PyG
Progress and Future

Matthias Fey
matthias@pyg.org
https://pyg.org
/pyg-team/pytorch-geometric

conda install pyg -c pyg
Graph Neural Networks

Message Passing Scheme

✓ Generalization of any neural network architecture
✓ Data-dependent computation

A new paradigm of how we define neural networks!

From CNNs to GNNs
Message Passing via continuous kernels

From Transformers to GNNs
Message Passing within a fully-connected graph
Implementing Graph Neural Networks is challenging

- **Sparsity and irregularity** of the underlying data
  How can we effectively parallelize irregular data of potentially varying size?

- **Heterogeneity** of the underlying data
  numerical, categorical, image and text features, potentially over different types of data

- **Inherently dynamic**
  it is hard to find scenarios in which graphs will not change over time

- **Various different requests on scalability**
  sparse vs. dense graphs, many small vs. single giant graphs, ...

- **Applicability to a set of diverse tasks**
  node-level vs. link-level vs. graph-level, clustering, pre-training, self-supervision, ...
PyTorch Geometric

PyG (PyTorch Geometric) is the PyTorch library to unify deep learning on graph-structured data

✓ simplifies implementing and working with Graph Neural Networks
✓ bundles state-of-the-art GNN architectures and training procedures
✓ achieves high GPU throughput on sparse data of varying size
✓ suited for both academia and industry
   flexible, comprehensive, easy-to-use
Design Principles

Graph-based Neural Network Building Blocks
- Message Passing layers
- Normalization layers
- Pooling & Readout layers
- ...

In-Memory Graph Storage, Datasets & Loaders
- Support for heterogeneous graphs
- 200+ benchmark datasets
- 10+ sampling techniques

Graph Transformations & Augmentations
- Graph iffusion
- Missing feature value imputation
- Mesh and Point Cloud support

Examples & Tutorials
- Learn practically about GNNs
- Videos, Colabs & Blogs
- Application-driven Graph ML Tutorials

Stanford CS224W Graph ML Tutorials
Design Principles

PyG is highly modular

✓ Access to 200+ datasets and 50+ transforms
✓ Access to a variety of mini-batch loaders
  Node-wise sampling, Subgraph-wise sampling, graph-wise batching
✓ Access to 80+ GNN layers, normalizations and readouts as neural network building blocks
  SAGEConv, GCNConv, GATConv, GINConv, PNAConv, ...

  and

20+ pre-defined models
  GraphSAGE, GCN, GAT, GIN, PNA, SchNet, DimeNet, ...

✓ Access to regular PyTorch loss functions and training routines
  Classification, Regression, Self-Supervision, ...
  Node-level, Link-level, Graph-level
Design Principles

✓ PyG is framework-specific
allows us to make use of recently released features right away
TorchScript for deployment, torch.fx for model transformations

✓ PyG keeps design principles close to vanilla PyTorch
If you are familiar with PyTorch, you already know most of PyG

✓ PyG fits nicely into the PyTorch ecosystem
Scaling up models via PyTorch Lightning
Explaining models via Captum
Ecosystem

The 🐍PyG ecosystem

... and many more!
Timeline

**Fast Graph Representation Learning with PyTorch Geometric**

- **Paper Release**: March'19
- **Open-Sourced**: Nov'17
- **Collaboration**: Feb'20
- **Stanford Partnership**
- **Heterogeneous GNNs**
- **Principled Aggregations**
- **Scalable Link Prediction**
- **Temporal Samplers**
- **Partnership Acquisition**: '21
- **Kumo.AI Partnership**
- **NVIDIA Partnership**
- **Intel Partnership**

- **PyG 2.0 Release**: Sep'21
- **PyG 2.1 Release**: Aug'22
- **PyG 2.2 Release**: Nov'22
- **Accelerations**
- **Scalability**
- **...**
Major Architecture Change

A new GNN engine: pyg-lib
Joint effort across many different partners

New Optimizations

- Improved GNN design via principled aggregations
- Improved scalability and pluggable graph backend support
Major Architecture Change

A new GNN engine: pyg-lib
Joint effort across many different partners

New Optimizations

- Improved GNN design via principled aggregations
- Improved scalability and pluggable graph backend support
Accelerating PyTorch Geometric

**pyg-lib**: A unified GNN engine for optimized low-level graph routines

- Joint effort of Kumo, NVIDIA, Intel & PyTorch
- Accelerating graph sampling routines
- Accelerating heterogeneous GNNs
- Accelerating sparse aggregations
- Speed-ups with no line of code change

/pyg-team/pyg-lib
Accelerating Heterogeneous GNNs

PyG 2.0 integrated heterogeneous graph and GNN support

✓ HeteroData: in-memory storage
✓ Metapath transformations
✓ Heterogeneous graph samplers
✓ Heterogeneous GNN layers
✓ Lazy initialization to elegantly support feature dimensions of varying size
✓ to_hetero(): A principled way to bring recent advancements of GNNs to heterogeneous graphs right away

