Improving Temporal Graph Network Messaging

Celeste Groux
McGill University
Montreal, Canada
celeste.groux@mail.mcgill.ca

ABSTRACT

Graph Representation Learning can be used to encode a graph structure to better accomplish various tasks such as community detection, link prediction and node classification. Graph Neural Network strategies have been used to construct many different encoder and decoder algorithms. That being said, most work has been done on static graphs. In dynamic networks, the additional temporal component requires adaptation. The Temporal Graph Network algorithm introduced by Rossi et al. for continuous time dynamic networks uses representation learning with a sequence-learning model and graph attention encoding approach. In this algorithm, each node has memory to store edge and node changes that affect it over time. Each change is introduced as input to a message function, and message are aggregated over batches to update the memory. Improvements to the message function and aggregations are explored, particularly to take into account neighbourhood changes and test learnable aggregation. The adapted TGN algorithm is then tested on the bipartite dynamic network Wikipedia and unipartite dynamic network Social Evolution for link prediction. Results for the proposed variants show very small improvements for both dynamic networks.

Author Keywords

Graph Neural Network; Continous Time Dynamic Network; Dynamic Link Prediction; Temporal Graph Network

CCS Concepts

•**Computing methodologies** → Neural networks;

INTRODUCTION

Graph data is naturally present in many areas, with examples including social networks, brain networks, protein networks, sensor networks, and more. Graph data requires different strategies to best understand its unique structure. Many algorithms have been developed and improved to accomplish tasks such as community and anomaly detection, centrality computation, link prediction, and node classification. Neural Network models have also been very helpful tools in completing such tasks by learning to represent the graph in a euclidean space [1]. Dynamic graphs then add an additional dimension and unique characteristic to graph data: time. Temporal graphs also naturally arise in many situations, and often studying the evolution of a network system over time is of interest. For instance, communication or recommendation systems can be represented as networks that change over time. The additional time component of dynamic networks does however add additional complexity, and researchers continue to work on and propose new algorithms using various approaches to study such graphs.

RELATED WORK

Representation learning techniques can be described as a model consisting of an encoder and decoder pair [4]. An encoder will take as input a network and outputs an embedding, while the decoder will use these embeddings and the respective representation to accomplish some task on the network, such as node classification.

Neural Network models are very widely used for such embeddings. Graph Convolutional Neural Networks (GCN) for instance are used by Yu et al. [10] to forecast traffic. Their 3D Temporal GCN algorithm learns how to relate different roads based on time series comparisons and merges temporal and spatial data within the GCN in an efficient way. In their 2018 paper [11], Zhang and Chen use a GNN to learn an improved heuristic suitable for the given network studied based on local subgraph patterns.

One encoding strategy is Recurrent Neural Networks (RNN) which have been used in sequence modelling and can be naturally extended to temporal networks embedding. Two such variants include Gated Recurrent Units (GRU) which was introduced by Cho et al. [2] and Long Short-Term Memory (LSTM) units. The adapted RNNs can handle variable sequence length input initial as a extension of a feedforward neural network model, but they also are able to capture long-term dependencies which otherwise can be difficult in more traditional reccurent units [3].

With the abundance of streaming data, making sense of continuous time dynamic networks can be even more challenging and GCNs are thus particularly useful for processing these vast amounts of data and learning patterns and representations. Researchers Ying et al. from Pinterest and Stanford developed Pinsage, a GCN based data efficient algorithm for a Pinterest recommendation system. It uses random walk methods and graph convolution to take into account both graph structure and node feature information for the one of the largest deep graph embeddings at billions of nodes and edges.

Three particular algorithms of interest are JODIE, TGAT, and DyRep [5, 9, 7]. TGAT stacks temporal graph attention layers to aggregate time and graph structure information in neighbourhoods. The stacked layers allow for learning of node embeddings with repspect to time and can learn inductively as well. JODIE uses RNNs to produce node embeddings, and

a particular part of the algorithm is the fact the embeddings are computed and projected in the future for every interaction. This projection function is learned to model the interactions seen in the system, and batch processing makes the algorithm highly efficient. DyRep employs unsupervised learning to learn a function to inductively produce low dimension time dependent node embeddings. It uses two processes to drive these embeddings, communication and association, which characterize different temporal interactions between nodes. All of these algorithms either compete with well established baselines or significantly out-perform them in tasks such as link prediction and node classification on their respective test sets.

Last year, Rossi et al. proposed the Temporal Graph Network (TGN) framework and algorithm which likewise adapts a GNN to learn on a dynamic graph [6]. What is interesting however is that this generic deep learning framework is shown to include all previous algorithms JODIE, TGAT, and DyRep as variants. Even so, the TGN-attn variant used for testing is shown to significantly outperform these other algorithms in the tasks of node classification and link prediction.

