

1.4 Creating Graphs from External Sources

(中文版)

The options to construct a `DGLGraph` from external sources include:

- [Conversion from external python libraries for graphs and sparse matrices](#) (NetworkX and SciPy).
- [Loading graphs from disk.](#)

The section does not cover functions that generate graphs by transforming from other graphs. See the API reference manual for an overview of them.

Creating Graphs from External Libraries

The following code snippet is an example for creating a graph from a SciPy sparse matrix and a NetworkX graph.

```
>>> import dgl
>>> import torch as th
>>> import scipy.sparse as sp
>>> spmat = sp.rand(100, 100, density=0.05) # 5% nonzero entries
>>> dgl.from_scipy(spmat) # from SciPy
Graph(num_nodes=100, num_edges=500,
      ndata_schemes={},
      edata_schemes={})

>>> import networkx as nx
>>> nx_g = nx.path_graph(5) # a chain 0-1-2-3-4
>>> dgl.from_networkx(nx_g) # from networkx
Graph(num_nodes=5, num_edges=8,
      ndata_schemes={},
      edata_schemes={})
```

Note that when constructing from the `nx.path_graph(5)`, the resulting `DGLGraph` has 8 edges instead of 4. This is because `nx.path_graph(5)` constructs an undirected NetworkX graph `networkx.Graph` while a `DGLGraph` is always directed. [In converting an undirected NetworkX graph into a `DGLGraph`](#), DGL internally converts undirected edges to two directed edges. Using directed NetworkX graphs `networkx.DiGraph` can avoid such behavior.

```
>>> nxg = nx.DiGraph([(2, 1), (1, 2), (2, 3), (0, 0)])
>>> dgl.from_networkx(nxg)
Graph(num_nodes=4, num_edges=4,
      ndata_schemes={},
      edata_schemes={})
```

! Note

DGL internally converts SciPy matrices and NetworkX graphs to tensors to construct graphs. Hence, these construction methods are not meant for performance critical parts.

See APIs: `dgl.from_scipy()`, `dgl.from_networkx()`.

Loading Graphs from Disk

There are many data formats for storing graphs and it isn't possible to enumerate every option. Thus, this section only gives some general pointers on certain common ones.

Comma Separated Values (CSV)

One very common format is CSV, which stores nodes, edges, and their features in a tabular format:

nodes.csv

age, title

43, 1

23, 3

...

edges.csv

src, dst, weight

0, 1, 0.4

0, 3, 0.9

...

There are known Python libraries (e.g. pandas) for loading this type of data into python objects (e.g., `numpy.ndarray`), which can then be used to construct a DGLGraph. If the backend framework also provides utilities to save/load tensors from disk (e.g., `torch.save()`, `torch.load()`), one can follow the same principle to build a graph.

See also: [Tutorial for loading a Karate Club Network from edge pairs CSV](#).

JSON/GML Format

Though not particularly fast, NetworkX provides many utilities to parse [a variety of data formats](#) which indirectly allows DGL to create graphs from these sources.

DGL Binary Format

DGL provides APIs to save and load graphs from disk stored in binary format. Apart from the graph structure, the APIs also handle feature data and graph-level label data. DGL also supports checkpointing graphs directly to S3 or HDFS. The reference manual provides more details about the usage.

See APIs: `dgl.save_graphs()`, `dgl.load_graphs()`.