MultiSage: Empowering GCN with Contextualized Multi-Embeddings on Web-Scale Multipartite Networks

Carl Yang*, Aditya Pal†, Andrew Zhai†, Nikil Pancha†, Jiawei Han‡, Charles Rosenberg†, Jure Leskovec†
*Emory University †Pinterest Inc. ‡University of Illinois at Urbana Champaign
*j.carlyang@emory.edu, †{apal, andrew, npancha, crosenberg, jure}@pinterest.com,‡hanj@illinois.edu

ABSTRACT

Graph convolutional networks (GCNs) are a powerful class of graph neural networks. Trained in a semi-supervised end-to-end fashion, GCNs can learn to integrate node features and graph structures to generate high-quality embeddings that can be used for various downstream tasks like search and recommendation. However, existing GCNs mostly work on homogeneous graphs and consider a single embedding for each node, which do not sufficiently model the multi-facet nature and complex interaction of nodes in real-world networks. Here, we present a *contextualized* GCN engine by modeling the multipartite networks of *target* nodes and their intermediate *context* nodes that specify the contexts of their interactions. Towards the neighborhood aggregation process, we devise a contextual masking operation at the feature level and a contextual attention mechanism at the node level to achieve interaction contextualization by treating neighboring target nodes based on intermediate context nodes. Consequently, we compute *multiple embeddings* for target nodes that capture their diverse facets and different interactions during graph convolution, which is useful for fine-grained downstream applications. To enable efficient web-scale training, we build a parallel random walk engine to pre-sample contextualized neighbors, and a Hadoop2-based data provider pipeline to pre-join training data, dynamically reduce multi-GPU training time, and avoid high memory cost. Extensive experiments on the bipartite Pinterest graph and tripartite OAG graph corroborate the advantage of the proposed system.

ACM Reference Format:
Carl Yang, Aditya Pal, Andrew Zhai, Nikil Pancha, Jiawei Han, Charles Rosenberg, Jure Leskovec. 2020. MultiSage: Empowering GCN with Contextualized Multi-Embeddings on Web-Scale Multipartite Networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi/10.1145/3394486.3403293

1 INTRODUCTION

Graph convolutional networks (GCNs) have recently received significant attention for the task of extracting information from large graphs [3, 12, 15, 16, 18, 22, 32]. This is in part due to their fundamental connection to spectral graph theory and thus provable representation power [4, 11, 38], as well as due to their promising performance on several graph mining benchmarks [10, 31, 45]. To harness the power of GCNs, GraphSage utilized batch-wise training through fixed-sized neighborhood sampling [10]. It was later adapted into a robust enterprise-scale version called PinSage [44] and deployed at Pinterest. PinSage was shown to be extremely effective at recommending similar pins based on the industry-scale pin-board graph. However, one key limitation of existing GCNs is that they cannot distinguish multi-facet node properties and complex node interactions, which manifests due to their homogeneous treatment of node links. As illustrated in Figure 1(a), in real-world industrial platforms like Pinterest, state-of-the-art GCN models mix all related nodes in a single embedding space.

Present work. In this work, we argue that nodes in a network are connected due to different reasons and are thus *close to each other in different ways*, which cannot be simultaneously captured by a single embedding. To this end, we propose MultiSage, which is based on a novel idea of *contextualized multi-embedding*, where we compute multiple embeddings for network nodes to capture their *contextualized interaction* in the corresponding *multiple embedding spaces*. Figure 1(b) illustrates the scenario where MultiSage retrieves and organizes nodes related to the query under different contexts in multiple embedding spaces. Our MultiSage answers two important questions: (1) how to find proper context; and (2) how to leverage context in massive real-world networks to facilitate effective and flexible downstream applications.

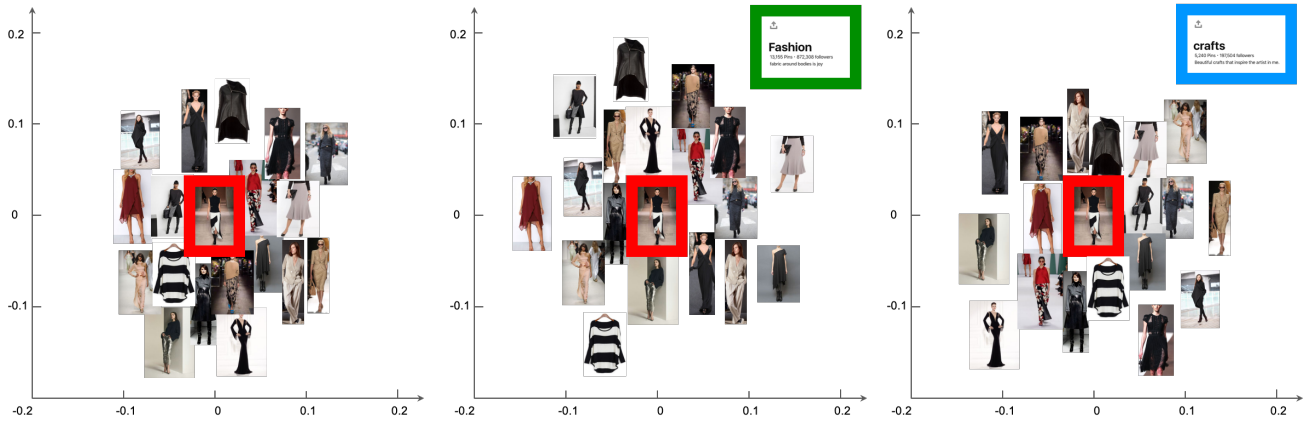
RQ 1: How to find proper context? Real-world applications often care most about particular types of nodes (*e.g.*, papers in academic graphs, users in social networks, *etc.* [8, 44, 48]). However, we observe that real-world networks are often multipartite, *i.e.*, including multiple types of nodes, which naturally provides the context of interactions between the nodes. Due to this observation, we find it beneficial to model *target nodes* and *context nodes* in multipartite networks, where the interactions between the target nodes can be subtly modeled via the help of context nodes.

To help illustrate this idea consider the example of Pinterest, where users interact with *pins* (*e.g.*, images in Figure 1) mostly by pinning them to personal *boards*, thus creating a massive bipartite pin-board graph. Since the embeddings of pins are critical for various downstream services like search and recommendation, we regard all pins as the target nodes and aim to compute high-quality contextualized multi-embeddings for them. At the same time, we regard each board node as a context node as it encodes key details of the relationship between the two pins that coexist on that board.

The intuition behind leveraging boards to contextualize interactions among pins is natural—for example, if the paths connecting two particular pins mostly pass through fashion boards, the embeddings of the two pins are likely close because they both describe fashion related items. Besides simplicity, we also find the

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403293



(a) PinSage Global Single-Embedding (b) MultiSage Contextualized Multi-Embedding

Figure 1: A toy example of related pin recommendation in Pinterest. In Subfigure (a), given the query pin (in red), PinSage computes a single pin embedding and mixes up all related pins. On the contrary, in Subfigure (b), we compute multiple pin embeddings based on different boards (*e.g.*, fashion and crafts), which naturally organizes related pins according to their contextualized distances to the query and effectively distinguishes relatedness in different perspectives (fashion models are drawn towards the query in the first space, whereas craft clothes in the second).

idea general—for example, on a publication network like OAG,[1] if two papers are mostly connected by paths passing through a data mining venue, they are likely close because they both study data mining problems.

