BLADE: Biased Neighborhood Sampling based Graph Neural Network for Directed Graphs

Srinivas Virinchi, Anoop Saladi
International Machine Learning, Amazon, Bengaluru, India
{virins,saladias}@amazon.com

ABSTRACT

Directed graphs are ubiquitous and have applications across multiple domains including citation, website, social, and traffic networks. Yet, the majority of research involving graph neural networks (GNNs) focus on undirected graphs. In this paper, we deal with the problem of node recommendation in non-attributed directed graphs. Specifically, given a directed graph and query node as input, the goal is to recommend top-k nodes that have a high likelihood of a link with the query node. Here we propose BLADE, a novel GNN to model directed graphs. In order to jointly capture link likelihood and link direction, we employ an asymmetric loss function and learn dual embeddings for each node, by appropriately aggregating features from its neighborhood. In order to achieve optimal performance on both low and high-degree nodes, we employ a biased neighborhood sampling scheme that generates locally varying neighborhoods which differ based on a node's connectivity structure. Extensive experimentation on several open-source and proprietary directed graphs show that BLADE outperforms state-of-the-art baselines by 6-230% in terms of HitRate and MRR for the node recommendation task and 10.5% in terms of AUC for the link direction prediction task. We perform ablation studies to accentuate the importance of biased neighborhood sampling employed in generating higher quality recommendations for both low-degree and high-degree query nodes. Further, BLADE delivers significant improvement in revenue and sales as measured through an A/B experiment.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Node Recommendation; Graph Neural Networks; Directed Graphs; Biased Neighborhood Sampling

ACM Reference Format:
Srinivas Virinchi, Anoop Saladi. 2022. BLADE: Biased Neighborhood Sampling based Graph Neural Network for Directed Graphs. In *WSDM'23, Feb 27–March 3, 2023, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

WSDM '23, Feb 27– March 3, 2023, Singapore

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Directed graphs have become an integral part of our lives. For example, we extensively rely on Google Maps for navigation. It can be modelled as a directed graph of locations connected by roads. There are innumerable such applications where the data can be modelled as a directed graph. The degree distribution of large-scale graphs is such that they contain more low-degree nodes compared to high-degree nodes, popularly referred to as power-law [14]. Majority of directed graphs are non-attributed i.e., no features for nodes and edges. Yet, the majority of research in this space focuses on undirected graphs, and assume that node/edge features are available. The above-mentioned points clearly motivate us to focus on modelling non-attributed directed graphs.

Problem. Consequently, in this paper, we address the node recommendation problem [15, 28] in directed graphs. Formally, we are given as input i) an *unweighted* and *non-attributed*[1] directed graph $G(V, E)$ where V and E are the set of nodes and edges respectively, ii) a query node $q \in V$. The goal is to recommend R_k^q, a set of top-k nodes that have a high likelihood of a link with q.

Prior Work and Limitations. Although GNNs [6, 11, 25] are popular, they cannot be extended to directed graphs as they fail to capture edge directions. Research in modelling directed graphs can be broadly categorized into random walk based models [9, 15, 28, 29] and GNNs [17, 22, 23, 27]. Prior work does not optimize for low-degree nodes, and can potentially result in an unsatisfying performance on low-degree nodes [21]. The inherent degree distribution contributes to this problem: 1) the sparse connectivity structure of low-degree nodes imply that, suboptimal node representation of any of its neighbor(s) results in a cascading effect on its node representation during model training. 2) during neighborhood sampling, every neighbor is treated equally irrespective of its degree, potentially leading to its low-degree neighbors being sampled with high probability. [13, 21] address the issue of degree bias to learn meaningful representations for both low and high-degree nodes. However, they are not suitable for non-attributed directed graphs, and existing work in directed graphs has not catered to this sub-optimal performance on low-degree nodes. Further, prior work in directed graphs [17, 22, 23, 27], either employ random initialization or one-hot encoding for nodes in directed graphs. These approaches can give rise to suboptimal performance.

1.1 Our Contribution

In light of the above discussion, we ask the following questions: (1) How do we jointly model link likelihood and link direction in directed graphs? (2) Can we improve the node recommendation

[1]There are no features or labels for nodes and edges.

performance for low-degree nodes in directed graphs? (3) How do we initialize node embeddings in directed graphs?

Accordingly, we present **B**iased **L**ocally **A**daptive **D**irection Awar**E** (BLADE), an efficient GNN model for directed graphs, to address the above questions. We delineate the contributions of BLADE below:

Direction Aware. In order to capture node asymmetry in directed graphs, we represent each node using *dual embeddings*. We employ an *asymmetric* loss function that appropriately aggregates node features from neighborhood to generate dual embeddings, by jointly preserving link strength and link direction in directed graphs.

Biased Locally Adaptive Neighborhood Sampling. We generate on-the-fly biased neighborhoods during model training. Specifically, for every seed node, we exploit the power-law degree distribution to estimate its neighborhood size i.e., larger neighborhood for low-degree seed nodes and smaller neighborhood for high-degree seed nodes. Generating varying size neighborhoods based on each node's local structure makes our neighborhood sampling *adapt locally*.

In contrast to performing uniform sampling, we provide differential treatment to each edge during neighborhood sampling. Given that the graph is unweighted, we estimate edge such that for any node, the probability of its high-degree neighbors being sampled is much higher when compared to its low-degree neighbors. This makes the proposed sampling *biased*. When both these bits work jointly, we expect an optimal performance on both low and high-degree nodes.

Graph Attention (GAT) [25] implicitly specifies different weights to different nodes during node aggregation. However, attention weights are computed based on a node's neighborhood features, and is agnostic of the structural properties of a graph i.e., degree, local clustering coefficient etc. Consequently, GAT's reliance on node features make them unsuitable for non-attributed graphs, and deems infeasible for large size graphs owing to its very slow training time.

