DynaGraph: Dynamic Graph Neural Networks at Scale

Mingyu Guan*, Anand Iyer*, Taesoo Kim*

*Georgia Institute of Technology  *Microsoft Research

GRADES-NDA 2022
Graph Neural Networks (GNNs)

- The recent past has seen an increasing interest in GNNs.
- Node embeddings are generated by combining graph structure and feature information.
- Most GNN models can fit into the Message Passing Paradigm.
Message Passing Paradigm

Current Neighbor States

Current Node State $h_{b}^{l-1}$
Message Passing Paradigm

Current Neighbor States

Current Node State $h_{t-1}^l$

Messages from Neighbors
Message Passing Paradigm

Current Neighbor States

Messages from Neighbors

Current Node State $h_{v}^{l-1}$

Aggregate and Reduce Received Messages
Message Passing Paradigm

Current Neighbor States

Messages from Neighbors

Current Node State $h_{v}^{l-1}$

Update

Next Node State $h_{v}^{l}$

Aggregate and Reduce Received Messages

$m_{v}^{l}$
Dynamic GNNs

• Most of existing GNN frameworks assume that the input graph is static.
• Real-world graphs are often dynamic in nature.
• Representation: a time series of snapshots of the graph.
• Common approach: Combine GNNs and RNNs.
  o GNNs for encoding spatial information (graph structure)
  o RNNs for encoding temporal information
LSTM

Gate $i$
- $W_{xi}x_t$
- $W_{hi}h_{t-1}$

Gate $f$
- $W_{xf}x_t$
- $W_{hf}h_{t-1}$

Gate $c$
- $W_{xc}x_t$
- $W_{hc}h_{t-1}$

Gate $o$
- $W_{xo}x_t$
- $W_{ho}h_{t-1}$

$E \rightarrow h_t$

GRU

Gate $r$
- $W_{xr}x_t$
- $W_{hr}h_{t-1}$

Gate $h$
- $W_{xh}x_t$
- $W_{hh}h'$

Gate $z$
- $W_{xz}x_t$
- $W_{hz}h_{t-1}$

$E \rightarrow h_t$

Time-independent
LSTM

Gate $i$
- $W_{xi}x_t$
- $W_{hi}h_{t-1}$

Gate $f$
- $W_{xf}x_t$
- $W_{hf}h_{t-1}$

Gate $c$
- $W_{xc}x_t$
- $W_{hc}h_{t-1}$

Gate $o$
- $W_{xo}x_t$
- $W_{ho}h_{t-1}$

Gate $e$
- $W_{he}x_t$
- $W_{ho}h_{t-1}$

GRU

Gate $r$
- $W_{xr}x_t$
- $W_{hr}h_{t-1}$

Gate $h$
- $W_{xh}x_t$
- $W_{hh}h'$

Gate $z$
- $W_{xz}x_t$
- $W_{hz}h_{t-1}$

Gate $e$
- $W_{xe}x_t$
- $W_{he}h_{t-1}$

Time-independent
Time-dependent
GraphLSTM

Gate $i$
\[
G_{\text{conv}}(x_t, W_{xi})
\] \[G_{\text{conv}}(h_{t-1}, W_{hi})
\]

Gate $f$
\[
G_{\text{conv}}(x_t, W_{xf})
\] \[G_{\text{conv}}(h_{t-1}, W_{hf})
\]

Gate $c$
\[
G_{\text{conv}}(x_t, W_{xc})
\] \[G_{\text{conv}}(h_{t-1}, W_{hc})
\]

Gate $o$
\[
G_{\text{conv}}(x_t, W_{xo})
\] \[G_{\text{conv}}(h_{t-1}, W_{ho})
\]

GraphGRU

Gate $r$
\[
G_{\text{conv}}(x_t, W_{xr})
\] \[G_{\text{conv}}(h_{t-1}, W_{hr})
\]

Gate $h$
\[
G_{\text{conv}}(x_t, W_{xh})
\] \[G_{\text{conv}}(h', W_{h' h})
\]

Gate $z$
\[
G_{\text{conv}}(x_t, W_{xz})
\] \[G_{\text{conv}}(h_{t-1}, W_{hz})
\]
Challenge #1: Redundant Neighborhood Aggregation

- Two categories of graph convolutions.
  - **Time-independent** graph convolution depends on current representations of nodes.
  - **Time-dependent** graph convolution depends on previous hidden states.

