a way to break the permutation-invariance of TGN, which is necessary to compute moving averages and autoregressive models. We are now able to prove:

Theorem 3 *There exists a formulation of TGNv2 that can represent persistent forecasting, moving average of order $k \in \mathbb{N}^+$, or any autoregressive model of order $k \in \mathbb{N}^+$ for any temporal graph with a bounded number of vertices.*

Our proof of Theorem 3 (Appendix B.3) leverages the existence of the node identification to 'index' into the memory vector to store information. From this, it follows that TGNv2 is strictly more expressive than TGN, as TGN is a special case of TGNv2.

3. Experiments

Table 1 shows our experimental results on the TGB benchmark. The top 3 rows are simple heuristics over ground-truth labels / ground-truth messages. 'Persistent Frcst (L)' and 'Moving Average (L)' refer to persistent forecasting and moving average over ground-truth labels respectively; while 'Moving Avg (M)' is a moving average over messages. The rest of the rows correspond to TG models. We describe the experimental details in Appendix C.

Evidently from Table 1, Moving Average (M) is a competitive method that outperforms all TG models. TGN (tuned) denotes the TGN that we trained using the same set of hyperparameters for TGNv2. Though TGN enjoys a performance boost with this set of hyperparameters, TGNv2 significantly outperforms TGN and all TG models on all datasets. Further, we can see that TGNv2 performs comparably to Moving Avg (M) on `tgbn-trade`, `tgbn-genre`, `tgbn-reddit` while TGN is beaten by Moving Avg (M) on all datasets.

Table 1: Main results. † are results obtained from Huang et al. (2023), while ‡ are obtained from Yu (2023). TGNv2 outperforms all current TG models by a large margin.

.

Method	tgbn-trade NDCG @ 10 ↑		tgbn-genre NDCG @ 10 ↑		tgbn-reddit NDCG @ 10 ↑		tgbn-token NDCG @ 10 ↑	
	Validation	Test	Validation	Test	Validation	Test	Validation	Test
Persistent Frcst (L)†	**0.860**	**0.855**	0.350	0.357	0.380	0.369	0.403	0.430
Moving Avg (L)†	0.841	0.823	**0.499**	**0.509**	**0.574**	**0.559**	**0.491**	**0.508**
Moving Avg (M)	0.793	0.777	0.478	0.472	0.499	0.481	0.402	0.415
JODIE‡	$0.394_{\pm 0.05}$	$0.374_{\pm 0.09}$	$0.358_{\pm 0.03}$	$0.350_{\pm 0.04}$	$0.345_{\pm 0.02}$	$0.314_{\pm 0.01}$		
TGAT‡	$0.395_{\pm 0.14}$	$0.375_{\pm 0.07}$	$0.360_{\pm 0.04}$	$0.352_{\pm 0.03}$	$0.345_{\pm 0.01}$	$0.314_{\pm 0.01}$		
CAWN‡	$0.393_{\pm 0.07}$	$0.374_{\pm 0.09}$						
TCL‡	$0.394_{\pm 0.11}$	$0.375_{\pm 0.09}$	$0.362_{\pm 0.04}$	$0.354_{\pm 0.02}$	$0.347_{\pm 0.01}$	$0.314_{\pm 0.01}$		
GraphMixer‡	$0.394_{\pm 0.17}$	$0.375_{\pm 0.11}$	$0.361_{\pm 0.04}$	$0.352_{\pm 0.03}$	$0.347_{\pm 0.01}$	$0.314_{\pm 0.01}$		
DyGFormer‡	$0.408_{\pm 0.58}$	$0.388_{\pm 0.64}$	$0.371_{\pm 0.06}$	$0.365_{\pm 0.20}$	$0.348_{\pm 0.02}$	$0.316_{\pm 0.01}$		
DyRep†	$0.394_{\pm 0.001}$	$0.374_{\pm 0.001}$	$0.357_{\pm 0.001}$	$0.351_{\pm 0.001}$	$0.344_{\pm 0.001}$	$0.312_{\pm 0.001}$	$0.151_{\pm 0.006}$	$0.141_{\pm 0.006}$
TGN†	$0.395_{\pm 0.002}$	$0.374_{\pm 0.001}$	$0.403_{\pm 0.010}$	$0.367_{\pm 0.058}$	$0.379_{\pm 0.004}$	$0.315_{\pm 0.020}$	$0.189_{\pm 0.005}$	$0.169_{\pm 0.006}$
TGN (tuned)	$0.445_{\pm 0.009}$	$0.409_{\pm 0.005}$	$0.443_{\pm 0.002}$	$0.423_{\pm 0.007}$	$0.482_{\pm 0.007}$	$0.408_{\pm 0.006}$	$0.251_{\pm 0.000}$	$0.200_{\pm 0.005}$
TGNv2 (ours)	$\mathbf{0.807}_{\pm \mathbf{0.006}}$	$\mathbf{0.735}_{\pm \mathbf{0.006}}$	$\mathbf{0.481}_{\pm \mathbf{0.001}}$	$\mathbf{0.469}_{\pm \mathbf{0.002}}$	$\mathbf{0.544}_{\pm \mathbf{0.000}}$	$\mathbf{0.507}_{\pm 0.002}$	$\mathbf{0.321}_{\pm \mathbf{0.001}}$	$\mathbf{0.294}_{\pm \mathbf{0.001}}$