RQ 2: How to leverage context? We propose MultiSage, which leverages ubiquitous graph context in real-world multipartite networks to extend GCN by injecting interaction contextualization into its critical neighborhood convolution process, where we dynamically compute multiple embeddings for each target node under the conditions implied by different context nodes. Particularly, we design a novel GCN architecture with a learnable contextual masking operation based on context node features for flexible feature-level embedding projection, and a three-way contextual attention mechanism for node-level neighbor reweighing during graph convolutions. To fully capture the rich information in web-scale networks, we further implement a parallel contextualized random walk engine and an efficient Hadoop2-based data provider pipeline to pre-join and dynamically feed training data to the multi-GPU model trainer, which allows scalable model training on massive networks with millions to billions of nodes.

We conduct extensive experiments and case studies on an enterprise Pinterest pin-board network as well as a public OAG publication network. The advantages of MultiSage are intriguing not only because it outperforms various state-of-the-art baselines with significant margins (9%-25% on MRR over the production model of PinSage) by incorporating rich and subtle network information, but also due to its corroborated utility in generating flexible and meaningful multi-embeddings that naturally paves the way to fine-grained search and recommendation.

[1] https://www.openacademic.ai/oag/

2 OUR APPROACH

2.1 Preliminaries

Following abundant recent works on GCN [3, 10, 22, 49], we take [15] proposed by Kipf and Welling as a representative to briefly recapitulate its main design. Particularly, the output of the $(l+1)$-th convolutional layer $\mathbf{H}^{(l+1)}$ of GCN is computed as follows

$$\mathbf{H}^{(l+1)} = \sigma\Big(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}\Big), \tag{1}$$

where $\tilde{\mathbf{A}}$ is the graph adjacency matrix with self-connections, $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$, $\mathbf{W}^{(l)}$ is the trainable layer-wise weight matrix, and $\sigma(\cdot)$ is a nonlinear activation function such as ReLU or Sigmoid. $\mathbf{H}^{(l)} \in \mathbb{R}^{N\times D_l}$ is the output of the l-th layer, with $\mathbf{H}^{(0)} = \mathbf{X}$, *i.e.*, the original node features.

One major drawback of [15] is the requirement of putting the whole graph (*i.e.*, $\tilde{\mathbf{A}}, \tilde{\mathbf{D}} \in \mathbb{R}^{N\times N}$) into the main memory (or GPU memory), which limits the training to graphs with only thousands of nodes. To address this issue, GraphSage [10] proposed to sample a fixed number of neighbors in each convolution layer and aggregate the neighborhood embedding as follows

$$\mathbf{h}^{(l+1)}_{\mathcal{N}(v)} = \text{AGGREGATE}(\{\mathbf{h}^{(l)}_u, \forall u \in \mathcal{N}(v)\}), \tag{2}$$

where $\mathcal{N}(v)$ is the sampled neighborhood of node v, and AGGREGATE is the aggregation function such as mean pooling.

To fully leverage the model capacity of GCN and scalability of GraphSage, PinSage [44] was developed at Pinterest for the particular task of related pin recommendations. To suit this real-world recommendation task, a series of techniques were adopted, while the major one lies in the triplet-wise optimization objective based on max-margin ranking as follows

$$\mathcal{J}(v_q, v_p, v_n) = \max\{0, \mathbf{h}^T_{v_q}\mathbf{h}_{v_n} - \mathbf{h}^T_{v_q}\mathbf{h}^L_{v_p} + \delta\}, \tag{3}$$

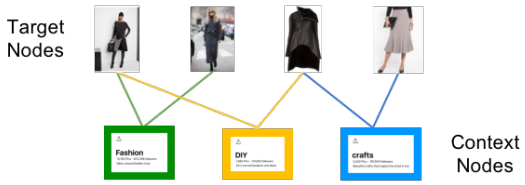


Figure 2: Target-context separation of Pinterest network.

where δ is a margin hyper-parameter. In each triplet (v_q, v_p, v_n), v_q and v_p are the *query* and *positive* nodes sampled from available training data (*e.g.*, related pin pairs generated from users' interactions with pins), while v_n is the *negative* node sampled from $P_n(v_q)$ (*i.e.*, the distribution of negative examples for v_q).

2.2 MultiSage

In this work, we leverage the heterogeneity of real-world networks by separating nodes into two main types: *target nodes* and *context nodes*. Our main focus is to learn embeddings of the target nodes, while using the context nodes to describe the relationship between the target nodes. Figure 2 gives an example of separating the multipartite Pinterest network into target nodes (*i.e.*, pins) and context nodes (*i.e.*, boards), where pins are related to each other via boards. Note that, we make two assumptions to achieve such desired simplifications of the otherwise complicated heterogeneous networks: (1) minimum domain knowledge is available to separate target nodes from context nodes; (2) most important interactions among target nodes involve context nodes. To show that both assumptions are general and realistic, we give examples of a few commonly used multipartite networks in Table 1. We note here that while in this work, we consider only one type of context node (boards), our framework can easily incorporate other types, such as users and sessions, simultaneously.

Similar to GraphSage, MultiSage learns an embedding for a given target node (say v - also referred to as ego target node) based on its neighboring target nodes ($\mathcal{N}_v$). However, unlike GraphSage which would employ the same aggregation function for all the neighbors, MultiSage leverages context nodes that underpin the interaction between the ego node and its neighbors. Let us first outline the importance of context during neighborhood aggregation via an example. In Figure 3 (a), when three neighbor pins are aggregated through mean pooling, the resulting neighborhood embedding simply lies in the center of the three pins, reflecting the same influence of all neighbors on the ego. In contrast in Figure 3 (b), two neighbor pins are connected via the fashion board while the other one via the crafts board, thus drawing the neighborhood

Dataset	Target	Context	Other context
IMDB [40]	movie	genre	director, actor
TCGA [43]	gene	pathway	disease, species
OAG [27]	paper	venue	author, keyword
Pinterest [6]	pin	board	user, session

Table 1: Target and context nodes on different networks.

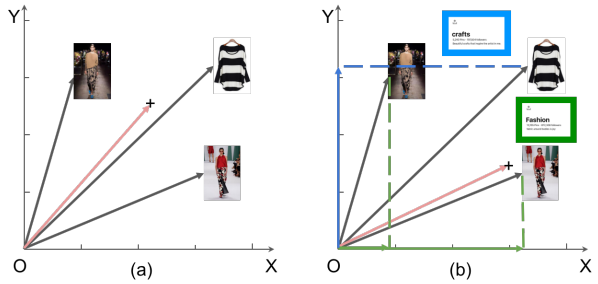


Figure 3: (a) Absence of context vs (b) presence of context.

embedding more into the fashion direction. Such contextualization over the target interaction is desirable, since each neighbor is similar to the ego from a particular perspective, and thus should influence the ego embedding more in the corresponding subspace. To achieve this, we need to retrieve the context node(s) between each ego and neighbor target node pair. For simplicity, we retrieve only one dominant context node that best encodes ego-neighbor interaction with a parallel contextualized random walk engine (details deferred to Section 2.3). As a result, each neighbor $u \in \mathcal{N}_v \subset \mathcal{T}$ of the ego $v \in \mathcal{T}$ is associated with a dominant context node $o \sim (v, u) \in C$.