TGN works by keeping a memory vector for each node. For each node or edge deletion or addition, a message function takes these as changes as input and produces as output a message to keep in the affected nodes memory. Changes to the network are processed in batches to aid in speed computation time, and so all messages for a single node from a given batch are aggregated into a single message for that batch. The memory is then updated with a RNN, node embeddings are produced, and then future edge and node class probabilities computed. Variants on node embeddings, memory functions, and memory updating produce these three variants of a TGN. TGN-attn however performs best with its variant including GRU memory updating, graph attention embedding function, and its respective message functions on test datasets. Even with high performance, the message functions and aggregation functions still leave room for improvement.

MOTIVATION

In this paper, we explore the effect of changing the functions for message passing and message aggregation. This could possibly improve the TGN predictions and thus allow for better modelling of systems that can be simulated as dynamic graphs, both continuous and discrete. As mentioned previously, there are many systems that can be modelled as dynamic graphs, and as more and more data becomes continuous and streaming, such as is the case with much of the data produced from online or social network interactions, it is important to have well-performing algorithms that can handle these large continuous dynamic sets of data. TGN is an algorithm that is able to handle these large datasets and is shown to perform very well compared to other recent well-performing algorithms, with average precision results above 97 for both the Wikipedia and Reddit datasets they tested on. Thus, it is worth investigating whether even more improvements can be made to this already well-performing algorithm.

Link prediction is a widely applicable task and is thus worth investigating and trying to improve, even if by small differences. Though this project focuses on link prediction, improvement in link prediction could also lead to additional improvement in node classification. TGN dynamic node classification results are shown as better than other well-performing dynamic network node classification algorithms, but the ROC AUC results still leaves much room for improvement at 87.81% and 67.06% respectively for the Wikipedia and Reddit datasets they tested on.

PROJECT DEFINITION

Message Function

The current message function used concatenates the inputs through the function of identity. Multiple improvements could be made for this function. First, in the case of a edge addition or deletion, this message function does not give any additional information on the new neighbourhood of a node. Additional information on a nodes neighbourhood could be influential in determining node labels or predicting link. One can think to look at the k-neighbourhood of a node. As this adds additional complexity and time for the algorithm to run, the trade-off between adding more neighbourhood information by increasing k can be studied.

Message Aggregation

The message aggregation functions tested in the TGN paper are mean message and most recent message. The mean message function averages all messages received for a given node, while the other message function keeps only the last message processed in a given batch. As we are considering a continuous time dynamic graph setting, keeping the last message only might lose some valuable information on how the graph changes over time. Also, when additional messages are shared to neighbours of affected nodes, selecting only the last message loses all the additional information one is giving to the graph. Further, the most recent message function outperformed the mean message function, but perhaps taking a simple average is not the best way to encapsulate all the message data. Perhaps a weighted average that combines both methods or a MLP aggregation could be used to better aggregate the total messages from a given batch to pass as a single message for node memory updating.

The task of link prediction will be tested with these various proposed changes to TGN for the message function and message aggregation. Additionally, the trade-off between more complex message functions and aggregation and increased time required to run the algorithm will be discussed. These strategies are compared two differently structured dynamic networks, the Wikipedia bipartite dynamic network used in the original paper and an additional unipartite dynamic network Social Evolution.

DATASETS

The datasets that will be used to test the algorithm include the Wikipedia dataset mentioned in the original paper and the and the Social Evolution (1 month) dataset used in the Causal Autonomous Walks (CAW) paper by Wang et al. [8]. The Wikipedia dataset is part of the Stanford Snap Dataset Network originally used for the paper on the JODIE alogrithm[1] and the

[1]Wikipedia: https://snap.stanford.edu/jodie/

1 month Social Evolution dataset can be found as part of the github for the CAW paper[2] [8].

The Wikipedia dataset is a bipartite graph with users as one set of nodes and pages as a second set of nodes. Edges represent users editing a page. This dynamic network's timespan is 30 days. Nodes have no features, but there are 172 text features for each edge representing the page edits made and the temporal component. There are 217 possible dynamic labels in the Wikipedia dataset. Labels for users represent whether they are banned or not. The wikipedia dataset was extracted from a public dataset of one month of edits. The 1000 most edited pages were chosen as items and the users who edited at least 5 pages where included in the network also.

The Social Evolution dataset is a unipartite graph tracking the proximity between students from an MIT dorm from October 2008 to May 2009 using their cellphones. The nodes represent students, and edges represent their proximity according to cellphone information. The dataset used in this paper is that of only the first 30 days. This dataset does not have node labels.

