Note

Click here to download the full example code

Make Your Own Dataset

This tutorial assumes that you already know the basics of training a GNN for node classification and how to create, load, and store a DGL graph.

By the end of this tutorial, you will be able to

- Create your own graph dataset for node classification, link prediction, or graph classification.

(Time estimate: 15 minutes)

`DGLDataset` Object Overview

Your custom graph dataset should inherit the `dgl.data.DGLDataset` class and implement the following methods:

- `__getitem__(self, i)`: retrieve the `i`-th example of the dataset. An example often contains a single DGL graph, and occasionally its label.
- `__len__(self)`: the number of examples in the dataset.
- `process(self)`: load and process raw data from disk.

Creating a Dataset for Node Classification or Link Prediction from CSV

A node classification dataset often consists of a single graph, as well as its node and edge features.

This tutorial takes a small dataset based on Zachary's Karate Club network. It contains

- A `members.csv` file containing the attributes of all members, as well as their attributes.
- An `interactions.csv` file containing the pair-wise interactions between two club members.

```
import urllib.request

import pandas as pd

urllib.request.urlretrieve(
    "https://data.dgl.ai/tutorial/dataset/members.csv", "./members.csv"
)
urllib.request.urlretrieve(
    "https://data.dgl.ai/tutorial/dataset/interactions.csv",
    "./interactions.csv",
)

members = pd.read_csv("./members.csv")
members.head()

interactions = pd.read_csv("./interactions.csv")
interactions.head()
```

This tutorial treats the members as nodes and interactions as edges. It takes age as a numeric feature of the nodes, affiliated club as the label of the nodes, and edge weight as a numeric feature of the edges.

Note

The original Zachary's Karate Club network does not have member ages. The ages in this tutorial are generated synthetically for demonstrating how to add node features into the graph for dataset creation.

Note

In practice, taking age directly as a numeric feature may not work well in machine learning; strategies like binning or normalizing the feature would work better. This tutorial directly takes the values as-is for simplicity.

```
import os

os.environ["DGLBACKEND"] = "pytorch"
import dgl
import torch
from dgl.data import DGLDataset


class KarateClubDataset(DGLDataset):
    def __init__(self):
        super().__init__(name="karate_club")

    def process(self):
        nodes_data = pd.read_csv("./members.csv")
        edges_data = pd.read_csv("./interactions.csv")
        node_features = torch.from_numpy(nodes_data["Age"].to_numpy())
        node_labels = torch.from_numpy(
            nodes_data["Club"].astype("category").cat.codes.to_numpy()
        )
        edge_features = torch.from_numpy(edges_data["Weight"].to_numpy())
        edges_src = torch.from_numpy(edges_data["Src"].to_numpy())
        edges_dst = torch.from_numpy(edges_data["Dst"].to_numpy())

        self.graph = dgl.graph(
            (edges_src, edges_dst), num_nodes=nodes_data.shape[0]
        )
        self.graph.ndata["feat"] = node_features
        self.graph.ndata["label"] = node_labels
        self.graph.edata["weight"] = edge_features

        # If your dataset is a node classification dataset, you will need to assign
        # masks indicating whether a node belongs to training, validation, and test set.
        n_nodes = nodes_data.shape[0]
        n_train = int(n_nodes * 0.6)
        n_val = int(n_nodes * 0.2)
        train_mask = torch.zeros(n_nodes, dtype=torch.bool)
        val_mask = torch.zeros(n_nodes, dtype=torch.bool)
        test_mask = torch.zeros(n_nodes, dtype=torch.bool)
        train_mask[:n_train] = True
        val_mask[n_train : n_train + n_val] = True
        test_mask[n_train + n_val :] = True
        self.graph.ndata["train_mask"] = train_mask
        self.graph.ndata["val_mask"] = val_mask
        self.graph.ndata["test_mask"] = test_mask

    def __getitem__(self, i):
        return self.graph

    def __len__(self):
        return 1


dataset = KarateClubDataset()
graph = dataset[0]

print(graph)
```

