ROLAND: Graph Learning Framework for Dynamic Graphs

Jiaxuan You, Tianyu Du, Jure Leskovec

March 20, 2024

Why Dynamic Graph

- **Real-world Graphs are Dynamic**:
  - Most graphs in real-world applications are dynamic, evolving over time.
- **Lack of Generalization in Static Graph Inference**:
  - Inference on static graphs often lacks generalization. It's more about semi-supervised learning.

An Example of Dynamic Graphs

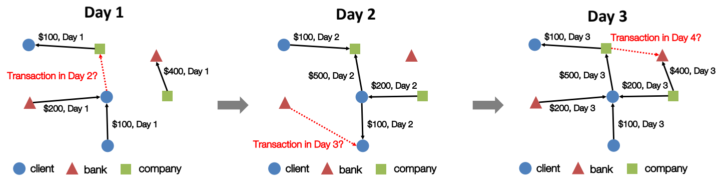


Figure: Example ROLAND use case for future link prediction on a dynamic transaction graph. We use information up to time t to predict potential edges at time $t+1$.

High Level Abstract

- **Conversion of Any GNN Model to Dynamic Graph Model**:
  - The ROLAND algorithm is capable of retrofitting any GNN model to cater to dynamic graphs, enhancing versatility and application scope.
- **Core Concept of Transitioning from Node Embeddings to Node States**:
  - The cornerstone of ROLAND's approach lies in transitioning from static node embeddings to evolving node states over time, capturing the temporal dynamics within the graph.

Overall Framework

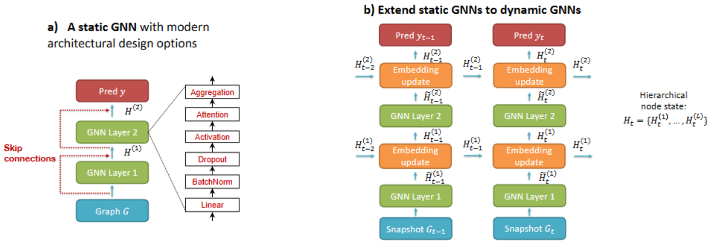


Figure: This figure illustrates the framework of a traditional static GNN alongside the corresponding Roland framework.

Graph Neural Networks (GNNs)

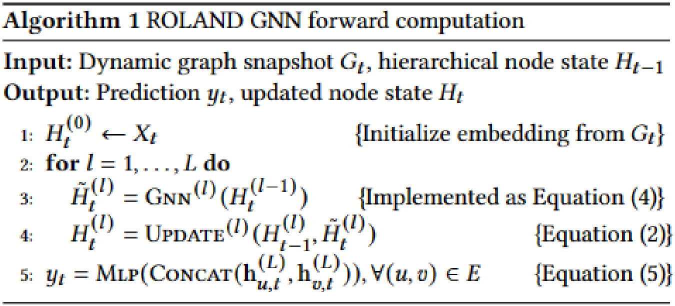
Graph Neural Networks (GNNs) operate on the principle of aggregating information from a node's neighbors to update its representation. The process can be formalized by the following equations:

$$\begin{aligned} \mathbf{m}_{u\to v}^{(l)} &= \textsc{Msg}^{(l)}(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}) \\ \mathbf{h}_v^{(l)} &= \textsc{Agg}^{(l)}\big(\{\mathbf{m}_{u\to v}^{(l)} \mid u \in \mathcal{N}(v)\}, \mathbf{h}_v^{(l-1)}\big) \end{aligned} \tag{1}$$

Here, $\mathbf{m}_{u\to v}^{(l)}$ represents the message from node u to node v at layer l, computed by the message function $\textsc{Msg}^{(l)}$ based on their embeddings $\mathbf{h}_u^{(l-1)}$ and $\mathbf{h}_v^{(l-1)}$ from the previous layer $(l-1)$. The updated embedding $\mathbf{h}_v^{(l)}$ for node v is then computed by the aggregation function $\textsc{Agg}^{(l)}$, which aggregates all incoming messages $\{\mathbf{m}_{u\to v}^{(l)}\}$ from v's neighbors $\mathcal{N}(v)$, along with v's previous embedding $\mathbf{h}_v^{(l-1)}$.

