GraphGym: Easy-to-use API for Graph Learning

Collaboration with Rex Ying, Matthias Fey, Jure Leskovec

Jiaxuan You
Stanford University
Overview of This Talk

Graph Learning Basics

Challenges in Applying Graph Learning

GraphGym
Machine Learning Tasks on Graphs

Node-level prediction
“What is the area of this research paper?”

Edge-level prediction
“Is this transaction fraudulent?”

Graph-level prediction
“Is this molecular graph toxic?”
Deep Learning Pipeline for Graphs

- **Benefits:**
  - Fully data-driven, few hand-engineering
  - Utilize rich feature information in graphs
Deep Learning Pipeline for Graphs

Key concepts

Input Graph → Graph Neural Network → Node embeddings → Prediction head (Node/edge/graph) → Predictions (Node/edge/graph) → Loss

Loss

Forward

Backward

J. You, R. Ying, J. Leskovec. Design Space of Graph Neural Networks, NeurIPS 2020
Key Concept: Node Embeddings

- **Intuition:** Map nodes to $d$-dimensional embeddings such that similar nodes in the graph are embedded close together.

- **How to learn the encoder function $ENC(\cdot)$?** **GNN!**

Jiaxuan You, Stanford University
Key Concept: Graph Neural Network (GNN)

- **Goal:** Learn meaningful node embeddings
- **Idea:** Iteratively aggregate information from a node’s neighborhood

Determine node computation graph

Propagate and transform information


Jiaxuan You, Stanford University
Overview of This Talk

Graph Learning Basics

Challenges in Applying Graph Learning

GraphGym

- Implementation
- Finding best designs
- Customization
Challenge 1: Implementing a GNN Pipeline

from torch_geometric.nn import GCNConv

How to manipulate data?

How to design GNN?

How to handle different prediction tasks?

How to evaluate performance?
Challenge 2: Finding Best GNN Designs

- **Observation**: The design space for GNN is huge
  - Easily define millions of GNN models
    - *GNN layer*: GCN, GraphSAGE, GAT, ...
    - *Aggregation*: Mean, Sum, Max,...
    - *Operators*: BatchNorm? Dropout? L2norm?, ...
    - Exponential to the design dimensions (e.g., $3^{10} = 59K$ designs)

- **Challenge**: Best GNN designs for different tasks are very different
  - A SOTA GNN in one task may perform badly in another task

<table>
<thead>
<tr>
<th>GNN Design</th>
<th>SOTA on Task A</th>
<th>SOTA on Task B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 8, 3, skipcat, sum)</td>
<td>0.785</td>
<td>0.736</td>
</tr>
<tr>
<td>(1, 4, 2, skipcat, max)</td>
<td>0.736</td>
<td>0.736</td>
</tr>
</tbody>
</table>
Challenge 3: Customizing a GNN Pipeline

- Suppose you want to apply a new GNN layer: ABConv

- **Common workflows:**
  - Fork an open-source pipeline that uses **ABConv**: Repurpose the pipeline to the dataset/task that you need
    - **Issue**: A new pipeline is needed, whenever you want to try something new
  - Adapt your existing GNN pipeline: Replace GCNConv with ABConv
    - **Issue**: Your core pipeline code needs to be altered, not scalable

- **Common issues**: laborious, inefficient, not scalable, ...
GraphGym Address The Challenges

- **Challenge 1: Implementing a GNN Pipeline**
  - **GraphGym** provides a modularized GNN Pipeline
  - Help users explore a giant GNN design space

- **Challenge 2: Finding Best GNN Designs**
  - **GraphGym** features a simple API for managing GNN experiments
  - Easily launch and analyze thousands of experiments

- **Challenge 3: Customizing a GNN Pipeline**
  - **GraphGym** allows users to easily register customized modules
  - Increase visibility and impact of SOTA research
Overview of This Talk

Graph Learning Basics

Challenges in Applying Graph Learning

- Implementation
- Finding best designs
- Customization

GraphGym

Graph Neural Network

Nodes embeddings

Prediction head
(Node/edge/graph)

Predictions
(Node/edge/graph)

Loss

Labels

PyG

Jiaxuan You, Stanford University
GraphGym: Modularized GNN Pipeline

create_loader() create_model() create_optimizer() compute_loss()
GraphGym: GNN Design Space

- **Core**: A GNN design space
- **Main design dimensions**
  - Intra-layer design
  - Inter-layer design
  - Learning configuration
  - 315K possible designs
- **Many other design dimension are available in GraphGym**
GraphGym: Experiment Configuration

- Each experiment is **fully described by a configuration file**
  - You can **always reproduce** the experiment, by keeping this config file

```python
python main.py --cfg example_node.yaml --repeat 3
```

---

Dataset

Training

Model

Optimizer

And more!

