Tensor Graph Neural Networks for Learning on Time Varying Graphs

Osman Asif Malik
University of Colorado Boulder
osman.malik@colorado.edu

Shashanka Ubaru
IBM Research
shashanka.ubaru@ibm.com

Lior Horesh
IBM Research
lhoresh@us.ibm.com

Misha E. Kilmer
Tufts University
misha.kilmer@tufts.edu

Haim Avron
Tel Aviv University
haimav@tauex.tau.ac.il

Abstract

Many irregular domains such as social networks, financial transactions, neuron connections, and natural language structures are represented as graphs. A variety of graph neural networks (GNNs) have been successfully applied for representation learning and prediction on such graphs. However, in many of the applications, the underlying graph changes over time and existing GNNs are inadequate for handling such time varying graphs. In this paper we propose a novel technique for learning embeddings of time varying graphs based on a tensor framework. The method extends the popular graph convolutional network (GCN) for learning representations of time varying graphs using the recently proposed tensor M-product technique. Numerical experiments on four real datasets demonstrate that our proposed method outperforms a baseline method when used for edge classification.

1 Introduction

Representation and learning from *time varying*, or *dynamic*, graphs have numerous applications. Examples include analyzing social networks [1], predicting collaboration in citation networks [15], detecting fraud and crime in financial networks [21, 25], traffic control [29], and understanding neuronal activities in the brain [5]. Recently, graph neural networks (GNNs) such as the graph convolutional network (GCN) [11] have become popular tools for exploring graph structured data [26]. However, most GNNs assume the graph to be static, and efficient GNN methods that handle time varying graphs are lacking. In this paper, we propose a novel tensor variant of GCN for learning embeddings of time varying graphs which we call *TensorGCN*. It captures correlation over time by leveraging the tensor M-product framework [3, 8, 9]. GCN was inspired by graph filtering theory, and we can establish a similar tensor based theory for our TensorGCN. Elements of our method can also be used as a preprocessing step for other dynamic graph methods.

Prediction on Time Varying Graphs: By *time varying graph*, we mean a sequence of graphs $(V, \mathbf{A}^{(t)}, \mathbf{X}^{(t)}), t \in \{1, 2, \ldots, T\}$, with a fixed set V of N nodes, adjacency matrices $\mathbf{A}^{(t)} \in \mathbb{R}^{N \times N}$, and graph feature matrices $\mathbf{X}^{(t)} \in \mathbb{R}^{N \times F}$ where $\mathbf{X}^{(t)}_{n:} \in \mathbb{R}^F$ is the feature vector consisting of F features associated with node n at time t. The graphs can be weighted, and directed or undirected. They can also have additional properties like (time varying) node and edge classes, which would be stored in a separate structure. Suppose we only observe the first $T' < T$ graphs in the sequence. The goal of our method is to use these observations to predict some property of the remaining $T - T'$ graphs. In this paper, we consider the problem of edge classification. Other potential applications are node classification and edge/link prediction.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

The Tensor M-Product Framework: This framework relies on a new definition of the product of two tensors, called the M-product [3, 8–10]. A distinguishing feature of this framework is that the M-product of two three-dimensional tensors is also three-dimensional, which is not the case for e.g. tensor contractions [2].

Definition 1.1 (M-transform)**.** Let $\mathbf{M} \in \mathbb{R}^{T\times T}$ be a mixing matrix. The *M-transform* of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I\times J\times T}$ is denoted by $\boldsymbol{\mathcal{A}} \times_3 \mathbf{M} \in \mathbb{R}^{I\times J\times T}$ and defined elementwise as

$$(\boldsymbol{\mathcal{A}} \times_3 \mathbf{M})_{ijt} = \sum_{k=1}^{T} \mathbf{M}_{tk}\boldsymbol{\mathcal{A}}_{ijk}.$$

We say that $\boldsymbol{\mathcal{A}} \times_3 \mathbf{M}$ is in the *transformed space*. Note that if $\mathbf{M}$ is invertible, then $(\boldsymbol{\mathcal{A}} \times_3 \mathbf{M}) \times_3 \mathbf{M}^{-1} = \boldsymbol{\mathcal{A}}$. Consequently, $\boldsymbol{\mathcal{A}} \times_3 \mathbf{M}^{-1}$ is the *inverse M-transform* of $\boldsymbol{\mathcal{A}}$.

