Scalable GNN Updates: More About PyTorch Geometric (PyG)

May 1st, 2019

Derrick Blakely

University of Virginia

https://qdata.github.io/deep2Read/

Table of contents

Introduction

Motivation

- GPUs work well on dense, repetitive data matrices where a single instruction can be applied to lots of data at once (SIMD)
- Graphs have irregular structures
- Adjacency matrix rows are sparse and can require control flow
- Samples vary in size
- PyTorch doesn't have a programming interface for GNNs

Goal

- Provide sparse GPU acceleration by creating CUDA kernels for COO-format matrices
- Make mini-batching simple
- Create a PyTorch programming interface
- Give users a library of pre-implemented GNN algorithms

Overview

Notation

Graphs:

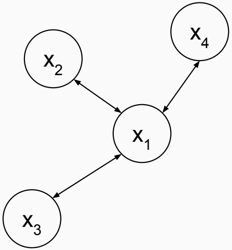
- $\mathcal{G} = (\mathsf{X}, (\mathsf{I}, \mathsf{E}))$ with $\mathsf{X} \in \mathbb{R}^{N \times F}$ and sparse adjacency tuple (I, E)
- $\mathsf{I} \in \mathbb{N}^{2 \times E}$ encodes edge indices in coordinate (COO) format
- $\mathsf{E} \in \mathbb{R}^{E \times D}$ holds D-dimensional edge features

Generalized neighborhood $\mathcal{N}$ aggregation:

$$x_i^{k+1} = \gamma^{k+1}\left(x_i^k, \square_{j \in \mathcal{N}(i)} \phi^{k+1}\left(x_i^k, x_j^k, e_{i,j}\right)\right)$$

- $\square_{j \in \mathcal{N}(i)}$: a differentiable permutation invariant function (e.g., summation, mean, etc)
- γ and ϕ: differentiable functions (e.g., MLPs)

Working Example



Sparse Adjacency Matrix **A**

0	1	1	1
1	0	0	0
1	0	0	0
1	0	0	0

→

Edge Indices **I**

Row	Col
0	1
0	2
0	3
1	0
2	0
3	0

Edge Features **E**

1
1
1
1
1
1

Neighborhood Aggregation as Message Passing

Implement a GCN algo by subclassing the MessagePassing by defining:

- aggregate $= \square$
- message $= \phi^{(k)}()$
- udpate $= \gamma^{(k)}()$

MessagePassing Interface

MessagePassing class performs:

- Gathers neighbors
- Computes messages (using above definition)
- Aggregates messages (using above)
- Scatters messages
- Computes updates

Computing a GNN Layer with Message Passing

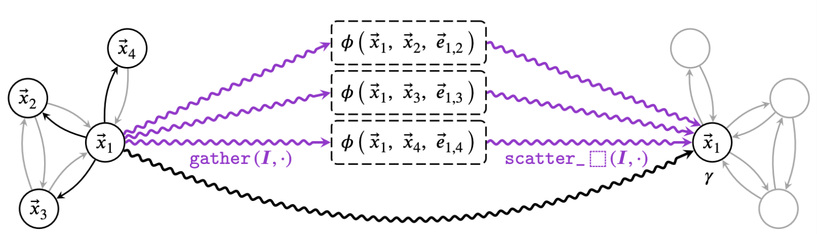


Figure 1: *Computing a GNN layer using gather and scatter methods based on edge indices* I. *Alternates between node-parallel and edge-parallel space.*

Extending the MessagePassing class

```python
class GCNConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super(GCNConv, self).__init__(aggr='add')  # "Add" aggregation.
        self.lin = torch.nn.Linear(in_channels, out_channels)

    def forward(self, x, edge_index):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix.
        edge_index = add_self_loops(edge_index, num_nodes=x.size(0))

        # Step 2: Linearly transform node feature matrix.
        x = self.lin(x)

        # Step 3-5: Start propagating messages.
        return self.propagate(edge_index, size=(x.size(0), x.size(0)), x=x)

    def message(self, x_j, edge_index, size):
        # x_j has shape [E, out_channels]

        # Step 3: Normalize node features.
        row, col = edge_index
        deg = degree(row, size[0], dtype=x_j.dtype)
        deg_inv_sqrt = deg.pow(-0.5)
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        return norm.view(-1, 1) * x_j

    def update(self, aggr_out):
        # aggr_out has shape [N, out_channels]

        # Step 5: Return new node embeddings.
        return aggr_out
```

Parallelization

- Their Gather-Apply-Scatter is *not* about parallelization; it's about the programming interface
- Simple extension of PyTorch's DataParallel class
- Only supports data-parallelism (no graph-parallelism)
- Doesn't help if graphs are too large to fit in GPU memory

```python
from itertools import chain
import torch
from torch_geometric.data import Batch

class DataParallel(torch.nn.DataParallel):
    def __init__(self, module, device_ids=None, output_device=None):
        super(DataParallel, self).__init__(module, device_ids, output_device)
        self.src_device = torch.device("cuda:{}".format(self.device_ids[0]))

    def forward(self, data_list):...


    def scatter(self, data_list, device_ids):...
```

Conclusions

Strengths

- MessagePassing programming interface is very intuitive
- Achieves GPU acceleration with sparse CUDA kernels over COO matrices

Weaknesses

- Assumes entire graph can fit in memory
- Doesn't partition graphs
- Doesn't introduce computation graph optimizations (like NGra or
- Only supports data-parallelism, which only works for graph classification and doesn't work for large graphs

Lessons Learned

- The programming interfaces of NGra and PyG are pretty much the same idea
- NGra is more cutting edge because it partitions graphs to avoid OOM and uses ring-based streaming multi-GPU support
- Gather-Apply-Scatter has lots of potential for GNNs
- Still opportunity for accelerating GNNs with smart computation graph partitioning and making improving parallelization