Raw feature transformation. Since $|\mathcal{T}|$ and $|C|$ can both be very large for a real-world graphs (*e.g.*, billions), identity based embedding is impractical. To this end, we adopt the common practice of feature based embedding for both target and context nodes, which learns to project and transform raw node features via stacked dense neural networks as follows

$$\begin{aligned} \mathbf{z}_t &= \text{ReLU}\big(\mathbf{W}_t^{(K)} \ldots \text{ReLU}(\mathbf{W}_t^{(1)}\mathbf{x}_t + \mathbf{b}_t^{(1)}) \ldots + \mathbf{b}_t^{(K)}\big), \\ \mathbf{z}_c &= \text{ReLU}\big(\mathbf{W}_c^{(K)} \ldots \text{ReLU}(\mathbf{W}_c^{(1)}\mathbf{x}_c + \mathbf{b}_c^{(1)}) \ldots + \mathbf{b}_c^{(K)}\big), \end{aligned} \quad (4)$$

where $\mathbf{x}_t$ and $\mathbf{x}_c$ are the raw features of target and context nodes, respectively, and $\{\mathbf{W}_t^{(k)}, \mathbf{b}_t^{(k)}, \mathbf{W}_c^{(k)}, \mathbf{b}_c^{(k)}, \forall k \in \{1, \ldots, K\}\}$ are the learnable parameters, which are independent of the graph size ($|\mathcal{T}|$ and $|C|$). $\mathbf{z}_t$ and $\mathbf{z}_c$ represent the embeddings of the target and the context nodes, respectively.

Contextual masking. The next step is to transform and aggregate target embeddings $\mathbf{z}_t$ based on context embeddings $\mathbf{z}_c$. Motivated by the example in Figure 3, we design and apply a *contextual masking operation* by first keeping the size of $\mathbf{z}_t$ and $\mathbf{z}_c$ to be the same, and then element-wise multiplying the embedding of the context node $\mathbf{z}_c(o)$ onto the embeddings of both ego and neighbor target nodes $\mathbf{z}_t(v)$ and $\mathbf{z}_t(u)$ as follows

$$\mathbf{z}_{t|c} = \mathbf{z}_t \otimes \mathbf{z}_c. \quad (5)$$

Note that, since the last embedding layer of $\mathbf{z}_c$ is ReLU, certain dimensions can be learned to be zero, which effectively "masks out" irrelevant dimensions, so as to project the target embeddings into particular subspaces directly controlled by the context embeddings, which exactly matches the geometric intuitions in Figure 3 (b).

Although the contextual masking operation is geometrically intuitive, to be comprehensive, we also explore and employ other schemes to compute $\mathbf{z}_{t|c}$. Motivated by the popular translation based relational models [2, 35], we compute $\mathbf{z}_{t|c} = \mathbf{z}_t \oplus \mathbf{z}_c$, which "translates" target embedding into different spaces by element-wise

summation with context embedding. Moreover, motivated by the power of dense neural networks [24], we also attach freely learnable dense neural networks to the concatenation of target and context embedding, *i.e.*, $\mathbf{z}_{t|c} = \text{ReLU}\big(\mathbf{W}_p(\mathbf{z}_t \odot \mathbf{z}_c) + \mathbf{b}_p\big)$.

Contextual attention. Contextual masking allows us to project different ego-neighbor pairs into various embedding subspaces, so as to emphasize contextualized interaction among target node pairs regarding particular embedding dimensions at the *feature level*. However, it does not consider the overall impact of different neighbors on the ego at the *node level*, especially under the consideration of different interaction context.

Motivated by the attention networks [29, 31], we design a novel contextual attention mechanism, by jointly computing an attention weight for each ego-context-neighbor triplet, $\alpha(v, o, u) =$

$$\frac{\exp\Big(\tau\big(\mathbf{a}^T[\mathbf{W}_{at}\mathbf{z}_t(v) \odot \mathbf{W}_{ac}\mathbf{z}_c(o) \odot \mathbf{W}_{at}\mathbf{z}_t(u)]\big)\Big)}{\sum_{u' \in \mathcal{N}_v, o' \sim (v,u')} \exp\Big(\tau\big(\mathbf{a}^T[\mathbf{W}_{at}\mathbf{z}_t(v) \odot \mathbf{W}_{ac}\mathbf{z}_t(o') \odot \mathbf{W}_{at}\mathbf{z}_t(u')]\big)\Big)} \tag{6}$$

where $\{\mathbf{a}, \mathbf{W}_{at}, \mathbf{W}_{ac}\}$ are the learnable attention parameters ($\mathbf{W}_{at}$ for target embedding and $\mathbf{W}_{ac}$ for context embedding), and τ is the LeakyReLU activation function following [23, 31]. $\alpha(v, o, u)$ is learned to assign different weights to neighbors based on the embedding of both context o and target pair (v, u), so as to allow graph convolution at each ego to raise attention to important neighbors and contexts. To further improve the capacity and stability, we exploit multi-head attention [29, 31] to compute the aggregated contextualized embedding as follows

$$\mathbf{z}_{\mathcal{N}_v}(x) = \sigma\Big(\frac{1}{D}\sum_{d=1}^{D}\sum_{u \in \mathcal{N}_v, o \sim (v,u)} \alpha^{(d)}(v, o, u)\ \mathbf{z}_{t|c}(x, o)\Big) \tag{7}$$

where σ is the Sigmoid function and x can be either v or u. While being expressive, our multi-head contextual attention mechanism is also feature based and cheap to compute. Particularly, all attention parameters $\{\mathbf{a}^{(d)}, \mathbf{W}_{at}^{(d)}, \mathbf{W}_{ac}^{(d)}, d \in \{1, \ldots, D\}\}$ are shared across all the links in the graph and independent of graph structure and size.

Training objective. For MultiSage, we directly add the contextualized ego embedding $\mathbf{z}_{\mathcal{N}_v}(v)$ and neighbor embedding $\mathbf{z}_{\mathcal{N}_v}(u)$ as the final MultiSage embedding $\mathbf{h}(v)$. For training loss, we follow the foot-steps of PinSage and take query-positive pairs (v_q, v_p), and apply the negative sampling strategy [37, 44] to generate query-positive-negative triplets (v_q, v_p, v_n). The MultiSage embeddings for v_q, v_p and v_n are optimized with the loss function in Eq. 3. In **Algorithm 1** (see Appendix), we provide the pseudo-code and implementation details of our proposed MultiSage *convolution* process. Figure 4 illustrates the detailed architecture of MultiSage.

2.3 Web-scale Implementation of MultiSage

Now we consider the implementation of MultiSage on web-scale multipartite networks.

Parallel contextualized random walk. The first complexity of MultiSage lies in the multi-layer graph convolutions, which requires the retrieval of multi-order neighbors on the graph during training. Since neighbors of a node can be almost anywhere on the graph due to the small world phenomenon [36], keeping adjacency

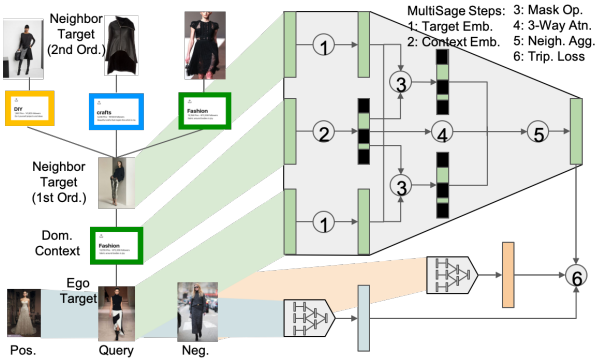


Figure 4: The overall neural architecture of MultiSage.

matrix in memory does not scale. Another approach is to compute minibatches with fixed numbers of neighbors [10], but such minibatches still become too large and complex when the convolution depth increases.