Evaluation. We perform extensive experimentation on several real-world and proprietary directed networks to demonstrate the superiority of BLADE compared to the state-of-the-art baselines. Ablation study confirms the crucial role of biased neighborhood sampling in boosting the performance of BLADE for both low and high-degree query nodes. Further, we show that proposed biased neighborhood sampling is comparable to conventional sampling in terms of running time, while yielding significant improvement in terms of recommendation quality. BLADE derives significant improvement in product sales and revenue as measured through an A/B experiment.

1.2 Related Work

In order to address the node recommendation problem [15, 28] in directed graphs, initial work relied on random walk models to capture node relationships in directed graphs. VERSE [24] and HOPE [15] propose learning two embeddings for each node to preserve higher order proximity, and node asymmetry in directed graphs. APP [28] captures asymmetry by preserving Rooted PageRank between nodes by relying on random walk with restart strategy. ATP [19] addresses the problem of question answering by embeddings nodes of directed graph by preserving node asymmetry. However, their approach is strictly restricted to directed acyclic graphs (DAGs), while real-world graphs are not acyclic. NERD [9] learns a pair of role-specific embeddings for each node using an alternating random walk strategy to capture edge strength and direction in directed graphs.

With GNNs being superior compared to random walk models, the contemporary research has shifted towards designing GNNs for directed graphs. DGCN [23] extends the spectral-based GCN model to directed graphs using first and second-order proximity to expand the receptive field of the convolution operation. APPNP [12] uses a GCN model to approximate personalized PageRank. DiGraphIB [22] builds upon the ideas of DGCN [23] and constructs a directed Laplacian of a PageRank matrix. It uses an inception module to share information between receptive fields. Gravity GAE [17] is a Graph Auto Encoder [10] integrated with the idea of gravity to address link prediction in directed graphs. DGGAN [29] is based on Generative Adversarial Network using a discriminator and two generators to jointly learn the source and target embedding of nodes. MagNet [27] proposes a GNN for directed graphs based on a complex Hermitian matrix. The magnitude of entries in the complex matrix encodes the graph structure, while the directional aspect is captured in the phase parameter. [26] addressed the same problem in the context of attributed directed graphs. During model training, prior work leverages fixed (same) size neighborhoods for every seed node, irrespective of its degree. Further, equal importance is assigned to every edge during neighborhood sampling. [7] proposes an adaptive layer based sampling scheme to reduce the computation and memory cost due to the uncontrollable neighborhood expansion across GCN layers, which is different from the biased adaptive neighborhood sampling employed by BLADE. [13, 21] address the issue of degree bias to learn meaningful representations for low-degree nodes. However, it is not suited for directed graphs and expects node/edge features as input. Graph Attention (GAT) [25] implicitly specifies different attention scores to each edge, which is computed as a function of a node's neighborhood features, and is agnostic of the structural properties of the graph i.e., degree, neighborhood sparsity etc. Consequently, it is unsuitable for non-attributed directed graphs. In order to bridge the gap from prior work, BLADE employs dual embeddings to mitigate asymmetry, and biased neighborhood sampling for improving the performance both on low-degree and high-degree nodes. We present the benefit of each component via ablation study.

2 BLADE

Motivation. Consider the directed product graph shown in Figure 1 where nodes correspond to products, and directed edges correspond to a co-purchase relation. For example, nodes A and B correspond to a phone case and phone respectively. Given B as input, we would like to recommend A, but given A as input, it may not be apt to recommend B because customers typically would purchase a phone case only while owning a phone. Vanilla GNNs cannot be directly leveraged on a directed graph, as it fails to capture node asymmetry. Basis the computational graph (Figure 1(a)), observe that vanilla GNNs fail to differentiate edge (A, B) and (B, A).

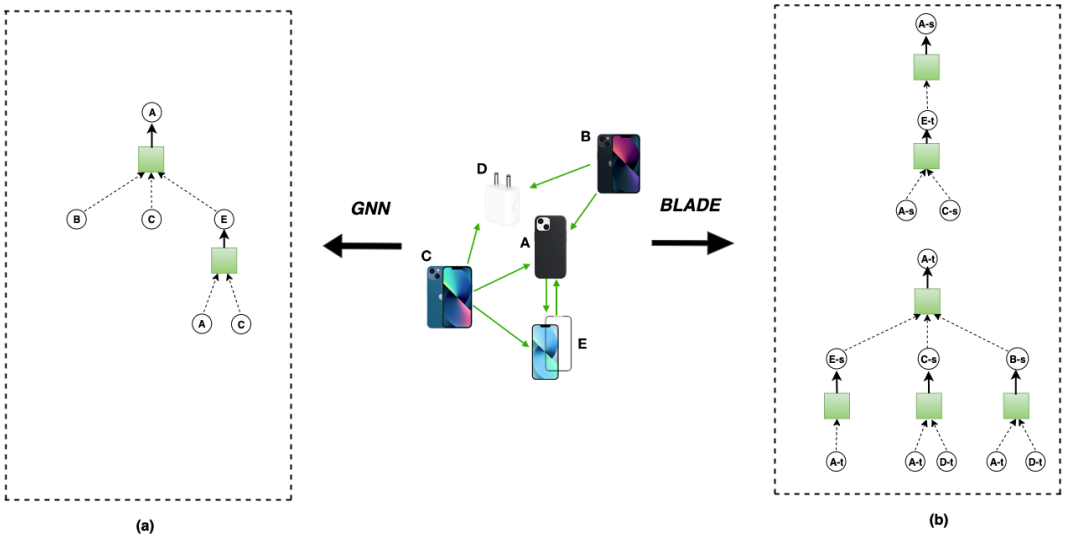

Figure 1: Computational graph of node A using a 2 layer vanilla GNN model (a) and BLADE model (b). Nodes B and C correspond to black and blue color phones of iPhone 13. A is a phone case, D is a charger and E is a screen guard. For any node u, u-s and u-t correspond to the source and target embedding of u respectively.