The ROLAND Forward Computation

Here's an overview of the forward computation algorithm:



```
Algorithm 1 ROLAND GNN forward computation
Input: Dynamic graph snapshot $G_t$, hierarchical node state $H_{t-1}$
Output: Prediction $y_t$, updated node state $H_t$
1: $H_t^{(0)} \leftarrow X_t$                                   {Initialize embedding from $G_t$}
2: for $l = 1, \ldots, L$ do
3:     $\tilde{H}_t^{(l)} = \textsc{Gnn}^{(l)}(H_t^{(l-1)})$      {Implemented as Equation (4)}
4:     $H_t^{(l)} = \textsc{Update}^{(l)}(H_{t-1}^{(l)}, \tilde{H}_t^{(l)})$      {Equation (2)}
5: $y_t = \textsc{Mlp}(\textsc{Concat}(\mathbf{h}_{u,t}^{(L)}, \mathbf{h}_{v,t}^{(L)})), \forall (u, v) \in E$      {Equation (5)}
```

Figure: The ROLAND algorithm workflow.

Update Functions

This work presents three methods for updating states:

1. **Moving Average:**

$$H_{t,v}^{(l)} = \kappa_{t,v} H_{t-1,v}^{(l)} + (1 - \kappa_{t,v}) \tilde{H}_{t,v}^{(l)} \tag{2}$$

where $\kappa_{t,v}$ is defined as:

$$\kappa_{t,v} = \frac{\sum_{\tau=1}^{t-1} |E_\tau|}{\sum_{\tau=1}^{t-1} |E_\tau| + |E_t|} \in [0, 1] \tag{3}$$

This formula adjusts the node state by blending its previous state with the newly computed state, weighted by the keep ratio $\kappa_{t,v}$, which is determined by the proportion of edges in the graph up to time t.

Update Functions

Continuing with the update methods:

2. *MLP*: Node embeddings are updated by a 2-layer MLP, $H_t^{(l)} = \textsc{Mlp}(\textsc{Concat}(H_{t-1}^{(l)}, \tilde{H}_t^{(l-1)}))$. This method employs a Multi-Layer Perceptron to fuse the information from the current and previous states.
3. *GRU*: Node embeddings are updated by a GRU cell, $H_t^{(l)} = \textsc{Gru}(H_{t-1}^{(l)}, \tilde{H}_t^{(l)})$, where $H_{t-1}^{(l)}$ is the hidden state and $\tilde{H}_t^{(l)}$ is the input for the GRU cell. Inspired by recurrent neural network architecture, this approach leverages GRU cells for temporal state updates.

Live-update Evaluation Setting

To evaluate the model using all snapshots, we propose a live-update evaluation procedure, encapsulating the following key steps:

1. Fine-tune the GNN model using newly observed data.
2. Evaluate predictions using new data.
3. Make predictions using historical information.

Scalable and Efficient Training for Dynamic GNNs

- Only the model $\textsc{Gnn}$, the new graph snapshot G_t, and the historical node states H_{t-1} are kept in GPU memory, optimizing memory use.
- Given that H_{t-1} encodes information up to time $t-1$, the model learns from H_{t-1} and the new snapshot G_t instead of the entire sequence of graphs. This incremental learning strategy significantly reduces computational complexity.
- The memory complexity of ROLAND is independent of the number of graph snapshots, enabling the training of GNNs on dynamic graphs with up to 56 million edges and 733 graph snapshots.

Fast Adaptation: Meta-training on Dynamic Graphs

- The conventional method of continuously fine-tuning a GNN with new data may not always yield the optimal model for future predictions, especially in graphs with temporal patterns.
- We propose the concept of a *meta-model* $\text{GNN}^{(meta)}$, which acts as an effective starting point for quickly generating specialized models for new, unseen prediction tasks.
- Our strategy employs the Reptile meta-learning algorithm for updating $\text{GNN}^{(meta)}$, striking a balance between adaptation and stability by calculating a moving average of trained models with a smoothing factor α.

Results in the Standard Fixed Split Setting

Table: We run each experiment with 3 random seeds and report the average performance together with the standard error.

	Bitcoin-OTC	Bitcoin-Alpha	UCI-Message
GCN	0.0025	0.0031	0.1141
DynGEM	0.0921	0.1287	0.1055
dyngraph2vecAE	0.0916	0.1478	0.0540
dyngraph2vecAERNN	**0.1268**	**0.1945**	0.0713
EvolveGCN-H	0.0690	0.1104	0.0899
EvolveGCN-O	0.0968	0.1185	**0.1379**
ROLAND Moving Average	0.0468 ± 0.0022	0.1399 ± 0.0107	0.0649 ± 0.0049
ROLAND MLP	0.0778 ± 0.0024	0.1561 ± 0.0114	0.0875 ± 0.0110
ROLAND GRU	$\mathbf{0.2203 \pm 0.0167}$	$\mathbf{0.2885 \pm 0.0123}$	$\mathbf{0.2289 \pm 0.0618}$
Improvement over best baseline	73.74%	48.33%	65.99%