4. Related Work

We believe this work is the first to address the limitations of TG models in dynamic node affinity prediction. Huang et al. (2023) were the first to point out this problem, highlighting that TGN and DyRep (Trivedi et al., 2018) are outperformed by heuristics over ground-truth labels. Yu (2023) extended this work and found that a suite of other TG models (JODIE (Kumar et al., 2019), TGAT (Xu et al., 2020), CAWN (Wang et al., 2022), TCL (Wang et al., 2021), GraphMixer (Cong et al., 2023), and DyGFormer (Yu et al., 2023)) all underperform in dynamic node affinity prediction. Despite still lagging behind heuristics over ground-truth labels, TGNv2 significantly outperforms all of the methods above, constituting what we believe to be **the first positive result in improving TG models for dynamic node affinity prediction**. Our method of augmenting TGNs with source-target identification to increase expressivity is most similar to the work of Sato et al. (2019), where they increased the expressivity of static GNNs via port numbering. Relatedly, other works demonstrated that breaking the permutation-invariance of static GNNs (e.g. by using RNNs to aggregate messages) led to empirical benefits (Xu and Veličković, 2024; Hamilton et al., 2018).

5. Conclusion

In this paper, we proposed to augment TGN with source-target identification. We proved that TGNv2 is strictly more expressive than TGN and consequently showed that TGNv2 achieves significantly higher performance than current TG models across all dynamic node affinity prediction datasets from TGB. In the future, we would like to close the remaining empirical gap between TGNv2 and the heuristics approaches. We believe this is because we formulated our message aggregator to output the last message, which was necessary to compare our results fairly with prior TGN experiments (Appendix C). To address this, we hope to explore more expressive aggregation functions. Moreover, we would like to further develop our work by studying TGNv2 on other TG tasks, such as dynamic link prediction.

References

Weilin Cong, Si Zhang, Jian Kang, Baichuan Yuan, Hao Wu, Xin Zhou, Hanghang Tong, and Mehrdad Mahdavi. Do we really need complicated model architectures for temporal networks?, 2023. URL https://arxiv.org/abs/2302.11636.

Songgaojun Deng, Huzefa Rangwala, and Yue Ning. Learning dynamic context graphs for predicting social events. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 1007–1016, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330919. URL https://doi.org/10.1145/3292500.3330919.

ZhengZhao Feng, Rui Wang, TianXing Wang, Mingli Song, Sai Wu, and Shuibing He. A comprehensive survey of dynamic graph neural networks: Models, frameworks, benchmarks, experiments and challenges, 2024. URL https://arxiv.org/abs/2405.00476.

Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):922–929, Jul. 2019. doi: 10.1609/aaai.v33i01.3301922. URL https://ojs.aaai.org/index.php/AAAI/article/view/3881.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018. URL https://arxiv.org/abs/1706.02216.

Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs, 2023.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19. ACM, July 2019. doi: 10.1145/3292500.3330895. URL http://dx.doi.org/10.1145/3292500.3330895.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks, 2021. URL https://arxiv.org/abs/2010.03409.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs, 2020.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks, 2020. URL https://arxiv.org/abs/2002.09405.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Approximation ratios of graph neural networks for combinatorial problems, 2019. URL https://arxiv.org/abs/1905.10261.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked label prediction: Unified message passing model for semi-supervised classification, 2021. URL https://arxiv.org/abs/2009.03509.

Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9: 79143–79168, 2021. ISSN 2169-3536. doi: 10.1109/access.2021.3082932. URL http://dx.doi.org/10.1109/ACCESS.2021.3082932.

Weiping Song, Zhiping Xiao, Yifan Wang, Laurent Charlin, Ming Zhang, and Jian Tang. Session-based social recommendation via dynamic graph attention networks. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19. ACM, January 2019. doi: 10.1145/3289600.3290989. URL http://dx.doi.org/10.1145/3289600.3290989.

Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Representation learning over dynamic graphs, 2018.

Lu Wang, Xiaofu Chang, Shuang Li, Yunfei Chu, Hui Li, Wei Zhang, Xiaofeng He, Le Song, Jingren Zhou, and Hongxia Yang. Tcl: Transformer-based dynamic graph modelling via contrastive learning, 2021. URL https://arxiv.org/abs/2105.07944.

Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. Inductive representation learning in temporal networks via causal anonymous walks, 2022. URL https://arxiv.org/abs/2101.05974.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs, 2020.

Kaijia Xu and Petar Veličković. Recurrent aggregators in neural algorithmic reasoning, 2024. URL https://arxiv.org/abs/2409.07154.

Guotong Xue, Ming Zhong, Jianxin Li, Jia Chen, Chengshuai Zhai, and Ruochen Kong. Dynamic network embedding survey, 2021. URL https://arxiv.org/abs/2103.15447.

Le Yu. An empirical evaluation of temporal graph benchmark, 2023. URL https://arxiv.org/abs/2307.12510.

Le Yu, Leilei Sun, Bowen Du, and Weifeng Lv. Towards better dynamic graph learning: New architecture and unified library, 2023. URL https://arxiv.org/abs/2303.13047.

Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, September 2020. ISSN 1558-0016. doi: 10.1109/tits.2019.2935152. URL http://dx.doi.org/10.1109/TITS.2019.2935152.

Appendix A. TGN Recap

For the reader's convenience, we restate the core modules of TGN. For a more thorough explanation of each module, we refer the reader to the original paper (Rossi et al., 2020).

Message Function. For each interaction between i and j, we construct two messages msg_s and msg_d:

$$\mathbf{m}_i(t) = \text{msg}_s(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \phi(\Delta t), e_{ij}(t)); \quad \mathbf{m}_j(t) = \text{msg}_d(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \phi(\Delta t), e_{ij}(t))$$

We can also opt to construct node messages if node events exist:

$$\mathbf{m}_i(t) = \text{msg}_n(\mathbf{s}_i(t^-), t, \mathbf{v}_i(t))$$

Message Aggregator.

$$\bar{\mathbf{m}}_i(t) = \text{agg}(\mathbf{m}_i(t_1), ..., \mathbf{m}_i(t_b))$$

Memory Updater.

$$\mathbf{s}_i(t) = \text{mem}(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)))$$

Embedding

$$\mathbf{z}_i(t) = g\big(\{\{h(\mathbf{s}_i(t), \mathbf{s}_j(t), e_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)) : j \in \mathcal{N}_i^L([0,t])\}\}\big)$$

Here, h is a learnable function, g is a permutation-invariant function such as a sum or mean, and L corresponds to the number of layers used for temporal message passing. We note that our formulation of the embedding layer is a more general version than in the TGN paper, and we can recover the original formulation by setting g to be a sum.

Appendix B. Proofs

B.1. Proof for Theorem 1

Theorem 1 *No formulation of TGN can represent a moving average of order $k \in \mathbb{N}^+$ for any temporal graph with a bounded number of vertices.*

Proof The main idea of the proof is that TGNs are unable to distinguish nodes whose messages are identical in every form but have different senders and/or recipients. To show this, we construct a temporal graph G with 3 nodes (Node 1, 2, and 3) and 'flip' it in such a way to yield a G' such that Node 1 in G' is sending different messages when compared to Node 1 in G but is indistinguishable from Node 1 in G from the point of view of TGNs (Figure 1).