- **Redundancy**: Graph convolutions in the same category perform same neighborhood aggregation.
Challenge #2: Inefficient Distributed Training

- No existing systems for training static GNNs, for example, DGL, support distributed dynamic GNN training in an efficient way.

- Static GNN training:
  - Partitioning both the graph structure and node features across machines.
  - Using data parallelism to train a static GNN.

- Can we partition each snapshot individually?
  - Partitioning and maintaining a large number of snapshots can be **expensive**.
  - The graph structure and the node features in each snapshot may vary.
Cached Message Passing

**Gate i**
- \( m_{xt} \rightarrow W_{xi} \rightarrow g_{xt}^{i} \)
- \( m_{h(t-1)} \rightarrow W_{hi} \rightarrow g_{h(t-1)}^{i} \)

**Gate f**
- \( m_{xt} \rightarrow W_{xf} \rightarrow g_{xt}^{f} \)
- \( m_{h(t-1)} \rightarrow W_{hf} \rightarrow g_{h(t-1)}^{f} \)

**Gate c**
- \( m_{xt} \rightarrow W_{xc} \rightarrow g_{xt}^{c} \)
- \( m_{h(t-1)} \rightarrow W_{hc} \rightarrow g_{h(t-1)}^{c} \)

**Gate o**
- \( m_{xt} \rightarrow W_{xo} \rightarrow g_{xt}^{o} \)
- \( m_{h(t-1)} \rightarrow W_{ho} \rightarrow g_{h(t-1)}^{o} \)

GraphLSTM

**E**

**Time-independent**

**Time-dependent**

**Typical Message Passing Paradigm of GNN:**

\[
m_{u \rightarrow v}^{l} = M^{l}(h_{v}^{l-1}, h_{u}^{l-1}, e_{u \rightarrow v}^{l-1})
\]

\[
m_{v}^{l} = \sum_{u \in N(v)} m_{u \rightarrow v}^{l}
\]

\[
h_{v}^{l} = U^{l}(h_{v}^{l-1}, m_{v}^{l})
\]
Cached Message Passing

Typical Message Passing Paradigm of GNN:

\[ m^l_{u \rightarrow v} = M^l(h^{l-1}_v, h^{l-1}_u, e^{l-1}_{u \rightarrow v}) \]

\[ m^l_v = \sum_{u \in N(v)} m^l_{u \rightarrow v} \]

\[ h^l_v = U^l(h^{l-1}_v, m^l_v) \]

The results after the message passing can be reused for all graph convolution in the same category.
Cached Message Passing

• Dynamic graphs are often trained using sequence-to-sequence models in a sliding-window fashion.

![Diagram of GraphRNN layers and sequence]

Teacher States (Ground Truth)
Cached Message Passing

- Dynamic graphs are often trained using sequence-to-sequence models in a sliding-window fashion.

![GraphRNN Diagram]

`GraphRNN`
Dynamic graphs are often trained using sequence-to-sequence models in a sliding-window fashion.

Neighborhood aggregation has already been performed in previous sequence(s)!
Cached Message Passing

GraphLSTM

Gate $i$

$\begin{align*}
W_{xi} & \rightarrow g_{xt}^i \\
m_{xt} & \\
W_{hi} & \rightarrow g_{ht-1}^i \\
m_{ht-1} & \\
\end{align*}$

Gate $f$

$\begin{align*}
W_{xf} & \rightarrow g_{xt}^f \\
m_{xt} & \\
W_{hf} & \rightarrow g_{ht-1}^f \\
m_{ht-1} & \\
\end{align*}$

Gate $c$

$\begin{align*}
W_{xc} & \rightarrow g_{xt}^c \\
m_{xt} & \\
W_{hc} & \rightarrow g_{ht-1}^c \\
m_{ht-1} & \\
\end{align*}$

Gate $o$

$\begin{align*}
W_{xo} & \rightarrow g_{xt}^o \\
m_{xt} & \\
W_{ho} & \rightarrow g_{ht-1}^o \\
m_{ht-1} & \\
\end{align*}$

$h_t$

Cache Store

Snapshot $t$

Snapshot $t-n$

$\begin{align*}
x_t & \\
h_{t-1} & \\
m_{xt-1} & \\
m_{ht-1} & \\
\end{align*}$
Cached Message Passing

