Learning on Streaming Graphs with Experience Replay

Massimo Perini
The University of Edinburgh
massimo.perini@ed.ac.uk

Giorgia Ramponi
ETH Zurich
giorgia.ramponi@inf.ethz.ch

Paris Carbone
KTH Royal Institute of Technology
parisc@kth.se

Vasiliki Kalavri
Boston University
vkalavri@bu.edu

ABSTRACT

Graph Neural Networks (GNNs) have recently achieved good performance in many predictive tasks involving graph-structured data. However, the majority of existing models consider static graphs only and do not support training on graph streams. While inductive representation learning can generate predictions for unseen vertices, these are only accurate if the learned graph structure and properties remain stable over time. In this paper, we study the problem of employing experience replay to enable *continuous graph representation learning* in the streaming setting. We propose two online training methods, *Random-Based Rehearsal-RBR*, and *Priority-Based Rehearsal-PBR*, which avoid retraining from scratch when changes occur. Our algorithms are the first streaming GNN models capable of scaling to million-edge graphs with low training latency and without compromising accuracy. We evaluate the accuracy and training performance of these experience replay methods on the node classification problem using real-world streaming graphs of various sizes and domains. Our results demonstrate that *PBR* and *RBR* achieve orders of magnitude faster training as compared to offline methods while providing high accuracy and resiliency to concept drift.

ACM Reference Format:
Massimo Perini, Giorgia Ramponi, Paris Carbone, and Vasiliki Kalavri. 2022. Learning on Streaming Graphs with Experience Replay. In *Proceedings of ACM SAC Conference (SAC'22).* ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Graph streams can represent continuous interactions and evolving relationships in a variety of datasets. The structure of social networks gradually changes as new friendship relations are formed and others disappear, online discussion networks grow as users interact with each other, and financial transaction networks expand with every new purchase. The streaming nature of real-world graphs poses a challenge to emerging Machine Learning (ML) applications to train adaptive models, capable of resilient continuous learning.

SAC'22, April 25 –April 29, 2022, Brno, Czech Republic

ACM ISBN 978-1-4503-8713-2/22/04…$15.00
https://doi.org/xx.xxx/xxx_x

Graph representation learning is a foundational methodology for understanding and analyzing complex graphs and their latent properties [17–19]. It involves learning the encoding for vertices, edges, and their features into low-dimensional spaces, such as a set of vectors, also known as *node embeddings*. Node embeddings enable the application of ML techniques, such as node classification, on graphs and have been used in recommender systems, social network analysis, chemical synthesis, and financial networks [15, 23, 46]. Embeddings can preserve the graph's structural and semantic properties (features) even at such low-dimensional spaces. However, most existing representation learning approaches either require the full graph to be known a priori during training [23, 53] or ignore associated attributes and assume that structural properties remain stable as new nodes and edges are added to the network [28].

Currently, there exist three approaches to incorporate graph changes in graph representation learning methods: (i) train on an initial graph snapshot and then use the trained model on future streaming updates, (ii) periodically retrain the model from scratch on snapshots of increasing size, or (iii) periodically retrain the model on snapshots consisting of *fresh* data only. The first method leads to deteriorating model accuracy since the data distribution can change arbitrarily over time [18, 42]. Whereas, the second method cannot be sustained over unbounded graph evolutions due to its increasing computational and spatial complexities. Finally, the latter approach can support concept drift, yet, it is prone to *catastrophic forgetting* [11], i.e., training a representation only on new data leads to (possibly complete) loss of previously learned training data. Given the size of most web-scale graphs and the rapid speed upon which change is introduced, there is an evident need of new methodologies for the timely yet accurate learning and updating of streaming graph representations.

Table 1 summarizes the capabilities of recent methods in graph representation learning in terms of (i) supporting incremental training on graph streams vs. requiring retraining from scratch and (ii) their ability to provide efficient re-training as the graph grows. ContinualGNN [44] is a recent replay-based method for streaming GNNs. However, as we show in Section 5.3, its training time requirements grow with the graph size and render it unsuitable for practical streaming scenarios.

Our contribution. We propose a streaming GNN model based on experience replay [27] that selects *rehearsal samples* and incrementally updates the network parameters while observing a stream of edges, as shown in Figure 1. Our model relies on two methods: *Random-Based Rehearsal-RBR* yields a uniform sample of the entire training graph while *Priority-Based Rehearsal-PBR* prioritizes data

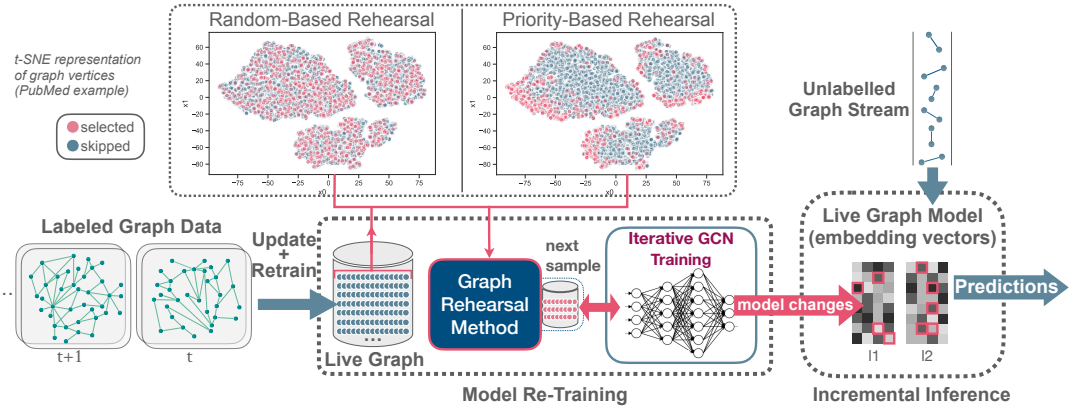


Figure 1: Overview of Experience Replay (*PBR*, *RBR*) and Incremental Inference on the PubMed dataset