Definition 1.2 (Facewise product)**.** Let $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I\times J\times T}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J\times K\times T}$ be two tensors. The *facewise product*, denoted by $\boldsymbol{\mathcal{A}} \triangle \boldsymbol{\mathcal{B}} \in \mathbb{R}^{I\times K\times T}$, is defined facewise as $(\boldsymbol{\mathcal{A}} \triangle \boldsymbol{\mathcal{B}})_{::t} \stackrel{\text{def}}{=} \boldsymbol{\mathcal{A}}_{::t}\boldsymbol{\mathcal{B}}_{::t}$.

Definition 1.3 (M-product)**.** Let $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I\times J\times T}$ and $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{J\times K\times T}$ be two tensors, and let $\mathbf{M} \in \mathbb{R}^{T\times T}$ be an invertible matrix. The *M-product*, denoted by $\boldsymbol{\mathcal{A}} \star \boldsymbol{\mathcal{B}} \in \mathbb{R}^{I\times K\times T}$, is defined as

$$\boldsymbol{\mathcal{A}} \star \boldsymbol{\mathcal{B}} \stackrel{\text{def}}{=} ((\boldsymbol{\mathcal{A}} \times_3 \mathbf{M}) \triangle (\boldsymbol{\mathcal{B}} \times_3 \mathbf{M})) \times_3 \mathbf{M}^{-1}.$$

In the original formulation of the M-product, $\mathbf{M}$ was chosen to be the Discrete Fourier Transform (DFT) matrix, which allows efficient computation using the Fast Fourier Transform (FFT) [3, 9, 10]. The framework was later extended for arbitrary invertible matrices (e.g. discrete cosine and wavelet transforms) [8], and for tensors of dimension greater than three [19]. The M-product framework allows one to elegantly generalize many classical numerical methods from linear algebra, and has been applied e.g. in neural networks [20], imaging [10, 19, 22], facial recognition [7], and tensor completion and denoising [16, 27, 28]. The M-product framework helps us define eigendecompositions and functions of tensors (called *t-eig* and *t-function*), see [9, 17] for details.

Other Related Work: There are many works that deal with graph neural networks when the graph and features are fixed/static; see e.g. [4, 6, 11] for GCNs, as well as the review papers [26, 30] and references therein. These methods cannot be directly applied to the dynamic setting we consider.

Manessi et al. [18] appear to be one of the first to apply GNNs to dynamic graphs. Their approach uses Long Short-Term Memory networks (LSTMs) along with GNNs in order to handle time variations. Seo et al. [23] present the Graph Convolutional Recurrent Network for graphs with time varying features. However, this method assumes that the edges are fixed over time, and is not applicable in our setting. Wang et al. [24] present a method called EdgeConv, which is a neural network (NN) approach that applies convolution operations on static graphs in a dynamic fashion. Their approach is not applicable when the graph itself is dynamic. Zhao et al. [29] develop a temporal GCN method called T-GCN, which they apply for traffic prediction. Their method assumes the graph remains fixed over time. Pareja et al. [21] present a variant of GCN where Gated Recurrent Units (GRUs) are coupled with a GCN to handle dynamic graphs. This paper is currently state-of-the-art. However, their approach is based on a heuristic RNN/GRU mechanism, which is not theoretically viable, and does not harness a tensor algebraic framework to incorporate time varying information. Newman et al. [20] present a tensor NN which utilizes the M-product tensor framework. Their approach can be applied to image and other high-dimensional data that lie on regular grids, and differs from ours since we consider data on time varying graphs.

2 Tensor Dynamic Graph Embedding

Our approach is inspired by the first order GCN in [11] for static graphs. For a graph with adjacency matrix $\mathbf{A}$ and feature matrix $\mathbf{X}$, a layer in that GCN takes the form $\mathbf{Y} = \sigma(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W})$, where $\tilde{\mathbf{A}} \stackrel{\text{def}}{=} \tilde{\mathbf{D}}^{-1/2}(\mathbf{A} + \mathbf{I})\tilde{\mathbf{D}}^{-1/2}$, $\tilde{\mathbf{D}}$ is diagonal with $\tilde{\mathbf{D}}_{ii} = 1 + \sum_j \mathbf{A}_{ij}$, $\mathbf{I}$ is the matrix identity, $\mathbf{W}$ is a matrix to be learned when training the NN, and σ is an activation function. Our goal is to convert this model to a tensor model by utilizing the M-product framework. To this end, we first introduce a tensor activation function $\hat{\sigma}$ which operates in the transformed space.

