AdamDGN: Adaptive Memory using Dynamic Graph Networks for Staleness Problem in Recommender System

IL-JAE KWON[†], Seoul National University, South Korea
KYUNG-MIN KIM, NAVER CLOVA, NAVER AI LAB, South Korea
JISU JEONG, NAVER CLOVA, NAVER AI LAB, South Korea
KYUYONG SHIN, NAVER CLOVA, NAVER AI LAB, South Korea
YOUNG-JIN PARK, NAVER CLOVA, NAVER AI LAB, South Korea
BYOUNG-TAK ZHANG, Seoul National University, South Korea

Graph Neural Networks (GNNs) have proved their effectiveness in various recommendation tasks with their ability to incorporate relational information. However, a staleness problem in a recommendation task has been less explored in the literatures on graph learning, evaluating their performances on datasets with unrealistic distribution of users. In practical applications, ratio of "cool" users, who cool down to a service, dominates that of loyal users yielding an extreme sparsity problem for recommender systems. In this paper, we bring DeepCluster strategy to a memory-based temporal graph model for an online adaptive graph learning method, AdamDGN, that allows all nodes to be adaptively updated as new events are introduced, irrespective of their involveness. We evaluated on Amazon product review datasets, and AdamDGN outperforms all other baselines with significant margins on both two datasets.

ACM Reference Format:
Il-jae Kwon[†], Kyung-Min Kim, Jisu Jeong, Kyuyong Shin, Young-Jin Park, and Byoung-Tak Zhang. 2021. AdamDGN: Adaptive Memory using Dynamic Graph Networks for Staleness Problem in Recommender System. 1, 1 (May 2021), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In recent years, learning from graph structured data emerged as a critical role in machine learning. While traditional machine learning methods have only considered data points that are spread on an euclidean space, increasing number of real world problems require an understanding on structured data. For example, in social media, everything is about interactions between social members. Many hidden attributes of a social member can be revealed by analyzing interactions that the social member makes with one another. Utilizing such rich information from interactions, we get to understand each social members in more in-depth manner. Various downstream tasks can be derived using

[†]This work done when the author worked at NAVER CLOVA.

Authors' addresses: Il-jae Kwon[†], kwonij2@snu.ac.kr, Seoul National University, Seoul, South Korea; Kyung-Min Kim, kyungmin.kim.ml@navercorp.com, NAVER CLOVA, and NAVER AI LAB, Seongnam, South Korea; Jisu Jeong, NAVER CLOVA, and NAVER AI LAB, Seongnam, South Korea, jisu.jeong@navercorp.com; Kyuyong Shin, NAVER CLOVA, and NAVER AI LAB, Seongnam, South Korea, ky.shin@navercorp.com; Young-Jin Park, NAVER CLOVA, and NAVER AI LAB, Seongnam, South Korea, young.j.park@navercorp.com; Byoung-Tak Zhang, Seoul National University, Seoul, South Korea, btzhang@bi.snu.ac.kr.

Manuscript submitted to ACM

in-depth representation of the social member such as political preference classification, friend suggestion or community detection. On the other hand, e-commerce market is another fine example of commonly used graph structured data, which in this case has heterogeneous kinds of nodes represented as users and items, connected by purchase histories as edges. By leveraging purchase histories between users and items, we can further recommend various items to users.

Many works on graph neural networks (GNNs) [10] have been proposed to exploit underlying rich attributes of graph structured data. GCNs [4] was one of the keystone works that promoted graph neural networks to a practical level. GCNs use an efficient layer-wise propagation rule by approximating the first-order of spectral graph convolution. By limiting the spectral convolution to the first-order, GCNs not only lightened computation cost of the operation but also alleviated the over-smoothing problem that previous spectral methods had. Message Passing Neural Networks (MPNN) [3] was presented as a general form of spatial convolution operation and treated GCNs as specific kind of a message passing process. In MPNNs, information between nodes is delivered directly by edges without visiting any spectral domains.