Category	Method	Incr. training	Scalability
Matrix factorization	Laplacian Eigenmaps [5], GraREP [3], DANE [3]	✗	✗
Random walk	DeepWalk [35], Node2Vec [17], LINE [40],CTDNE [34]	✗	✗
	DNE [10]	✓	✗
Autoencoder	DNGR [7], SDNE [43]	✗	✗
	DynGEM [16]	✓	✗
Inductive GNN	GAT [42], DGNN [29], DySAT [37], GraphSAGE [18]	✗	✗
	ContinualGNN [44]	✓	✗
	PBR & RBR	✓	✓

Table 1: Graph representation learning approaches

points based on the model prediction error, aiming to preserve the decision boundary learnt by the classifier. Our model limits retraining to a *fixed, curated subset of the graph stream*, while avoiding the case of catastrophic forgetting. As a result, the training performance of *RBR* and *PBR* is independent of the graph size, allowing them to scale to million-edge graphs. Our evaluation shows that *RBR* and *PBR* achieve prediction accuracy comparable to that of offline retraining, offer quick convergence, and address concept drift effectively. At the same time, they save hours of training time.

Our code is publicly available in a github repository [2].

2 PROBLEM SETTING

In this section we provide the necessary background and we formally define the problem we address in this paper.

Streaming Graph Model. Many real-world problems can be modelled as graph streams. For example, in a social network a vertex is added for a new user and an edge is created for new friendship relationships. Our streaming graph model builds on a series of *graph changes* defined as follows.

Definition 2.1. Given a time/step domain $t \in \mathbb{N}$, a *graph change* at time t consists of a tuple $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$, where $\mathcal{V}_t$ is a set of vertices, and $\mathcal{E}_t \subseteq \{(v_i, v_j)$ s.t. $v_i, v_j \in \mathcal{V}_t$ and $v_i \neq v_j\}$.

Graph changes can be used to compose graph snapshots (we will use the terms graph and graph snapshot interchangeably).

Definition 2.2. A *graph snapshot* $\mathcal{G}_{t\to t'} = (\mathcal{V}_{t\to t'}, \mathcal{E}_{t\to t'})$ is an instance of a graph that satisfies the following for $t, t' \in \mathbb{N}$, $t < t'$: For any $\mathcal{G}_{k\in\mathbb{N}} = (\mathcal{V}_k, \mathcal{E}_k)$:
$\mathcal{G}_k \subset \mathcal{G}_{t\to t'}$ s.t. $\mathcal{V}_k \subseteq \mathcal{V}_{t\to t'}$ and $\mathcal{E}_k \subseteq \mathcal{E}_{t\to t'} \iff k \in [t, t']$.

For simplicity we will denote as $\overrightarrow{\mathcal{G}_t} = (\overrightarrow{\mathcal{V}_t}, \overrightarrow{\mathcal{E}_t})$ a graph snapshot $\mathcal{G}_{0\to t}$ that incorporates the complete set of changes up to step t.

Edges and vertices can have associated sets of features. We denote as $\mathcal{N}^k(v)$ the set of k-hop neighbors of vertex $v \in \mathcal{V}$.

Node Classification. The task of *node classification* is a supervised learning problem that aims to learn an assignment of a label (class) to each node in the graph. As an example, consider a citation network whose nodes are articles and edges link articles of the same author. A node classification task would involve inferring the topic of a given article or its probability to be accepted to a conference. To facilitate the application of ML algorithms to graphs, embedding techniques transform graph vertices and structural features into low dimensional representations h_v for every vertex $v \in \overrightarrow{\mathcal{V}_t}$. Recent work has proposed learning node embeddings in an inductive way, so that each node is represented by the aggregation of a subset of its neighbors [18]. Hence, if a new node appears, and it does not change the data distribution, it can still be properly represented by its neighbors. In this work, we use GraphSAGE [18], an inductive model that constructs the aggregation as follows:

$$h_v \leftarrow \sigma(f(v, \{u \text{ s.t. } u \in \mathcal{N}^k(v)\}))$$

where σ is the sigmoid function and f is an aggregation function such as MEAN, LSTM or AggregatePOOL [18].

Problem definition Given a stream of graph changes $\mathcal{G}_0, \ldots, \mathcal{G}_t, \mathcal{G}_{t+1}$, the streaming graph embedding problem seeks to continuously incorporate $\mathcal{G}_{t+1}$ into the model built on the snapshot $\overrightarrow{\mathcal{G}_t}$. Straightforward solutions include 1) training the model on $\mathcal{G}_{t+1}$, i.e., using only the newly added nodes and edges or 2) retraining the model on the complete snapshot $\overrightarrow{\mathcal{G}_{t+1}}$ from scratch to incorporate all changes.

The first approach leads to catastrophic forgetting [11], where the model does not remember past information ($\overrightarrow{\mathcal{G}_t}$) and becomes biased on the new data. The second approach is expensive for high update rates and as graphs grow larger in size. We target the supervised single-task scenario, where the embedding model is used to assign labels to an unlabeled graph stream. Our goal is to sustain the classification accuracy in the presence of class distribution changes and concept drift.