To simplify the multi-layer architecture of GCN while maintaining its long-range propagation ability, we consider [16], which shows that a single-layer GCN with neighbors sampled according to personalized PageRank scores can mimic the behavior of multi-layer GCNs while avoiding graph over-smoothing. To leverage this observation, we use a parallelizable random walk engine to sample a fixed number of higher-order neighbors for each target node and store them in a database before model training.

The random walk engine also retrieves the context nodes between an ego-neighbor pair of target nodes. However, it is not realistic to keep all the intermediate context nodes as it will blow up in size and pose challenges to the design of the contextualized graph convolution. It is also not necessary, because although each ego node can have multiple facets, usually only one facet is active when it interacts with a particular neighbor node. Therefore, we use a lightweight greedy algorithm based on stream data processing [1] to retrieve only one dominant context node per ego-target node pair. This is achieved by running multiple random-walks from the ego node and identifying the most frequently visited context node when a target node is also visited in the same random-walk. **Algorithm 2** (see Appendix) provides details of our *parallel contextualized random walk*.

Hadoop2-based data provider. The second complexity lies in the retrieval and joining of node features. For example, consider a Pinterest graph with 1 billion pins, each with 64-dim *float* visual features and 128-dim *short-int* textual features. The *feature store* of all pins will take 1TB space. During training and inference, again since the neighbor of a node in the current batch can be anywhere on the graph, neighbor retrieval is fast enough only if the 1TB graph is completely stored in memory. This approach, though expensive, has been adopted for PinSage under the support of the Linux HugePage technology [44]. However, it is hard to acquire many dedicated machines with such large memory for rapid model development and training, and it constrains the inclusion of additional data and signals as a path to further model improvements.

To remove the requirements of large memory, we develop a Hadoop2-based data provider with pre-computed neighbor datasets and AWS S3 streaming. The idea is to sample and fix the neighbor lists $\mathcal{S}$ of all target nodes $\mathcal{T}$ on $\mathcal{G}$ with Algorithm 2 offline, pre-join the large feature stores with $\mathcal{S}$ and store the results on S3 cloud. During training and inference of MultiSage, it is then possible to dynamically prepare minibatches of nodes with neighbors, both already joined with the features, so as to avoid the heavy joining operation online. We develop an efficient and robust pipeline that handles tremendous amounts of data (more than 60TB intermediate data during a full join of all neighbor lists with the feature stores), and synchronize the model training and inference on multiple GPUs with S3 streaming through dedicated stream data loaders.

3 EXPERIMENTS

3.1 Experimental Setup

Dataset preparation. From Pinterest, we collect a pin-board graph of 76M pins, 15M boards, and 2.7B pin-board links. In addition, we collect a paper-venue graph of 87M papers, 46K venues and 87M paper-venue links from OAG [26, 27] where all pairs of identical papers have different titles, to further evaluate the generalizability of MultiSage. Since in Pinterest and OAG, the most important use cases for search and recommendation are around pins and papers respectively, we model pins/papers as the target nodes, and boards/venues/keywords as the context nodes.

For Pinterest, we collect training data based on the *repin signals*. A repin pair $(q, i) \in \mathcal{R}$ is created when a user stores a pin i after exploring potentially related pins of q, which is a valuable signal indicating that the user likes the recommendation of i based on q (i and q are related in certain ways). A good model should be able to learn to capture the similarity between i and q. We remove repin pairs with low frequencies (*e.g.*, less than 2 times) to reduce noise, and keep a maximum number of repin pairs for the same query (*e.g.*, 20) to reduce bias. In our experiments, we collect a set of 75M repin pairs. We create separate validation sets from all training data by randomly sampling 1M from the 75M pairs.

For OAG, we collect 14M pairs of identical papers on Microsoft Academic Graph and Aminer. Since most pairs are easy to classify simply based on paper contents like titles, we further pick 72K from the 14M pairs with different titles for model training and evaluation. Similarly to Pinterest, a random set of 10K pairs are separated from the 72K pairs, which is only used for evaluation.

Baseline algorithms. We mainly compare with the state-of-the-art production model PinSage currently deployed at Pinterest, together with a few other commonly used embedding methods leveraging various signals with advanced mainstream deep learning models [44]. The baselines we compare include

- **Visual**: The unified visual embedding used as pin features.
- **Textual**: The conceptnet textual embedding used as pin and paper features.
- **Combined**: The combination of visual and textual embeddings used as pin features.
- **Pixie** [6]: Graph-based ranking through aggregating random-walk visiting counts deployed at Pinterest, which essentially computes the popular Personalized PageRank scores [13].
- **PinSage** [44]: The state-of-the-art production pipeline that computes pin embeddings by incorporating visual, textual and graph signals currently deployed at Pinterest (V5).
- **GAT** [31]: Our implementation of graph attention networks based on PinSage V5 for scalable training.
- **HAN** [34]: Our implementation of heterogeneous graph attention networks based on PinSage V5 for scalable training.

Note that, only HAN and our MultiSage variants can model the context nodes between target nodes, whereas Pixie, PinSage and GAT can only model the homogeneous network of target nodes by ignoring their different interaction contexts.

Evaluation metrics. We evaluate the performance of different models based on a separate set of evaluation node pairs $\mathcal{P}_{eval}$. For each pair $p = (q, i) \in \mathcal{P}_{eval}$, we use q as a query and then compute a set of metrics based on the ranking position of i among a fixed pool $\mathcal{I}$ of 1M evaluation nodes. The ranking is done based on exact cosine similarity for all embedding methods except *Pixie*, which directly returns a rank list of nodes given a query. Similar to [44], we compute the scaled Mean Reciprocal Rank as $MRR = \frac{1}{|\mathcal{P}_{eval}|} \sum_{(q,i)\in\mathcal{P}_{eval}} \frac{1}{R_{i,q}/M_s}$, where $R_{i,q}$ is the rank of i among $\mathcal{I}$. M_s is the scaling factor ensuring the difference between large ranks to be still noticeable. We use $M_s = 10$ which is smaller than 100 used in [44], because we now use a smaller pool $\mathcal{I}$ of nodes for faster evaluations and also because all baselines including PinSage have been largely improved in the past one year, which makes the ability to distinguish between small ranking differences more important. For the same reasons, for *recall@K* (short as *REC@K*, which is defined as the fraction of queries q where i is ranked among the top K among $\mathcal{I}$), we use small K's such as 10 and 1 to subtly compare the models.

In addition to the ranking based metrics which only measure the relative distances among relevant and irrelevant nodes, we also compute the average cosine distances (DC) and Euclidean distances (DE) among the embeddings of all nodes ($DC*$ and $DE*$) and all relevant nodes pairs ($DC+$ and $DE+$). This helps us to understand the absolute distribution of nodes in the embedding space. A good embedding method should be able to spread all nodes further apart to occupy vast areas in the embedding space (large $D*$'s), while putting relevant nodes close to each other (small $D+$'s).