Given the size of each dynamic network and computational considerations, a subset of the dynamic network datasets are used. In the case of Wikipedia, the first 35% of interactions are included, and for Social Evolution, the first 45% of interactions are included. The Wikipedia dataset used for testing thus has 55,116 nodes and 4,948 nodes, and the Social Evolution dataset has 61,626 interactions and 66 nodes.

The network data is organized in the following format in their respective csv files. Each line represents an interaction or edge between nodes of the network. These interaction are saved as comma seperated values indicating the user, item, timestamp, state label, and edge features. The timestamp is in cardinal format as opposed to datetime. State labels for Wikipedia are 1 only if the state of the user changes, and otherwise they are 0.

METHODOLOGY

The proposed message function and aggregation changes all follow the same general process. This process begins from an interaction in the network. As mentioned previously, these are processed in batches which makes the algorithm more efficient. For each batch of interactions, there is a respective batch of messages that are created by concatenating the edge interaction information into a tensor. These are the raw messages. The message function then transforms each raw message $rm_i(t)$ for node i into a message or messages $m_i(t)$. This is where additional proposed variants are introduced. This message function includes 3 steps. First, the nodes that will be influenced are selected. Second, the selection of what message to propagate is made. Then, a final transformation is applied to the raw messages. All the transformed messages $m_i(t)$ corresponding to the same nodes in the batch are then collected and aggregated into a final message to put into memory. A diagram from the original paper of the general training process is shown in Figure 1.

[2]Social Evolution 1 month: https://github.com/snap-stanford/CAW

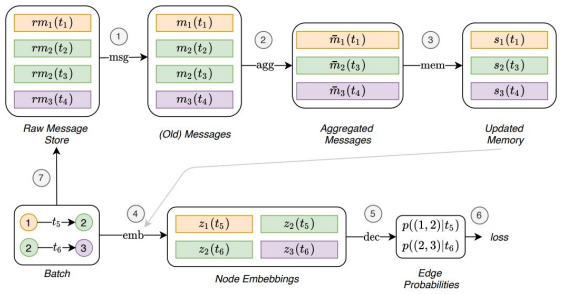


Figure 1: The original diagram from Rossi et al.'s TGN paper depicting the processing of edge interactions over time in batches. It also show the particular order of the process, which is chosen so that the computation of the memory can influence the model loss and thus obtain a gradient.

Message Function

1. Nodes to Influence

The message function begins with the selection of which nodes to influence. These nodes are selected from the k^{th} neighbourhood of the node of interest i for the given raw message $rm_i(t)$. This can be computed through the use of the network's adjacency matrix. These neighbour nodes are stored in a list $nhbd_i$. A parameter j is then chosen which specifies the number of neighbours to affect with the raw message. There are then multiple options for to select a subset of these neighbours. First, a uniform sample of j nodes in $nbhd_i$ can be made for a random selection of nodes to affect. Secondly, we can base the selection off of centrality measures. If selecting the degree centrality sample, the j nodes of $nbhd_i$ with highest degree are kept. In this case, the degree of each node in the network is computed for the dynamic graph at time t using the adjacency matrix. The $nbhd_i$ list can then be sorted according to the respective node temporal degrees. If the selecting the closeness centrality sample, the j nodes of $nbhd_i$ with the highest closeness are kept. Again, in this case the temporal graph at time t is found with its respective adjacency matrix, and then the closeness centrality is computed using this matrix. The $nbhd_i$ nodes are then sorted according to their closeness to select the top j nodes. Other centrality measures could be used, but this study focuses on these two options. Note also that if the parameter j is set to 0, the entire neighbourhood in the list $nbhd_i$ will be affected. Let the selected j nodes to affect using the given raw message be denoted $nbhd_f$.

2. Message to Propogate

A choice must then be made as to what message to propagate to the neighbourhood. The basic option is to copy the raw message $rm_i(t)$ for each node m in $nbhd_f$. A second option is to create an edited tensor message to propagate to these neighbourhood nodes. This is done by concatenating new information with the information in the original tensor, which includes data on source nodes memory, the destination nodes, edge features, and source time encoding. The new information includes this same data but for the edge that connects the $nbhd_f$ node and the original node i. Thus, for node m in

$nbhd_f$, the node m's memory, and edge features for the edge connecting m and i would be concatenated with the original tensor to create a new raw message.

3. Final Transformation

A final modification is done before obtaining the final message or messages. There are multiple options in this step also. These are variants already included in the TGN algorithm proposed by Rossi et al. and include Identity and MLP transformations. For Identity, the messages from step 2 in the message function process are unaltered, and thus they become the final messages $m_m(t)$ for the nodes m in $nhbd_f$. For MLP, a multi-layer perceptron (MLP) neural network is used to transform the information in the raw message tensor from step 2 of the message funtion. It learns how to transform this data with respect to minimizing loss during training. Rossi et al. stick to Identity for simplicity in their paper, but here the effect of the MLP transformation is also viewed with the additional proposed neighbourhood changes made in earlier steps of the message function. It should be noted also that the number of final messages can increase up to j times more than the initial number of raw messages.