Cache Store

Snapshot $t$

... Snapshot $t-n$

$\begin{align*}
&x_t \\
&h_{t-1}
\end{align*}$

$\begin{align*}
&\text{Msg. Passing} \\
&\text{Msg. Passing}
\end{align*}$

$\begin{align*}
&\text{GraphLSTM} \\
&\text{Gate } i \\
&m_{x_t} \\
&m_{h_{t-1}} \\
&W_{xi} \\
&W_{hi}
\end{align*}$

$\begin{align*}
&g^{i}_{xt} \\
&g^{i}_{h_{t-1}}
\end{align*}$

$\begin{align*}
&\text{Gate } f \\
&m_{x_t} \\
&m_{h_{t-1}} \\
&W_{xf} \\
&W_{hf}
\end{align*}$

$\begin{align*}
&g^{f}_{xt} \\
&g^{f}_{h_{t-1}}
\end{align*}$

$\begin{align*}
&\text{Gate } c \\
&m_{x_t} \\
&m_{h_{t-1}} \\
&W_{xc} \\
&W_{hc}
\end{align*}$

$\begin{align*}
&g^{c}_{xt} \\
&g^{c}_{h_{t-1}}
\end{align*}$

$\begin{align*}
&\text{Gate } o \\
&m_{x_t} \\
&m_{h_{t-1}} \\
&W_{xo} \\
&W_{ho}
\end{align*}$

$\begin{align*}
&g^{o}_{xt} \\
&g^{o}_{h_{t-1}}
\end{align*}$

$\begin{align*}
&E \\
&+ A \\
&+ A \\
&+ A
\end{align*}$

$\begin{align*}
&h_t
\end{align*}$
Cached Message Passing

- Cache Store
  - Snapshot $t$
  - $m_{x_t}$
  - $m_{h_{t-1}}$
  - Snapshot $t-n$
  - $m_{x_{t-n}}$
  - $m_{h_{t-n-1}}$

- PUT
  - $x_t$
  - $h_{t-1}$

- GraphLSTM
  - Gate $i$
    - $m_{x_t}$
    - $W_{xi}$
    - $g_{xt}^i$
    - $m_{h_{t-1}}$
    - $W_{hi}$
    - $g_{h_{t-1}}^i$
  - Gate $f$
    - $m_{x_t}$
    - $W_{xf}$
    - $g_{xt}^f$
    - $m_{h_{t-1}}$
    - $W_{hf}$
    - $g_{h_{t-1}}^f$
  - Gate $c$
    - $m_{x_t}$
    - $W_{xc}$
    - $g_{xt}^c$
    - $m_{h_{t-1}}$
    - $W_{hc}$
    - $g_{h_{t-1}}^c$
  - Gate $o$
    - $m_{x_t}$
    - $W_{xo}$
    - $g_{xt}^o$
    - $m_{h_{t-1}}$
    - $W_{ho}$
    - $g_{h_{t-1}}^o$