In order to model directed graphs, we employ two embeddings for each node i.e., source (s) and target (t) embedding in BLADE. Specifically, representing A and B each using two embeddings, permits us to differentiate edge (A, B) and (B, A). For any directed edge (u, v), the *rational of BLADE would be to embed the source embedding of u and the target embedding of v closely in the embedding space.*

We look at a node's out-neighbors when generating its source embedding, and at in-neighbors when generating its target embedding. WLOG, we consider node A to illustrate the working of BLADE. The source embedding of A is aggregated basis the target embedding of E (out-neighbor of A). Similarly, the target embedding of A would be aggregated basis the source embeddings of B, C and E (in-neighbors of A). The resulting computational graph corresponding to the source and target embedding of A generated by BLADE is shown in Figure 1(b). We can extend this aggregation step to other nodes, and across multiple hops.

After the source and target embeddings of every node is generated using BLADE, we map the query node to the source node embedding space, the potential candidates to the target embedding space, and perform a nearest neighbor lookup to recommend nodes of interest for a query node. Observe that, when we employ BLADE on an undirected graph, the computational graph of the source and target embedding for every node becomes the same.

Initial Node Embeddings. Table 1 shows the notation that we will follow hereafter. Recall that G is unweighted, directed, and the nodes V do not contain any features. In order to initialize the node features (X), we broadly have two options: 1) random initialization: the initial node representation is sampled from a probability distribution like uniform distribution [-1,1], normal distribution $N(0, 1)$, Xavier uniform and Xavier normal distribution etc. This form of initialization is random, and does not characterize any aspect of the node. 2) random walk based initialization: we generate random walks with restart (RWR) starting from each node, inspired by the power of PageRank [28] in directed graphs. We set the restart probability to 0.15. We generate the node embeddings based on the generated walks similar to Node2Vec [4], and expect it to perform better than random initialization as it captures the underlying connectivity structure of the graph.

Dual Node Embeddings Generation: Forward Pass. Algorithm 1 describes the forward pass to generate the source and target embedding of each node using BLADE assuming that the model is already trained. It expects an unweighted and non-attributed directed graph G and initial node features $X_u, \forall u \in V$ as input.

Table 1: Notation

Notation	Description
G	unweighted and non-attributed directed graph
V	set of nodes in G
$V_i \in V$	node i in G
E	set of edges in G
X_i	input embedding of node i
$(h_i^s)^l, (h_i^t)^l$	hidden source and target embedding of node i in layer l
θ_i^s, θ_i^t	source and target embedding of node i
$q \in V$	query node
R_k^q	top-k node recommendations for query node q
d_i^{in}	in-degree of node i
d_i^{out}	out-degree of node i
N_i	first hop out-neighbors of node i
N_i^{adpt}	adaptive first hop out-neighbors of node i

Initialize the hidden source and target representation of node u at layer 0 with X_u. At layer l, for each node u, a) aggregate its source embedding using the target embedding of its out-neighbors from $(l-1)^{th}$ layer (line 4), b) aggregate its target embedding using the source embedding of its in-neighbors from $(l-1)^{th}$

Algorithm 1 BLADE's node embedding generation

Input: graph $G = (V, E)$; initial node embedding $\{X_u, \forall u\}$; Number of GNN layers L; weights $W^l,\ \forall l$
Output: source embedding θ^s_u and target embedding θ^t_u, $\forall u \in V$

1: $(\mathrm{h}^s_u)^0 \leftarrow \mathrm{X}_u$; $(\mathrm{h}^t_u)^0 \leftarrow \mathrm{X}_u\ \forall u \in V$
2: **for** l=1,...L **do**
3: **for** $u \in V$ **do**
4: $(h^s_u)^l \leftarrow \sigma\left(\sum_{(u,v)\in E} (h^t_v)^{l-1} W^l\right)$
5: $(h^t_u)^l \leftarrow \sigma\left(\sum_{(v,u)\in E} (h^s_v)^{l-1} W^l\right)$
6: **end for**
7: $(h^s_u)^l \leftarrow (\mathrm{h}^s_u)^l / ||(h^s_u)^l||_2, \forall u \in V$
8: $(h^t_u)^l \leftarrow (\mathrm{h}^t_u)^l / ||(h^t_u)^l||_2, \forall u \in V$
9: **end for**
10: $\theta^s_u \leftarrow (\mathrm{h}^s_u)^L, \forall u \in V$
11: $\theta^t_u \leftarrow (\mathrm{h}^t_u)^L, \forall u \in V$

layer (line 5). W^l and σ correspond to fully connected layer at step l and ReLU activation function respectively. We normalize the embeddings to a unit norm (lines 7, 8). We repeat this process for L steps to generate the final source and target representation of all nodes $\{\theta^s_u, \theta^t_u\}\ \forall u \in V$ (line 10, 11). We leverage the generated embeddings to recommend nodes.

Node recommendation: Given a query node q, we use θ^s_q, the source embedding of q, to perform a nearest neighbor lookup in the target embedding space of all the nodes to recommend, R^q_k, a set of top-k related nodes. Specifically, for a query node $q \in V$, we compute a relevance score with respect to a candidate node $v \in V$, $rel(q, v)$, as shown in Equation 1.

$$rel(q, v) = (\theta^s_q)^\intercal (\theta^t_v) \tag{1}$$

Observe that $rel(q, v) \neq rel(v, q)$ which enables to capture the asymmetry in relationship between two nodes.