Message Aggregation

Message aggregation then transforms these many messages to store a single message in every affected nodes memory. First all the messages corresponding to a unique node are gathered. Then, there are multiple possible transformations. The first two are implemented by Rossi et al. These are the last message aggregation function and mean message aggregation which stores the last message only in the nodes memory and stores a mean of all messages for the node respectively. The additional function proposed is that of a neural network MLP. This can be used to determine the best way to aggregate the possibly many different messages for a unique node into the most helpful information for the model to store in its memory. Like for the final raw message transformation, the MLP function is tied to the loss of the model and thus receives a gradient to create the learned function.

The general procedure of processing dynamic interaction and updating memory using these additional proposed variations are summarized in Figure 2. Each step lists the possible options that can be chosen to get different variants of the algorithm. The effect of each option and their interactions with other option will be analysed.

EXPERIMENT SETUP

For each dataset, the interactions and nodes are split for three different testing scenarios: old node, new node, and inductive setting. This will allow for the evaluation of transductive and inductive learning settings, as well as for testing on interactions and nodes not seen while training the TGN model. The old node subset of the data will be used to train the TGN model, and the new node subset of the data will be used to test on nodes that are not seen in the training. The inductive setting nodes then are used for additional inductive testing. These splits for each subset are shown in Table 1.

Each test of changes the TGN model will consist of the different combinations of changes to the message function and aggregation. Table 2 lists the variants that are tested and what combinations of changes they each include. Note that each variant uses graph attention as its embedding. As well, for variants that include sharing message to its neighbourhood, mean message aggregation is used. Otherwise the 0-Rec-ID variant uses last message aggregation. The message that is propagated is a copy of the original message for all these variants. Only this subset of all proposed changes is tested given the time frame of the project and computational challenges. Future work would include all the testing of all possible changes proposed. The name used to refer to each variant is also given in Table 2.

		Wikipedia	*Social Evolution*
Total	Interactions	55,116	61,626
	Nodes	4,948	66
Old Nodes	Training Interactions	29,116	9,244
	Training Nodes	3,389	50
	Test Interactions	8,267	9,244
	Test Nodes	9,244	51
New Nodes	Validation Interactions	8,268	728
	Validation Nodes	1,933	41
	Test Interactions	4,068	3,040
	Test Nodes	1,214	49
Inductive Setting	Test Nodes	494	6

Table 1: Splits used for testing on Wikipedia and Social Evolution datasets.

For each variant of the TGN algorithm, the given model will be run over 30 epochs and repeated for 4 runs. This will allow for a mean performance to be computed. Other hyper-parameters such as learning rate, batch size, and early stopping patience are kept the same as in the default best performing version of TGN in the paper at 0.0001, 200, and 5 respectively. The default best TGN variant in Rossi et al.'s paper is used as the baseline for comparison of results. This is the first variant listed in Table 2, 0-Rec-ID.

These variants of the TGN algorithm are tested over the both the transductive and induction task for link prediction. In the transductive task, future links are predicted using nodes from observed during past training, while for the inductive task, the future links are predicted without having seen the nodes previously. The performance statistics that are computed to evaluate the algorithm are area under curve (AUC) and average precision (AP). These performance measures are both already implemented in the original TGN framework code.

Note that these variants are run on a PC with an Intel i7 6700k CPU, two Nvidia GTX 970 GPUs, and 32G of DDR4 Ram.

RESULTS

The results obtained for each variant tested are given in Figure 3 for the Wikipedia and Social Evolution. The differences that are seen are very small, though there is more variation in the Social Evolution dataset. This is clear by noting that the y values in the plots span a range of 0.08 for the Wikipedia

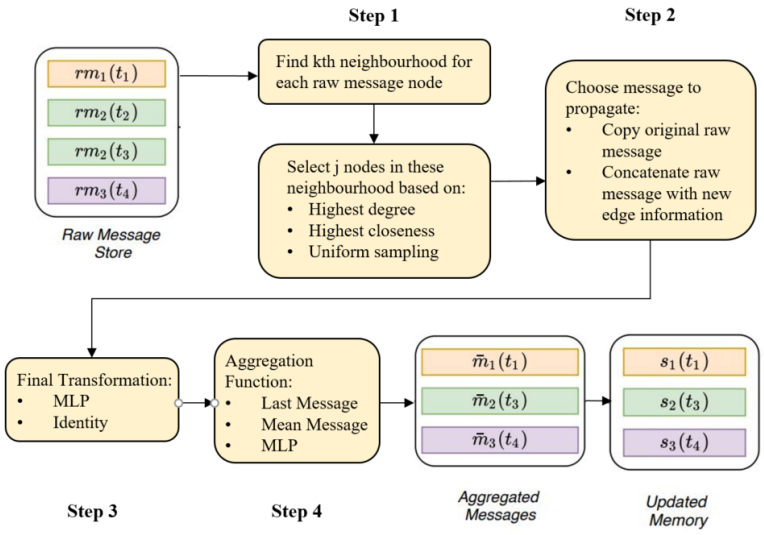


Figure 2: The diagram summarizes the general steps and proposed additions to the message function and aggregation.

