Data handling in PyG

Giovanni Pellegrini[1,2,3]

SML[1] Lab, University of Trento, Italy
TIM[2]
EIT DIGITAL[3]

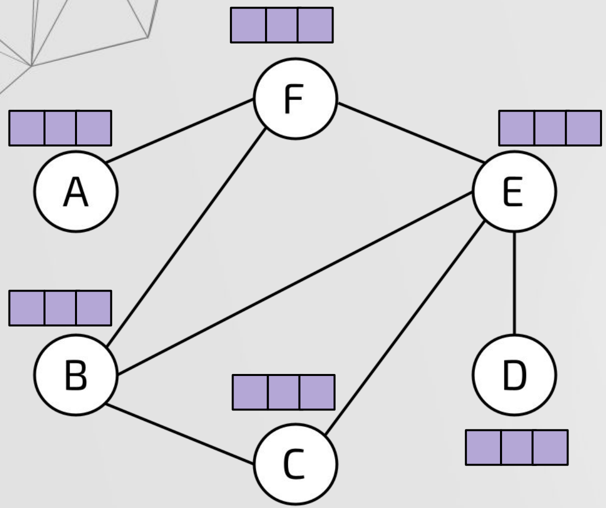
01 Prediction Tasks on Graphs

In our tutorials we mainly covered 4 tasks on graphs:

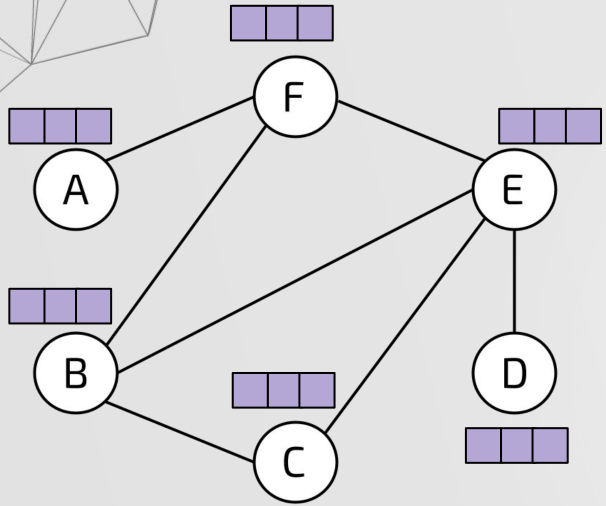
- Node prediction
- Graph prediction
- Edge prediction (property)
- Edge prediction (link between two nodes)

01 Prediction Tasks on Graphs

Node prediction

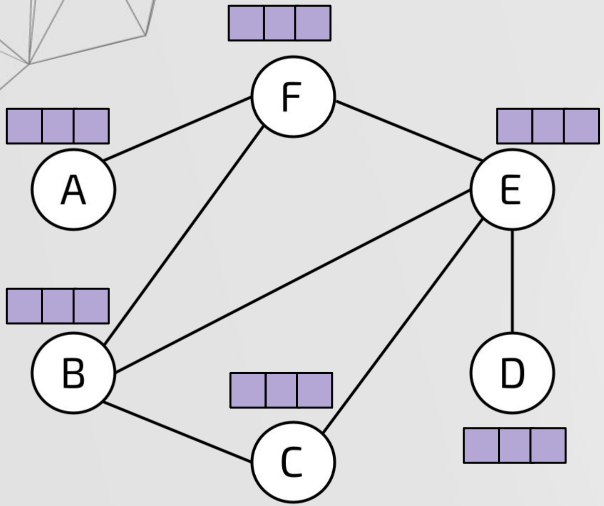


Node prediction

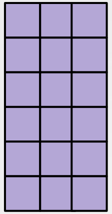


$$\mathbf{X}^{t+1} = \mathrm{GNN}(\mathbf{W}^t, \mathbf{X}^t, \mathbf{A}) \qquad t = 1, \ldots, k$$