Learning parameters: Backward Pass. We discuss how to train BLADE i.e. learn the weight matrix W^l for each layer in Algorithm 2. Conventional mini-batch sampling scheme employed in [6, 11, 25]: a) samples fixed neighborhood size for each node, b) assigns equal probability to each neighbor during sampling. However, low-degree nodes still suffer from suboptimal performance [21], owing to degree-bias and sparse neighborhood. We present the details of different components involved in training BLADE:

1. Locally Adaptive (Estimating Neighborhood Size). During biased sampling, we estimate the neighborhood size of each node based on its degree. Specifically, we sample smaller neighborhood for high-degree nodes, and larger neighborhood for low-degree nodes. This relates to a power-law distribution which is shown in Equation 2.

$$p(d) = \left(\frac{\alpha - 1}{d_{min}}\right)\left(\frac{d}{d_{min}}\right)^{-\alpha},\ d \geq d_{min} \tag{2}$$

d refers to the in-degree of a node, d_{min} refers to the minimum in-degree of all nodes in the graph, and α is the power-law coefficient. Note that given a graph, the power-law coefficient α is unknown, and it can be estimated as follows:

$$\mathcal{L} = ln \prod_{u\in V} \left(\frac{\alpha - 1}{d_{min}}\right)\left(\frac{indegree(u)}{d_{min}}\right)^{-\alpha}$$

Algorithm 2 Learning BLADE's parameters

Input: graph $G = (V, E)$; node features $\{X_u, \forall u \in V\}$; Number of GNN layers L; number of epochs e; n_{neg} negative links per node
Output: weights $W^l,\ \forall l \in \{1, 2, ..L\}$

1: $\mathrm{d}_{min} \leftarrow min\{indegree(u),\ \ \forall u \in V\}$
2: $\alpha \leftarrow 1 + \frac{|V|}{\sum_{i=1}^{|V|} ln \frac{indegree(i)}{d_{min}}}$
3: **for** $u \in V$ **do**
4: $\mathrm{n}_{pos_u} \leftarrow indegree(u)^{-\alpha}$
5: $\mathrm{n}_{pos_u} \leftarrow minmaxscale(n_{pos_u})$
6: **end for**
7: **for** $(u,v) \in E$ **do**
8: $\mathrm{escore}_{uv} \leftarrow outdegree(u) * indegree(v)$
9: **end for**
10: $(\mathrm{h}^s_u)^0 \leftarrow \mathrm{X}_u$; $(\mathrm{h}^t_u)^0 \leftarrow \mathrm{X}_u\ \forall u \in V$
11: **for** epoch=1,...e **do**
12: $\mathrm{E}_{pos} \leftarrow ProbSample(E, escore, n_{pos})$
13: $\mathrm{E}_{neg} \leftarrow UniformNegativeSampling(g, n_{neg})$
14: $\mathrm{posloss} \leftarrow -\sum_{(u,v)\in E_{pos}} log(\sigma(\theta^s_u.\theta^t_v))$
15: $\mathrm{negloss} \leftarrow -\sum_{(u,v)\in E_{neg}} log(\sigma(1 - \theta^s_u.\theta^t_v))$
16: $\mathrm{asymposloss} \leftarrow -\sum_{\substack{(u,v)\in E_{pos} \wedge \\ (v,u)\notin E_{pos}}} log(\sigma(\theta^s_u.\theta^t_v))$
17: $\mathrm{asymnegloss} \leftarrow -\sum_{\substack{(u,v)\in E_{pos} \wedge \\ (v,u)\notin E_{pos}}} log(\sigma(1 - \theta^s_v.\theta^t_u))$
18: $\mathrm{loss} \leftarrow posloss + negloss + asymposloss + asymnegloss$
19: Perform BackPropagation
20: **end for**

Solving for $\frac{d\mathcal{L}}{d\alpha} = 0$, we get

$$\alpha = 1 + \frac{|V|}{\sum_{i=1}^{|V|} ln \frac{indegree(i)}{d_{min}}} \tag{3}$$

We employ Equation 3 in Algorithm 2 (lines 1-2) to estimate α. The time complexity to estimate α including computing the degree of every node is O(|V| + |E|).

For each node u, we estimate, n_{pos_u}, the neighborhood size of u to be sampled (Algorithm 2 line 3). This would make n_{pos_u} small for high-degree nodes and large for low-degree nodes, compared to a fixed value. We scale $n_{pos_u}, \forall u$ between 5 and 50 (line 4). The time complexity to estimate n_{pos_u} for all nodes is O(|V|).

2. Biased (Assign high probability to high-degree neighbors). In order to bias the neighborhood sampling towards high-degree nodes, we assign a score to every edge inspired by the preferential attachment model [2, 18] as shown below:

$$escore_{uv} = outdegree(u) * indegree(v) \tag{4}$$

Based on Equation 4, for every end-vertex u, the edges connected to its high-degree neighbors would be assigned a higher score (line 8 in Algorithm 2). This would assign high degree neighbors a higher chance to be chosen during sampling. Notably, this would mitigate degree bias, and achieve improved performance both on low and high-degree nodes. For every node u, we sample n_{pos_u} neighbors from its neighbors using the computed edge scores. The time complexity to compute the edge scores is O(|E|).

3. Training. For each node, we sample positive edges E_{pos} basis the estimated edge scores *escore* and neighborhood sizes n_{pos} (line 12). We sample n_{pos} random (uniform) distribution negative (non-existing) links for every node (line 13). The time taken to perform sampling is $O\left(|V| * (n_{pos_u} + n_{neg})\right)$. n_{neg} and $n_{pos_u} (\leq 50 \ \ \forall u)$ are constants making the time complexity of sampling $O(|V|)$.
4. Asymmetric Loss. We employ an asymmetric loss function to learn the model parameters (line 18 in Algorithm 2). It can be decomposed into four terms. For every positive link (u, v) (from E_{pos}), we force the dot product of source embedding of u and target embedding of v to be high (line 14). For every negative link (u, v) (from E_{neg}), we force the dot product of source embedding of u and target embedding of v to be low (line 15). The third and the fourth terms in the loss explicitly captures the edge direction i.e., asymmetry in relationship between nodes. Specifically, for each one-way directed link (u, v) i.e. $(u, v) \in E_{pos} \wedge (v, u) \notin E_{pos}$, we want the model to assign a high score to (u, v) and a small score to (v, u) (lines 16–17). We perform back propagation to update the model parameters.
Complexity. In terms of training BLADE, the time complexity of estimating α, n_{pos} and *escore* (Algorithm 2 lines 1–9) is O(|V|+|E|). The time complexity of sampling is O(|V|) as the number of epochs is constant. The backpropagation and inference procedure (Algorithm 1) is same for any model, as long as the model architecture is asymptotically similar. We use FAISS [8] to perform efficient nearest neighbor lookup. Our model uses O($|V|$) space to store embeddings corresponding to $|V|$ nodes. This makes BLADE efficient and easily deployable in an e-commerce production environment, where the graph consists of millions of edges.

