torch_geometric.utils

degree(*index, num_nodes=None, dtype=None*) [source]

Computes the (unweighted) degree of a given one-dimensional index tensor.

Parameters:
- **index** (*LongTensor*) – Index tensor.
- **num_nodes** (*int*, *optional*) – The number of nodes, *i.e.* `max_val + 1` of `index`. (default: `None`)
- **dtype** (`torch.dtype`, optional) – The desired data type of the returned tensor.

Return type: `Tensor`

scatter_(*name, src, index, dim_size=None*) [source]

Aggregates all values from the `src` tensor at the indices specified in the `index` tensor along the first dimension. If multiple indices reference the same location, their contributions are aggregated according to `name` (either `"add"`, `"mean"` or `"max"`).

Parameters:
- **name** (*string*) – The aggregation to use (`"add"`, `"mean"`, `"max"`).
- **src** (*Tensor*) – The source tensor.
- **index** (*LongTensor*) – The indices of elements to scatter.
- **dim_size** (*int*, *optional*) – Automatically create output tensor with size `dim_size` in the first dimension. If set to `None`, a minimal sized output tensor is returned. (default: `None`)

Return type: `Tensor`

softmax(*src, index, num_nodes=None*) [source]

Computes a sparsely evaluated softmax. Given a value tensor `src`, this function first groups those values along the first dimension based on the indices specified in `index`, and then proceeds to compute the softmax individually for each group.

Parameters:
- **src** (*Tensor*) – The source tensor.
- **index** (*LongTensor*) – The indices of elements for applying the softmax.
- **num_nodes** (*int*, *optional*) – The number of nodes, *i.e.* `max_val + 1` of `index`. (default: `None`)

Return type: `Tensor`

---

dropout_adj(*edge_index, edge_attr=None, p=0.5, force_undirected=False, num_nodes=None, training=True*) [source]

Randomly drops edges from the adjacency matrix `(edge_index, edge_attr)` with propability `p` using samples from a Bernoulli distribution.

Parameters:

- **edge_index** (*LongTensor*) – The edge indices.
- **edge_attr** (*Tensor, optional*) – Edge weights or multi-dimensional edge features. (default: `None`)
- **p** (*float, optional*) – Dropout probability. (default: `0.5`)
- **force_undirected** (*bool, optional*) – If set to `True`, will either drop or keep both edges of an undirected edge. (default: `False`)
- **num_nodes** (*int, optional*) – The number of nodes, *i.e.* `max_val + 1` of `edge_index`. (default: `None`)
- **training** (*bool, optional*) – If set to `False`, this operation is a no-op. (default: `True`)

---

is_undirected(*edge_index, num_nodes=None*) [source]

Returns `True` if the graph given by `edge_index` is undirected.

Parameters:

- **edge_index** (*LongTensor*) – The edge indices.
- **num_nodes** (*int, optional*) – The number of nodes, *i.e.* `max_val + 1` of `edge_index`. (default: `None`)

Return type: bool

---

to_undirected(*edge_index, num_nodes=None*) [source]

Converts the graph given by `edge_index` to an undirected graph, so that $(j, i) \in \mathcal{E}$ for every edge $(i, j) \in \mathcal{E}$.

Parameters:

- **edge_index** (*LongTensor*) – The edge indices.
- **num_nodes** (*int, optional*) – The number of nodes, *i.e.* `max_val + 1` of `edge_index`. (default: `None`)

Return type: `LongTensor`

---

contains_self_loops(*edge_index*) [source]

Returns `True` if the graph given by `edge_index` does not contain self-loops.

Parameters: **edge_index** (*LongTensor*) – The edge indices.

Return type: bool

`remove_self_loops`(*edge_index, edge_attr=None*) [source]

Removes every self-loop in the graph given by `edge_index`, so that $(i, i) \notin \mathcal{E}$ for every $i \in \mathcal{V}$.