dataset, and 0.4 in the case of the Social Evolution dataset plots. Also, very similar patterns are seen for Old Node AUC, Old Node AP, New Node AUC, and New Node AP between all of the Wikipedia plots, and also separately between the Social Evolution plots.

We find that the best performing variant for Wikipedia was the 10-Uni-MLP variant with an approximate improvement from the baseline of 0.01 in all measures except Old Node AP, while for Social Evolution the best variant is not as clear but is between 5-Rec-ID and 5-Deg-MLP also with improvement around 0.01.

Neighbourhood Effect

The results using variants of different neighbourhood size do not display any particular pattern. For both the Wikipedia and Social Evolution datasets, there is no clear pattern in the neighbourhood size results, comparing 5 neighbours or 10 neighbours, that produced better results.

Type of Sample

Figure 3 displays the Wikipedia and Social Evolution Old Node AP results in a bar chart coloured according the the type of neighbourhood sample chosen. For Wikipedia, the uniform sample performed best. The degree and recent sample were similar in this dataset, but the degree sample is slightly better. For Social Evolution, the uniform sample instead performed the worst. The degree and recent samples results however were found to be similar.

Message Transformation

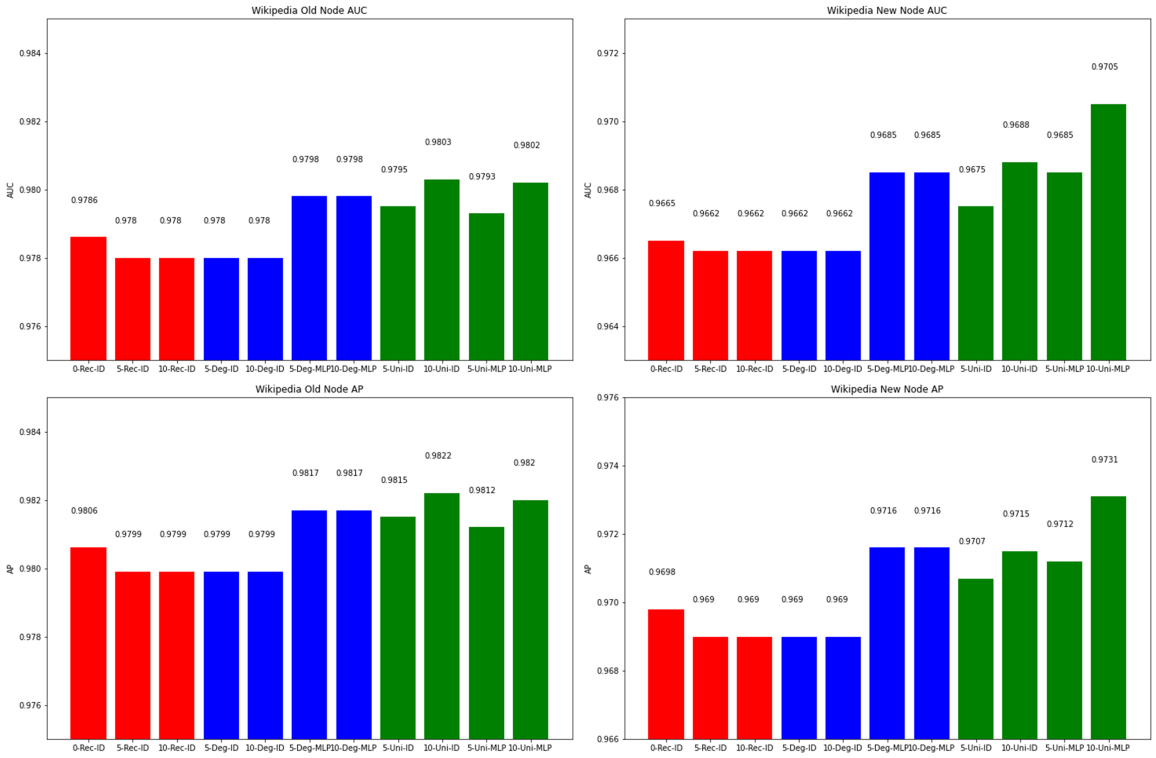
Table 3 shows the average results for the baseline, variants using Identity message transformation, and variants using MLP message transformation. We see that the variants with MLP transformation perform slightly better on the Wikipedia dataset, while identity transformation gives results similar to the baseline. For Social evolution, the average results for MLP transformation variants are again slightly better than the identity transformation, but in this case the baseline results are similar to the mean MLP transformation results.