3 EXPERIMENTS

In this section, we evaluate the performance of BLADE against state-of-the-art GNN models for directed graphs. Specifically, we aim to answer the following evaluation questions:
EQ1: Is BLADE able to deliver superior performance for the node recommendation task compared to existing baselines?
EQ2: How effective is BLADE in capturing node asymmetry i.e. predicting the correct edge direction?
EQ3: How does initializing node embeddings impact the performance of BLADE for recommending nodes?
EQ4: How does using varying neighborhood size used in BLADE compare to a fixed neighborhood size employed by traditional GNN models for node recommendation?
EQ5: How does biased (non-uniform) edge sampling affect the performance of BLADE compared to an unbiased uniform edge sampling and graph attention based edge weighing?
EQ6: What is the runtime overhead incurred by the biased neighborhood sampling employed in BLADE compared to minibatch sampling employed by Vanilla GCN during model training?
Public Datasets. We employ six open-source directed graph datasets for benchmarking the performance of different models for node recommendation task. The overview of the datasets is shown in Table 2. All the open-source graphs are taken from SNAP and KONECT databases. WikiVote contains Wikipedia voting data from its inception till January 2008. Nodes in the network represent Wikipedia users and a directed edge from node i to node j represents that user i voted on user j. Cora and DBLP are citation networks where we represent the papers as nodes, and a directed edge from node i to node j represents that paper i cites paper j. Twitter is a social network of users where a directed edge from node i to node j represents that user i follows user j. Amazon1 and Amazon2 correspond to directed co-purchase networks. Each node in the graph is a product and a directed edge from node i to node j indicates that customers bought product i before product j.
Proprietary Datasets. We also employ two proprietary directed graph datasets for benchmarking BLADE against existing baselines. E-comm1 and E-comm2 are proprietary datasets sampled from different stores of an e-commerce major. Each node in the proprietary graph corresponds to a product, and a directed edge corresponds to a co-purchase relation between the two products. Owing to confidentialy, we do not disclose how the edge directions have been crafted in these datasets.

Observe that all the graph datasets are unweighted, non-attributed and directed. The graph datasets are selected to capture diverse structural aspects of a graph i.e., domain, size, directed edges % and average degree. The graph datasets range from 7K-5.5M nodes and 50K-31M edges.
Implementation Details. We implemented BLADE using DGL and PyTorch. We observe that a learning rate of 10^{-4} using Adam's optimizer works the best. We use (L =) 3 layer GNN model for BLADE. After generating the node embeddings, we perform nearest neighbor lookup to suggest top-k ($k \in \{5, 10, 20\}$) node recommendations. All the experiments are conducted on a 64-core machine with a 488 GB RAM running Linux. For all models, we learn 128 dimensional node embeddings trained for a maximum of 30 epochs. We repeat all the experiments 10 times and report the average value across the runs.

Table 2: Directed Graph Datasets used in experiments

Dataset	|V|	|E|	Average Degree	% Directed Edges
WikiVote	7,115	103,689	14.57	94.34
DBLP	12,590	49,759	3.95	99.53
Cora	23166	91,500	3.95	94.87
Twitter	81,306	1,768,149	21.75	51.83
Amazon1	259,005	1,206,557	4.66	45.72
Amazon2	403,394	3,387,388	8.4	44.26
E-comm1	1,988,703	14,152,887	7.3	76.33
E-comm2	5,541,372	31,759,139	5.76	79.97

Baselines. We previously discussed state-of-the-art models in directed graphs (Section 1.2). In order to evaluate BLADE, we choose the most competitive baselines as follows:
(1) We choose APP [28] and NERD [9] as they deliver superior performance compared to deepwalk[16], node2vec [4], LINE [20], HOPE[15] and VERSE [24]. These are the best performing random walk based models suitable for directed graphs.
(2) DGGAN [29] is a GAN [3] based model which outperforms deepwalk [16], node2vec [4], LINE [20], APP[28], HOPE [15] and graph auto-encoder based models like VGAE [10], Gravity GAE [17] etc.
(3) MagNet [27] is a GNN model that uses a complex Hermitian matrix to encode the graph structure, and it outperforms GraphSage [6], GAT [25], DGCN [23] and APPNP [12].

In summary, we compare BLADE against APP [28], NERD [9], DGGAN [29] and MagNet [27]. We use publicly released code repositories for all the baselines and employ parameter tuning to choose the best parameters for each baseline.