Node prediction



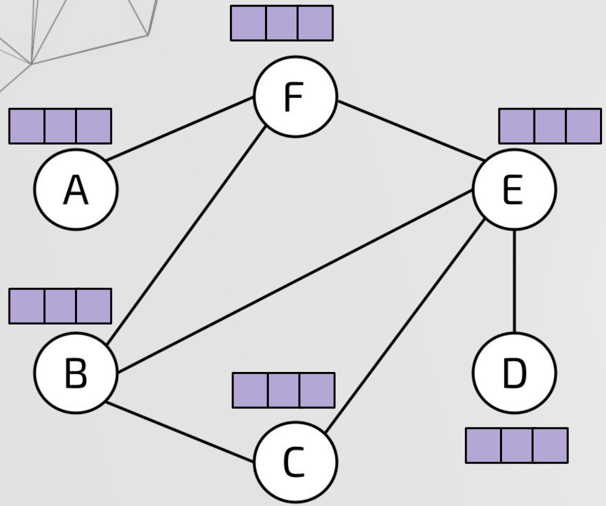
$\mathbf{X}^k$

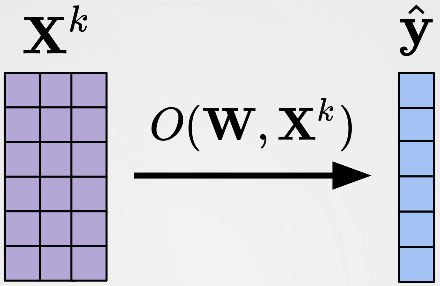


$$\mathbf{X}^{t+1} = \mathrm{GNN}(\mathbf{W}^t, \mathbf{X}^t, \mathbf{A}) \qquad t = 1, \ldots, k$$

01 Prediction Tasks on Graphs

Node prediction



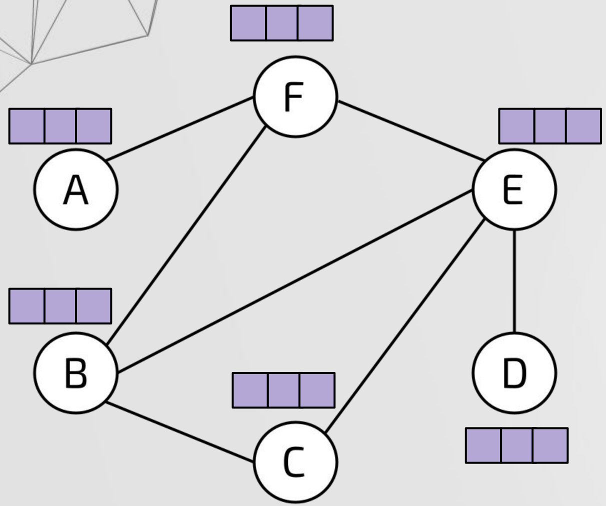


$$\mathbf{X}^{t+1} = \mathrm{GNN}(\mathbf{W}^t, \mathbf{X}^t, \mathbf{A}) \qquad t = 1, \ldots, k$$

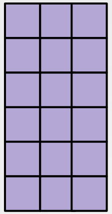
$O(\mathbf{W}, \mathbf{X}^k)$ $\longrightarrow$ node readout function