Furthermore, we compute the sets of top-K retrieved nodes for both q and i ($TopK(q)$ and $TopK(i)$, respectively), and thus compute the average sizes of intersection ($INT = |TopK(q) \cap TopK(i)|$) and union ($UNI = |TopK(q) \cup TopK(i)|$) of the two sets, as well as their *Jaccard index* ($JAC = INT/UNI$). These metrics help us to further understand how query nodes and positive nodes are distributed in the embedding space. A good embedding method should put q and i closer in the embedding space, in the sense that their neighborhoods $TopK(q)$ and $TopK(p)$ has higher overlap, leading to large values of INT, small values of UNI, and large values of JAC.

Due to paper space limit, we put more details about dataset preparation and model training in Appendix B.

3.2 Quantitative Evaluation

Overall comparison. We first compare MultiSage against all baseline methods on the related pin recommendation task based on repin pairs, which is the most classic evaluation scenario in

Pinterest	MRR	REC@1	REC@10	DC+	DC*	DE+	DE*	INT	UNI	JAC
Visual	0.4406	0.1710	0.3606	0.4194	0.6337	0.9101	1.1255	23.32	174.67	0.1506
Textual	0.5741	0.1888	0.4965	0.3414	0.7614	0.7549	1.2050	31.78	166.21	0.1917
Combined	0.4438	0.1731	0.3635	0.4190	0.6340	0.9096	1.1258	23.44	174.53	0.1512
Pixie	0.3093	0.0418	0.2169	N/A	N/A	N/A	N/A	21.32	176.65	0.1351
PinSage	0.8759	0.4928	0.8234	0.2655	0.9279	0.7161	1.3593	47.30	150.69	0.3302
GAT	0.8880	0.5357	0.8665	0.2532	0.9343	0.7060	1.3618	48.70	149.24	0.3572
HAN	0.9013	0.5653	0.8838	0.2501	0.9415	0.6907	1.3558	50.29	148.83	0.3672
MultiSage-2	**0.9569**	**0.6215**	**0.9326**	**0.2316**	**0.9655**	**0.6660**	**1.3871**	**53.95**	**144.04**	**0.3906**
OAG	MRR	REC@1	REC@10	DC+	DC*	DE+	DE*	INT	UNI	JAC
Textual	0.1418	0.0273	0.0399	0.1081	0.4788	0.2814	1.0557	33.10	164.87	0.2193
Pixie	0.3126	0.1054	0.2642	N/A	N/A	N/A	N/A	36.58	160.76	0.2517
PinSage	0.5682	0.1845	0.5193	0.1238	0.6381	0.3179	1.1577	41.13	156.80	0.2935
GAT	0.6059	0.2355	0.5498	0.1104	0.6416	0.2908	1.2022	43.02	155.70	0.3144
HAN	0.6214	0.2641	0.5749	0.1005	0.6543	0.2869	1.2383	44.96	154.49	0.3200
MultiSage-2	0.6874	0.3270	0.6455	0.0836	0.6989	**0.2542**	1.2769	48.63	148.97	0.3602
MultiSage-3	**0.7026**	**0.3614**	**0.6875**	**0.0814**	**0.7127**	0.2583	**1.3058**	**51.63**	**145.40**	**0.3891**

Table 2: Performance of state-of-the-art large-scale embedding methods for general recommendation.

Method	Home Decoration			Women's Fashion		
	MRR	REC@1	REC@10	MRR	REC@1	REC@10
PinSage	0.8021/0.8067	0.4195/0.4257	0.7439/0.7460	0.7537/0.7545	0.3754/0.3759	0.6838/0.6976
MultiSage	**0.8407/0.8488**	**0.4899/0.5160**	**0.7954/0.8146**	**0.8058/0.8294**	**0.4363/0.4711**	**0.7533/0.7806**

Table 3: Off-task utility of embeddings produced by PinSage and MultiSage in shopping recommendation.

Pinterest due to its close connection to user engagement [44]. Similarly, we also present results on same paper identification based on the paper pairs on OAG. Table 2 shows the comprehensive set of evaluation metrics computed on both datasets. The differences in the performances between MultiSage and baselines all passed the standard paired t-test with p-value 0.01.

MultiSage-2 is our model on bipartite networks (*i.e.*, pin-board networks of Pinterest and paper-venue networks of OAG). As we can see, it consistently outperforms all baselines with significant margins in all cases, which provides a strong signals towards its effectiveness and robustness in utilizing network contexts. In Particular, the performance gain over PinSage and GAT clearly demonstrates its broader model capacity regarding context nodes, while the improvements over HAN also corroborates its appropriate model design for handling the multipartite contexts.

To show that MultiSage is general and lends itself to modeling multipartite networks with more than two types of nodes, we further incorporate keywords into the OAG network to form a tripartite network by linking each paper to its keywords. During convolution (Algorithm 1), we compute two sets of context embeddings for each paper based on both the neighboring venues and keywords, which have the same neural architecture (Eq. 4-7) but different sets of learnable parameters. We observe that MultiSage-3, which is computed on the tripartite network, can lead to additional performance gain, thus indicating the generalizability of our proposed system.

Off-task analysis. In addition to related pin recommendation, we now focus on the comparison between MultiSage and the production model PinSage to evaluate how the learned pin embeddings can influence the performance of other important but not directly related tasks. For instance, besides the repin signals that quantify user engagement, shopping signals are impactful in monetization. However, since high-quality shopping pairs are expensive and scarce, can an embedding model trained based on repin signals be useful in a shopping recommendation task?

Table 3 presents the off-task utility of pin embeddings on shopping recommendation. As we can observe, the gap between MultiSage and PinSage change from those evaluated on repin pairs, but the advantage of MultiSage is clearly maintained, indicating its general beneficial effects on different related tasks. We further randomly take out half of the shopping pairs from the evaluation set and mix into the training data of repin pairs. The second value in the last two rows of Table 3 are computed from embeddings trained with the mixture of repin and shopping pairs. As we can observe, MultiSage is able to improve significantly on the shopping metrics given additional shopping pairs for training, while the performance of PinSage almost stays the same. Note that, although related, similarity among repin pairs and shopping pairs might be slightly different. The results indicate that such subtle differences might be more effectively captured by MultiSage, which deliberately takes the contextualized proximity of nodes into account.