We proceed by way of contradiction. Assume that there exists a particular formulation of TGN that can implement a moving average of order k for any temporal graph with a bounded number of vertices. We initialise all memory vectors to be the zero vector, as per TGN's original formulation. Now, consider the following sequence of events that implicitly define the temporal graph G:

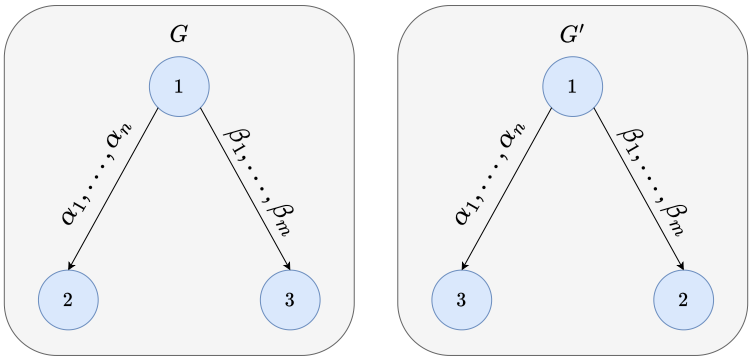

Figure 1: Graphs G and G'. Clearly Node 1 in G and G' are sending different sequences of messages, but TGNs are unable to distinguish them.

1. Node 1 sends Node 2 a series of n events with features $\alpha_1, \ldots, \alpha_n$ at timestamps $t_1, \ldots, t_n$.
2. Node 1 sends Node 3 a series of m events with features $\beta_1, \ldots, \beta_m$ at timestamps $t_{n+1}, \ldots, t_{n+m}$.

where $t_1 < \cdots < t_n < t_{n+1} < \cdots < t_{n+m}$, $n \geq k$, and $m \geq k$. Let $\bar{\alpha} = \frac{\alpha_{n-k+1} + \cdots + \alpha_n}{k}$, the moving average of order k of $\alpha_1, ..., \alpha_n$. Similarly, define $\bar{\beta}$ to be the moving average of order k of $\beta_1, ..., \beta_m$. Suppose we compute Node 1's embedding at time t_T where $t_T > t_{n+m}$. We assume that no node updates are done, and all $\mathbf{v}_1(t) = \mathbf{v}_2(t) = \mathbf{v}_3(t)$ for all t. Node 1 receives n messages from its interactions with Node 2:

$$\mathbf{m}_1(t_i) = \text{msg}_s(\mathbf{0}, \mathbf{0}, t_i, \alpha_i) \qquad \forall i \in [1, \ldots, n]$$

Similarly, Node 1 receives m messages from its interactions with Node 3:

$$\mathbf{m}_1(t_{n+i}) = \text{msg}_s(\mathbf{0}, \mathbf{0}, t_{n+i}, \beta_i) \qquad \forall i \in [1, \ldots, m]$$

Node 1 then aggregates the messages it receives and updates its memory:

$$\begin{aligned} \tilde{\mathbf{m}}_1(t_T) &= \text{agg}(\mathbf{m}_1(t_1), \ldots, \mathbf{m}_1(t_{n+m})) \\ \mathbf{s}_1(t_T) &= \text{mem}(\tilde{\mathbf{m}}_1(t_T), \mathbf{0}) \end{aligned}$$

We can do the same set of calculations for Node 2 and Node 3:

$$\begin{aligned} \mathbf{m}_2(t_i) &= \text{msg}_d(\mathbf{0}, \mathbf{0}, t_i, \alpha_i) \qquad \forall i \in [1, \ldots, n] \\ \mathbf{m}_3(t_{n+i}) &= \text{msg}_d(\mathbf{0}, \mathbf{0}, t_{n+i}, \beta_i) \qquad \forall i \in [1, \ldots, m] \\ \tilde{\mathbf{m}}_2(t_T) &= \text{agg}(\mathbf{m}_2(t_1), \ldots, \mathbf{m}_2(t_n)) \\ \tilde{\mathbf{m}}_3(t_T) &= \text{agg}(\mathbf{m}_3(t_{n+1}), \ldots, \mathbf{m}_3(t_{n+m})) \\ \mathbf{s}_2(t_T) &= \text{mem}(\tilde{\mathbf{m}}_2(t_T), \mathbf{0}) \\ \mathbf{s}_3(t_T) &= \text{mem}(\tilde{\mathbf{m}}_3(t_T), \mathbf{0}) \end{aligned}$$