Parameters:
- **edge_index** (*LongTensor*) – The edge indices.
- **edge_attr** (*Tensor, optional*) – Edge weights or multi-dimensional edge features. (default: `None`)

Return type: (`LongTensor`, `Tensor`)

`add_self_loops`(*edge_index, num_nodes=None*) [source]

Adds a self-loop $(i, i) \in \mathcal{E}$ to every node $i \in \mathcal{V}$ in the graph given by `edge_index`.

Parameters:
- **edge_index** (*LongTensor*) – The edge indices.
- **num_nodes** (*int, optional*) – The number of nodes, *i.e.* `max_val + 1` of `edge_index`. (default: `None`)

Return type: `LongTensor`

`contains_isolated_nodes`(*edge_index, num_nodes=None*) [source]

Returns `True` if the graph given by `edge_index` contains isolated nodes.

Parameters:
- **edge_index** (*LongTensor*) – The edge indices.
- **num_nodes** (*int, optional*) – The number of nodes, *i.e.* `max_val + 1` of `edge_index`. (default: `None`)

Return type: bool

`to_dense_batch`(*x, batch, fill_value=0*) [source]

Given a sparse batch of node features $\mathbf{X} \in \mathbb{R}^{(N_1 + \ldots + N_B) \times F}$ (with N_i indicating the number of nodes in graph i), creates a dense node feature tensor $\mathbf{X} \in \mathbb{R}^{B \times N_{\max} \times F}$ (with $N_{\max} = \max_i^B N_i$). In addition, a second tensor holding $[N_1, \ldots, N_B] \in \mathbb{N}^B$ is returned.

Parameters:

- **x** (*Tensor*) – Node feature matrix $\mathbf{X} \in \mathbb{R}^{(N_1 + \ldots + N_B) \times F}$.
- **batch** (*LongTensor*) – Batch vector $\mathbf{b} \in \{0, \ldots, B-1\}^N$, which assigns each node to a specific example.
- **fill_value** (*float*, *optional*) – The value for invalid entries in the resulting dense output tensor. (default: `0`)

Return type: (`Tensor`, `LongTensor`)

`normalized_cut`(*edge_index, edge_attr, num_nodes=None*) [source]

Computes the normalized cut $\mathbf{e}_{i,j} \cdot \left(\frac{1}{\deg(i)} + \frac{1}{\deg(j)} \right)$ of a weighted graph given by edge indices and edge attributes.

Parameters:

- **edge_index** (*LongTensor*) – The edge indices.
- **edge_attr** (*Tensor*) – Edge weights or multi-dimensional edge features.
- **num_nodes** (*int*, *optional*) – The number of nodes, *i.e.* `max_val + 1` of `edge_index`. (default: `None`)

Return type: `Tensor`

`grid`(*height, width, dtype=None, device=None*) [source]

Returns the edge indices of a two-dimensional grid graph with height `height` and width `width` and its node positions.

Parameters:

- **height** (*int*) – The height of the grid.
- **width** (*int*) – The width of the grid.
- **dtype** (`torch.device`, optional) – The desired data type of the returned position tensor.
- **dtype** – The desired device of the returned tensors.

Return type: (`LongTensor`, `Tensor`)

`dense_to_sparse`(*tensor*) [source]

Converts a dense adjacency matrix to a sparse adjacency matrix defined by edge indices and edge attributes.

Parameters: **tensor** (*Tensor*) – The dense adjacency matrix.

Return type: (`LongTensor`, `Tensor`)

---

sparse_to_dense(*edge_index, edge_attr=None, num_nodes=None*) [source]

Converts a sparse adjacency matrix given by edge indices and edge attributes to a dense adjacency matrix.

Parameters:
- **edge_index** (*LongTensor*) – The edge indices.
- **edge_attr** (*Tensor*) – Edge weights or multi-dimensional edge features.
- **num_nodes** (*int*, *optional*) – The number of nodes, *i.e.* `max_val + 1` of `index`. (default: `None`)

Return type: `Tensor`

---

to_scipy_sparse_matrix(*edge_index, edge_attr=None, num_nodes=None*) [source]

Converts a graph given by edge indices and edge attributes to a scipy sparse matrix.