Ablation tests. To demonstrate the utility of our proposed techniques of contextual masking and contextual attention, we compare

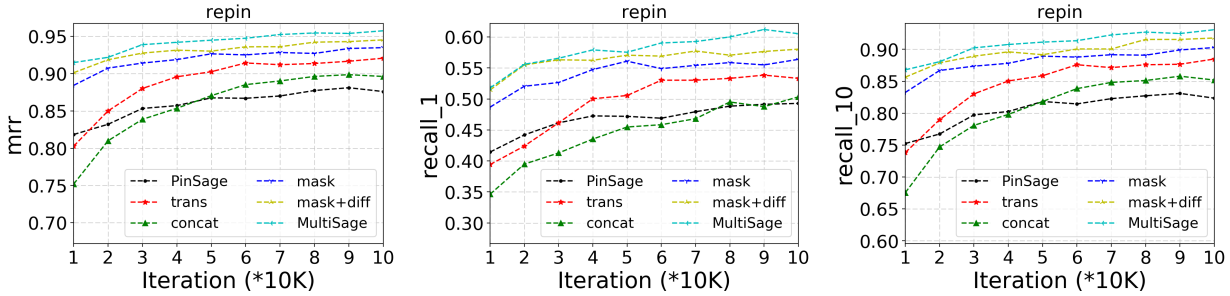


Figure 5: Performance of different model variants on *Pinterest*. Scores are computed every 10K iterations on the testing data.

multiple MultiSage variants with different model designs as introduced in the previous section, which are inspired by common practices in recent neural network models. We also conduct ablation tests with several variants of MultiSage:

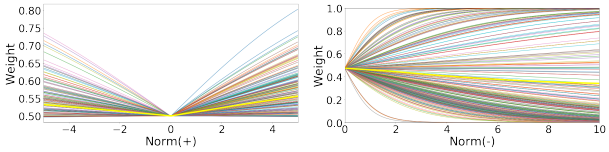
- **trans**: Adding context, target embeddings before aggregation.
- **concat**: Concatenating context embeddings to target embeddings and computing a learnable projection before aggregation.
- **mask**: Applying the learnable contextual masking operation to the target embedding according to Eq. 5 before aggregation.
- **mask-diff**: Contextual mask plus difference based aggregation (more details in Section 3.3.1).
- **mask-atn (MultiSage)**: Contextual masking plus contextual attention based aggregation according to Eq. 6, which constitutes the final version of our proposed MultiSage model.

Figure 5 shows the main metrics we measured during the training of different model variants on the Pinterest graph. As we can observe, the full MultiSage model with contextual masking and contextual attention is able to converge most rapidly to stable performance and outperform all other model variants, indicating the reasonableness of our model design in achieving efficient and effective contextualized multi-embedding.

Scalability study. We study the scalability of our MultiSage pipeline from three perspectives: parameter complexity, runtime complexity, and storage complexity.

As we claim in Section 2, the model parameter complexity of MultiSage is independent of the graph sizes. Empirically, the training time complexity is linear with the number of training batches. In practice, as similar to PinSage, we find MultiSage to converge before a full epoch through all training pairs, which results in only a slightly longer runtime than PinSage within the same pipeline and hardware settings, due to the processing of additional training features and updating of additional model parameters. However, training on the AWS GPU machines synchronized with our Hadoop2-based data provider pipeline leads to around a 25% reduction in runtime in comparison with the original Pinsage pipelines, while maintaining extremely close evaluation metrics.

Regarding storage, the original pipeline of PinSage requires the whole training graph (*e.g.*, over 2TB for the production graph in Pinterest) to be stored in memory for fast random access during neighbor aggregation, which was supported by Linux HugePage. As for our Hadoop2-based MultiSage pipeline, we completely remove



(a) Summation of ego and neighbor (b) Difference of ego and neighbor

Figure 6: Attention over different self-neighbor pairs.

this memory requirement by pre-joining and storing all data on AWS S3 and using dedicated stream data loaders to train the model on multiple GPUs. As a direct benefit, this new training pipeline reduces the resources needed for MultiSage model training, and allows us to further increase the size of training graphs by adding various other features and signals upon availability.

3.3 Qualitative Exploration

Model visualizations. Although the usage of attention is intuitive for weighing the neighbors during graph convolution [31], it is unclear exactly how attention exactly weighs different neighbors. One interesting assumption could be that since GCN is ultimately conducting graph-based feature smoothing [18], attention might help stabilize this process by assigning less weight to more different neighbors based on the current embeddings. Based on this assumption, we designed the difference-based aggregation function by computing a weight for each neighbor $u \in \mathcal{N}_v$ based on the embedding distance distance between u and the ego v, which is shown to be beneficial in Figure 5 (mask+diff). Here, through particularly designed model visualizations, we aim to further study how the attention mechanism works and why it is better than simple difference-based weights.

In Figure 6, we fix the ego embedding to all zeros, while varying each of the 256 dimensions of the neighbor embedding so as to change the norms of the (a) summation and (b) difference of ego and neighbor embeddings along the X-axis. The Y-axis denotes the average attention weights yielded by the attention network learned on the Pinterest dataset. Each colored curve corresponds to the changes on one of the 256 dimensions, while the thick yellow curve denotes the average of all thin curves. As we can observe from both subfigures, attention learns to put different stress on different

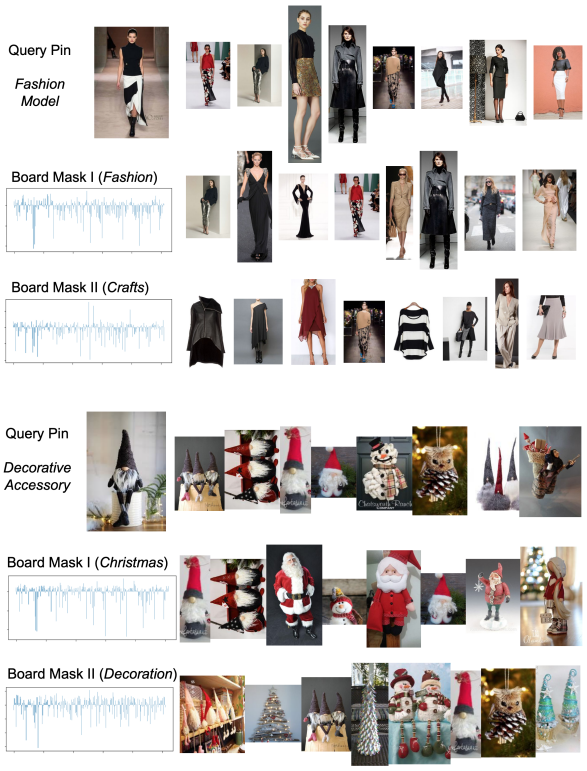


Figure 7: Examples of Pinterest pins recommended under different board-indicated semantic contexts.

dimensions. Moreover, in (a), attention generally puts more weights to neighbors that sum up with egos to have larger norms, and in (b), it puts slightly less weights to neighbors that are more different from egos. In this sense, attention generalizes and is more powerful than simple difference-based weights. In the multi-head attention setting, we observe the aggregated behaviors of multiple attention networks to be similar to those of the single-head attention, but the particular neighbor weights are assigned based on slightly different embedding dimensions, which could have further improved the attention capacity.

Case studies. In the previous experiments, although we compute multi-embeddings during graph convolution, we only use the aggregated embeddings to evaluate the overall item similarity due to the lack of evaluation data for multi-embeddings. Now we showcase how to leverage the multi-embeddings we learn for fine-grained applications such as contextualized recommendation. In the future, it will be important to further study the quality of multi-embeddings in a more principled and quantitative fashion.

Rather than generalized recommendation [44], in contextualized recommendation, we aim to rank items (*e.g.*, pins) based on contexts (*e.g.*, boards), which is flexibly personalized towards fine-grained semantics. In Figure 7, we use two random real cases from Pinterest generated based on the learned MultiSage model to demonstrate its utility in the important but seldom studied task of contextualized recommendation in the real world. Specifically, given a query pin of a fashion model (decorative accessory), rather than returning a list of generally relevant pins, we can choose arbitrary boards like fashion and crafts (Christmas and decoration), apply the corresponding learned board-based contextual masking on the base embedding of both query and candidates, and retrieve lists of pins that are relevant to the query in the corresponding perspectives. Such flexible contextualization can be easily combined with any existing search or recommendation services to enable fine-grained effective and interpretable personalization.