---

Jiaxuan You, Stanford University
GraphGym: Launching a Batch of Experiments

- **Grid search** over experimental settings
  ```python
  # (1) dataset configurations
  dataset.format format ['PyG']
  dataset.name dataset ['TU_ENZYMES', 'TU_PROTEINS']
  dataset.task task ['graph']
  # (2) model configurations
  gnn.layers_pre_mp l_pre [1, 2]
  gnn.layers_mp l_mp [2, 4, 6, 8]
  gnn.layers_post_mp l_post [2, 3]
  gnn.stage_type stage ['skipsum', 'skipconcat']
  gnn.agg agg ['add', 'mean', 'max']
  ```
  - Run different datasets
  - Run different models

- **Launching** GNN experiments in parallel (in 2 lines!)
  - Easily scale to **thousands** of experiments (and fully utilize all the GPUs)

  ```bash
  # generate configs
  python configs_gen.py --config example.yaml --grid example.txt --out_dir configs
  # launch batch of experiments
  bash run_batch.sh configs/example_grid_example.txt
  ```
  - Repeat each experiment for 3 random seeds
  - Run 8 experiments simultaneously
GraphGym: Analyze a Batch of Experiments

- Automatically summarize experiment results and figures

Understand the best design: Rank for each design choice

Aggregated performance
GraphGym: Register Customized Modules

Register a new GNN layer to GraphGym

```python
class ExampleConv(nn.Module):
    def __init__(self, dim_in, dim_out, bias=False, **kwargs):
        super(ExampleConv, self).__init__()
        self.model = ExampleConvLayer(dim_in, dim_out, bias=bias)

    def forward(self, batch):
        batch.x = self.model(batch.x, batch.edge_index)
        return batch

register_layer('exampleconv', ExampleConv)
```

Supported customized modules:
- Activations
- Customized configurations
- Feature augmentations
- Feature encoders
- GNN heads
- GNN layers
- Data loaders
- Loss functions
- GNN network architectures
- Optimizers
- GNN global pooling layers
- GNN stages
- GNN training pipelines
- Data transformations

The new ExampleConv layer is ready to be used
Application: GraphGym in Research

- **Goal**: Use GraphGym to validate a newly proposed GNN layer (ID-GNN)
- GraphGym can ensure a **fair comparison**:
  - We **fix all the other the hyperparameters** except switching the GNN Layer
  - We **control the computational budget for all the models** to be the same
- We want to show ID-GNN can consistently outperform GCN
  - 4 GNN backbones * 20 node/edge/graph level tasks * 3 random seeds * 2 GNN layers (ID-GNN vs. GCN) = 480 experiments
- With GraphGym, launching 480 experiments takes 3 lines of code!

```python
1 # generate configs
2 python configs_gen.py --config baseline.yaml --config_budget baseline.yaml --grid idgnn.txt
3 # run batch of configs
4 # Args: config_dir, num of repeats, max jobs running, sleep time
5 bash run_batch.sh configs/baseline_grid_idgnn 3 10 1
6 #aggregate results for the batch
7 python agg_batch.py --dir results/baseline_grid_idgnn
```

Run in 10 parallel processes, only taking a few hours to finish
GraphGym in Domain Applications

- **GraphGym in domain applications:** financial predictions
  - Apply GNN to transaction networks with millions of nodes and edges
  - Knowledge of GraphGym design space + implementation
- Task 1: Fraud detection
  - Naïve GNN implementation: AUC 0.64
  - GraphGym implementation: AUC 0.94+
- Task 2: Future transaction prediction
  - Naïve GNN implementation: AUC 0.68
  - GraphGym implementation: AUC 0.88+

Jiaxuan You, Stanford University
GraphGym: History and Future

- **GraphGym was originally proposed in 2020**
  - **Paper:** Design Space for Graph Neural Networks, NeurIPS 2020 spotlight
  - **Code:** Originally released at [https://github.com/snap-stanford/GraphGym](https://github.com/snap-stanford/GraphGym)

- **GraphGym has become a core component of PyG 2.0**
  - `pip install torch_geometric`
  - `git clone https://github.com/pyg-team/pytorch_geometric.git`
  - `bash graphgym/run_single.sh`  # run a single GNN experiment
  - `bash graphgym/run_batch.sh`  # run a batch of GNN experiments

- Excited to be a core PyG team member
- We are continuously working on better and deeper integration with PyG
Summary of This Talk

Graph Learning Basics

Challenges in Applying Graph Learning
- Implementation
- Finding best designs
- Customization

GraphGym

Paper: Design Space for Graph Neural Networks (NeurIPS 2020 spotlight)
Try it out: git clone https://github.com/pyg-team/pytorch_geometric.git