01 Prediction Tasks on Graphs

Graph prediction



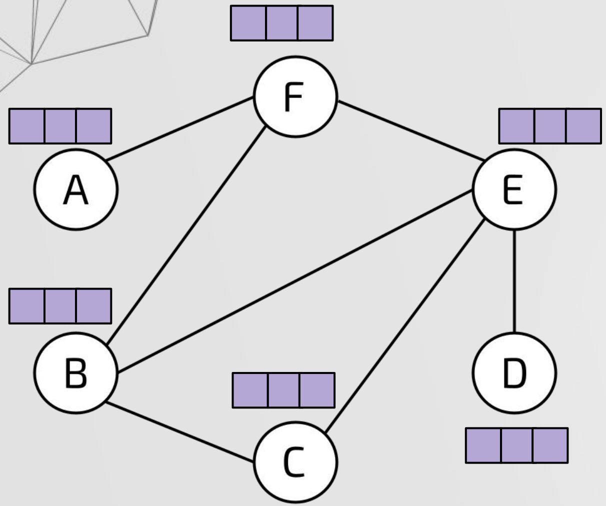
$$\mathbf{X}^k$$

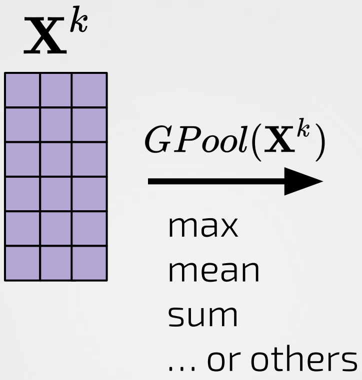


$$\mathbf{X}^{t+1} = \mathrm{GNN}(\mathbf{W}^t, \mathbf{X}^t, \mathbf{A}) \qquad t = 1, \ldots, k$$

01 Prediction Tasks on Graphs

Graph prediction



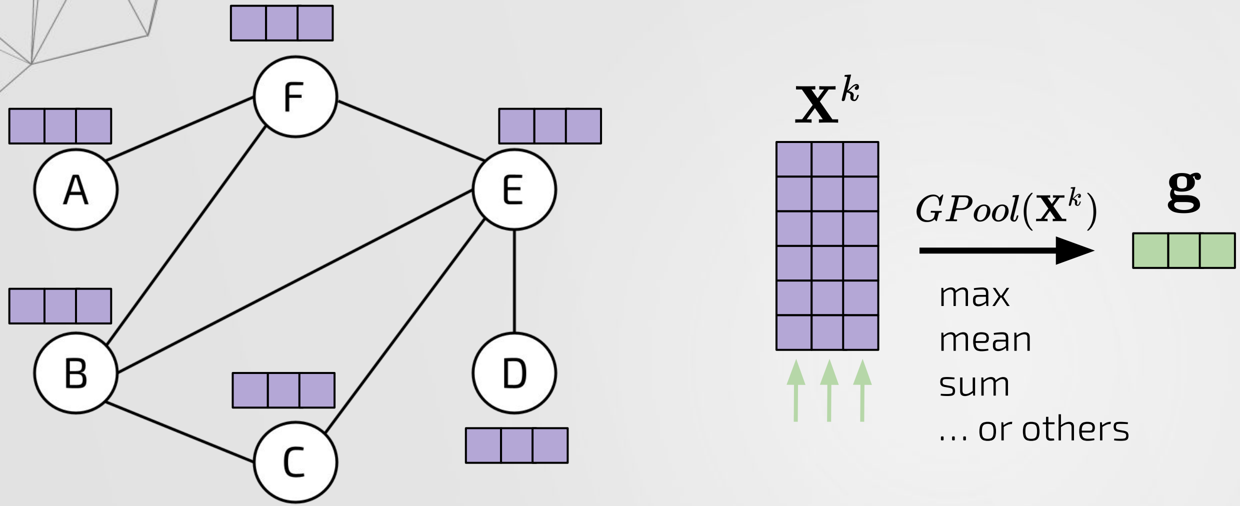


$$\mathbf{X}^{t+1} = \mathrm{GNN}(\mathbf{W}^t, \mathbf{X}^t, \mathbf{A}) \qquad t = 1, \ldots, k$$