Then, as we have assumed no node events have occurred, and all $\mathbf{v}_i(t_T)$ are the same, then we can ignore them during the embedding computation:

$$\begin{aligned}\mathbf{z}_1(t_T) &= g\big(\{\{h(\mathbf{s}_1(t_T), \mathbf{s}_2(t_T), \alpha_i) : i \in [1, \dots n]\}\} \cup \{\{h(\mathbf{s}_1(t_T), \mathbf{s}_3(t_T), \beta_i) : i \in [1, \dots m]\}\}\big) \\ &= [0, \bar{\alpha}, \bar{\beta}]^T\end{aligned}$$

as per our assumption. Consider now the flipped temporal graph G' with events:

1. Node 1 sends Node 3 a series of n events with features $\alpha_1, \dots, \alpha_n$ at timestamps $t_1, \dots, t_n$.

2. Node 1 sends Node 2 a series of m events with features $\beta_1, \dots, \beta_m$ at timestamps $t_{n+1}, \dots, t_{n+m}$.

where $t_1 < \dots < t_n < t_{n+1} < \dots < t_{n+m}$, $n \geq k$, $m \geq k$ as before. Suppose we are again to compute the node embeddings at time $t_T > t_{n+m}$. Node 1 receives n messages from its interactions with Node 3:

$$\mathbf{m}'_1(t_i) = \text{msg}_s(\mathbf{0}, \mathbf{0}, t_i, \alpha_i) \qquad \forall i \in [1, \dots, n]$$

Node 1 receives m messages from its interactions with Node 2:

$$\mathbf{m}'_1(t_i) = \text{msg}_s(\mathbf{0}, \mathbf{0}, t_{n+i}, \beta_i) \qquad \forall i \in [1, \dots, m]$$

But observe that Node 1 receives the same set of messages as it did in G. Therefore, $\mathbf{s}'_1(t_T)$ must be equal to $\mathbf{s}_1(t_T)$:

$$\begin{aligned}\tilde{\mathbf{m}}'_1(t_T) &= \text{agg}(\mathbf{m}'_1(t_1), \dots, \mathbf{m}'_1(t_{n+m})) \\ &= \text{agg}(\mathbf{m}_1(t_1), \dots, \mathbf{m}_1(t_{n+m})) \\ &= \tilde{\mathbf{m}}_{\mathbf{1}}(t_T) \\ \mathbf{s}'_1(t_T) &= \text{mem}(\tilde{\mathbf{m}}'_1(t_T), \mathbf{0}) \\ &= \text{mem}(\tilde{\mathbf{m}}_1(t_T), \mathbf{0}) \\ &= \mathbf{s}_1(t_T)\end{aligned}$$

Further, we can see that Node 3 in G' receives the same set of messages as Node 2 in G, and Node 2 in G' receives the same set of messages as Node 3 in G. Following the same

reasoning, we can conclude that:

$$\begin{aligned}
\tilde{\mathbf{m}}'_2(t_T) &= \text{agg}(\mathbf{m}'_2(t_{n+1}), \ldots, \mathbf{m}'_2(t_{n+m})) \\
&= \text{agg}(\mathbf{m}_3(t_{n+1}), \ldots, \mathbf{m}_3(t_{n+m})) \\
&= \tilde{\mathbf{m}}_{\mathbf{3}}(t_T) \\
\mathbf{s}'_2(t_T) &= \text{mem}(\tilde{\mathbf{m}}'_2(t_T), \mathbf{0}) \\
&= \text{mem}(\tilde{\mathbf{m}}_3(t_T), \mathbf{0}) \\
&= \mathbf{s}_3(t_T) \\
\tilde{\mathbf{m}}'_3(t_T) &= \text{agg}(\mathbf{m}'_3(t_1), \ldots, \mathbf{m}'_3(t_n)) \\
&= \text{agg}(\mathbf{m}_2(t_1), \ldots, \mathbf{m}_2(t_n)) \\
&= \tilde{\mathbf{m}}_{\mathbf{2}}(t_T) \\
\mathbf{s}'_3(t_T) &= \text{mem}(\tilde{\mathbf{m}}'_3(t_T), \mathbf{0}) \\
&= \text{mem}(\tilde{\mathbf{m}}_2(t_T), \mathbf{0}) \\
&= \mathbf{s}_2(t_T)
\end{aligned}$$