Recommender system is one of the popular downstream task of GNNs that is widely used in the real applications such as friend recommendation, movie recommendation and purchase recommendation [2]. Recommendation task aims to predict possible links from a heterogeneous graph containing two types of nodes; users and items. Taking advantage of GNNs' ability to exploit rich structural attribute of data, GNN based methods achieved remarkable success in recommender system [6, 11, 13, 14]. However, unlike recommendation on static graphs, there has been limited studies done for recommender systems on dynamically evolving graphs, also called as temporal graphs. To solve recommendation problem in dynamic graphs, it is important to learn temporal representation out of a sequence of events and learn how user's preference evolves. Recurrent Neural Networks (RNNs) and Transformers are popular building block used to understand temporal interaction data [5, 9, 12] on top of traditional GNNs. Traditional GNNs can be applied to snapshots of dynamic graph (discrete-time dynamic graph) where temporal networks such as RNNs aggregate all embeddings from snapshots to solve temporal recommeder system. However dicrete-time dynamic graph may overlook some critical information as edge addition or deletion, ending up in incomplete representation learning. In contrast to discrete-time dynamic model, recent literature tackled temporal recommender system by utilizing continuous-time graph [5, 9, 12], outperforming discrete-time models. TGN [9] is one of the successful models that uses GRU embedded node memory as state vector to aggregate history interactions followed by GNNs to build node representation. JODIE [5], on the other hand, had similar memory structure but used time-based prediction MLP module instead of GNNs. They both tested on preprocessed datasets that were built from selected users who have at least 5 interaction histories. However these preprocessed datasets are not practical in real world application, where majority of users easily cool down to a service, leaving only a few interaction histories left to refer on.

In this paper we propose AdamDGN that performs well in the real world setting, taking advantage of cluster adaptation stage of our own. Our model uses clustering and adaptation stage to update out-dated memories along with other frequently updated memories which are in the same cluster, in order to provide up-to-date representation for cool users. In the next section, we illustrates some preliminaries for our model. In Sections 3 and 4, we propose our novel model, AdamDGN, with experimental results.

2 PRELIMINARIES

2.1 DeepCluster

DeepCluster [7] is a clustering method that jointly learns parameters of Convolutional Neural Networks (CNNs) and cluster assignments of resulting features. It iteratively clusters on features, using the k-means algorithm with cluster assignments as supervision signals to train parameters of CNNs. More precisely, DeepCluster first determines cluster assignments y_n of input n and the centroid matrix C with the following equation:

$$\min_{C\in\mathbb{R}^{d\times k}} \frac{1}{N}\sum_{n=1}^{N} \min_{y_n\in\{0,1\}^k} \left\| f_\theta(x_n) - Cy_n \right\|_2^2 \quad \text{such that} \quad y_n^\top 1_k = 1. \tag{1}$$

where f_θ corresponds to CNNs. Then these assignments act as pseudo-labels when training the weights of CNN. To prevent trivial solutions such as assigning all inputs to a single cluster, a small random perturbation of centroid for non-empty cluster or sampling of inputs with a uniform distribution over the classes are used.

2.2 Temporal Graph Networks (TGNs)

TGNs [9] operate on a continuous-time dynamic graph built based on a sequence of events rather than a snapshot of the graph. A typical example of an event can be an interaction with another node or node-wise change. A memory module in TGNs keeps the contexts of nodes acquired from historical events, and embedding networks exploit the memory to learn the temporal properties of the nodes.

Formally, the embeddings of the graph nodes at time t, $\mathbf{Z}(t) = (\mathbf{z}_1(t), ..., \mathbf{z}_{n(t)}(t))$ can be formulated as follows:

$$\textbf{Message Function}: \mathbf{m}_i(t) = \mathrm{msg_s}(\mathbf{s}_i(t^-), \mathbf{s}_j(t^-), \Delta t, \mathbf{e}_{ij}(t)), \qquad \mathbf{m}_j(t) = \mathrm{msg_d}(\mathbf{s}_j(t^-), \mathbf{s}_i(t^-), \Delta t, \mathbf{e}_{ij}(t)) \tag{2}$$

$$\textbf{Message Aggregator}: \hat{\mathbf{m}}_i(t) = \mathrm{agg}(\mathbf{m}_i(t_1), ..., \mathbf{m}_i(t_b)) \tag{3}$$

$$\textbf{Memory Updater}: \mathbf{s}_i(t) = \mathrm{mem}(\hat{\mathbf{m}}_i(t), \mathbf{s}_i(t^-)) \tag{4}$$

$$\textbf{Node Embedding}: \mathbf{z}_i(t) = \mathrm{emb}(i,t) = \sum_{j\in N_i^k([0,t])} h(\mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}, \mathbf{v}_i(t), \mathbf{v}_j(t)) \tag{5}$$

where the equation 2 computes a message involving source and target nodes i, and j, respectively. $s_i(t^-)$ corresponds to the node i's memory block just before time t, and e_{ij} denotes event embedding. If there are multiple events involving the same node i in the same batch, they are aggregated with the equation 3. $\hat{m}_i(t)$, which summarizes the incoming events for node i within a batch, yields the update of node i's memory block $s_i(t)$ (equation 4). For interaction events including node i and j, the memory blocks of both nodes are updated. To compute the temporal embedding $z_i(t)$ of node i, the embedding networks use the memory blocks of node i and its neighborhood in the equation 5. $N_i^k([0,t])$ denotes the k-hop neighborhood of node i until time t. The choices of msg, agg, mem, emb can be optional, ranging from simple functions like concatenation, mean, or MLP to more complex ones such as GRU or GNNs with attention.