Duplicate message passing modules

GNN layer for edge type 1
GNN layer for edge type 2
GNN layer for edge type 3

Group output by destination type
to_hetero() is a powerful tool but lacks parallelism across edge types

pyg-lib supports concurrent type-dependent transformations via NVIDIA CUTLASS integration

✓ Flexible to implement most heterogeneous GNN operators with

✓ Efficient, even on sparse types or on a large number of types

Find out more in the Accelerating PyG with NVIDIA GPUs talk later!
pyg-lib leverages a variety of techniques to further accelerate neighbor sampling routines

- Pre-allocation of random numbers
- Vector-based mapping of nodes for smaller node types
- Faster hashmap implementation
- 10x to 15x speed-ups

Find out more in the Accelerating PyG with Intel CPUs talk later!
Announcements

Major Architecture Change

A new GNN engine: pyg-lib
Joint effort across many different partners

New Optimizations

Improved GNN design via principled aggregations

Improved scalability and pluggable graph backend support
Principled Aggregations

Choice of neighborhood aggregation is a central topic in Graph ML research.

- **Input**
- **sum - multiset**
- **mean - distribution**
- **max - set**

**Xu et al.: How Powerful Are Graph Neural Networks?**

**Corso et al.: Principal Neighborhood Aggregation for Graph Nets**

---

**Permutation Invariant Aggregators**

- **SoftMaxSum**
- **Mean**
- **Max**
- **Min**

**PowerMeanSum**

**PowerMean**

**ogbn-proteins**

- **ogbn-products**

**Li et al.: Deeper-GCN: All You Need to Train Deeper GCNs**

**Performance**

- **Number of layers**
- **PowerMean**
- **Max**
- **Mean**
- **Sum**

---

18
Principled Aggregations

PyG makes the concept of aggregations a first-class principle

- Access to all kinds of simple, advanced, learnable and exotic aggregations
  Median, Softmax, Attention, LSTM, ...

- Fully-customize and combine aggregations within MessagePassing or for global pooling

- Aggregations will pick up the best format to accelerate computation
  scatter reductions, degree bucketing, ...

- Further optimization via fusion possible (TBD)
Principled Aggregations

The *different flavors* of implementing aggregations

Gather & Scatter
- very flexible 😊
- fast for sparse graphs 😊
- memory-inefficient 😭

Sparse MatMul
- less flexible 😫
- very fast 😊
- memory-efficient 😋

Degree Bucketing
- any aggregation 😊
- memory-inefficient 😭
- padding/seq. iteration 😭

Individual Kernel
- not flexible at all 😫
- memory-efficient 😊
- very fast 😊

PyG >= 0.1
PyG >= 1.6
PyG >= 2.1
PyG >= 2.2
Announcements

Major Architecture Change

A new GNN engine: pyg-\textit{lib}
Joint effort across many different partners

New Optimizations

Improved GNN design via principled aggregations

Improved scalability and pluggable graph backend support
PyG simplifies implementing scalable link prediction tasks

Separation between message passing edges `edge_index` and supervision edges `edge_label_index`

Only minor changes required to auto-scale your link prediction model

Sampler creates a unified subgraph by sampling from both endpoints

```
data = Reddit(root_dir)
train_data, _, _ = RandomLinkSplit(data)
train_loader = LinkNeighborLoader(
    train_data, num_neighbors=[25, 10])
for train_data in train_loader:
    ...
    h = model.encode(train_data.x, train_data.edge_index)
pred = model.decode(h, train_data.edge_label_index)
    loss = criterion(pred, train_data.edge_label) 
    ...
```
Previously, PyG was limited to single-node in-memory datasets.

With PyG, we aim to support any backend by providing FeatureStore and GraphStore abstractions:

- Disentangles feature fetching from graph sampling routines.
- Allows for distributed server/client architectures.
- Allows for out-of-memory backends, e.g., via memory-mapped I/O or by connecting to graph databases.

Find out more in the Scaling-up PyG talk later!
**Automatic Mixed Precision**

```python
with torch.amp.autocast():
    out = model(data.x, data.edge_index)
```

**Temporal Graph Samplers**

```python
loader = NeighborLoader(
    data, num_neighbors=[25, 10], time=...)
```

**Explainability**

- Explain predictions across *any* GNN model, dataset, and task
  - *out-of-the-box*

- **Captum**

**Model Milestones**

- ✓ Deep GNNs with 1000+ layers
  - Li *et al.*: Training Graph Neural Networks with 1000 Layers

- ✓ GNNs on heterophily graphs
  - Lim *et al.*: Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods

- ✓ ... and *many more!*
**Conclusion**

**PyG** bundles the *state-of-the-art* in Graph Representation Learning

- 80+ GNN architectures
- 200+ benchmark datasets
- 50+ graph transformations
- *Dedicated* sparsity-aware CUDA kernels
- Multi-GPU support
- Support for scalability techniques
- Heterogeneous graph support
- GNN Design Space Exploration

We are constantly encouraged to make **PyG** even better!

---

https://pyg.org

[github](https://github.com/pyg-team/pytorch-geometric)

**team@pyg.org**

conda install pyg -c pyg