Therefore:

$$\begin{aligned}
\mathbf{z}'_1(t_T) &= g\big(\big\{\big\{h(\mathbf{s}'_1(t_T), \mathbf{s}'_3(t_T), \alpha_i) : i \in [1, \ldots n]\big\}\big\} \cup \big\{\big\{h(\mathbf{s}'_1(t_T), \mathbf{s}'_2(t_T), \beta_i) : i \in [1, \ldots m]\big\}\big\}\big) \\
&= g\big(\big\{\big\{h(\mathbf{s}_1(t_T), \mathbf{s}_2(t_T), \alpha_i) : i \in [1, \ldots n]\big\}\big\} \cup \big\{\big\{h(\mathbf{s}_1(t_T), \mathbf{s}_3(t_T), \beta_i) : i \in [1, \ldots m]\big\}\big\}\big) \\
&= \mathbf{z}_1(t_T) \\
&= [0, \bar{\alpha}, \bar{\beta}]^T
\end{aligned}$$

which is a contradiction, as a moving average of order k would've computed $[0, \bar{\beta}, \bar{\alpha}]^T$ for G'. ■

B.2. Proof for Corollary 2

Corollary 2 *No formulation of TGN can represent an autoregressive model of order $k \in \mathbb{N}^+$ for any temporal graph with a bounded number of vertices.*

Proof Our proof for Theorem 2 proceeds very similarly to Theorem 1. Notice that in our proof of Theorem 1, we did not make use of the fact that $\bar{\alpha}$ and $\bar{\beta}$ are moving averages. Therefore, if our auto-regressive model has weights $w_1, \ldots, w_k$, then we can define $\bar{\alpha}$ and $\bar{\beta}$ to be:

$$\bar{\alpha} = \sum_{i=1}^{k} w_i \alpha_{n-i}$$

$$\bar{\beta} = \sum_{i=1}^{k} w_i \beta_{n-i}$$

and consequently proceeding in the same manner as we did in Theorem 1. ■

B.3. Proof for Theorem 3

Theorem 3 *There exists a formulation of TGNv2 that can represent persistent forecasting, moving average of order $k \in \mathbb{N}^+$, or any autoregressive model of order $k \in \mathbb{N}^+$ for any temporal graph with a bounded number of vertices.*

Proof We first prove the theorem for the case of moving averages of order k, and extend that to apply to persistent forecasting and autoregressive models. Let the maximum number of nodes encountered in the temporal graph be n, and assign each node an identifier such that each node is uniquely identified by an $i \in [0, \dots, n-1]$. We initialise all memory vectors $\mathbf{s}_i$ to be $\mathbf{0} \in \mathbb{R}^{nk}$.

Next, denote $e_{ij}^{(l)}$ be the feature of the l-th message that i sends to j, and let $M(i,j,t)$ return the index of the most recent message that i sent to j at time t. We assume that the batch size is 1, which means that as soon as a message is sent, we update the memory vectors. Further, for the time being, we ignore the formulation of msg_d, and assume that all the messages received by the aggregator are messages constructed by msg_s.

Our goal is to find a formulation of TGNv2 such that, for each timestamp t, it computes $\mathbf{z}_i(t)[j] = \frac{1}{k}\sum_{x=0}^{k-1} e_{ij}^{(M(i,j,t)-x)}$. We assume that the moving average is defined for all values of t by letting $e_{ij}^{(l)} = 0$ for all negative l. We now concretely define the formulation of TGNv2, and subsequently show that it computes the moving average. Now, suppose some node i sends j a message at time t. Let msg_s be formulated as:

$$\text{msg}_s(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \phi_t(\Delta t), e_{ij}(t), \phi_n(i), \phi_n(j)) = [e_{ij}(t), j]$$

i.e. msg_s simply outputs a 2-element vector with the feature of the event message and the index of the destination node. Since our batch size is 1, the aggregator only receives at most one message. We let the aggregator be the identity function. The main idea of the proof is in the formulation of the memory module, which takes advantage of the node index of the destination to 'store' the newest feature message between i and j.

We introduce some machinery to aid our formulation. Consider a block matrix $\mathbf{B} \in \mathbb{R}^{nk \times nk}$ with n matrices $\mathbf{B}_1, \dots, \mathbf{B}_n \in \mathbb{R}^{k \times k}$ in its diagonal:

$$\mathbf{B} = \begin{bmatrix} \mathbf{B}_1 & \dots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{B}_n \end{bmatrix}$$