3 ADAMDGN: ADAPTIVE MEMORY USING DYNAMIC GRAPH NEURAL NETWORK

In this paper, we propose AdamDGN, an online adaptive memory model for dynamic graph learning, to solve the staleness problem in real world data. Key concern is how to correctly recommend items to cool users who have limited purchase histories. In order to tackle this concern, we take advantage of DeepCluster [7] method with our novel adaptation method to update cool users along with loyal users.

AdamDGN has 3 stages for one train cycle: Aggregation stage, Clustering stage and Adaptation stage. As AdamDGN is an online training method, every learning cycle occurs as new batch of sample sequences are fed.

3.1 Aggregation Stage

In the first stage, the aggregation stage, model computes aggregated representations for each nodes based on their latest memory and newly introduced messages from batch of events. Each event has source node and destination node with it's edge attribute. Batch of N events is defined as $\mathbf{B} = (\mathbf{s}, \mathbf{d}, \mathbf{t}, \mathbf{e})$, when $\mathbf{s} \in \mathbb{R}^{N\times 1}$, $\mathbf{d} \in \mathbb{R}^{N\times 1}$ are tensors of source nodes and destination nodes with $\mathbf{e} \in \mathbb{R}^{N\times E}$ as matrix of edge attributes. First we need to make a message vector from new event. The message of the single event is computed as following.

$$\mathbf{msg}_i^t = \mathrm{MSG}\left(\mathbf{m}_i^{t-1}, \mathbf{m}_j^{t-1}, \Delta\mathbf{t}, \mathbf{e}_{ij}^t\right) \tag{6}$$

, while $\mathbf{e}_{ij}$ is an edge attribute for the dynamic edge and $\mathbf{mem}_i$ is static memory of node i. There are several choices for the MSG function, where we chose simple concatenation for our model. Now memory for node i, at time stamp t can be computed as following.

$$\mathbf{m}_i^t = \mathrm{RNN}\left(\mathbf{msg}_i^t, \mathbf{m}_i^{t-1}\right) \tag{7}$$

There are several choices for aggregation method but we used simple RNN cell that takes message $\mathbf{msg}_i^t$ as an input and $\mathbf{m}_i^{t-1}$ as a state for simplicity.

3.2 Clustering Stage

Once memory for all involving nodes are computed, model now moves to the clustering stage. In this stage, a variant of Online Deep Cluster method is used for online learning. As new batches of events are fed, novel involving nodes will be added to the existing clusters. We first compute the soft cluster assignments of the memory $\mathbf{m}_i^t$ by computing similarities with centroid matrix, $\mathbf{C}$.

$$\tilde{l}_i^t = \mathrm{softmax}\left(\mathbf{m}_i^t \mathbf{C}^{t\top}\right), \text{ when } \tilde{l}_i^{t\top}\mathbf{1} = 1 \tag{8}$$

Then, hard cluster label for node memory i can be computed as below.

$$l_i^t = \mathrm{argmax}\left(\tilde{l}_i^t\right) \tag{9}$$

Once clustering is over, centroid matrix, $\mathbf{C}$, is updated by averaging updated memories.

$$c_l^t = \mathbb{E}_{i\in l}\, \mathbf{m}_i^t \tag{10}$$

As the original paper on DeepCluster [7] points out, there can be a trivial solution to cluster all nodes in a single cluster. To avoid this trivial solution, we redirect clusters that have less than certain amount of elements to re-cluster and divide the largest cluster into half to keep balanced number of elements throughout the clusters.

3.3 Adaptation Stage

As dynamic edges are created, memories for few involving nodes get updated. However, memories for nodes that are excluded from these events, keep holding their out-dated memory resulting in poor recommendation. To tackle this problem, we synchronize the static memories to the movements of centroids, so that local update on involving nodes globally affects other memories.

For adaptation, we use pseudo label $\tilde{l}_i^t$ to compute weight sum of centroids. Adapted memories are computed as follow.

$$\mathbf{m}_i^{t+1} = (1 - \beta)\mathbf{m}_i + \beta(\tilde{l}_i^t \mathrm{C}) \tag{11}$$

, while β is a hyperparameter that determines how much adaptation on memories to apply. If β is 1, memory will be fully adopted to movements of centroids.