$GPool(\mathbf{X}^k) \longrightarrow$ global pooling function

01 Prediction Tasks on Graphs

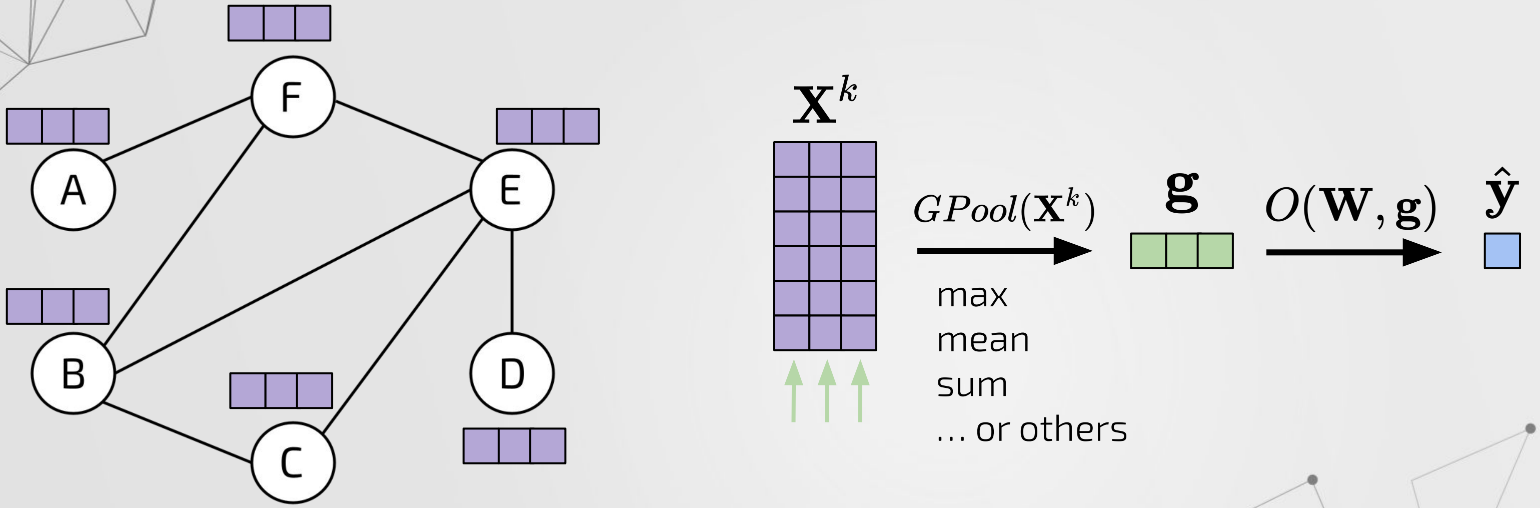
Graph prediction



$$\mathbf{X}^{t+1} = \mathrm{GNN}(\mathbf{W}^t, \mathbf{X}^t, \mathbf{A}) \qquad t = 1, \ldots, k$$

$GPool(\mathbf{X}^k) \longrightarrow$ global pooling function

Graph prediction

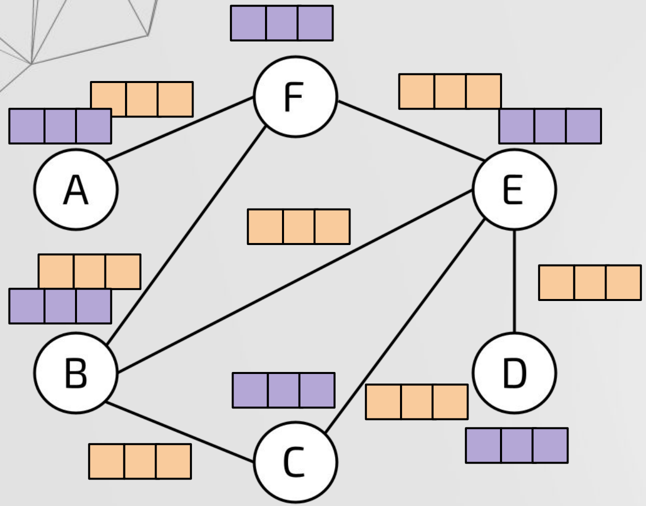


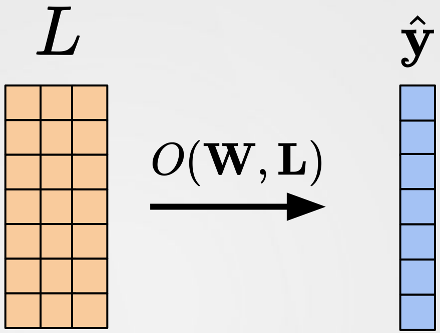
$$\mathbf{X}^{t+1} = \mathrm{GNN}(\mathbf{W}^t, \mathbf{X}^t, \mathbf{A}) \qquad t = 1, \ldots, k$$