Parameters:
- **edge_index** (*LongTensor*) – The edge indices.
- **edge_attr** (*Tensor*, *optional*) – Edge weights or multi-dimensional edge features. (default: `None`)
- **num_nodes** (*int*, *optional*) – The number of nodes, *i.e.* `max_val + 1` of `index`. (default: `None`)

---

from_scipy_sparse_matrix(*A*) [source]

“Converts a scipy sparse matrix to edge indices and edge attributes.

Parameters: **A** (*scipy.sparse*) – A sparse matrix.

---

to_networkx(*data, node_attrs=None, edge_attrs=None*) [source]

Converts a data object graph to a networkx graph.

Parameters:
- **data** (*torch_geometric.data.Data*) – The data object.
- **node_attrs** (*iterable of str*, *optional*) – The node attributes to be copied. (default: `None`)
- **edge_attrs** (*iterable of str*, *optional*) – The edge attributes to be copied. (default: `None`)

---

from_networkx(*G*) [source]

“Converts a networkx graph to a data object graph.

Parameters: **G** (*networkx.Graph*) – A networkx graph.

---

one_hot(*src, num_classes=None, dtype=None*) [source]

Converts labels into a one-hot format.

Parameters:
- **src** (*Tensor*) – The labels.
- **num_classes** (*int or list, optional*) – The number of classes. (default: `None`)
- **dtype** (`torch.dtype` , optional) – The desired data type of the returned tensor.

Return type: `Tensor`

---

accuracy(*pred, target*) [source]

Computes the accuracy of correct predictions.

Parameters:
- **pred** (*Tensor*) – The predictions.
- **target** (*Tensor*) – The targets.

Return type: int

---

true_positive(*pred, target, num_classes*) [source]

Computes the number of true positive predictions.

Parameters:
- **pred** (*Tensor*) – The predictions.
- **target** (*Tensor*) – The targets.
- **num_classes** (*int*) – The number of classes.

Return type: `LongTensor`

---

true_negative(*pred, target, num_classes*) [source]

Computes the number of true negative predictions.

Parameters:
- **pred** (*Tensor*) – The predictions.
- **target** (*Tensor*) – The targets.
- **num_classes** (*int*) – The number of classes.

Return type: `LongTensor`

---

false_positive(*pred*, *target*, *num_classes*) [source]

Computes the number of false positive predictions.

Parameters:
- **pred** (*Tensor*) – The predictions.
- **target** (*Tensor*) – The targets.
- **num_classes** (*int*) – The number of classes.

Return type: `LongTensor`

---

false_negative(*pred*, *target*, *num_classes*) [source]

Computes the number of false negative predictions.

Parameters:
- **pred** (*Tensor*) – The predictions.
- **target** (*Tensor*) – The targets.
- **num_classes** (*int*) – The number of classes.

Return type: `LongTensor`

---

precision(*pred*, *target*, *num_classes*) [source]

Computes the precision: $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$.

Parameters:
- **pred** (*Tensor*) – The predictions.
- **target** (*Tensor*) – The targets.
- **num_classes** (*int*) – The number of classes.

Return type: `Tensor`

---

recall(*pred*, *target*, *num_classes*) [source]

Computes the recall: $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$.

Parameters:
- **pred** (*Tensor*) – The predictions.
- **target** (*Tensor*) – The targets.
- **num_classes** (*int*) – The number of classes.

Return type: `Tensor`

`f1_score`(*pred*, *target*, *num_classes*) [source]

Computes the F_1 score: $2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$.

Parameters:

- **pred** (*Tensor*) – The predictions.
- **target** (*Tensor*) – The targets.
- **num_classes** (*int*) – The number of classes.

Return type: `Tensor`