3.4 Train

To initialize the model, memories and centroids are set to tensors of zeros. For each batch of events, we randomly select equal number of negative destination nodes, d^*, as positive destination nodes, d.

For time stamp t, resulting vector from Aggregation Stage 7 are now fed to decoder module to compute probability for link prediction task. There are few options for decoder module and we used a simple MLP.

$$\mathbf{p}_{pos} = \mathrm{Dec}\ (\mathbf{m_s}, \mathbf{m_d}),\ \ \mathbf{p}_{neg} = \mathrm{Dec}\ (\mathbf{m_s}, \mathbf{m_{d^*}}) \tag{12}$$

Now from link probability score from positive and negative samples, we now compute binary cross entropy loss for contrastive learning.

$$\mathbf{L}_{link} = \mathrm{BCE}\ (\mathbf{p}_{pos}, \mathbf{p}_{neg}) \tag{13}$$

On top of the main loss function, $\mathbf{L}_{link}$, we added self-supervised training goal for clustering. Under the prior that similar nodes will have similar pseudo label for clusters, we add $\mathbf{L}_{pseudo}$ which is computed as following.

$$\mathbf{p'}_{pos} = \tilde{l}_s^{\intercal}\, \tilde{l}_d,\ \ \mathbf{p'}_{neg} = \tilde{l}_s^{\intercal}\, \tilde{l}_{d^*} \tag{14}$$

$$\mathbf{L}_{pseudo} = \mathrm{BCE}\ (p'_{pos}, p'_{neg}) \tag{15}$$

At last, $\mathbf{L}_{total}$ for training is a weight sum between $\mathbf{L}_{link}$ and $\mathbf{L}_{pseudo}$ with hyperparameter α that decides how much pseudo loss we should consider.

$$\mathbf{L}_{total} = (1 - \alpha)\mathbf{L}_{link} + \alpha\mathbf{L}_{pseudo} \tag{16}$$

Once training is over for each batch, memory now moves to memory update mode where Clustering Stage and Adaptation Stage take place.

4 EXPERIMENT

4.1 Dataset

Previous works on continuous dynamic graph link prediction task held their experiments on preprocessed data that contains only the nodes with redundant edges. For instance, Reddit and Wikipedia datasets were filtered out to cool users who have less than five interaction histories. However, these datasets are far departed from the real world data in which huge portion of users are cool users, as shown in Table 1. Therefore in our experiment we tested on every users who have at least two interactions, only filtering out cold start users to focus on our problem. For experiment, we selected five categories of Amazon's purchase review dataset [8]: "Appliances", "Books", "Clothing, Shoes and Jewelry",

	Sequence Length	Users	Items	Avg. Degree of User	User ≤ 5	User = 2
Appliances	150,299	63,614	12,832	2.36	98.8%	78.9%
Books	901,976	269,959	14,680	3.34	91.5%	61.2%
Clothing, Shoes and Jewelry	871,874	325,356	12,832	2.68	95.3%	66.4%
Electronics	837,365	288,223	22,508	2.91	93.7%	61.8%
Movies and TV	997,559	247,689	83,604	4.03	88.9%	39.4%

Table 1. Analysis on Amazon datasets that were used for the experiment. In order to level how many cool users exist in Amazon data, we added two columns. Column name "User ≤ 5" indicates percentage of users who have less or equal to 5 purchase histories out of total unique users and in a similar vein, "User = 2" indicates percentage of users who have only 2 purchase histories.

Baselines		Appliances	Books	Clothing Shoes & Jewelry	Electronics	Movies & TV
Trend	AP	0.615 ± 0.038	0.647 ± 0.057	0.670 ± 0.072	0.665 ± 0.053	0.547 ± 0.014
	AUC	0.621 ± 0.037	0.654 ± 0.056	0.676 ± 0.072	0.670 ± 0.052	0.548 ± 0.014
TGAT	AP	0.533 ± 0.017	0.643 ± 0.022	0.648 ± 0.054	0.752 ± 0.025	0.618 ± 0.019
	AUC	0.560 ± 0.026	0.720 ± 0.026	0.652 ± 0.086	0.829 ± 0.023	0.678 ± 0.022
JODIE	AP	0.674 ± 0.026	0.730 ± 0.032	0.808 ± 0.035	0.835 ± 0.040	0.625 ± 0.018
	AUC	0.730 ± 0.026	0.800 ± 0.031	0.872 ± 0.028	0.873 ± 0.033	0.665 ± 0.020
TGN	AP	0.556 ± 0.016	0.631 ± 0.019	0.750 ± 0.023	0.667 ± 0.024	0.577 ± 0.016
	AUC	0.599 ± 0.025	0.706 ± 0.023	0.830 ± 0.021	0.714 ± 0.026	0.626 ± 0.021
AdamDGN	AP	**0.845 ± 0.025**	**0.844 ± 0.022**	**0.894 ± 0.019**	**0.847 ± 0.021**	**0.833 ± 0.024**
	AUC	**0.824 ± 0.025**	**0.896 ± 0.015**	**0.919 ± 0.013**	**0.904 ± 0.013**	**0.812 ± 0.025**