$GPool(\mathbf{X}^k) \longrightarrow$ global pooling function

$O(\mathbf{W}, \mathbf{g}) \longrightarrow$ graph readout function

Edge prediction (property)



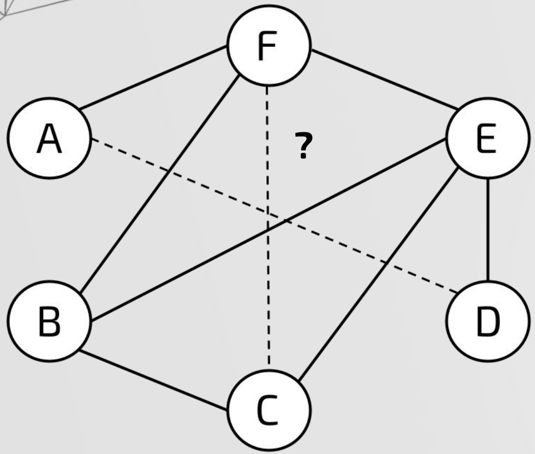


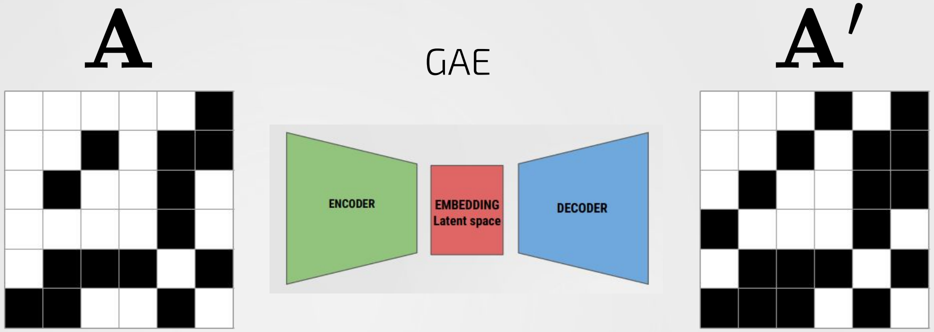
$$\mathbf{l}_{u,v} = \frac{1}{2}(\mathbf{x}_u^k, \mathbf{x}_v^k)$$

$O(\mathbf{W}, \mathbf{L})$ ⟶ edge readout function

01 Prediction Tasks on Graphs

Edge prediction (link)





02 Data submodule

Two modules for data handling:

- **torch_geometric.Data** -> classes and methods for creating and managing (collection of) graphs
- **torch_geometric.Datasets** -> module with a collection of datasets

02 Data submodule

torch_geometric.data.data.Data : base class for representing a graph

CLASS `Data` (**x**=None, **edge_index**=None, **edge_attr**=None, **y**=None, **pos**=None, **normal**=None, **face**=None, **kwargs) [source]

A plain old python object modeling a single graph with various (optional) attributes:

PARAMETERS:

- **x** (*Tensor, optional*) – Node feature matrix with shape `[num_nodes, num_node_features]`. (default: `None`)
- **edge_index** (*LongTensor, optional*) – Graph connectivity in COO format with shape `[2, num_edges]`. (default: `None`)
- **edge_attr** (*Tensor, optional*) – Edge feature matrix with shape `[num_edges, num_edge_features]`. (default: `None`)
- **y** (*Tensor, optional*) – Graph or node targets with arbitrary shape. (default: `None`)
- **pos** (*Tensor, optional*) – Node position matrix with shape `[num_nodes, num_dimensions]`. (default: `None`)
- **normal** (*Tensor, optional*) – Normal vector matrix with shape `[num_nodes, num_dimensions]`. (default: `None`)
- **face** (*LongTensor, optional*) – Face adjacency matrix with shape `[3, num_faces]`. (default: `None`)