Definition 2.1. Let $\mathcal{A} \in \mathbb{R}^{I \times J \times T}$ be a tensor and σ an elementwise activation function. We define the activation function $\hat{\sigma}$ as $\hat{\sigma}(\mathcal{A}) \stackrel{\text{def}}{=} \sigma(\mathcal{A} \times_3 \mathbf{M}) \times_3 \mathbf{M}^{-1}$.

We can now define our proposed dynamic graph embedding. Let $\mathcal{A} \in \mathbb{R}^{N \times N \times T}$ be a tensor with frontal slices $\mathcal{A}_{::t} = \tilde{\mathbf{A}}^{(t)}$, where $\tilde{\mathbf{A}}^{(t)}$ is the normalization of $\mathbf{A}^{(t)}$. Moreover, let $\mathcal{X} \in \mathbb{R}^{N \times F \times T}$ be a tensor with frontal slices $\mathcal{X}_{::t} = \mathbf{X}^{(t)}$. Finally, let $\mathcal{W} \in \mathbb{R}^{F \times F' \times T}$ be a weight tensor. We define our dynamic graph embedding as $\mathcal{Y} = \mathcal{A} \star \mathcal{X} \star \mathcal{W} \in \mathbb{R}^{N \times F' \times T}$. This computation can also be repeated in multiple layers. For example, a 2-layer formulation would be of the form

$$\mathcal{Y} = \mathcal{A} \star \hat{\sigma}(\mathcal{A} \star \mathcal{X} \star \mathcal{W}^{(0)}) \star \mathcal{W}^{(1)}.$$

One important consideration is how to choose the matrix $\mathbf{M}$ which defines the M-product. For time varying graphs, we choose $\mathbf{M}$ to be lower triangular and banded so that the frontal slice $(\mathcal{A} \times_3 \mathbf{M})_{::t}$ will be a linear combination of the adjacency matrices $\mathcal{A}_{::\max(1,t-b+1)}, \ldots, \mathcal{A}_{::t}$, where we refer to b as the "bandwidth" of $\mathbf{M}$. Specifically, the entries in $\mathbf{M}$ on the diagonal and first $b-1$ subdiagonals are set to 1, and all other entries are set to zero. $\mathbf{M}$ is then normalized so that each row has unit ℓ_1-norm. This choice ensures that each frontal slice $(\mathcal{A} \times_3 \mathbf{M})_{::t}$ only contains information from current and past graphs that are close temporally. Another possibility is to treat $\mathbf{M}$ as a parameter matrix to be learned from the data. The M-product framework lets us define a tensor graph Fourier transform. This helps us establish the relation between our TensorGCN and graph filtering, similar to the original GCN in [4] (we omit details here due to the space constraint).

An embedding $\mathcal{Y} \in \mathbb{R}^{N \times F' \times T}$ can now be used for various prediction tasks, like node and edge classification and link prediction. In Section 3, we apply our method for edge classification by using a model similar to that in [21]. Given an edge between nodes m and n at time t, the predictive model is $p(m, n, t) \stackrel{\text{def}}{=} \text{softmax}(\mathbf{U}[(\mathcal{Y} \times_3 \mathbf{M})_{m:t}, (\mathcal{Y} \times_3 \mathbf{M})_{n:t}]^\top)$ where $(\mathcal{Y} \times_3 \mathbf{M})_{m:t} \in \mathbb{R}^{F'}$ and $(\mathcal{Y} \times_3 \mathbf{M})_{n:t} \in \mathbb{R}^{F'}$ are row vectors, $\mathbf{U} \in \mathbb{R}^{C \times 2F'}$ is a weight matrix, and C the number of classes.