Table 2. Experimental results on dynamic graph link prediction task. Our model achieved best results on every datasets in a large gap. Second best results were highlighted on blue.

"Electronics" and "Movies and TV". For the ones that have too long sequence of events, we used first one million events for experiment. 70% of events sequence was used for training and 15% each for validation and testing. Batch size for events were fixed to 300 throughout the experiment. For validation and test, equal number of randomly selected negative samples were used.

4.2 Results

For experiment, we evaluated our model with four other baselines: Trend, TGAT [1], JODIE [5] and TGN [9]. For Trend model, we made a model to recommend on any items that were bought in previous batch of events and not to recommend others.

From Table 2, our model out-performed every other models in both AP and AUC score by a large gap in every datasets. Other than our model, JODIE performed the second best among other temporal graph models. TGN and TGAT showed poor performance even though TGN was a state-of-the-art performing model on public datasets: Reddit and Wikipedia. We assume that due to the sparsity of constructed graph, resulted from dominant number of cool users, graph attention network could not propagate enough messages from neighboring nodes. On the other hand, JODIE uses time dependent MLP module to predict how an old memory might change during the time difference, independent of number of interactions. Results show that in such setting where significant number of cool users exist, predicting future embedding of out-dated node is better than relying on message passing on sparse graph. AdamDGN uses clustering and adaptation stage to update out-dated memory along with other memories that are in the same cluster. Once loyal user's memory get frequently updated, nodes that are clustered in the same cluster will also get updated. As a result, clustering and adaptation strategy that our model uses, performs better than prediction by MLP or GNNs.

Interesting result is that AUC score from our model and Trend model has positive correlation throughout the datasets. Since Trend model recommends items that were purchased by other users a lot in a close past, model performs best when there is an explicit trend on purchase. As a result, Trend performs best in the order of Clothing Shoes & Jewelry, Electronic, Books, Appliances and Movies & TV, which is the same order from AdamDGN's result. We can easily infer from this correlation, that our model can successfully follow the trend of purchase, exploiting the benefit of cluster wise adaptations.

5 CONCLUSION

In this paper, we propose AdamDGN, an online adaptive memory model for graph learning to tackle staleness problem that can easily happen on real world recommender system. Our model uses cluster wise adaptation strategy to update cool users' outdated memories alongside to loyal users' up-to-date memories. Through experiments on Amazon product review data, we proved that our model performed best on every dataset. Not only for recommender system, but our model can also be used on other graph learning tasks facing sparsity problem which we leave it for a future work.

REFERENCES

[1] da Xu, chuanwei ruan, evren korpeoglu, sushant kumar, and kannan achan. 2020. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJeW1yHYwH

[2] Yang Gao, Yi-Fan Li, Yu Lin, Hang Gao, and Latifur Khan. 2020. Deep Learning on Knowledge Graph for Recommender System: A Survey. *arXiv preprint arXiv:2004.00387* (2020).

[3] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. PMLR, 1263–1272.

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

[5] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *Proceedings of the 25th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM.

[6] Zhiwei Liu, Lin Meng, Jiawei Zhang, and Philip S Yu. 2020. Deoscillated Graph Collaborative Filtering. *arXiv preprint arXiv:2011.02100* (2020).

[7] Armand Joulin Mathilde Caron, Piotr Bojanowski and Matthijs Douze. 2018. Deep Clustering for Unsupervised Learning of Visual Features. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[8] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 188–197.

[9] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML 2020 Workshop on Graph Representation Learning*.

[10] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[11] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. 2020. Learning to Hash with Graph Neural Networks for Recommender Systems. In *Proceedings of The Web Conference 2020*. 1988–1998.

[12] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. Dyrep: Learning representations over dynamic graphs. In *International Conference on Learning Representations*.

[13] Xiao Wang, Ruijia Wang, Chuan Shi, Guojie Song, and Qingyong Li. 2020. Multi-component graph convolutional collaborative filtering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 6267–6274.

[14] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.