Out:

```
/home/ubuntu/prod-
doc/readthedocs.org/user_builds/dgl/checkouts/1.1.x/tutorials/blitz/6_load_data.py:107:
UserWarning: The given NumPy array is not writable, and PyTorch does not support non-writable
tensors. This means writing to this tensor will result in undefined behavior. You may want to
copy the array to protect its data or make it writable before converting it to a tensor. This
type of warning will be suppressed for the rest of this program. (Triggered internally at
../torch/csrc/utils/tensor_numpy.cpp:199.)
  nodes_data["Club"].astype("category").cat.codes.to_numpy()
Graph(num_nodes=34, num_edges=156,
      ndata_schemes={'feat': Scheme(shape=(), dtype=torch.int64), 'label': Scheme(shape=(),
dtype=torch.int8), 'train_mask': Scheme(shape=(), dtype=torch.bool), 'val_mask': Scheme(shape=
(), dtype=torch.bool), 'test_mask': Scheme(shape=(), dtype=torch.bool)}
      edata_schemes={'weight': Scheme(shape=(), dtype=torch.float64)})
```

Since a link prediction dataset only involves a single graph, preparing a link prediction dataset will have the same experience as preparing a node classification dataset.

Creating a Dataset for Graph Classification from CSV

Creating a graph classification dataset involves implementing `__getitem__` to return both the graph and its graph-level label.

This tutorial demonstrates how to create a graph classification dataset with the following synthetic CSV data:

- `graph_edges.csv`: containing three columns:
  - `graph_id`: the ID of the graph.
  - `src`: the source node of an edge of the given graph.
  - `dst`: the destination node of an edge of the given graph.
- `graph_properties.csv`: containing three columns:
  - `graph_id`: the ID of the graph.
  - `label`: the label of the graph.
  - `num_nodes`: the number of nodes in the graph.

```
urllib.request.urlretrieve(
    "https://data.dgl.ai/tutorial/dataset/graph_edges.csv", "./graph_edges.csv"
)
urllib.request.urlretrieve(
    "https://data.dgl.ai/tutorial/dataset/graph_properties.csv",
    "./graph_properties.csv",
)
edges = pd.read_csv("./graph_edges.csv")
properties = pd.read_csv("./graph_properties.csv")

edges.head()

properties.head()


class SyntheticDataset(DGLDataset):
    def __init__(self):
        super().__init__(name="synthetic")

    def process(self):
        edges = pd.read_csv("./graph_edges.csv")
        properties = pd.read_csv("./graph_properties.csv")
        self.graphs = []
        self.labels = []

        # Create a graph for each graph ID from the edges table.
        # First process the properties table into two dictionaries with graph IDs as keys.
        # The label and number of nodes are values.
        label_dict = {}
        num_nodes_dict = {}
        for _, row in properties.iterrows():
            label_dict[row["graph_id"]] = row["label"]
            num_nodes_dict[row["graph_id"]] = row["num_nodes"]

        # For the edges, first group the table by graph IDs.
        edges_group = edges.groupby("graph_id")

        # For each graph ID...
        for graph_id in edges_group.groups:
            # Find the edges as well as the number of nodes and its label.
            edges_of_id = edges_group.get_group(graph_id)
            src = edges_of_id["src"].to_numpy()
            dst = edges_of_id["dst"].to_numpy()
            num_nodes = num_nodes_dict[graph_id]
            label = label_dict[graph_id]

            # Create a graph and add it to the list of graphs and labels.
            g = dgl.graph((src, dst), num_nodes=num_nodes)
            self.graphs.append(g)
            self.labels.append(label)

        # Convert the label list to tensor for saving.
        self.labels = torch.LongTensor(self.labels)

    def __getitem__(self, i):
        return self.graphs[i], self.labels[i]

    def __len__(self):
        return len(self.graphs)


dataset = SyntheticDataset()
```

```
graph, label = dataset[0]
print(graph, label)
```

Out:

```
Graph(num_nodes=15, num_edges=45,
      ndata_schemes={}
      edata_schemes={}) tensor(0)
```

Creating Dataset from CSV via `CSVDataset`

The previous examples describe how to create a dataset from CSV files step-by-step. DGL also provides a utility class `CSVDataset` for reading and parsing data from CSV files. See 4.6 Loading data from CSV files for more details.

```
# Thumbnail credits: (Un)common Use Cases for Graph Databases, Michal Bachman
# sphinx_gallery_thumbnail_path = '_static/blitz_6_load_data.png'
```

Total running time of the script: (0 minutes 0.571 seconds)

Download Python source code: 6_load_data.py

Download Jupyter notebook: 6_load_data.ipynb