4 RELATED WORK

While the development of MultiSage is driven by the novel concept of contextualized multi-embeddings, we draw connections between it and various widely applied techniques in related fields.

Message passing on graphs. As an instance of GCN-based models, MultiSage can be regarded as message passing neural networks [5, 9, 32]. Existing works learn the same filter functions for all nodes [3, 10, 15, 42, 44] or edges [17, 25, 31, 47]. A few recent works extend GCN to the heterogeneous network setting [25, 34, 41, 46] by simultaneously modeling multiple types of nodes and links. However, they do not focus on any particular types of nodes and devise dedicated GCN layers to each type of links, thus bringing in lots of unnecessary complexity and instability.

Local and global graph convolutions. The power of GCN comes from its learnable convolutions among graph local neighborhoods [15], which is closely related to spectral smoothing in the Fourier domain [4, 11]. Empirical studies [10, 31, 45] and theoretical analyses [22, 38] also show the benefit of stacking multiple convolution layers to capture the global graph structures. However, more convolution layers lead to more complex and less stable models, which are harder to train and scale, and sometimes lead to worse performance due to the phenomenon of over smoothing [18]. To capture both local and global structures while avoiding the drawbacks in complexity and instability, we join the spirit of [16, 32, 44] to reduce the GCN-core of MultiSage to single-layer and use random walk with restart to account for both direct and higher-order neighbors during graph aggregation.

Relational models. Although seldom considered in traditional network mining [28], relational models have been widely studied on knowledge bases [2, 19, 20, 35, 39]. They jointly model entities and relations in correlated embedding spaces, which are connected by particularly designed translation functions. For example, TransE [2] models the entity on one side of the relation as a summation of the entity on the other side and the relation itself, while some others transform the embedding spaces before the translations through linear summation [19, 35] or projection [20, 39]. For MultiSage, we find the projection (*i.e.*, masking) operation more powerful, because it changes the distance metric in the translated spaces, which allows the modeling of different contextualized interactions between the same pairs of nodes.

Conditional similarity networks. Conditional similarity is proposed in [30] to model the possibly different similarities between the same pairs of images, conditioned on different semantic aspects of focus. MultiSage takes a close approach in spirit to model the different node interactions on graphs under different contexts.

Recent works on network multi-embedding. During the writing of this paper, we notice several recent works on network conditional similarity and multi-embeddings [7, 14, 21, 33], which reinforces the approach taken in this work. In particular, [7, 21] leverage additional graph clustering algorithms to find latent aspects of node similarity, whereas [14, 33] leverage fixed categories or textual patterns to define the different conditions. MultiSage is closer to [14, 33], since it also relies on explicit data in addition to the target networks. However, MultiSage is essentially different from [14, 33], due to its ability for flexible contextualization in continuous feature spaces, unified leverage of the powerful GCN models, and remarkable scalability.

5 CONCLUSIONS

In this work, for the first time, we propose to improve GCN with novel contextualized multi-embeddings. To leverage rich contexts on real-world multipartite networks, we design a powerful GCN engine that flexibly computes multiple embeddings for the target nodes based on the context nodes during graph convolution. Extensive experiments and case studies demonstrate the effectiveness and efficiency of our proposed GCN engine in both general and fined-grained recommendation in the real world. For future work, it would be important to further study the quality of multi-embeddings in a principled and quantitative manner, as well as how much the contextualized embeddings can help in recommending less popular and more fresh items under different contexts.

REFERENCES

[1] Charu C Aggarwal. 2007. *Data streams: models and algorithms*. Vol. 31. Springer Science & Business Media.
[2] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NIPS*.
[3] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. Fastgcn: fast learning with graph convolutional networks via importance sampling. In *ICLR*.
[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*.
[5] Alberto Garcia Duran and Mathias Niepert. 2017. Learning graph representations with embedding propagation. In *NIPS*.
[6] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *WWW*.
[7] Alessandro Epasto and Bryan Perozzi. 2019. Is a Single Embedding Enough? Learning Node Representations that Capture Multiple Social Contexts. In *WWW*.
[8] Shaohua Fan, Junxiong Zhu, Xiaotian Han, Chuan Shi, Linmei Hu, Biyu Ma, and Yongliang Li. 2019. Metapath-guided Heterogeneous Graph Neural Network for Intent Recommendation. In *KDD*.
[9] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *ICML*.
[10] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*.
[11] David K Hammond, Pierre Vandergheynst, and Rémi Gribonval. 2011. Wavelets on graphs via spectral graph theory. *ACHA* 30, 2 (2011), 129–150.
[12] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. In *NIPS*.
[13] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *WWW*.
[14] Tianwen Jiang, Tong Zhao, Bing Qin, Ting Liu, Nitesh V Chawla, and Meng Jiang. 2019. The Role of: A Novel Scientific Knowledge Graph Representation and Construction Model. In *KDD*.
[15] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[16] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Combining neural networks with personalized pagerank for classification on graphs. In *ICLR*.
[17] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *KDD*.
[18] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*.
[19] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*.
[20] Hanxiao Liu, Yuexin Wu, and Yiming Yang. 2017. Analogical inference for multi-relational embeddings. In *ICML*.
[21] Ninghao Liu, Qiaoyu Tan, Yuening Li, Hongxia Yang, Jingren Zhou, and Xia Hu. 2019. Is a Single Vector Enough? Exploring Node Polysemy for Network Embedding. In *KDD*.
[22] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph Convolutional Networks with EigenPooling. In *KDD*.
[23] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *ICML*.
[24] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. 2017. On the expressive power of deep neural networks. In *ICML*.
[25] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*.
[26] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. 2015. An overview of microsoft academic service (mas) and applications. In *WWW*.
[27] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD*.
[28] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *KDD*.
[29] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.
[30] Andreas Veit, Serge Belongie, and Theofanis Karaletsos. 2017. Conditional similarity networks. In *CVPR*.
[31] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.
[32] Saurabh Verma and Zhi-Li Zhang. 2019. Stability and Generalization of Graph Convolutional Neural Networks. In *KDD*.
[33] Hao Wang, Tong Xu, Qi Liu, Defu Lian, Enhong Chen, Dongfang Du, Han Wu, and Wen Su. 2019. MCNE: An End-to-End Framework for Learning Multiple Conditional Network Representations of Social Network. In *KDD*.
[34] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous Graph Attention Network. In *WWW*.
[35] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*.
[36] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world' networks. *nature* 393, 6684 (1998), 440.
[37] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. 2017. Sampling matters in deep embedding learning. In *ICCV*.
[38] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks?. In *ICLR*.
[39] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning in knowledge bases. In *ICLR*.
[40] Carl Yang, Yichen Feng, Pan Li, Yu Shi, and Jiawei Han. 2018. Meta-graph based hin spectral embedding: Methods, analyses, and insights. In *ICDM*.
[41] Carl Yang, Jieyu Zhang, and Jiawei Han. 2019. Neural Embedding Propagation on Heterogeneous Networks. In *ICDM*.
[42] Carl Yang, Jieyu Zhang, Haonan Wang, Sha Li, Myungwan Kim, Matt Walker, Yiou Xiao, and Jiawei Han. 2020. Relation Learning on Social Networks with Multi-Modal Graph Edge Variational Autoencoders. In *WSDM*.
[43] Carl Yang, Peiye Zhuang, Wenhan Shi, Alan Luu, and Pan Li. 2019. Conditional Structure Generation through Graph Variational Generative Adversarial Nets. In *NIPS*.
[44] 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 *KDD*.
[45] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical graph representation learning with differentiable pooling. In *NIPS*.
[46] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V. Chawla. 2019. Heterogeneous Graph Neural Network. In *KDD*.
[47] Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. 2018. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. In *UAI*.
[48] Jun Zhao, Zhou Zhou, Ziyu Guan, Wei Zhao, Wei Ning, Guang Qiu, and Xiaofei He. 2019. IntentGC: a Scalable Graph Convolution Framework Fusing Heterogeneous Information for Recommendation. In *KDD*.
[49] Dingyuan Zhu, Ziwei Zhang, Peng Cui, and Wenwu Zhu. 2019. Robust Graph Convolutional Networks Against Adversarial Attacks. In *KDD*.