Experimental Setup. In order to answer the evaluation questions, we setup the experiments (similar to [28], [27], [6]) as follows:

- Node Recommendation Task: For each graph dataset, we use 75%, 5% and 20% non-overlapping edges for training, validation and testing respectively. For a ground-truth recommendation (u, v) (from the test data), where u is the query node, we retrieve top-k node recommendations (R_k^q) suggested by each model. In order to evaluate the quality of recommendations, we use HitRate@k and MRR@k for k in $\{5, 10, 20\}$. This provides an insight into how a model captures relationship between nodes. We will employ this setup to answer EQ1, EQ3, EQ4, EQ5 and EQ6.
- Directed link prediction Task: For each graph dataset, we use 75%, 5% and 20% non-overlapping edges for training, validation and testing respectively. In this task, we consider only one-way directed edges ($(u, v) \in G \wedge (v, u) \notin G$) as positive links. We reverse the direction of the positive links to create negative links while testing. We want to evaluate how different models capture the edge direction; a good model assigns a high score to the correct edge (positive link), and a low score to the reverse edge (negative link). We evaluate the model performance using AUC for this task. We employ this setup to answer EQ2.

4 RESULTS

In this section, we present results corresponding to the performance of different models in jointly capturing node relationship (Section 4) and edge direction (Section 4). We also perform ablation study (Section 4.1) to analyze the impact of different optimizations employed in BLADE like adaptive neighborhood size, biased neighborhood sampling and random walk with restart based initial node representations. For proprietary datasets, we do not show the absolute values, and results relative to an internal baseline is presented owing to confidentiality.

EQ1. Node Recommendation Task The results for the node recommendation task is shown in Table 3. We observe that BLADE consistently outperforms existing baselines in terms of both HitRate and MRR. DGGAN delivers the second best performance, but it fails to complete model training in 48 hours on the proprietary datasets indicated by OOT (Out Of Time)[2]. It does not scale to real-world datasets; DGGAN [29] demonstrate their model efficacy on small datasets (the biggest graph employed in their experiment has 15K nodes). MagNet is not applicable for node recommendation task, and we evaluate it against BLADE only for the link prediction task. Overall, BLADE captures relationship between nodes the best compared to the existing baselines.

EQ2. Link Direction Prediction Task We present the performance of different models in predicting edge direction in Table 4. We observe that BLADE outperforms existing baselines in terms of AUC. We observe that MagNet and DGGAN perform the second best. As mentioned earlier, DGGAN is marked OOT for the largest datasets. From Table 3 and Table 4, we infer that BLADE is able to jointly capture relationship between nodes and edge direction in directed graphs the best compared to state-of-the-art baselines.

[2]OOT: Out of time. The model training could not be completed in 48 hours.

4.1 Ablation Study

Recall that BLADE employs different optimizations like employing adaptive neighborhood size, biased neighborhood sampling and random walk with restart based initial node representations. In order to analyse the impact of these optimizations on the performance of BLADE, and deduce the importance of each optimization, we perform ablation study on the Twitter dataset which we present next.

EQ3. Initial Node Embeddings (X) In this work, we consider unweighted and non-attributed directed graphs where the nodes and edges have no explicit features or weights. However, GNNs require initial node representations for both training and inference. To this end, we explore different ways to set the initial nodes embeddings:
(1) BLADE-U: The embedding is sampled from a uniform distribution in $[-1, 1]$.
(2) BLADE-N: The embedding is sampled from a standard normal distribution (mean 0 and standard deviation 1).
(3) BLADE-XU: The embedding is sampled from a Xavier uniform distribution.
(4) BLADE-XN: The embedding is sampled from a Xavier normal distribution.
(5) BLADE-RWR: We generate random walks with restart based walks which we leverage to generate node embeddings similar to DeepWalk [16], Node2Vec [4] etc.

We present the impact of initial node embeddings on the performance of BLADE in Table 5 in terms of HitRate and MRR. We find that among different distributions, initializing the embeddings using a standard normal distribution (BLADE-N) performs the best. This is consistent with the results shown in [1]. However, we observe that initializing the node embeddings using random walk with restart based model (BLADE-RWR) delivers the best performance; RWR based embeddings captures the node connectivity structure. Note that the results corresponding to BLADE in Table 3 and Table 4 employ RWR based initial node representations.

EQ4. Impact of Adaptive Neighborhood Size Recall that a GNN employs fixed size neighborhood batch sampling during training. Irrespective of the node characteristic, for every node it samples a fixed number of neighbors in every batch. Both GNNs and random walk based models do not capture the node representations for low-degree nodes the best. In order to overcome this issue, BLADE employs an adaptive neighborhood size based on the node degree as explained earlier. The impact of using a variable neighborhood size on BLADE is shown in Table 6. Observe that having an adaptive neighborhood size with BLADE performs better than having a fixed neighborhood size. Further, to perform a deeper

Table 3: Node Recommendation. Best results are in bold

Dataset	k	HitRate@k				MRR@k			
		NERD	APP	DGGAN	BLADE	NERD	APP	DGGAN	BLADE
WikiVote	5	0.0068	0.0031	0.0158	**0.0164**	0.0028	0.0013	0.0075	**0.0079**
	10	0.0121	0.0064	0.0282	**0.0302**	0.0035	0.0017	0.0091	**0.0097**
	20	0.0236	0.0151	0.0497	**0.0589**	0.0043	0.0023	0.0106	**0.0116**
DBLP	5	0.0086	0.0098	0.0201	**0.0462**	0.0042	0.0033	0.0092	**0.0194**
	10	0.0141	0.0283	0.0337	**0.0978**	0.005	0.0057	0.011	**0.0261**
	20	0.0249	0.0515	0.0526	**0.1751**	0.0057	0.0073	0.0123	**0.0313**
Cora	5	0.0183	0.0638	0.0146	**0.1136**	0.0083	0.0192	0.0076	**0.0359**
	10	0.0373	0.1239	0.023	**0.2683**	0.0108	0.031	0.0087	**0.0563**
	20	0.0671	0.2283	0.0401	**0.4287**	0.0128	0.0382	0.0098	**0.0675**
Twitter	5	0.0213	0.0452	0.008	**0.0564**	0.0064	0.0198	0.004	**0.0223**
	10	0.0546	0.0901	0.0139	**0.1173**	0.0107	0.0257	0.0047	**0.0303**
	20	0.1211	0.1629	0.0219	**0.2152**	0.0152	0.0306	0.0053	**0.0369**
Amazon1	5	0.253	0.3163	0.3195	**0.333**	0.1133	0.1338	0.1298	**0.2105**
	10	0.3808	0.5635	0.5784	**0.6142**	0.1306	0.1674	0.1781	**0.2492**
	20	0.4882	0.6917	0.71	**0.7544**	0.1381	0.1767	0.185	**0.2593**
Amazon2	5	0.1868	0.1838	0.1758	**0.2013**	0.0849	0.0563	0.056	**0.1043**
	10	0.3515	0.3752	0.3742	**0.4498**	0.1065	0.0717	0.0717	**0.1187**
	20	0.4927	0.5701	0.5848	**0.6498**	0.1165	0.0808	0.0818	**0.1327**
E-comm1	5	0.62x	2.14x	OOT	**3.57x**	0.61x	1.02x	OOT	**1.65x**
	10	1.94x	4.01x	OOT	**6.25x**	0.71x	1.26x	OOT	**2x**
	20	3.14x	6.95x	OOT	**9.98x**	0.79x	1.46x	OOT	**2.25x**
E-comm2	5	1.28x	1.23x	OOT	**3.87x**	0.71x	0.67x	OOT	**1.87x**
	10	1.85x	1.99x	OOT	**6.25x**	0.78x	0.77x	OOT	**2.18x**
	20	2.63x	3.31x	OOT	**9.12x**	0.84x	0.85x	OOT	**2.38x**