Epoch Time

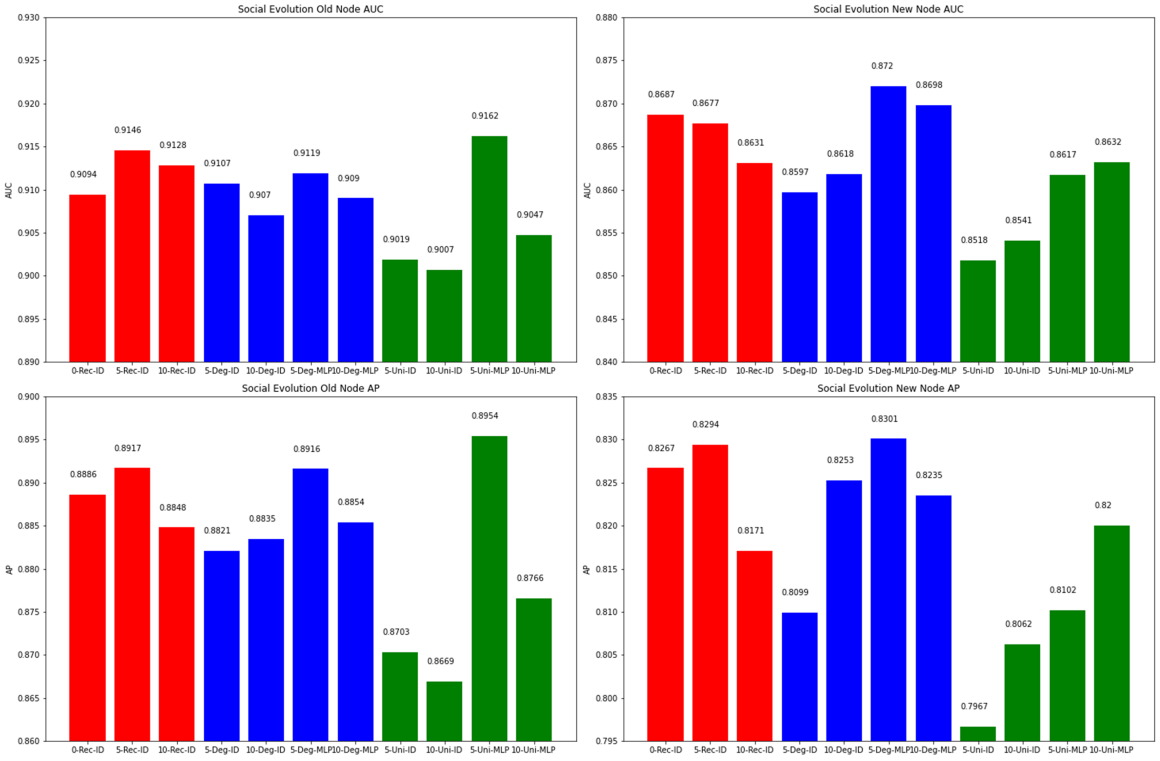
The average times needed to run one epoch in the algorithm for each variant and each dataset are listed in Table 4. As expected, the baseline 0-Rec-ID is the quickest as the only variant without additional neighbourhood message sharing taking approximately 3min and 4min to run 30 epochs for Wikipedia and Social Evolution respectively. All other variants that do involve the neighbourhood in the message function take longer, being in the range of 9-15min for Wikipedia and 18-22min for Social Evolution. There is also a pattern where the variants that include 10 neighbourhood nodes also take 4-5 seconds per epoch more than the same variant with 5 nodes instead. This results in a difference of approximately 2.5min over 30 epochs. Interestingly, including MLP transformation did not make the algorithm take more time to run as the time required was similar to a simple identity transformation.

DISCUSSION

The results show that there is little difference between variants that included neighbourhood information and the baseline TGN model used for comparison. Adding message sharing to the neighbourhood in this way did not show significant changes in the results to conclude that this improves the TGN algorithm performance. More testing on other datasets and larger test sizes would be good to investigate further. Though there are variants that perform slightly better than the baseline 0-Rec-ID, the differences are very small for both Wikipedia and Social Evolution, though larger in Social Evolution. Perhaps the variants proposed do not have as large of an impact because all variants use graph attention as their embedding.