3 Numerical Experiments

Here, we present results for edge classification on four datasets[1]: The Bitcoin Alpha and OTC transaction datasets [12], the Reddit body hyperlink dataset [13], and a chess results dataset [14]. The bitcoin datasets consist of transaction histories for users on two different platforms. Each node is a user, and each directed edge indicates a transaction and is labeled with an integer between -10 and 10 which indicates the senders trust for the receiver. We convert these labels to two classes: positive (trustworthy) and negative (untrustworthy). The Reddit dataset is build from hyperlinks from one subreddit to another. Each node represents a subreddit, and each directed edge is an interaction which is labeled with -1 for a hostile interaction or $+1$ for a friendly interaction. We only consider those subreddits which have a total of 20 interactions or more. In the chess dataset, each node is a player, and each directed edge represents a match with the source node being the white player and the target node being the black player. Each edge is labeled -1 for a black victory, 0 for a draw, and $+1$ for a white victory. For each node-time pair (n, t) in these graphs, we compute the number of outgoing and incoming edges and use these two numbers as features.

The bitcoin and Reddit datasets are heavily skewed, with about 90% of edges labeled positively, and the remaining labeled negatively. Since the negative instances are more interesting to identify (e.g. to prevent financial fraud or online hostility), we use the F1 score to evaluate the experiments on these datasets, treating the negative edges as the ones we want to identify. The classes are more well-balanced in the chess dataset, so we use accuracy to evaluate those experiments.

The data is first temporally partitioned into T graphs (T is different for each dataset). Since the corresponding adjacency matrices are very sparse for these datasets, we apply the same technique as in [21] and add the entries of each adjacency matrix $\mathbf{A}^{(t)}$ to the following $l-1$ adjacency matrices $\mathbf{A}^{(t+1)}, \ldots, \mathbf{A}^{(t+l-1)}$, where we refer to l as the "edge life." The adjacency tensor $\mathcal{A}$ is then constructed as described in Section 2. Note that this only affects $\mathcal{A}$, and that the added edges are not treated as real edges in the classification problem. After this, the T frontal slices of $\mathcal{A}$ are divided into S_{train} training slices, S_{val} validation slices, and S_{test} testing slices, which come sequentially after each other; see Figure 1 and Table 1.

[1] We provide links to these datasets in Section A of the supplementary material.

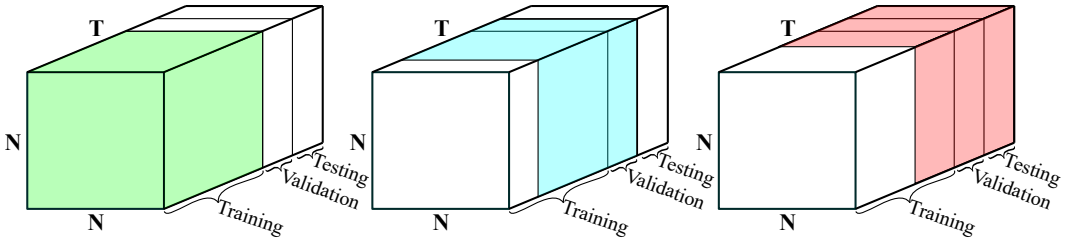

Figure 1: Partitioning of $\mathcal{A}$ into training, validation and testing data.

Table 1: Partitioning and results for each dataset. * and † indicate if the results are given in terms of F1 score or accuracy, respectively; higher is better in both cases. "Symmetric" and "Asymmetric" indicate if the adjacency matrices were symmetrized or not.

	Partitioning			Symmetric		Asymmetric	
Dataset	S_{train}	S_{val}	S_{test}	Proposal	Baseline	Proposal	Baseline
Bitcoin OTC*	95	20	20	**0.3103**	0.0769	**0.3529**	0.3317
Bitcoin Alpha*	95	20	20	**0.2207**	0.1538	**0.2331**	0.2100
Reddit*	66	10	10	**0.2071**	0.1966	**0.2028**	0.1805
Chess†	80	10	10	**0.4713**	0.4369	**0.4708**	0.4342