Table 4: Link Direction prediction (AUC). Best results are in bold

Model	Dataset							
	WikiVote	DBLP	Cora	Twitter	Amazon1	Amazon2	E-comm1	E-comm2
NERD	63.67	52.9	53.2	63.2	50.9	53.4	7.1x	2.48x
APP	51.1	52.3	51.87	50.13	51.36	52.29	0.1x	0.15x
DGGAN	67.71	55.8	57.42	71.46	53.05	53.47	OOT	OOT
MagNet	67.5	52.2	59.93	61.52	51.6	56.93	12.65x	6.62x
BLADE	**68.88**	**61.7**	**65.75**	**78.54**	**55.83**	**57.6**	**14.72x**	**12.53x**

Table 5: Impact of Initial Node Embeddings on BLADE's performance for Twitter Dataset

Model	HitRate@20	MRR@20
BLADE-U	0.185	0.0318
BLADE-N	0.2137	0.0354
BLADE-XU	0.212	0.0352
BLADE-XN	0.2108	0.0349
BLADE-RWR	**0.2152**	**0.0369**

analysis, we split the test edges into low[3] and high-degree bins based on the out-degree of the source node. Results indicate that using BLADE with an adaptive neighborhood size improves both the performance on both low-degree and high-degree test edges. Note that the results corresponding to BLADE in Table 3 and Table 4 employ sampling using adaptive neighborhood size.

[3]For Twitter dataset, a node having less than an in-degree of 10 is considered to be a low-degree node.

EQ5. Impact of Biased Edge Sampling Graph neural networks assign equal importance to every edge in unweighted graphs having no node features. An exception to this is GAT [25], which provides different attention scores to every neighbor. However, attention based models consume significant time and memory for training and are very slow on real-world e-commerce datasets. In order to provide a differential treatment, we present a bias towards high-degree neighbors inspired by the preferential attachment mechanism [2, 18]. We use the computed edge scores during neighborhood sampling i.e. for every node i, we sample n_{pos_i} neighbors based on the edge score. We show the impact of using a biased edge sampling scheme on the performance of BLADE in Table 7. We observe that using the computed edge scores during neighborhood sampling shows an improvement in performance compared to an unbiased sampling procedure. Further, observe that BLADE using a GAT layer trained for 30 epochs perform very poorly (5x drop in HitRate and MRR). BLADE using GAT based layer takes 12x more training time compared to BLADE using a GCN layer. GAT models converge

Table 6: Impact of Adaptive Neighborhood Size on BLADE's performance for Twitter Dataset

Model	HitRate@20			MRR@20		
	All test edges	Test Edges with low-degree source nodes	Test Edges with high-degree source nodes	All test edges	Test Edges with low-degree source nodes	Test Edges with high-degree source nodes
BLADE (5 neighbors)	0.2118	0.3935	0.1684	0.0366	0.0698	0.0286
BLADE (10 neighbors)	0.2122	0.3933	0.1686	0.0367	0.0696	0.0288
BLADE (25 neighbors)	0.212	0.3926	0.1686	0.0367	0.0694	0.0288
BLADE (50 neighbors)	0.2118	0.3926	0.1683	0.0366	0.0694	0.0287
BLADE (All neighbors)	0.2119	0.3922	0.1686	0.0366	0.0696	0.0287
BLADE (Adaptive)	**0.2152**	**0.3999**	**0.1706**	**0.0369**	**0.0702**	**0.0289**

very slowly and need to be trained for 500-1000 epochs [5, 25] to derive optimal performance, making it unsuitable for e-commerce graphs. Note that the results corresponding to BLADE in Table 3 and Table 4 employ the biased edge sampling scheme during training and inference of BLADE.

Table 7: Impact of Biased Edge Sampling on BLADE's performance for Twitter Dataset

Model	HitRate@20	MRR@20
BLADE (3-layer 4-head GAT)	0.0469	0.0088
BLADE (Uniform Edge Sampling)	0.2148	0.0368
BLADE (Biased Edge Sampling)	**0.2152**	**0.0369**

EQ6. Running Time Comparison From results, we observed the crucial role of biased sampling in BLADE while modelling directed graphs. We have also previously discussed the time complexity of biased sampling, and shown that it is asymptotically similar to random sampling employed in Vanilla GCN. Here, we show the run time comparison of BLADE against vanilla GCN in Table 8. Observe that the there is only a slight difference between the per epoch time of BLADE and Vanilla GCN. Specifically, this slight increase is due to the biased sampling employed by BLADE, which can be accommodated for deriving significant gains compared to baselines basis experiments. This clearly indicates the efficiency of BLADE and paves way for deploying it in an e-commerce production environment, where the graph consists of millions of edges.