3 LEARNING ON STREAMING GRAPHS

Figure 1 shows our end-to-end solution to the streaming graph embedding problem and visualizes the effect of node rehearsal on the PubMed Diabetes network [33] (t-SNE representation). As graph data arrive in batches (t, $t+1$, etc.) new vertices and edges are incorporated in a live graph model and trigger retraining via experience replay. In the single-task scenario, it is crucial to access and reuse previous data, since memory-less continuous learning approaches [4, 25] can only learn effectively in multi-task settings [21, 32]. To this end, we use sampling to maintain and reuse information across consecutive model training instances.

3.1 Selecting Rehearsed Nodes

RBR yields a uniform sample of the overall data while *PBR* prioritizes data points which are more relevant than others with regard to preserving the decision boundary learnt by the classifier. Essentially, rehearsed nodes with *PBR* resemble support vectors in that they are the most *challenging* nodes to classify. Iterative Graph-Convolutional Network (GCN) training algorithms can exploit rehearsed data to generate effective vector-based graph representations without the need to train on the full dataset. Updates are fed to an incremental inference module which applies point model updates instead of completely replacing the model. We describe the sampling methods in detail next.

Random Sampling (*RBR*). At every time step t, the *RBR* strategy selects a set of past vertices $\mathcal{S}_t \subset \overrightarrow{\mathcal{V}_t}$ at random. This simple technique permits to reuse past data to update the model in the new snapshot. We expect this approach to work well when the previous data are i.i.d., so that sampling at random selects a representative set of the vertices. When class data is unbalanced, however, uniform random sampling is not adequate and a more involved strategy is required. Note that the purpose of *RBR* is not to maintain a uniform random sample of the graph seen so far, as training on a single sample would bias the model towards the selected nodes. Instead, a new sample is used for each backpropagation step and the model is trained on multiple possibly overlapping random samples of fixed size to achieve good performance.

Priority Sampling (*PBR*). Priority-Based Rehearsal samples nodes according to their *priority index*. At step t, we define the priority index for vertex $v \in \overrightarrow{\mathcal{V}_t}$ as follows:

$$p_v \propto \alpha \min(L^t(v), \epsilon) \tag{1}$$

where $L^t(v)$ is the loss value at time t of the vertex v[1], α is a smoothing parameter that determines the prioritization importance and ϵ is a small constant to avoid that the probability becomes 0. Essentially, we compute the priority index based on the model prediction error, so that nodes with higher prediction errors have a higher probability of being selected during sampling. Using this method, the training effort is focused on the nodes where the model is sub-optimal. For instance, if vertices have unbalanced classes, *PBR* rehearses a larger number of *rare* vertices, leading to least forgetting rare classes since their errors decrease at a slower rate, focusing more on harder vertices as the model improves. The effect is similar to online hard mining [20] and boosting [12], and is also analogous to a Deep Q reinforcement learning network trained with Prioritized Experience Replay [39], leading to faster learning while avoiding training stagnation.

[1]We use the last-seen loss computed during each training instance on the retrained samples to update the priority index.

Algorithm 1: *PBR* Online Training Algorithm

```
1  Inputs: b = batch_size; m = max_priority; ϵ; γ; α
2  G :: (V, E) = nil; // the dynamic graph
3  H :: dict[V, FLOAT] = nil; // vertex priorities
4  θ = Xavier Initialization; // neural network weights
5  Function update(G_t = (V_t, E_t)):
6      G = G_t→;
7      foreach v ∈ V_t do
8          H(v) = m; // add new vertex to the dictionary
9          foreach n ∈ v.neighbors do
10             H(n) = min(m, H(n) + γ / n.degree) ;
11         end
12     end
13 Function train(G_t = (V_t, E_t)):
14     update(G_t);
15     sample = prioritizedSample(H, b, α) ;
16     losses = L(sample, {v.label | v ∈ sample}) ;
17     θ = SGD(θ, losses);
18     foreach v ∈ sample do
19         H[v] = max(losses[v], ϵ) ;
20     end
```

Algorithm 1 shows the pseudocode of online training with *PBR*. The training process is divided in two main functions: *update* and *train*. The *update* function takes as input the a graph change $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ and assigns priorities to the graph vertices. In this function, we introduce a bonus for new vertices arriving with an unknown loss value, giving them maximum priority m (line 8). Then we increase the probability of sampling the neighbors of a new vertex by a $\frac{\gamma}{\text{neighbor.degree}}$, where γ is a small constant value (line 10). The *train* function performs the training of the neural network. At this point, we generate a mini-batch of b vertices using the *PBR* approach in line 15. Then, we compute the loss value of every sampled vertex (line 16) and these values are used to train the neural network parameters θ. At the end of the process, we use the loss values to update the priority indexes of the sampled vertices (line 19).

3.2 Implementation

We implemented *RBR* and *PBR* on the GraphSAGE [18] GNN using DGL, a high-performance framework-agnostic Python package for deep learning on graphs that leverages modern hardware. Compared to similar approaches, GraphSAGE keeps its computational

footprint constant and can scale to large graphs by restricting aggregation to neighborhood samples. Our methods, though, are not specific to GraphSAGE and can be used with other GNNs.

According to Algorithm 1, a new fixed-size sample is drawn during each backpropagation step. In practice, though, we draw multiple independent and possibly overlapping samples as soon as the change at step t has been integrated in the graph model and use them as needed. *PBR* is implemented using a *sum-tree* data structure. The leaf nodes of the tree store the vertex priorities and every other node is the sum of its children. This data structure supports updates and sample generation with a $O(\log |V|)$ complexity. While updating the priority index is more computationally expensive than using random sampling, in Section 5.3 we show that the additional overhead is not significant in practice.

To generate a prediction for vertex v, GCNs aggregate the embeddings of its neighbors up to depth d. A naive implementation of inference leads to redundant computations for vertices with overlapping neighborhoods. Instead, we employ an incremental algorithm to identify dependencies between new and existing vertices and recompute only those embeddings that are affected by changes. While interesting, we leave the details of the incremental inference algorithm out of the scope of this paper.