We choose to use an embedding $\mathcal{Y}_{\text{train}} = \mathcal{A}_{::(1:S_{\text{train}})} \star \mathcal{X}_{::(1:S_{\text{train}})} \star \mathcal{W}$ for training. When computing the embeddings for the validation and testing data, we still need S_{train} frontal slices of $\mathcal{A}$, which we get by using a sliding window of slices. This is illustrated in Figure 1, where the green, blue and red blocks show the frontal slices used when computing the embeddings for the training, validation and testing data, respectively. The embeddings for the validation and testing data are $\mathcal{Y}_{\text{val}} = \mathcal{A}_{::(S_{\text{val}}+1:S_{\text{val}}+S_{\text{train}})} \star \mathcal{X}_{::(S_{\text{val}}+1:S_{\text{val}}+S_{\text{train}})} \star \mathcal{W}$ and $\mathcal{Y}_{\text{test}} = \mathcal{A}_{::(S_{\text{val}}+S_{\text{test}}+1:T)} \star \mathcal{X}_{::(S_{\text{val}}+S_{\text{test}}+1:T)} \star \mathcal{W}$, respectively. We choose to learn $\mathcal{W}$ directly in the transformed domain, where we assume that each frontal slice is the same (i.e., that $(\mathcal{W} \times_3 \mathbf{M})_{::t} = (\mathcal{W} \times_3 \mathbf{M})_{::t'}$ for all t, t') in order to reduce the number of learnable parameters. For training, we use the cross entropy loss function:

$$\text{loss} = -\sum_{t} \sum_{(m,n)\in E_t} \sum_{c=1}^{C} \alpha_c f(m,n,t)_c \log(p(m,n,t)_c), \tag{1}$$

where $f(m,n,t) \in \mathbb{R}^C$ is a one-hot vector encoding the true class of the edge (m,n) at time t, and $\alpha \in \mathbb{R}^C$ is a vector summing to 1 which contains the weight of each class. Since the bitcoin and Reddit datasets are so skewed, we weigh the minority class more heavily in the loss function for those datasets, and treat α as a hyperparameter; see Section B of the supplement for details.

The experiments are implemented in PyTorch with some preprocessing done in Matlab. Our code will eventually be made available at `https://github.com/OsmanMalik`. In the experiments, we use an edge life of $l = 10$ and bandwidth $b = 20$. Since the graphs in the considered datasets are directed, we also investigate the impact of symmetrizing the adjacency matrices. Table 1 shows the results for our method and a 1-layer version of the GCN baseline described in [21]. Our method outperforms the baseline in all cases. For the bitcoin datasets, symmetrizing the adjacency matrices decreases performance, especially for the baseline. For the Reddit and chess datasets, symmetrization does not have a significant impact on either method.

4 Conclusion

We have presented a novel approach for embedding of time varying, or dynamic, graphs which leverages the tensor M-product framework. We used it for edge classification in experiments on four real datasets, outperforming a baseline method in all cases. Future research directions include developing the theoretical guarantees for the method, investigating optimal structure and learning of the transform matrix $\mathbf{M}$, designing deep networks utilizing our tensor embedding, and using the method for other prediction tasks.

References

[1] Tanya Y. Berger-Wolf and Jared Saia. A framework for analysis of dynamic social networks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 523–528. ACM, 2006.

[2] Richard L. Bishop and Samuel I. Goldberg. *Tensor Analysis on Manifolds*. Courier Corporation, 2012.

[3] Karen Braman. Third-order tensors as linear operators on a space of matrices. *Linear Algebra and its Applications*, 433(7):1241–1253, 2010.

[4] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[5] Fabrizio De Vico Fallani, Jonas Richiardi, Mario Chavez, and Sophie Achard. Graph analysis of functional brain networks: Practical issues in translational neuroscience. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1653):20130521, 2014.

[6] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[7] Ning Hao, Misha E. Kilmer, Karen Braman, and Randy C. Hoover. Facial recognition using tensor-tensor decompositions. *SIAM Journal on Imaging Sciences*, 6(1):437–463, 2013.

[8] Eric Kernfeld, Misha Kilmer, and Shuchin Aeron. Tensor–tensor products with invertible linear transforms. *Linear Algebra and its Applications*, 485:545–570, 2015.

[9] Misha E. Kilmer and Carla D. Martin. Factorization strategies for third-order tensors. *Linear Algebra and its Applications*, 435(3):641–658, 2011.

[10] Misha E. Kilmer, Karen Braman, Ning Hao, and Randy C. Hoover. Third-order tensors as operators on matrices: A theoretical and computational framework with applications in imaging. *SIAM Journal on Matrix Analysis and Applications*, 34(1):148–172, 2013.

[11] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[12] Srijan Kumar, Francesca Spezzano, V. S. Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 221–230. IEEE, 2016.