(a) Wikipedia



(b) Social Evolution

Figure 3: Results for Wikipedia and Social Evolution grouped by type of neighbourhood sample. Red indicates a recent neighbour sample, blue indicates a neighbour sample based on highest degree, and green indicates a uniform neighbour sample.

Variant Name	*# Neighbours*	*Nbhd Sample*	*Msg Transformation*
0-Rec-ID	0	Recent	ID
5-Rec-ID	5	Recent	ID
10-Rec-ID	10	Recent	ID
5-Deg-ID	5	Degree	ID
10-Deg-ID	10	Degree	ID
5-Uni-ID	5	Uniform	ID
10-Uni-ID	10	Uniform	ID
5-Deg-MLP	5	Degree	MLP
10-Deg-MLP	10	Degree	MLP
5-Uni-MLP	5	Uniform	MLP
10-Uni-MLP	10	Uniform	MLP

Table 2: Different variants of TGN that were tested. Note that mean message aggregation is used for all variants with non-zero neighbourhood sample.

Test Statistic	*0-Rec-ID*	*ID*	*MLP*
Wikipedia			
Old Node AUC	0.9786	0.9786	0.9798
Old Node AP	0.9806	0.9806	0.9817
New Node AUC	0.9665	0.9669	0.9690
New Node AP	0.9698	0.9697	0.9719
Social Evolution			
Old Node AUC	0.9094	0.9079	0.9104
Old Node AP	0.8886	0.8799	0.8873
New Node AUC	0.8687	0.8597	0.8667
New Node AP	0.8267	0.8141	0.8210

Table 3: Results compared for baseline and then neighbourhood information sharing variants with ID or MLP message transformations.

Variant Name	*Wiki (s)*	*x30 (min)*	*Social (s)*	*x30 (min)*
0-Rec-ID	6	3.1	9	4.4
5-Rec-ID	19	9.7	36	18.2
10-Rec-ID	24	11.9	40	19.8
5-Deg-ID	26	13.1	41	20.3
10-Deg-ID	31	15.4	44	21.8
5-Uni-ID	21	10.5	38	18.9
10-Uni-ID	25	12.7	41	20.4
5-Deg-MLP	26	13.2	41	20.7
10-Deg-MLP	31	15.6	44	22.0
5-Uni-MLP	21	10.5	41	18.8
10-Uni-MLP	26	12.7	44	20.3

Table 4: Approximate running time per epoch and for 30 epochs for each variant tested.

This means that message sharing is already included in some part of the algorithm, and perhaps the additional message sharing in the message function does not provide enough extra information to be lead to significant improvement.

It is interesting also to see that different types of neighbourhood samples performed best in Wikipedia and Social Evolution. In social Evolution, the uniform sample performed much worse compared to both degree and recent sampling. A highest degree sample and recent sample could perform better than uniform in such a network that is modelling social interactions as it better represents how new links are formed. New social connections could be more likely linked to your recent interactions for instance. As well, sharing neighbourhood information according to highest degree models how your highest degree connections might have a higher influence on your future connections. In Wikipedia, the uniform sample performed best, with recent and degree sample performing similarly. This shows how the bipartite structure results in a different type sample being preferred. The Wikipedia network does not have these same social network intuitions for users writing onto pages, and they do not seem to translate as well either to the bipartite case. We see that in this different case, uniform random sample performs best, though only slightly so.

When grouped by neighbourhood sample size, we see no particular pattern among the variant performance results. This was surprising, as one might expect that the results show that either 5 or 10 is better for the sharing of messages in the neighbourhood, though it might also differ according to sample type, showing that neighbourhood information either helps or hinders the predictions. We find however that a neighbourhood size of 5 is better in some cases, while in others the size of 10 is better, without any clear pattern in these results. The results being so varied seem to indicate that the best neighbourhood sample size depends on the data and the other variant options selected such as message transformation and type of sample.

When grouped instead by final message transformation, there is a pattern. MLP performs better than ID for both Wikipedia and Social Evolution. It is interesting to note that the baseline model uses instead ID as its final message transformation, while for tested variants MLP did slightly better than ID. Perhaps, the slight improvement was not considered to be worth the additional time needed for the MLP transformation. However, we find that the additional time required for MLP variants versus the same variant with ID transformation remains almost the same. Thus, MLP seems to be best to use as a final message transformation for any of the proposed variants.

We find that the best performing variants for Wikipedia and Social Evolution were the 10-Uni-MLP and between 5-Rec-ID and 5-Deg-MLP respectively and they have small improvements over the baseline TGN algorithm. We note that there is no pattern between the best variants. This reflects how the results for the very differently structured datasets also are very dissimilar when it comes to differences based on the neighbourhood sample size, type of sample, and message transformation. It would be interesting to analyze other networks that are similar in structure to Wikipedia and Social Evolution respectively

and find out whether these variants perform best also for these similar networks.