4 EXPERIMENTAL METHODOLOGY

We trained our models on a machine with a NVIDIA V100 GPU and a Intel Xeon Gold 6132 Processor, running CentOS 7, Pytorch 1.5.1, Tensorflow 2.2.0, DGL 0.5 and CUDA 10.1. We repeat each experiment 6 times and we report average results and the 95% confidence interval. Table 2 presents the datasets and parameters of our experimental evaluation. We have selected four publicly available graphs from different domains to evaluate the various class distribution changes and concept drift scenarios we describe in the next section. All datasets have associated timestamps in either their edges or their vertices and are ingested chronologically in batches (snapshots) to emulate an online scenario. Arxiv, PubMed and Bitcoin are modeled as streams of vertices while Reddit is modeled as an edge stream. In all experiments, we fix the sample size for *RBR* and *PBR* to 64 vertices, $\alpha = 1$, and $\gamma = 0.1$.

4.1 Evaluation scenarios

To study the performance and robustness of experience replay in GNNs, we consider four scenarios (and corresponding datasets) of changes in class distribution, skew, and concept drift.

4.1.1 Stable class distribution. The first scenario considers a *well-behaving* graph whose class distribution per snapshot remains stable throughout its temporal evolution. Arxiv [47] (ogbn-arxiv) is a representative dataset of such a graph. It models the citation network between CS arXiv papers. Nodes represent papers and edges indicate that a paper cites another. Each paper comes with a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. Papers are associated with the publishing date, thus, we ingest this graph as a vertex stream. Classes correspond to 40 subject areas of arXiv CS papers, such as cs.AI, cs.LG, and cs.OS, manually labeled by the paper's authors and arXiv moderators.

4.1.2 Abrupt class distribution shift. The second scenario aims to evaluate a case of sudden change in the graph evolution. Pubmed Diabetes [33] is a citation network extracted from the PubMed database. Each vertex (publication) has an associated timestamp that denotes the paper's publication date and a TF/IDF weighted word vector from a dictionary of 500 unique words. Vertices are classified into one of three classes. We ingest this dataset as a vertex stream in snapshots of size 50 (cf. Table 2). As the graph evolves, two sudden events change its class distribution. First, only classes 1 and 2 are present in the graph until snapshot 35, when class 0 is introduced. Second, all vertices after snapshot 80 belong to class 0. From that point on, no new examples from classes 1 and 2 appear.

4.1.3 Class distribution skew. The third scenario considers the presence of high skew in the class distribution across snapshots. We use Reddit [19], a graph of posts and comments, as the representative dataset. Every post (node) is labeled after the community or *subreddit* it belongs to. The post features consist of 602 values: the average embedding of the post title and its comments, the post's score, and the number of comments associated with the post. Text features have been generated using a 300-dimensional GloVe CommonCrawl word vector. Posts are linked with an edge if they have been commented by the same user and each edge bears the timestamp of the first comment. The graph has 41 classes, all of which appear in all snapshots. However, the class distribution is highly skewed with a few communities contributing significantly more examples than the others. Moreover, the class distribution changes across snapshots. As this dataset is an edge stream, incoming edges are more likely to have vertex endpoints belonging to popular classes. As a consequence, vertices from popular classes can appear multiple times in the same batch of changes.

4.1.4 Concept drift. The final scenario is an unfavorable case where the statistical properties of classes change over time. We use the Bitcoin [46] graph of 200K partially labeled transactions as a representative example of such an irregular dataset. Edges represent the flow of Bitcoin currency from one transaction to the next one. A timestamp is associated with each transaction, representing an approximate time when it was broadcasted to the bitcoin network. The dataset categorizes the nodes as "licit", "illicit" or "unknown". A node is illicit if the corresponding transaction has been created by an entity that belongs to an illicit category (scams, malware, terrorist organizations, ransomware, Ponzi schemes, etc.). 2% of the nodes are illicit while 21% of them are licit. Unknown vertices are not considered as a separate class. Vertices with unknown labels modify the graph structure but do not contribute to the training or test sets. However, the network learns to predict the label of a known vertex using also features from its unknown neighbors. There are 166 features associated with each node and the temporal information is encoded by a time step running from 1 to 49, spaced with an interval of about two weeks. The statistical properties of the licit and illicit classes change over time, making the classification task too hard for GNNs.

4.2 Snapshot construction

To evaluate the accuracy of *RBR* and *PBR* in a streaming scenario, we create a sequence of temporal snapshots by taking into account

Dataset	Type	|V|	|E|	Classes	Ingestion	Backprop.	Timestamp	Re-train period	Test period
Arxiv	Citation	169K	1.16M	40	50	2	Publication date	8300 vertices	700 vertices
PubMed	Citation	20K	44K	3	50	2	Pub. date	6500 vertices	200 vertices
Reddit	Social	220K	5M	41	1200	50	First comment	720K edges	20K edges
Bitcoin	Transaction	204K	234K	2	204	6	T. broadcasted	70K vertices	2040 vertices

Table 2: Datasets and parameters used for experimental evaluation.

the timestamps associated with the vertices or edges in each graph. We create snapshots so that the number of mutations (vertex or edge additions) they introduce is constant over time. We use two strategies to split the graphs of Table 2 into training and test data.

The *default* strategy creates a 70 : 30 split of training and test data, where each new vertex in a snapshot is added to either the test or the train set at random. As a consequence, the test set at time step t consists of a random sample of vertices that arrived in steps [0-t]. The test set is therefore independent of the train set, albeit it follows the same probability distribution. This strategy is the default evaluation approach established by the previous work [18, 24, 42] and allows us to evaluate the performance of our methods on vertices *similar* to the ones in the train set.