Table 8: Training time per epoch (s)

Dataset	Vanilla GCN	BLADE
WikiVote	0.64	1.3
DBLP	0.78	0.95
Cora	0.94	2.2
Twitter	11.38	16.79
Amazon1	22.6	32.28
Amazon2	43.2	55.03
E-comm1	164.78	184.82
E-comm2	398.66	471.16

4.2 Production A/B Test

We further evaluate the performance of BLADE in recommending co-purchasable productions by conducting an A/B test in two different marketplaces. For the control group, we use an incumbent approach based on product co-purchases, while for the treatment group, we show the recommendations generated from BLADE. We run the experiments for 4 weeks and observe +160% improvement on product sales and +175% improvement on profit gain. All the results are statistically significant with p-value < 0.05. These results show the product recommendations generated from BLADE can significantly improve customer shopping experience in discovering potentially related products of interest.

5 CONCLUSIONS

In this paper, we presented BLADE, a novel Graph Neural Network, for node recommendation in directed graphs. In order to train the model, we employ an asymmetric loss function to jointly capture link likelihood and link direction. During training, we also employ a locally adaptive neighborhood sampling approach which is biased towards high-degree neighbors. We leverage the trained GNN model to generate dual embeddings for each node, by appropriately aggregating features from its neighborhood. Extensive offline experiments show that BLADE outperforms state-of-the-art baselines for the node recommendation and link prediction task on both publicly available and proprietary directed graph datasets. We perform ablation study to highlight the benefit of different components of BLADE. We evaluate the efficiency of BLADE both through run time complexity analysis and benchmarking. The efficiency of BLADE makes it deployable on large e-commerce graphs. Further, BLADE delivers significant improvement in revenue and sales as measured through an A/B experiment.

REFERENCES

[1] Ralph Abboud, Ismail Ilkan Ceylan, Martin Grohe, and Thomas Lukasiewicz. 2020. The surprising power of graph neural networks with random node initialization. *arXiv preprint arXiv:2010.01179* (2020).

[2] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of modern physics* 74, 1 (2002), 47.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. In *NIPS*.

[4] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *22nd ACM SIGKDD*. 855–864.

[5] Hantao Guo, Rui Yan, Yansong Feng, Xuesong Gao, and Zhanxing Zhu. 2020. Simplifying Graph Attention Networks with Source-Target Separation. In *ECAI 2020*. IOS Press, 1166–1173.

[6] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *NIPS* 30 (2017).

[7] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive sampling towards fast graph representation learning. *Advances in neural information processing systems* 31 (2018).

[8] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* 7, 3 (2021), 535–547.

[9] Megha Khosla, Jurek Leonhardt, Wolfgang Nejdl, and Avishek Anand. 2019. Node representation learning for directed graphs. In *ECML PKDD*. 395–411.

[10] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).

[11] Thomas N Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. *ICLR* (2017).

[12] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. *ICLR* (2019).

[13] Zemin Liu, Trung-Kien Nguyen, and Yuan Fang. 2021. Tail-GNN: Tail-Node Graph Neural Networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 1109–1119.

[14] Buddhika Nettasinghe and Vikram Krishnamurthy. 2021. Maximum Likelihood Estimation of Power-law Degree Distributions via Friendship Paradox-based Sampling. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 6 (2021), 1–28.

[15] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *SIGKDD*. 1105–1114.

[16] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *SIGKDD*. 701–710.

[17] Guillaume Salha, Stratis Limnios, Romain Hennequin, Viet-Anh Tran, and Michalis Vazirgiannis. 2019. Gravity-inspired graph autoencoders for directed link prediction. In *CIKM*. 589–598.

[18] Herbert A Simon. 1955. On a class of skew distribution functions. *Biometrika* 42, 3/4 (1955), 425–440.

[19] Jiankai Sun, Bortik Bandyopadhyay, Armin Bashizade, Jiongqian Liang, P Sadayappan, and Srinivasan Parthasarathy. 2019. Atp: Directed graph embedding with asymmetric transitivity preservation. In *AAAI*, Vol. 33. 265–272.

[20] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.

[21] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Yiqi Wang, Jiliang Tang, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. 2020. Investigating and mitigating degree-related biases in graph convoltuional networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1435–1444.

[22] Zekun Tong, Yuxuan Liang, Changsheng Sun, Xinke Li, David Rosenblum, and Andrew Lim. 2020. Digraph inception convolutional networks. *NIPS* 33 (2020), 17907–17918.

[23] Zekun Tong, Yuxuan Liang, Changsheng Sun, David S Rosenblum, and Andrew Lim. 2020. Directed graph convolutional network. *arXiv preprint arXiv:2004.13970* (2020).

[24] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. 2018. Verse: Versatile graph embeddings from similarity measures. In *Proceedings of the 2018 World Wide Web Conference*. 539–548.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. (2018).

[26] Saladi Anoop Virinchi, Srinivas and Abhirup Mondal. [n. d.]. Recommending Related Products Using Graph Neural Networks in Directed Graphs. In *ECML PKDD*.

[27] Xitong Zhang, Yixuan He, Nathan Brugnone, Michael Perlmutter, and Matthew Hirn. 2021. Magnet: A neural network for directed graphs. *NIPS* 34 (2021).

[28] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable graph embedding for asymmetric proximity. In *AAAI*, Vol. 31.

[29] Shijie Zhu, Jianxin Li, Hao Peng, Senzhang Wang, Philip S Yu, and Lifang He. 2021. Adversarial directed graph embedding. *AAAI* (2021).