[13] Srijan Kumar, William L. Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. In *Proceedings of the 2018 World Wide Web Conference*, pages 933–943. International World Wide Web Conferences Steering Committee, 2018.

[14] Jérôme Kunegis. Konect: The koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

[15] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 177–187. ACM, 2005.

[16] Canyi Lu, Jiashi Feng, Yudong Chen, Wei Liu, Zhouchen Lin, and Shuicheng Yan. Tensor robust principal component analysis: Exact recovery of corrupted low-rank tensors via convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5249–5257, 2016.

[17] Kathryn Lund. The tensor t-function: A definition for functions of third-order tensors. *arXiv preprint arXiv:1806.07261*, 2018.

[18] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, 2020.

[19] Carla D. Martin, Richard Shafer, and Betsy LaRue. An order-p tensor factorization with applications in imaging. *SIAM Journal on Scientific Computing*, 35(1):A474–A490, 2013.

[20] Elizabeth Newman, Lior Horesh, Haim Avron, and Misha Kilmer. Stable Tensor Neural Networks for Rapid Deep Learning. *arXiv preprint arXiv:1811.06569*, 2018.

[21] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E. Leisersen. EvolveGCN: Evolving graph convolutional networks for dynamic graphs. *arXiv preprint arXiv:1902.10191*, 2019.

[22] Oguz Semerci, Ning Hao, Misha E. Kilmer, and Eric L. Miller. Tensor-based formulation and nuclear norm regularization for multienergy computed tomography. *IEEE Transactions on Image Processing*, 23(4):1678–1693, 2014.

[23] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[24] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018.

[25] Mark Weber, Jie Chen, Toyotaro Suzumura, Aldo Pareja, Tengfei Ma, Hiroki Kanezashi, Tim Kaler, Charles E. Leiserson, and Tao B. Schardl. Scalable Graph Learning for Anti-Money Laundering: A First Look. *arXiv preprint arXiv:1812.00076*, 2018.

[26] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[27] Zemin Zhang and Shuchin Aeron. Exact tensor completion using t-SVD. *IEEE Transactions on Signal Processing*, 65(6):1511–1526, 2016.

[28] Zemin Zhang, Gregory Ely, Shuchin Aeron, Ning Hao, and Misha Kilmer. Novel methods for multilinear data completion and de-noising based on tensor-SVD. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3842–3849, 2014.

[29] 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*, 2019.

[30] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Supplementary Material

A Links to Datasets

- The Bitcoin Alpha dataset is available at https://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html.
- The Bitcoin OTC dataset is available at https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html.
- The Reddit dataset is available at https://snap.stanford.edu/data/soc-RedditHyperlinks.html. Note that we use the dataset with hyperlinks in the body of the posts.
- The chess dataset is available at http://konect.uni-koblenz.de/networks/chess.

B Further Details on the Experiment Setup

When partitioning the data into T graphs, as described in Section 3, if there are multiple data points corresponding to an edge (m, n) for a given time step t, we only add that edge once to the corresponding graph and set the label equal to the sum of the labels of the different data points. For example, if bitcoin user m makes three transactions to user n during time step t with ratings 10, 2 and -1, then we add the edge (m, n) once to graph t with label $10 + 2 - 1 = 11$.

For training, we run gradient descent with a learning rate of 0.01 and momentum of 0.9 for 10,000 iterations. For each 100 iterations, we compute and store the performance of the model on the validation data. As mentioned in Section 3, the weight vector α in the loss function (1) is treated as a hyperparameter in the bitcoin and Reddit experiments. Since these datasets all have two edge classes, let α_0 and α_1 be the weights of the minority (negative) class and majority (positive) class, respectively. Since these parameters add to 1, we have $\alpha_1 = 1 - \alpha_0$. For both our proposed TensorGCN and the baseline, we repeat the bitcoin and Reddit experiments once for each $\alpha_0 \in \{0.75, 0.76, \ldots, 0.95\}$. For each model and dataset, we then find the best stored performance of the model on the validation data across all α_0 values. We then treat the corresponding model as the trained model, and report its performance on the testing data in Table 1. The results for the chess experiment are computed in the same way, but only for a single vector $\alpha = [1/3,\ 1/3,\ 1/3]$.