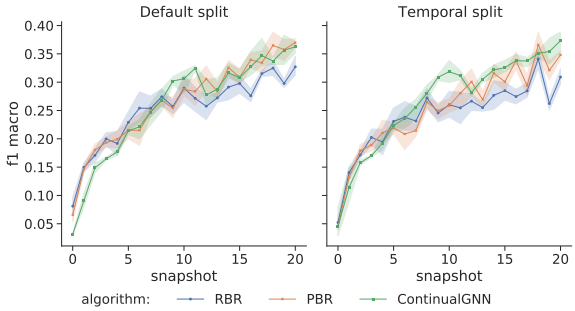
To faithfully simulate a streaming setting, we further employ a *temporal* splitting strategy to construct the test and train sets, similar to the approach introduced in an anti-money laundering study [46]. At snapshot t, we generate the test set using vertices of snapshot $t+1$ and the train set using vertices that arrived before t. As a result, the test set captures the actual graph evolution and is not guaranteed to follow the same distribution as the train set.

5 EXPERIMENTAL EVALUATION

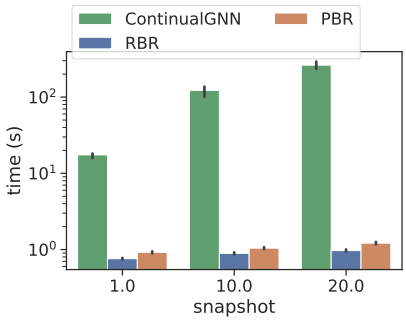
We evaluate two key aspects of *RBR* and *PBR*. First, we examine **the quality of the embeddings** they generate using a node classification task. To that end, we are interested in understanding (i) how our methods behave upon encountering changes in the class distribution, and (ii) how their convergence behavior compares to other approaches given a large graph with multiple classes. The results of Sections 5.1 and 5.2 demonstrate that *RBR* and *PBR* achieve the same or higher accuracy as state-of-the-art online methods and offline retraining on the entire graph, they converge quickly, and they are robust to changes. Second, we evaluate **the execution runtime benefits** that our methods provide by avoiding to retrain on the full network. The results of Sections 5.1 and 5.3 show that *RBR* and *PBR* save hours of training time when compared to offline training and are orders of magnitude faster than existing continual learning methods.

5.1 Comparison with ContinualGNN

We first compare the performance of *RBR* and *PBR* with that achieved by the state-of-the-art method, ContinualGNN [44], on the supervised node classification task. We select the largest dataset, Reddit, and calculate macro F1 for the test set for every snapshot[2]. Like *PBR* and *RBR*, ContinualGNN is a replay-based method that aims to sample *important* nodes for incremental training. Figure 2 shows



(a) f1 Macro (Reddit)



(b) Training latency in s (Reddit)

Figure 2: Comparison of classification performance and training latency with ContinualGNN on the Reddit dataset.

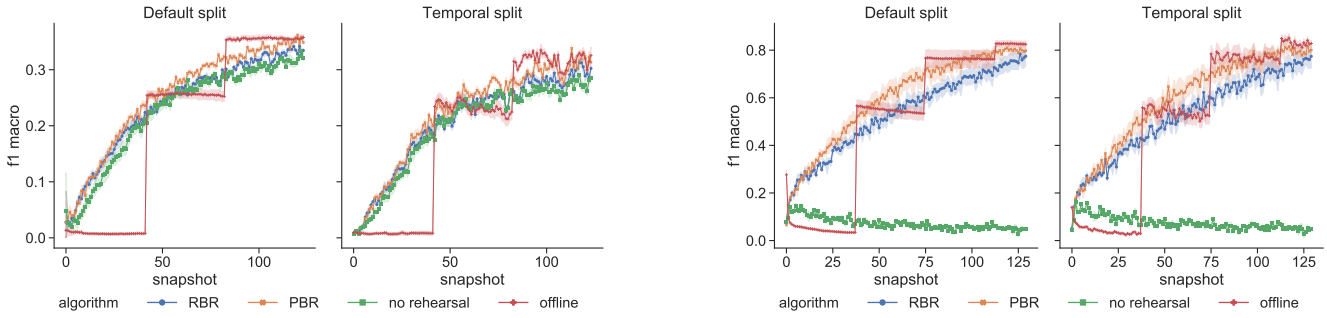
the classification and latency results for the first 20 snapshots. We observe that *PBR*, *RBR*, and ContinualGNN achieve similar classification accuracy, however, the training latency of ContinualGNN grows proportionally to the size of the graph stream. At snapshot 20, ContinualGNN takes over 300*s* to retrain its model. On the other hand, the training latency of *PBR* and *RBR* remains constant over time and well below 1*s*.

5.2 Classification performance

We now further compare our online methods with two additional baselines on all four datasets: (i) *Offline* corresponds to periodically training over the full graph with multiple epochs and (ii) *No-Rehearsal* corresponds to training over the new vertices in a snapshot. We expect the latter baseline to perform well if the class distribution does not change across consecutive snapshots.

The experiments in this section favor the offline training strategy by not taking into account the training execution time when evaluating classification performance. Even if periodic retraining on the entire graph requires a substantial amount of time, we assume that training is instant and the model becomes available immediately. Training performance is evaluated separately in Section 5.3.

[2] *F1 Macro* computes the F1 for each class and then takes the unweighted mean: $\sum_k F1_k \cdot \frac{1}{k}$.



(a) Arxiv (b) Reddit

Figure 3: Evolution of the classification performance (Macro F1) as the graph stream is ingested continuously over time.

