Chapter 5: Training Graph Neural Networks

(中文版)

Overview

This chapter discusses how to train a graph neural network for node classification, edge classification, link prediction, and graph classification for small graph(s), by message passing methods introduced in Chapter 2: Message Passing and neural network modules introduced in Chapter 3: Building GNN Modules.

This chapter assumes that your graph as well as all of its node and edge features can fit into GPU; see Chapter 6: Stochastic Training on Large Graphs if they cannot.

The following text assumes that the graph(s) and node/edge features are already prepared. If you plan to use the dataset DGL provides or other compatible `DGLDataset` as is described in Chapter 4: Graph Data Pipeline, you can get the graph for a single-graph dataset with something like

```
import dgl

dataset = dgl.data.CiteseerGraphDataset()
graph = dataset[0]
```

Note: In this chapter we will use PyTorch as backend.

Heterogeneous Graphs

Sometimes you would like to work on heterogeneous graphs. Here we take a synthetic heterogeneous graph as an example for demonstrating node classification, edge classification, and link prediction tasks.

The synthetic heterogeneous graph `hetero_graph` has these edge types:

- `('user', 'follow', 'user')`
- `('user', 'followed-by', 'user')`
- `('user', 'click', 'item')`
- `('item', 'clicked-by', 'user')`
- `('user', 'dislike', 'item')`

- `('item', 'disliked-by', 'user')`

```
import numpy as np
import torch

n_users = 1000
n_items = 500
n_follows = 3000
n_clicks = 5000
n_dislikes = 500
n_hetero_features = 10
n_user_classes = 5
n_max_clicks = 10

follow_src = np.random.randint(0, n_users, n_follows)
follow_dst = np.random.randint(0, n_users, n_follows)
click_src = np.random.randint(0, n_users, n_clicks)
click_dst = np.random.randint(0, n_items, n_clicks)
dislike_src = np.random.randint(0, n_users, n_dislikes)
dislike_dst = np.random.randint(0, n_items, n_dislikes)

hetero_graph = dgl.heterograph({
    ('user', 'follow', 'user'): (follow_src, follow_dst),
    ('user', 'followed-by', 'user'): (follow_dst, follow_src),
    ('user', 'click', 'item'): (click_src, click_dst),
    ('item', 'clicked-by', 'user'): (click_dst, click_src),
    ('user', 'dislike', 'item'): (dislike_src, dislike_dst),
    ('item', 'disliked-by', 'user'): (dislike_dst, dislike_src)})

hetero_graph.nodes['user'].data['feature'] = torch.randn(n_users, n_hetero_features)
hetero_graph.nodes['item'].data['feature'] = torch.randn(n_items, n_hetero_features)
hetero_graph.nodes['user'].data['label'] = torch.randint(0, n_user_classes, (n_users,))
hetero_graph.edges['click'].data['label'] = torch.randint(1, n_max_clicks, (n_clicks,)).float()
# randomly generate training masks on user nodes and click edges
hetero_graph.nodes['user'].data['train_mask'] = torch.zeros(n_users,
dtype=torch.bool).bernoulli(0.6)
hetero_graph.edges['click'].data['train_mask'] = torch.zeros(n_clicks,
dtype=torch.bool).bernoulli(0.6)
```

Roadmap

The chapter has four sections, each for one type of graph learning tasks.

- 5.1 Node Classification/Regression
- 5.2 Edge Classification/Regression
- 5.3 Link Prediction
- 5.4 Graph Classification
- 5.5 Use of Edge Weights