Define the block-permutation matrix $\mathbf{P} \in \mathbb{R}^{nk \times nk}$:

$$\mathbf{P} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} \\ \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \dots & \mathbf{I} & \mathbf{0} \end{bmatrix}$$

where $\mathbf{I}$ is the $\mathbf{R}^{k\times k}$ identity matrix. Observe that $\mathbf{PBP}^T$ cyclically shifts the order of $\mathbf{B}_1, \ldots, \mathbf{B}_n$ in $\mathbf{B}$ by one:

$$\mathbf{PBP}^T = \begin{bmatrix} \mathbf{B}_n & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_1 & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \ldots & \mathbf{0} & \mathbf{B}_{n-1} \end{bmatrix}$$

Similarly, define the permutation matrix $\mathbf{Q}$ that analogously shifts the elements of a vector cyclically by 1:

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & \ldots & 1 \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 1 & 0 \end{bmatrix}$$

In order to compute the moving average, then each time we receive a message, we need to 'make room' in our memory vector to store the message. We introduce the shift matrix $\mathbf{S}$, which is a $k \times k$ matrix that is obtained by taking the $(k-1) \times (k-1)$ identity matrix and sufficiently padding the top row and rightmost column with zeroes which, when applied to a vector $\mathbf{v} \in \mathbb{R}^k$, keeps the top $k-1$ elements and discards the last element:

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & \ldots & 0 & 0 \\ 1 & 0 & \ldots & 0 & 0 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{bmatrix}$$

Define the generator block matrix $\mathbf{X} \in \mathbb{R}^{nk\times nk}$ that has the same form as $\mathbf{B}$ and consists of n matrices $\mathbf{B}_1, \ldots, \mathbf{B}_n$, but with $\mathbf{B}_1 = \mathbf{S}$ and all other $\mathbf{B}_i = \mathbf{I}$:

$$\mathbf{X} = \begin{bmatrix} \mathbf{S} & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \ldots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \ldots & \mathbf{0} & \mathbf{I} \end{bmatrix}$$

Similarly, define the generator vector $\mathbf{y} \in \mathbb{R}^{nk} = [1, 0, \ldots, 0]^T$. Next, define $f(j) = (\mathbf{P})^j \mathbf{X} (\mathbf{P}^T)^j$, which cyclically shifts the block matrices in $\mathbf{X}$ a total number of j times, and $p(j) = \mathbf{Q}^j \mathbf{y}$, which cyclically shifts the elements of $\mathbf{y}$ a total number of j times. Now, let the memory module be:

$$\begin{aligned} \mathbf{s}_i(t) &= \text{mem}(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)) \\ &= \text{mem}([e_{ij}(t), j], \mathbf{s}_i(t^-)) \\ &= f(j) \cdot \mathbf{s}_i(t^-)) + p(j) \cdot e_{ij}(t) \end{aligned}$$

It is quite easy to see, via a straightforward induction, that at every timestamp t, $\mathbf{s}_i(t)$ stores the k most recent messages sent to j in the 'subarray' $\mathbf{s}_i(t)[jk : jk + k - 1]$. Consequently, making $\mathbf{z}_i(t)$ compute the moving average is straightforward. Define the aggregator matrix $\mathbf{A} \in \mathbb{R}^{n \times nk}$ to be:

$$\mathbf{A}[m, n] = \begin{cases} \frac{1}{k} & \text{if } mk \leq n \leq mk + k - 1 \\ 0 & \text{otherwise} \end{cases}$$

and let our embedding layer be defined as:

$$\begin{aligned} \mathbf{z}_i(t) &= g(\{\{h(\mathbf{s}_i(t), \mathbf{s}_j(t), e_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)) : j \in \mathcal{N}_i^L([0, t]\}\}) \\ &= \frac{1}{|\mathcal{N}_i^L([0, t]|} \sum_{j=0}^{|\mathcal{N}_i^L([0,t]|} \mathbf{A} \cdot \mathbf{s}_i(t) \\ &= \mathbf{A} \cdot \mathbf{s}_i(t) \end{aligned}$$

which is the moving average of order k, as multiplying $\mathbf{A}$ with $\mathbf{s}_i(t)$ has the effect of summing the k most recent messages for each node and multiplying the sum by $\frac{1}{k}$. From this, we can see that the theorem holds for persistent forecasting as persistent forecasting is moving average with $k = 1$. Subsequently, we can adapt our proof above to hold for autoregressive models of any order k by formulating the aggregator matrix $\mathbf{A}$ to have the autoregressive weights $w_k, \ldots, w_1$ in entries where 1 is present.