	Pubmed						Bitcoin					
	Default split			Temporal split			Default split			Temporal split		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Offline	0	1	0.63	0	0.88	0.36	0.33	0.76	0.59	0.28	0.6127	0.47
No-rehearsal	0.02	1	0.62	0	1	**0.63**	0.36	0.62	0.49	0.31	**0.73**	**0.49**
RBR	0.4	1	0.73	0	0.87	0.45	0.49	0.74	0.64	0.29	0.6796	0.48
PBR	0.22	1	**0.76**	0	**0.95**	0.52	0.49	**0.77**	**0.66**	0.32	**0.73**	**0.49**

Table 3: Min, max, and average Macro F1 measurements for all methods on the classification task.

Arxiv (Stable class distribution). Figure 3a plots the F1 macro score for the classification task on the Arxiv graph over time. Each step in the x-axis corresponds to an addition of 14 batches of new vertices of size given in Table 2. F1 macro is arguably low for all methods, including *Offline*[3]. Yet, the online methods show competitive performance and improve over time, matching the accuracy of full retraining. This dataset is an example of a well-behaving, predictable graph with regard to temporal change. The fact that class distribution per snapshot remains relatively stable throughout the graph's evolution means that even the naive *No-Rehearsal* method achieves the same F1 score as offline training.

Reddit (Skew). Let us now examine the case of an edge stream. Recall that when a set of edge changes are added to the graph, some of the newly added vertices will appear multiple times. Figure 3b plots the F1 macro metric over time for the node classification task on Reddit. In the default split strategy, *PBR* achieves the highest score during the entire experiment. *RBR* matches its performance only at step 130. *Offline* can only improve its performance after re-training, while *No-Rehearsal* performs poorly throughout the evolution of the graph. The results of the temporal split strategy show almost identical behavior. This is expected, as all classes are present in the graph since the beginning and the temporal split constructs snapshots with similar class distribution as that of the default split.

Pubmed (Abrupt shift). Let us now examine a case of concept drift. Table 3 reports F1 macro results for the Pubmed dataset. This is a vertex stream with 3 classes, 2 of which are present in the graph since the beginning, while the third class appears later. In the default split case, during the first 10 steps, *RBR* and *PBR* perform equally well and outperform *Offline* training, which initiates a full retrain every 33 steps. From snapshot 11, we observe a steep increase in accuracy for the online methods. *PBR* and *No-Rehearsal* improve their performance quickly, while random takes more steps to reach the same level of accuracy. At snapshot 66, a significant number of examples from the third class is introduced in the dataset. The *Offline* method cannot predict this change without retraining. The online methods also experience an accuracy drop, however, *PBR* is the only method able to quickly correct its model. After snapshot 80 every new node ingested belongs to the same class. While our methods continue to correctly predict classes of new nodes, the accuracy of *No-Rehearsal* drops below 0.2. The effects of the abrupt shift in class distribution become clearer in the temporal shift experiment. Since all new examples in snapshots 80-100 belong to the same class, the performance of *Offline* drops drastically without retraining. In this case, *No-Rehersal* achieves perfect accuracy as it only considers new vertices anyway. Further, the superiority of *PBR* over uniform random sampling becomes clear. *PBR* is capable of selecting the right rehearsal nodes and quickly improve accuracy.

Bitcoin (Concept drift). The Bitcoin dataset is a 2-classes unbalanced dataset, with both classes available since the beginning. Table 3 shows the min, max, and average F1 macro achieved over the evolution of the graph stream. During the first 18 steps of the default split, all methods exhibit similar performance. From snapshot 19 on, there is a steep increase in accuracy for the *PBR* and *RBR* methods, where *PBR* converges faster and achieves a higher macro F1 score. We attribute the difference between *RBR* and *PBR* to the fact that priority sampling selects more rehearsal vertices from the minority class. The *Offline* method improves its performance at snapshot 33 because of the retrain, while the performance of *No-Rehearsal* remains low and stable throughout the graph evolution. This behavior is due to too few vertices of the minority class being added in each snapshot, preventing the neural network from learning the difference between the two classes when minority

[3]When plotting accuracy instead of F1 macro, all methods reach a value of ~ 0.69, which is in agreement with the Open Graph Benchmark leaderboard (https://ogb.stanford.edu/docs/leader_nodeprop/#ogbn-arxiv)

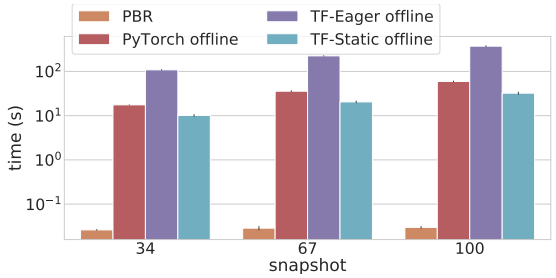


Figure 4: Training execution time on graph snapshots of increasing size for three offline methods and *PBR* (Pubmed).

nodes from the previous snapshots are not used for training. It is also worth noting that after retraining, the prediction performance of the *Offline* method degrades over time, while both *RBR* and *PBR* maintain high F1 scores. This happens because the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways [46]. *Offline* cannot generate good predictions for the vertices added in the test set in each snapshot after training and this causes the steady drop in the macro F1 score. The unpredictable changes in the statistical properties of the licit and illicit classes alongside the fact that the majority of examples are not labeled contribute to all four methods demonstrating degraded performance for temporal split. The *Offline* method fails to achieve F1 score higher than ≈ 0.6. This behavior is in agreement with the results of the recent study that introduces this dataset [46], where a Random Forest classifier outperforms the GCN methods. Nevertheless, *RBR* and *PBR* perform as well as the offline method, even in this unfavorable scenario.