02 Data submodule

torch_geometric.data.batch.Batch : data object that represents a collection of graphs

CLASS `Batch` (batch=None, ptr=None, **kwargs) [source]

A plain old python object modeling a batch of graphs as one big (disconnected) graph. With `torch_geometric.data.Data` being the base class, all its methods can also be used here. In addition, single graphs can be reconstructed via the assignment vector `batch`, which maps each node to its respective graph identifier.

CLASSMETHOD `from_data_list` (data_list, follow_batch=[], exclude_keys=[]) [source]

Constructs a batch object from a python list holding `torch_geometric.data.Data` objects. The assignment vector `batch` is created on the fly. Additionally, creates assignment batch vectors for each key in `follow_batch`. Will exclude any keys given in `exclude_keys`.

PROPERTY `num_graphs`

Returns the number of graphs in the batch.

`to_data_list` ( ) → List[torch_geometric.data.data.Data] [source]

Reconstructs the list of `torch_geometric.data.Data` objects from the batch object. The batch object must have been created via `from_data_list()` in order to be able to reconstruct the initial objects.

02 Data submodule

torch_geometric.data.cluster.ClusterData & torch_geometric.data.cluster.ClusterLoader : group nodes into smaller subgraphs and load them in batches for faster computation on large graphs

CLASS **ClusterData** (**data**, **num_parts**: int, **recursive**: bool = False, **save_dir**: Optional[str] = None, **log**: bool = True) [source]

Clusters/partitions a graph data object into multiple subgraphs, as motivated by the "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks" paper.

PARAMETERS:

- **data** (*torch_geometric.data.Data*) – The graph data object.
- **num_parts** (*int*) – The number of partitions.
- **recursive** (*bool, optional*) – If set to `True`, will use multilevel recursive bisection instead of multilevel k-way partitioning. (default: `False`)
- **save_dir** (*string, optional*) – If set, will save the partitioned data to the `save_dir` directory for faster re-use. (default: `None`)
- **log** (*bool, optional*) – If set to `False`, will not log any progress. (default: `True`)

CLASS **ClusterLoader** (**cluster_data**, ****kwargs**) [source]

The data loader scheme from the "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks" paper which merges partioned subgraphs and their between-cluster links from a large-scale graph data object to form a mini-batch.

Note

Use `torch_geometric.data.ClusterData` and `torch_geometric.data.ClusterLoader` in conjunction to form mini-batches of clusters. For an example of using Cluster-GCN, see examples/cluster_gcn_reddit.py or examples/cluster_gcn_ppi.py.

02 Data submodule

torch_geometric.data.sampler.NeighborSampler : samples a specific number of nodes in a neighborhood

CLASS `NeighborSampler` (**edge_index**: Union[torch.Tensor, torch_sparse.tensor.SparseTensor], **sizes**: List[int], **node_idx**: Optional[torch.Tensor] = `None`, **num_nodes**: Optional[int] = `None`, **return_e_id**: bool = `True`, **transform**: Optional[Callable] = `None`, ****kwargs**) [source]

PARAMETERS:

- **edge_index** (*Tensor or SparseTensor*) – A `torch.LongTensor` or a `torch_sparse.SparseTensor` that defines the underlying graph connectivity/message passing flow. `edge_index` holds the indices of a (sparse) symmetric adjacency matrix. If `edge_index` is of type `torch.LongTensor`, its shape must be defined as `[2, num_edges]`, where messages from nodes `edge_index[0]` are sent to nodes in `edge_index[1]` (in case `flow="source_to_target"`). If `edge_index` is of type `torch_sparse.SparseTensor`, its sparse indices `(row, col)` should relate to `row = edge_index[1]` and `col = edge_index[0]`. The major difference between both formats is that we need to input the *transposed* sparse adjacency matrix.
- **sizes** (*[int]*) – The number of neighbors to sample for each node in each layer. If set to `sizes[l] = -1`, all neighbors are included in layer `l`.
- **node_idx** (*LongTensor, optional*) – The nodes that should be considered for creating mini-batches. If set to `None`, all nodes will be considered.
- **num_nodes** (*int, optional*) – The number of nodes in the graph. (default: `None`)

03 Datasets submodule

torch_geometric.datasets.Dataset : base class for implementing a dataset

CLASS `Dataset` (root=None, transform=None, pre_transform=None, pre_filter=None) [source]

Dataset base class for creating graph datasets. See here for the accompanying tutorial.

PARAMETERS:

- **root** (*string, optional*) – Root directory where the dataset should be saved. (optional: `None`)
- **transform** (*callable, optional*) – A function/transform that takes in an `torch_geometric.data.Data` object and returns a transformed version. The data object will be transformed before every access. (default: `None`)
- **pre_transform** (*callable, optional*) – A function/transform that takes in an `torch_geometric.data.Data` object and returns a transformed version. The data object will be transformed before being saved to disk. (default: `None`)
- **pre_filter** (*callable, optional*) – A function that takes in an `torch_geometric.data.Data` object and returns a boolean value, indicating whether the data object should be included in the final dataset. (default: `None`)

03 Datasets submodule

Two types of datasets can be implemented, using **Dataset** class or **InMemoryDataset** class (which extends **Dataset**):

InMemoryDataset is a dataset that fits entirely in the memory (RAM), it is loaded once. Four methods need to be implemented:

- **torch_geometric.data.InMemoryDataset.raw_file_names()**: A list of files in the raw_dir which needs to be found in order to skip the download.
- **torch_geometric.data.InMemoryDataset.processed_file_names()**: A list of files in the processed_dir which needs to be found in order to skip the processing.
- **torch_geometric.data.InMemoryDataset.download()**: Downloads raw data into raw_dir.
- **torch_geometric.data.InMemoryDataset.process()**: Processes raw data and saves it into the processed_dir.

Dataset is used also for large datsets, in which data is loaded and stored to files during the computation, methods to be implemented:

- **torch_geometric.data.Dataset.len()**: Returns the number of examples in your dataset.
- **torch_geometric.data.Dataset.get()**: Implements the logic to load a single graph.

03 Datasets submodule

Torch_geometric.transforms: list of functions to perform transformation of graphs data:

'ToSparseTensor',
'ToUndirected',
'Constant',
'Distance',
'Cartesian',
'OneHotDegree',
'TargetIndegree',
'LinearTransformation',
'RandomScale',
'RandomRotate',
'RandomShear',
'NormalizeFeatures',
'AddSelfLoops',
'RemoveIsolatedNodes',

03 Datasets submodule

torch_geometric.data.DataLoader : class for composing batches of graphs in a dataset

CLASS **DataLoader** (dataset, batch_size=1, shuffle=False, follow_batch=[], exclude_keys=[], **kwargs) [source]

Data loader which merges data objects from a `torch_geometric.data.dataset` to a mini-batch.

PARAMETERS:

- **dataset** (*Dataset*) – The dataset from which to load the data.
- **batch_size** (*int*, *optional*) – How many samples per batch to load. (default: `1`)
- **shuffle** (*bool*, *optional*) – If set to `True` , the data will be reshuffled at every epoch. (default: `False`)
- **follow_batch** (*list or tuple*, *optional*) – Creates assignment batch vectors for each key in the list. (default: `[]`)
- **exclude_keys** (*list or tuple*, *optional*) – Will exclude each key in the list. (default: `[]`)
- ****kwargs** (*optional*) – Additional arguments of `torch.utils.data.DataLoader` .