In our constructions above, we assumed that our batch size is 1 and we ignored the fact that nodes are receiving messages from msg_d. To adapt the proof for an arbitrary batch size, we can define the aggregator module to concatenate all incoming messages, and then during the memory update, we 'unpack' this concatenation and apply our logic above for each message. Finally, to handle messages from msg_d, we can expand the size of the message vector by 1 to include a 'tag' that is nonzero if and only if the message originates from msg_d. Then, the aggregator module can drop messages from msg_d by inspecting this tag, leaving us with messages from msg_s – which we have shown how to handle.

■

Appendix C. Experiment Details

The code to reproduce our experiments can be found at https://anonymous.4open.science/r/TGNv2-submission-neurreps-08D5.

For both TGN and TGNv2, we utilise the same choices of core modules where applicable and use the same hyperparameters in order to make the results as comparable as possible. We repeat each experiment run three times with three different random seeds, each time picking the best-performing model on the validation set, and reporting the mean and standard deviation NDCG @ 10 on both the validation and test set. For our experiments, our choice of core module largely follows the choices made in TGB's experiments with TGN (Huang et al., 2023) for dynamic node affinity prediction:

Message Function. Our message function concatenates its inputs. For example, in the case of TGN:

$$\text{msg}_s(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \phi(\Delta t), e_{ij}(t)) = [\mathbf{s}_i(t^-) \circ \mathbf{s}_j(t^-) \circ \phi(\Delta t) \circ e_{ij}(t)]^T$$
$$\text{msg}_d(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \phi(\Delta t), e_{ij}(t)) = [\mathbf{s}_j(t^-) \circ \mathbf{s}_i(t^-) \circ \phi(\Delta t) \circ e_{ij}(t)]^T$$

where $\circ$ is a concatenation operator. Since there are no node events in the TGB dataset, we chose to ignore formulating the message function for node events.

Message Aggregator. We set the message aggregator to take the last message in a batch:

$$\begin{aligned}\bar{\mathbf{m}}_i(t) &= \text{agg}(\mathbf{m}_i(t_1), ..., \mathbf{m}_i(t_b)) \\ &= \mathbf{m}_i(t_b)\end{aligned}$$

Memory Updater. We set the memory updater to be a GRU:

$$\begin{aligned}\mathbf{s}_i(t) &= \text{mem}(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-))) \\ &= \text{GRU}(\bar{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)))\end{aligned}$$

where $\mathbf{s}_i \in \mathbb{R}^{d_{memory}}$.

Embedding We set the embedding module to be one layer of TransformerConv (Shi et al., 2021) with 2 heads and a dropout value of 0.1. For efficiency, we only use the last x neighbours for temporal message passing. We describe the value of x for each experiment in Table 2. We set $\mathbf{z}_i(t)$ to have a dimension of $d_{embedding}$.

Decoder The decoder takes in the embedding for each node and outputs the node affinities for all other nodes. For our decoder, we chose an MLP with 2 layers + ReLU. Both layers have dimensionality $d_{decoder}$.

Time / Node Encoder We set $\phi_t(t) = \cos(w_t \cdot t)$ and $\phi_n(i) = \cos(w_n \cdot i)$ where $w_t \in \mathbb{R}^{d_{time}}$ and $w_n \in \mathbb{R}^{d_{node}}$.

Hyperparameters For moving average over the ground-truth labels (Moving Average (L)), we set $k = 7$. For moving average over messages (Moving Average (M)), we set $k = 2048$ for `tgbn-trade`, `tgbn-genre`, `tgbn-reddit`, and $k = 512$ for `tgbn-token` due to memory issues. We use a constant learning rate schedule for all experiments, except for `tgbn-trade`, where we decay the learning rate by 0.5 every 250 epochs. We use the Adam optimiser (Kingma and Ba, 2017) to train our models. We set a global hidden dimension d, that is used in all places where we need to select a dimension, i.e. $d = d_{memory} = d_{embedding} = d_{decoder} = d_{time} = d_{node}$. The hyperparameters that we chose can be found in Table 2.

Table 2: Hyperparameters for TGB experiments, for both TGN and TGNv2.

	`tgbn-trade`	`tgbn-genre`	`tgbn-reddit`	`tgbn-token`
Learning Rate	1e-3	1e-4	1e-4	1e-4
Batch Size	200	200	200	200
Epochs	750	50	50	50
d	784	784	784	1024
No. of temporal neighbours x	25	30	30	10