5.3 Training performance

We now evaluate the performance benefits of *PBR* and *RBR* over periodically performing full model retraining. We implemented GraphSAGE on TensorFlow and PyTorch and evaluate three execution modes. *PyTorch* is our default implementation. It relies on the DGL [45] high performance Python package for deep learning on graphs that leverages modern hardware. The computational graph is generated dynamically and rebuilt after each iteration of training. *Tensorflow-eager* also relies on DGL, but operations between tensors have been implemented using Tensorflow 2. Since DGL does not currently support symbolic execution, we enabled *EagerExecution*, an imperative programming environment that operates like PyTorch, evaluating operations immediately without building the computation graph first. *Tensorflow-static* relies on Tensorflow 2 optimized symbolic execution. In this mode, the computation graph is static. We created and connected all the variables at the beginning. As a consequence, the graph is built once and can be reused across training iterations.

Figure 4 plots the training time of the offline methods and that of *PBR* for three snapshots of the Pubmed dataset. We observe that the cost of offline training is proportional to the size of the input graph, while *PBR* achieves orders of magnitude lower training time. *PBR*'s performance depends only on the size of the rehearsed nodes sample and the number of backpropagation iterations, thus, its training execution time remains constant regardless of the graph size. Comparing the performance of offline training methods, we see that *TF-static* outperforms the other frameworks. As expected, static graph generation has a positive impact on performance, however, we observe that *PyTorch* is highly competitive with *TF-static*, even though it generates the computation graph dynamically.

6 RELATED WORK

The majority of existing graph representation learning methods target static graphs only [23]: factorization-based approaches [3, 5, 26], random walk-based methods [10, 17, 34, 35, 40], and autoencoders [7, 43], and graph neural networks [8, 9, 13, 18, 22, 30, 41, 49–51]. These methods require retraining when accuracy degrades or in the case of concept drifts. Xie et al. [48] have contributed a survey covering recent efforts in dynamic graph representation learning. Of those, DynGEM [16] relies on deep autoencoders to maintain stable embeddings over time but it does not handle attributed graphs. DNE [10] is a selection mechanism to choose and train the vertices affected by changes and update their representations. LINE [40] is based on random-walk methods and introduces a loss function for characterizing first-order and second-order proximities. DySAT [31, 38] constructs embeddings that take into account the structural and temporal properties of the graph, while Béres et al. [6] propose a method that relies on streaming algorithms to construct node representations. Yet, these methods do not take into account the features of vertices and edges. Several recent approaches support temporal graphs, in the sense that they optimize learning the graph's *temporal evolution*. These approaches ingest a graph as a series of snapshots, yet they require all snapshots to be available a priori at training time [14, 52, 53]. The work by Rossi et al. [36] supports streaming graphs, however it only considers updates in the most recently received batch. Finally, ContinualGNN [44] is the work closest to our approach. As we showed in Section 5.3, while ContinualGNN achieves good classification accuracy on par with *PBR*, its training latency grows with the graph size, making it unsuitable for streaming applications in practice.

7 CONCLUSION

We have introduced the problem of continuous graph representation learning from streams of changes and propose the use of experience replay, a pragmatic approach with classification performance benefits and guaranteed low retraining cost for fast model updates. We introduced two lightweight sampling methods, *Priority-Based Rehearsal-PBR* and *Random-Based Rehearsal-RBR*, to select rehearsal nodes for each batch of changes. Our results demonstrate that experience replay achieves good accuracy and convergence, comparable to that of offline training on the entire graph. At the same time, it provides fast retraining, since it uses only a fraction of the overall ingested training data.

8 ACKNOWLEDGEMENTS

The authors are pleased to acknowledge that the computational work reported on in this paper was performed on the Shared Computing Cluster which is administered by Boston University's Research Computing Services [1].

REFERENCES

[1] BU Shared Computing Cluster. www.bu.edu/tech/support/research/. Last access: December 2021.

[2] Online GNN Learning Repository. https://github.com/MassimoPerini/online-gnn-learning. Last access: December 2021.

[3] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. 2013. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web - WWW '13*. ACM Press, New York, New York, USA, 37–48. https://doi.org/10.1145/2488388.2488393

[4] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. 2018. Memory Aware Synapses: Learning What (not) to Forget. In *Computer Vision – ECCV 2018*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). Springer International Publishing, Cham, 144–161.

[5] Mikhail Belkin and Partha Niyogi. 2001. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic (NIPS'01)*. MIT Press, Cambridge, MA, USA, 585–591. http://dl.acm.org/citation.cfm?id=2980539.2980616

[6] Ferenc Béres, Domokos M Kelen, Róbert Pálovics, and András A Benczúr. 2019. Node embeddings in dynamic graphs. *Applied Network Science* 4, 1 (2019), 64.

[7] Shaosheng Cao and Wei Lu. 2016. Deep Neural Networks for Learning Graph Representations. In *AAAI (AAAI'16)*. AAAI Press, 1145–1152. http://dl.acm.org/citation.cfm?id=3015812.3015982www.aaai.org

[8] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *International Conference on Learning Representations*.

[9] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 257–266.

[10] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. 2018. Dynamic network embedding: An Extended Approach for Skip-gram based Network Embedding: An Extended Approach for Skip-gram based Network Embedding. *IJCAI International Joint Conference on Artificial Intelligence* 2018-July (2018), 2086–2092. https://doi.org/10.24963/ijcai.2018/288

[11] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.

[12] M Galar, A Fernandez, E Barrenechea, H Bustince, and F Herrera. 2012. A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42, 4 (2012), 463–484. https://doi.org/10.1109/TSMCC.2011.2161285

[13] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1416–1424.

