3.1 DGL NN Module Construction Function

(中文版)

The construction function performs the following steps:

1. Set options.
2. Register learnable parameters or submodules.
3. Reset parameters.

```
import torch.nn as nn

from dgl.utils import expand_as_pair

class SAGEConv(nn.Module):
    def __init__(self,
                 in_feats,
                 out_feats,
                 aggregator_type,
                 bias=True,
                 norm=None,
                 activation=None):
        super(SAGEConv, self).__init__()

        self._in_src_feats, self._in_dst_feats = expand_as_pair(in_feats)
        self._out_feats = out_feats
        self._aggre_type = aggregator_type
        self.norm = norm
        self.activation = activation
```

In construction function, one first needs to set the data dimensions. For general PyTorch module, the dimensions are usually input dimension, output dimension and hidden dimensions. For graph neural networks, the input dimension can be split into source node dimension and destination node dimension.

Besides data dimensions, a typical option for graph neural network is aggregation type (`self._aggre_type`). Aggregation type determines how messages on different edges are aggregated for a certain destination node. Commonly used aggregation types include `mean`, `sum`, `max`, `min`. Some modules may apply more complicated aggregation like an `lstm`.

`norm` here is a callable function for feature normalization. In the SAGEConv paper, such normalization can be l2 normalization: $h_v = h_v / \lVert h_v \rVert_2$.

```
# aggregator type: mean, pool, lstm, gcn
if aggregator_type not in ['mean', 'pool', 'lstm', 'gcn']:
    raise KeyError('Aggregator type {} not supported.'.format(aggregator_type))
if aggregator_type == 'pool':
    self.fc_pool = nn.Linear(self._in_src_feats, self._in_src_feats)
if aggregator_type == 'lstm':
    self.lstm = nn.LSTM(self._in_src_feats, self._in_src_feats, batch_first=True)
if aggregator_type in ['mean', 'pool', 'lstm']:
    self.fc_self = nn.Linear(self._in_dst_feats, out_feats, bias=bias)
self.fc_neigh = nn.Linear(self._in_src_feats, out_feats, bias=bias)
self.reset_parameters()
```

Register parameters and submodules. In SAGEConv, submodules vary according to the aggregation type. Those modules are pure PyTorch nn modules like `nn.Linear`, `nn.LSTM`, etc. At the end of construction function, weight initialization is applied by calling `reset_parameters()`.

```
def reset_parameters(self):
    """Reinitialize learnable parameters."""
    gain = nn.init.calculate_gain('relu')
    if self._aggre_type == 'pool':
        nn.init.xavier_uniform_(self.fc_pool.weight, gain=gain)
    if self._aggre_type == 'lstm':
        self.lstm.reset_parameters()
    if self._aggre_type != 'gcn':
        nn.init.xavier_uniform_(self.fc_self.weight, gain=gain)
    nn.init.xavier_uniform_(self.fc_neigh.weight, gain=gain)
```