- Msg. Passing
  - PUT
  - $+ A$

- $E$
  - $h_t$
Cached Message Passing

Cache Store

Snapshot $t$
- $m_{xt}$
- $m_{ht-1}$

Snapshot $t-n$
- $m_{xt-n}$
- $m_{ht-n-1}$

GraphLSTM

Gate $i$
- $m_{xt}$
- $W_{xi}$
- $g^i_{xt}$

Gate $f$
- $m_{xt}$
- $W_{xf}$
- $g^f_{xt}$

Gate $c$
- $m_{xt}$
- $W_{xc}$
- $g^c_{xt}$

Gate $o$
- $m_{xt}$
- $W_{xo}$
- $g^o_{xt}$

Gate $i$
- $m_{ht-1}$
- $W_{hi}$
- $g^i_{ht-1}$

Gate $f$
- $m_{ht-1}$
- $W_{hf}$
- $g^f_{ht-1}$

Gate $c$
- $m_{ht-1}$
- $W_{hc}$
- $g^c_{ht-1}$

Gate $o$
- $m_{ht-1}$
- $W_{ho}$
- $g^o_{ht-1}$

PUT
- $m_{xt}$
- $m_{ht-1}$

GET
- $m_{xt}$
- $m_{ht-1}$

$x_t$

$h_{t-1}$

$E$
Cached Message Passing

GraphLSTM

Cache Store

Snapshot $t$

$\mathbf{m}_{x_{t}}$

$\mathbf{m}_{h_{t-1}}$

PUT

PUT

GET

GET

GET

GET

$\mathbf{m}_{x_{t}}$

$\mathbf{m}_{h_{t-1}}$

$\mathbf{W}_x$

$\mathbf{W}_h$

$\mathbf{g}_x$

$\mathbf{g}_h$

Gate $i$

$\mathbf{g}^i_{x_{t}}$

$\mathbf{g}^i_{h_{t-1}}$

$\mathbf{x}_{t}$

$\mathbf{h}_{t-1}$

Msg. Passing

Msg. Passing

Gate $f$

$\mathbf{g}^f_{x_{t}}$

$\mathbf{g}^f_{h_{t-1}}$

GET

GET

$\mathbf{m}_{x_{t}}$

$\mathbf{m}_{h_{t-1}}$

$\mathbf{W}_x$

$\mathbf{W}_h$

$\mathbf{g}_x$

$\mathbf{g}_h$

Gate $c$

$\mathbf{g}^c_{x_{t}}$

$\mathbf{g}^c_{h_{t-1}}$

GET

GET

$\mathbf{m}_{x_{t}}$

$\mathbf{m}_{h_{t-1}}$

$\mathbf{W}_x$

$\mathbf{W}_h$

$\mathbf{g}_x$

$\mathbf{g}_h$

Gate $o$

$\mathbf{g}^o_{x_{t}}$

$\mathbf{g}^o_{h_{t-1}}$

GET

GET

$\mathbf{m}_{x_{t}}$

$\mathbf{m}_{h_{t-1}}$

$\mathbf{W}_x$

$\mathbf{W}_h$

$\mathbf{g}_x$

$\mathbf{g}_h$

$\mathbf{h}_t$

$\mathbf{E}$
Distributed DGNN Training

**Partitioned Snapshots & Input Features**

- Layer 1
- Layer 2
- Layer K

**Sliding Window**

\[ M_1 \]

\[ M_2 \]

\[ M_3 \]

\[ M_4 \]
DynaGraph API

**cache()**  
Cache caller function outputs; do nothing if already cached.

**msg_pass()**  
Computes intermediate message passing results.

**update()**  
Computes output representation from intermediate message passing results.

**integrate()**  
Integrates a GNN into a GraphRNN to create a dynamic GNN.

**stack_seq_model()**  
Stacks dynamic GNN layers to an encoder-decoder structure.
Implementation & Evaluation

• Implemented on Deep Graph Library (DGL) v0.7
• Evaluated using 8 machines, each with 2 NVIDIA Tesla V100 GPUs
  - **METR-LA**: 207 nodes/snapshots, |F|=2, |S|= 34K
  - **PEMS-BAY**: 325 nodes/snapshots, |F|=2, |S|= 52K
  - **METR-LA-Large**: 0.4m nodes/snapshots, |F|=128, |S|= 34k
  - **PEMS-BAY-Large**: 0.7m nodes/snapshots, |F|=128, |S|= 52k
• Several Dynamic GNN architectures
  - GCRN-GRU, GCRN-LSTM [ICONIP ‘18]
  - DCRNN [ICLR ‘18]
DynaGraph Single-Machine Performance

Up to 2.31x Speedup
DynaGraph Distributed Performance

Up to 2.23x Speedup

![Bar chart showing average epoch time comparison between DGL and DynaGraph for different models and datasets.](chart.png)
DynaGraph Scaling

Throughput (snapshots/sec)

# Machines (# GPUs)

DGL

DynaGraph

GCRN-GRU

GCRN-LSTM
Summary

• Supporting dynamic graphs is increasingly important for enabling many GNN applications.
  • Existing GNN systems mainly focus on static graphs and static GNNs.
  • Dynamic GNN architectures combine GNN techniques and temporal embedding techniques like RNNs.

• DynaGraph enables dynamic GNN training at scale.
  • Several techniques to reuse intermediate results.
  • Efficient distributed training.
  • Outperforms state-of-the-art solutions.

Thank you!
Contact: mingyu.guan@gatech.edu