Results in the Live-update Settings

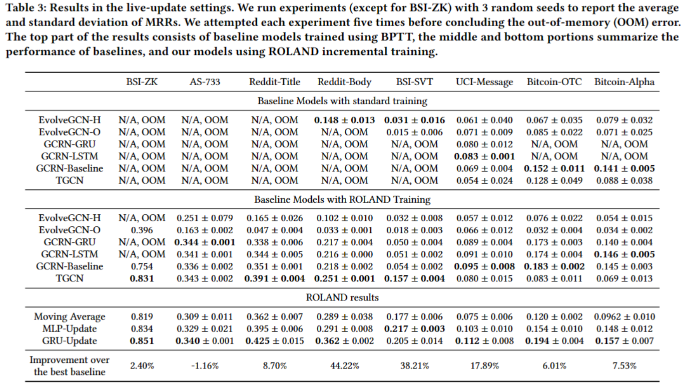


Table 3: Results in the live-update settings. We run experiments (except for BSI-ZK) with 3 random seeds to report the average and standard deviation of MRRs. We attempted each experiment five times before concluding the out-of-memory (OOM) error. The top part of the results consists of baseline models trained using BPTT, the middle and bottom portions summarize the performance of baselines, and our models using ROLAND incremental training.

	BSI-ZK	AS-733	Reddit-Title	Reddit-Body	BSI-SVT	UCI-Message	Bitcoin-OTC	Bitcoin-Alpha
Baseline Models with standard training								
EvolveGCN-H	N/A, OOM	N/A, OOM	N/A, OOM	**0.148 ± 0.013**	**0.031 ± 0.016**	0.061 ± 0.040	0.067 ± 0.035	0.079 ± 0.032
EvolveGCN-O	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.015 ± 0.006	0.071 ± 0.009	0.085 ± 0.022	0.071 ± 0.025
GCRN-GRU	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.080 ± 0.012	N/A, OOM	N/A, OOM
GCRN-LSTM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	**0.083 ± 0.001**	N/A, OOM	N/A, OOM
GCRN-Baseline	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.069 ± 0.004	**0.152 ± 0.011**	**0.141 ± 0.005**
TGCN	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	N/A, OOM	0.054 ± 0.024	0.128 ± 0.049	0.088 ± 0.038
Baseline Models with ROLAND Training								
EvolveGCN-H	N/A, OOM	0.251 ± 0.079	0.165 ± 0.026	0.102 ± 0.010	0.032 ± 0.008	0.057 ± 0.012	0.076 ± 0.022	0.054 ± 0.015
EvolveGCN-O	0.396	0.163 ± 0.002	0.047 ± 0.004	0.033 ± 0.001	0.018 ± 0.003	0.066 ± 0.012	0.032 ± 0.004	0.034 ± 0.002
GCRN-GRU	N/A, OOM	**0.344 ± 0.001**	0.338 ± 0.006	0.217 ± 0.004	0.050 ± 0.004	0.089 ± 0.004	0.173 ± 0.003	0.140 ± 0.004
GCRN-LSTM	N/A, OOM	0.341 ± 0.001	0.344 ± 0.005	0.216 ± 0.000	0.051 ± 0.002	0.091 ± 0.010	0.174 ± 0.004	**0.146 ± 0.005**
GCRN-Baseline	0.754	0.336 ± 0.002	0.351 ± 0.001	0.218 ± 0.002	0.054 ± 0.002	**0.095 ± 0.008**	**0.183 ± 0.002**	0.145 ± 0.003
TGCN	**0.831**	0.343 ± 0.002	**0.391 ± 0.004**	**0.251 ± 0.001**	**0.157 ± 0.004**	0.080 ± 0.015	0.083 ± 0.011	0.069 ± 0.013
ROLAND results								
Moving Average	0.819	0.309 ± 0.011	0.362 ± 0.007	0.289 ± 0.038	0.177 ± 0.006	0.075 ± 0.006	0.120 ± 0.002	0.0962 ± 0.010
MLP-Update	0.834	0.329 ± 0.021	0.395 ± 0.006	0.291 ± 0.008	**0.217 ± 0.003**	0.103 ± 0.010	0.154 ± 0.010	0.148 ± 0.012
GRU-Update	**0.851**	**0.340** ± 0.001	**0.425 ± 0.015**	**0.362 ± 0.002**	0.205 ± 0.014	**0.112 ± 0.008**	**0.194 ± 0.004**	**0.157 ± 0.007**
Improvement over the best baseline	2.40%	-1.16%	8.70%	44.22%	38.21%	17.89%	6.01%	7.53%

Figure: Table 3: Results in the live-update settings.

Thank You

Thank you for your attention!

Questions?