The time needed to run the best variants for Wikipedia and Social Evolution is approximately 4 and 5 times longer than the baseline models. This is not too large to rule out using entirely, but it is a trade-off to consider when choosing what variant to use. Interestingly, all variants that included neighbourhood information had similar running time. One might have expected a larger difference between 5 to 10 neighbours, between the different types of sample, or between MLP and ID transformations. However we find them all to be comparable. Compared to the expected results with larger variation and larger run times for each variant, this is a good result to find.

CONCLUSION

In this paper, changes were proposed to the continuous dynamic network algorithm TGN in the message function and message aggregation. The changes of sharing message to a node's neighbourhood based on a recent, uniform, and highest degree sample were tested for neighbourhood sizes of 5 and 10. Then, these variants were tested with the final message transformation of MLP or ID, and mean message aggregation. These variants were tested on two very differently structured networks: the Wikipedia bipartite dynamic network and the Social Evolution unipartite dynamic network. The TGN variant with best performance in the original paper is used as a baseline for comparison of results which included 0 neighbours and ID message transformation. The proposed variants otherwise kept all other parameters and hyper-parameters the same as the baseline TGN.

We find that the best performing variants for Wikipedia and Social Evolution were the 10-Uni-MLP and between 5-Rec-ID and 5-Deg-MLP respectively, with small improvements over the baseline TGN algorithm. The results are very close for all variants and the baseline on Wikipedia and Social Evolution, though there was greater variation for Social Evolution. Further runs and tests on other networks might be necessary to confirm whether these best performing variants have a significant increase, though it may be small. The size of the neighbourhood sample was not shown to have a particular pattern, but there were patterns in the results for both neighbourhood sample type and final transformation. For Wikipedia, the uniform sample performed best, while for Social Evolution, the uniform performed the worst. Instead the degree and recent samples were best and comparable for the Social Evolution network. For final message transformation, the MLP transformation was found to be better for both the Wikipedia and Social Evolution networks. Time needed to run the variants with additional neighbourhood information was approximately 4-5 times the baseline variant, totalling approximately 12 and 20 minutes for 30 epochs for Wikipedia and Social Evolution respectively. This is thus a small enough difference to consider using the proposed variants.

Future work to continue this project would include looking at all proposed variations to the message function and message aggregation. Due to time and computational considerations, only a small portion were tested, but looking at the highest closeness sample, using an edited message instead of a copy, and using an MLP for message aggregation are all changes that can be later explored. Additionally, testing on additional dynamic networks would allow for a confirmation of a general pattern in the performance of the variants. It also will allow for an analysis of whether the best performing variants are specific to a network, a group of network with similar properties, or are even more general. Finally, a future task would be to study the effect of using a different embedding method with the proposed message function and message aggregation changes. This could provide evidence to support whether the neighbourhood message sharing has a small impact due to the effect of using graph attention embedding.

REFERENCES

[1] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2016. Geometric deep learning: going beyond Euclidean data. *CoRR* abs/1611.08097 (2016). http://arxiv.org/abs/1611.08097

[2] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR* abs/1409.1259 (2014). http://arxiv.org/abs/1409.1259

[3] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR* abs/1412.3555 (2014). http://arxiv.org/abs/1412.3555

[4] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2019. Relational Representation Learning for Dynamic (Knowledge) Graphs: A Survey. *CoRR* abs/1905.11485 (2019). http://arxiv.org/abs/1905.11485

[5] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. *CoRR* abs/1908.01207 (2019). http://arxiv.org/abs/1908.01207

[6] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *CoRR* abs/2006.10637 (2020). https://arxiv.org/abs/2006.10637

[7] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2018. Representation Learning over Dynamic Graphs. *CoRR* abs/1803.04051 (2018). http://arxiv.org/abs/1803.04051

[8] Yanbang Wang, Yen-Yu Chang, Yunyu Liu, Jure Leskovec, and Pan Li. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. *CoRR* abs/2101.05974 (2021). https://arxiv.org/abs/2101.05974

[9] Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive Representation Learning on Temporal Graphs. *CoRR* abs/2002.07962 (2020). https://arxiv.org/abs/2002.07962

[10] Bing Yu, Mengzhang Li, Jiyong Zhang, and Zhanxing Zhu. 2019. 3D Graph Convolutional Networks with Temporal Graphs: A Spatial Information Free Framework For Traffic Forecasting. *CoRR* abs/1903.00919 (2019). http://arxiv.org/abs/1903.00919

[11] Muhan Zhang and Yixin Chen. 2018. Link Prediction Based on Graph Neural Networks. *CoRR* abs/1802.09691 (2018). http://arxiv.org/abs/1802.09691