[14] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. 2020. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems* 187 (2020), 104816.

[15] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (7 2018), 78–94. https://doi.org/10.1016/J.KNOSYS.2018.03.022

[16] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. 2018. DynGEM: Deep Embedding Method for Dynamic Graphs. (5 2018).

[17] Aditya Grover and Jure Leskovec. 2016. node2vec. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*. ACM Press, New York, New York, USA, 855–864. https://doi.org/10.1145/2939672.2939754

[18] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30*, I Guyon, U V Luxburg, S Bengio, H Wallach, R Fergus, S Vishwanathan, and R Garnett (Eds.). Curran Associates, Inc., 1024–1034. http://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.pdf

[19] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *CoRR* (2017), 1–24. https://doi.org/doi:10.6084/m9.figshare.97898

[20] Alexander Hermans, Lucas Beyer, and Bastian Leibe. 2017. In Defense of the Triplet Loss for Person Re-Identification. *CoRR* abs/1703.0 (2017). http://arxiv.org/abs/1703.07737

[21] Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. 2018. Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines. *arXiv e-prints* (2018), arXiv:1810.12488.

[22] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive Sampling Towards Fast Graph Representation Learning. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). 4558–4567.

[23] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. 2020. Representation Learning for Dynamic Graphs: A Survey. *Journal of Machine Learning Research* 21, 70 (2020), 1–73.

[24] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[25] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. [n. d.]. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114, 13 (3 [n. d.]), 3521 – 3526. https://doi.org/10.1073/pnas.1611835114

[26] Jundong Li, Harsh Dani, Xia Hu, Jiliang Tang, Yi Chang, and Huan Liu. 2017. Attributed Network Embedding for Learning in a Dynamic Environment. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17* (11 2017), 387–396. https://doi.org/10.1145/3132847.3132919

[27] Long-Ji Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8, 3 (1992), 293–321. https://doi.org/10.1007/BF00992699

[28] Xi Liu, Ping-Chun Hsieh, Nick Duffield, Rui Chen, Muhe Xie, and Xidao Wen. 2019. Real-time streaming graph embedding through local actions. In *Companion Proceedings of The 2019 World Wide Web Conference*. 285–293.

[29] Yao Ma, Ziyi Guo, Zhaochun Ren, Eric Zhao, Jiliang Tang, and Dawei Yin. 2018. Streaming Graph Neural Networks. (10 2018). http://arxiv.org/abs/1810.10627

[30] Yao Ma, Suhang Wang, Charu C Aggarwal, and Jiliang Tang. 2019. Graph convolutional networks with eigenpooling. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 723–731.

[31] Torricelli Maddalena, Karsai Márton, and Gauvin Laetitia. 2020. weg2vec: Event embedding for temporal networks. *Scientific Reports (Nature Publisher Group)* 10, 1 (2020).

[32] Davide Maltoni and Vincenzo Lomonaco. 2019. Continuous learning in single-incremental-task scenarios. *Neural Networks* 116 (8 2019), 56–73. https://www.sciencedirect.com/science/article/pii/S0893608019300838

[33] Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. 2012. Query-driven Active Surveying for Collective Classification. In *Workshop on Mining and Learning with Graphs (MLG)*.

[34] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. 2018. Continuous-Time Dynamic Network Embeddings. In *Companion Proceedings of the The Web Conference 2018 (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 969–976. https://doi.org/10.1145/3184558.3191526

[35] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: online learning of social representations. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (2014), 701–710. https://doi.org/10.1145/2623330.2623732

[36] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *arXiv preprint arXiv:2006.10637* (2020).

[37] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2018. Dynamic Graph Representation Learning via Self-Attention Networks. (12 2018). http://arxiv.org/abs/1812.09430

[38] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2018. Dynamic graph representation learning via self-attention networks. *arXiv preprint arXiv:1812.09430* (2018).

[39] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized Experience Replay. *International Conference on Learning Representations* (2016). http://arxiv.org/abs/1511.05952

[40] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web (WWW '15)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1067–1077. https://doi.org/10.1145/2736277.2741093

[41] Dinh V Tran, Nicolò Navarin, and Alessandro Sperduti. 2018. On filter size in graph convolutional networks. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1534–1541.

[42] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations (ICLR)*. http://arxiv.org/abs/1710.10903

[43] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural Deep Network Embedding. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 1225–1234. https://doi.org/10.1145/2939672.2939753

[44] Junshan Wang, Guojie Song, Yi Wu, and Liang Wang. 2020. Streaming graph neural networks via continual learning. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1515–1524.

[45] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Hao Zhang, Haibin

Lin, Junbo Zhao, Jinyang Li, Alexander J Smola, and Zheng Zhang. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *ICLR Workshop on Representation Learning on Graphs and Manifolds* (2019). https://arxiv.org/abs/1909.01315

[46] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I Weidele, Claudio Bellei, Tom Robinson, and Charles E Leiserson. 2019. Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics. *arXiv preprint arXiv:1908.02591* (2019).

[47] Marinka Zitnik Yuxiao Dong Hongyu Ren Bowen Liu Michele Catasta Jure Leskovec Weihua Hu, Matthias Fey. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687* (2020).

[48] Yu Xie, Chunyi Li, Bin Yu, Chen Zhang, and Zhouhua Tang. 2020. A Survey on Dynamic Network Embedding. arXiv:cs.SI/2006.08093

[49] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826* (2018).

[50] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. ACM, New York, NY, USA, 974–983. https://doi.org/10.1145/3219819.3219890

[51] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[52] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process. *Aaai 2018* (2018), 571–578. http://yangy.org/works/dynamictriad/dynamic_triad.pdf

[53] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding temporal network via neighborhood formation. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2857–2866.