APPENDIX A: Algorithm Pseudo-Code and Implementation Details

MultiSage convolution. With the novel contextual masking operation and contextual attention mechanism introduced in Section 2.3.1, we formulate our proposed MultiSage convolution procedure in Algorithm 1. The contextual masking operation in Line 3 allows the graph convolution to be focused on subspaces indicated by corresponding interaction context at the feature level, while the contextual attention mechanism in Line 4 allows the graph convolution to bias towards neighbors with certain important interaction context at the node level. Line 6 applies normalization to the final MultiSage embedding for stable training and efficient nearest neighbor search in the subsequent tasks like recommendation.

Algorithm 1 MultiSage Convolution

Input: ego feature $\mathbf{x}_v$, neighbor features $\{\mathbf{x}_u, \forall u \in \mathcal{N}_v\}$, context features $\{\mathbf{x}_c, \forall c \sim (v,u), u \in \mathcal{N}_v\}$
Output: MultiSage embedding $\mathbf{h}_v$ for ego node v

1: Comp. target embedding $\mathbf{z}_t(v), \mathbf{z}_t(u)$ *w.r.t.* Eq. 4
2: Comp. context embedding $\mathbf{z}_c(c)$ *w.r.t.* Eq. 4
3: Comp. contextualized embedding $\mathbf{z}_{t*c}(v), \mathbf{z}_{t*c}(u)$ *w.r.t.* Eq. 5
4: Comp. aggregated embedding $\mathbf{z}_{\mathcal{N}_v}(v), \mathbf{z}_{\mathcal{N}_v}(u)$ *w.r.t.* Eq. 6-8
5: $\mathbf{h}_v \leftarrow \mathbf{z}_{\mathcal{N}_v}(v) + \mathbf{z}_{\mathcal{N}_v}(u)$
6: $\mathbf{h}_v \leftarrow \mathbf{h}_v / ||\mathbf{h}_v||_2$

Parallel contextualized random walk. Algorithm 2 outlines the details of our proposed parallel contextualized random walk algorithm. For each start target node v, we use vis[v], ctx[v] and dom[v] to record the visiting count, dominant context, and the 'dominance' of the context of each end target. In Line 7, a random walk path of length ζ is generated through the standard random walk with restart on graph $\mathcal{G}$, with path[1] as the first context node and path[-1] as the last target node visited by the walk. Lines 8-15 describe the particular greedy algorithm for context retrieving, which gets motivation from stream data processing [1]. It does not always guarantee the retrieval of the most dominant context in a stream, but can find the near-optimal one in most cases. More importantly, it is light in both computation and memory, which keeps the whole procedure highly efficient and parallelizable.

Note that, PinSage also leverages short random walks to sample node neighborhoods, but it does not replace multi-layer convolution of direct neighbors with the single-layer convolution of higher-order neighbors, and it does not consider contexts in random walks.

Algorithm 2 Parallel Contextualized Random Walk

Input: graph $\mathcal{G} = \{\mathcal{T}, \mathcal{C}, \mathcal{E}\}$, walk length ζ, number of walks κ, number of threads ξ, number of neighbors to keep s
Output: sampled contextualized neighbor lists $\mathcal{S} = \{\{v : [(c,u), \forall u \in \mathcal{N}_v, c \sim (v,u)]\}, \forall v \in \mathcal{T}\}$

1: $\forall v \in \mathcal{T}$, $\mathcal{S}[v] \leftarrow \emptyset$, vis[$v$], ctx[$v$], dom[$v$]$\leftarrow$ defaultdict(int)
2: Start a pool Ω of ξ threads in parallel
3: **for** v in $|\mathcal{T}|$ **do**
4: **with** ω = rand(Ω)
5: **for** i in κ **do**
6: path $\leftarrow$ rand_walk(v, ζ)
7: vis[v][path[-1]] ++
8: **if** ctx[v][path[-1]] == path[1] **then**
9: dom[v][path[-1]] ++
10: **else**
11: **if** dom[v][path[-1]]>0 **then**
12: dom[path[-1]] - -
13: **else**
14: dom[v][path[-1]] ++
15: ctx[v][path[-1]] = path[1]
16: **end if**
17: **end if**
18: **end for**
19: $\mathcal{S}[v] \leftarrow$ top s target-context pairs *w.r.t.* vis[v] & ctx[v]
20: **end for**

APPENDIX B: Experimental Setup Details

Dataset Preparation. All node features we use in the Pinterest and OAG datasets are summarized as follows.

- Pinterest:
  - A 2048-dim *pin unified embedding* is computed based on the visual signals from the canonical image of each pin;
  - A 300-dim *pin conceptnet embedding* is computed based on the textual signals from the texts associated with each pin;
  - A 1-dim *pin degree vector* is computed based on how many neighboring boards each pin is connected to on the pin-board graph;
  - A 300-dim *board conceptnet embedding* is computed based on words and phrases in the board names and board descriptions;
  - A 300-dim *board categorical vector* is computed based on the category labels annotated for each board.
- OAG:
  - A 300-dim *paper conceptnet embedding* is computed based on the textual contents of papers;
  - A 300-dim *venue topic vector* is computed based on pre-computed latent topic distributions of venues trough LDA.
  - A 300-dim *keyword conceptnet embedding* is computed based on the words inside each keyword.

Training details. As for model parameters, we use two separate three-layer feedforward neural networks (FNN) with sizes $2349 \rightarrow 1024 \rightarrow 256$ for target embedding for Pinterest ($300 \rightarrow 256 \rightarrow 128$ for OAG), and use another two-layer FNN with sizes $600 \rightarrow 256$ for context embedding for Pinterest ($300 \rightarrow 128$ for OAG). A three-layer FNN with sizes $768 \rightarrow 128 \rightarrow 1$ is used for the computation of three-way attention weights for Pinterest ($384 \rightarrow 64 \rightarrow 1$ for OAG). All FNNs are with ReLU activations. On both datasets, we set the number of attention heads to 10, the training batch size to 256 and the number of epochs to 100K. We always use the Adam optimizer with learning rate 0.0001. The random walk parameters ζ, κ, ξ and s are empirically set to 10, 10K, 128, and 20, respectively, and the random walk is restarted with probability 0.1 at each target node. We deploy the data provider pipeline described in Section 2.3 on a Hadoop2 cluster with 378 d2.8xlarge Amazon AWS nodes. Model training is then done in parallel on a p2.16xlarge AWS machine with 8 GeForce GTX 1080 Ti GPUs.