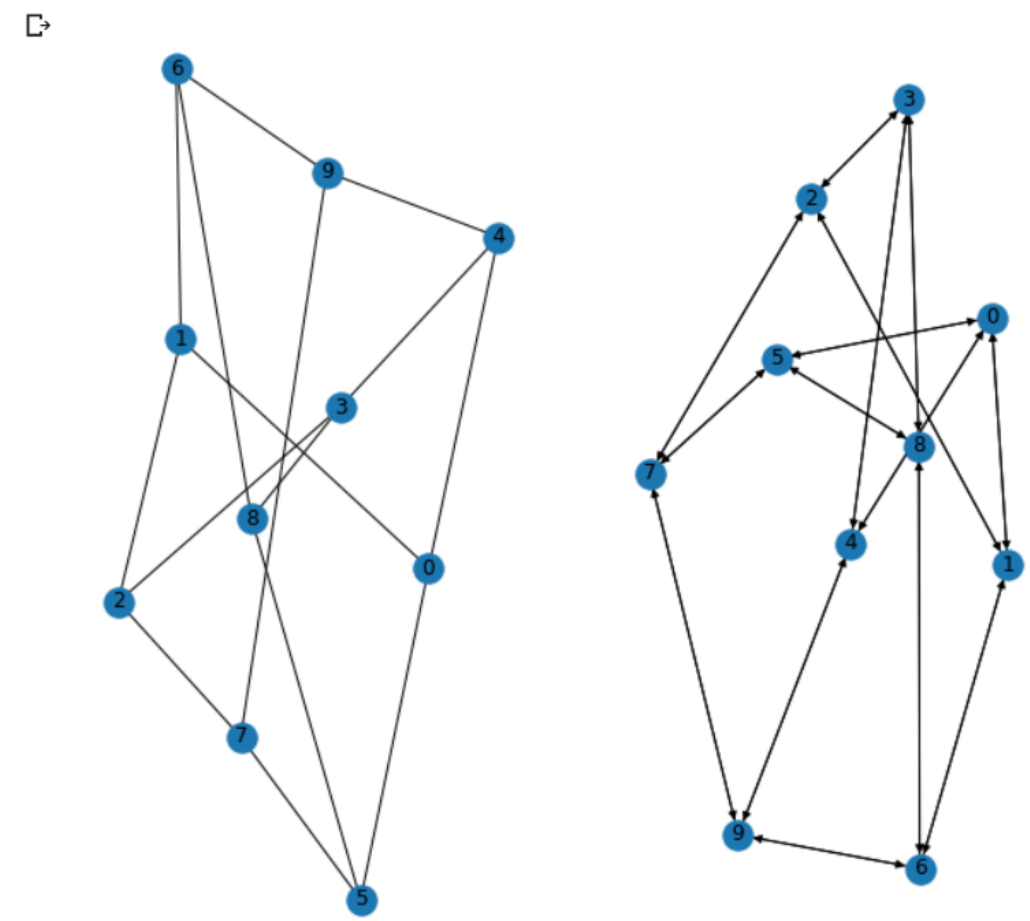
ML Course 2019 : Tutorial-6

Weizmann Institute Of Science

15.05.2019

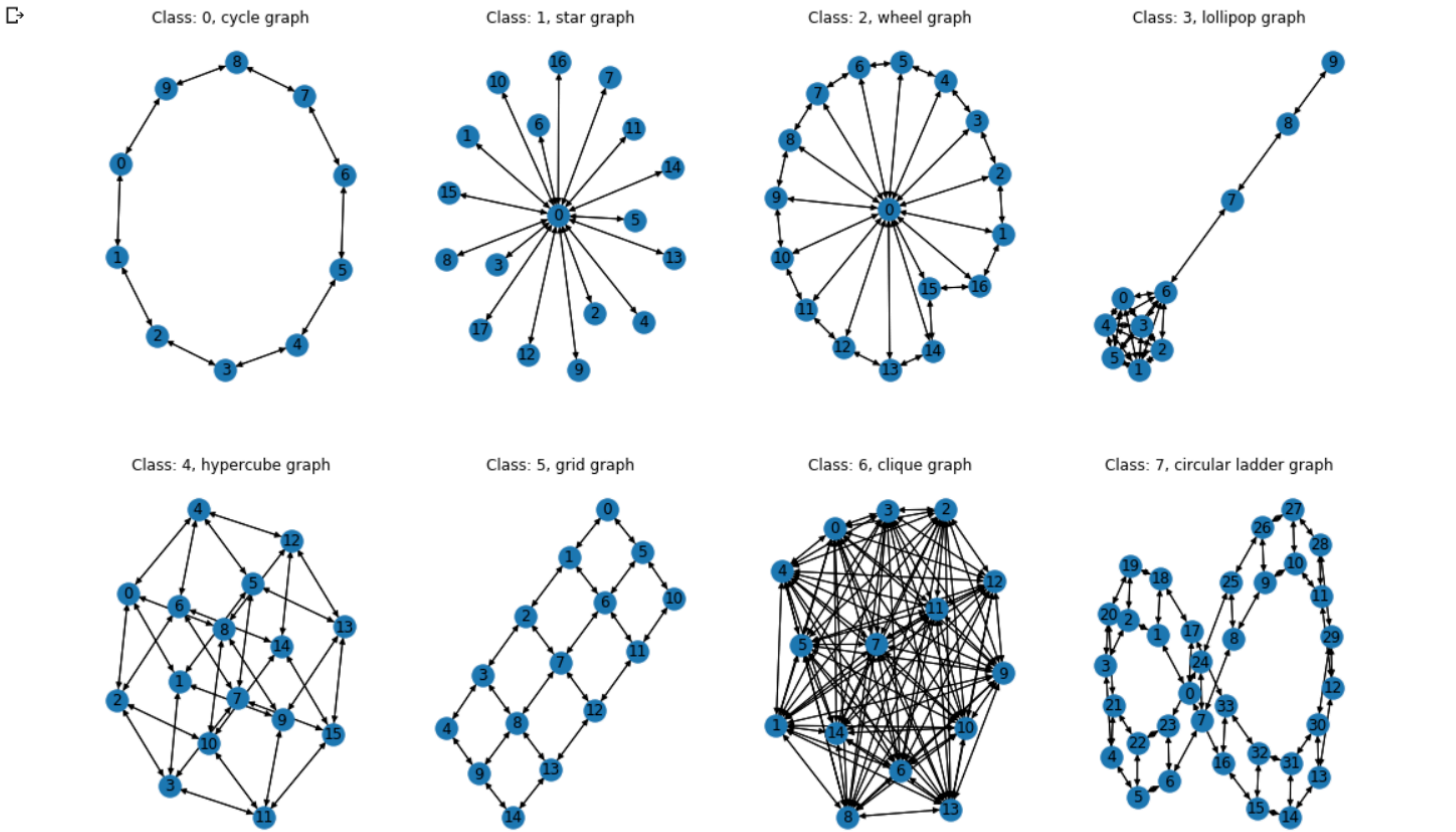
Introduction to graphs


```
[ ] g_nx = nx.petersen_graph() # -- a graph with 10 vertices and 15 edges ----
    g_dgl = dgl.DGLGraph(g_nx)

    import matplotlib.pyplot as plt
    plt.rcParams["figure.figsize"]=10,10
    plt.subplot(121)
    nx.draw(g_nx, with_labels=True)
    plt.subplot(122)
    nx.draw(g_dgl.to_networkx(), with_labels=True)

    plt.show()
```

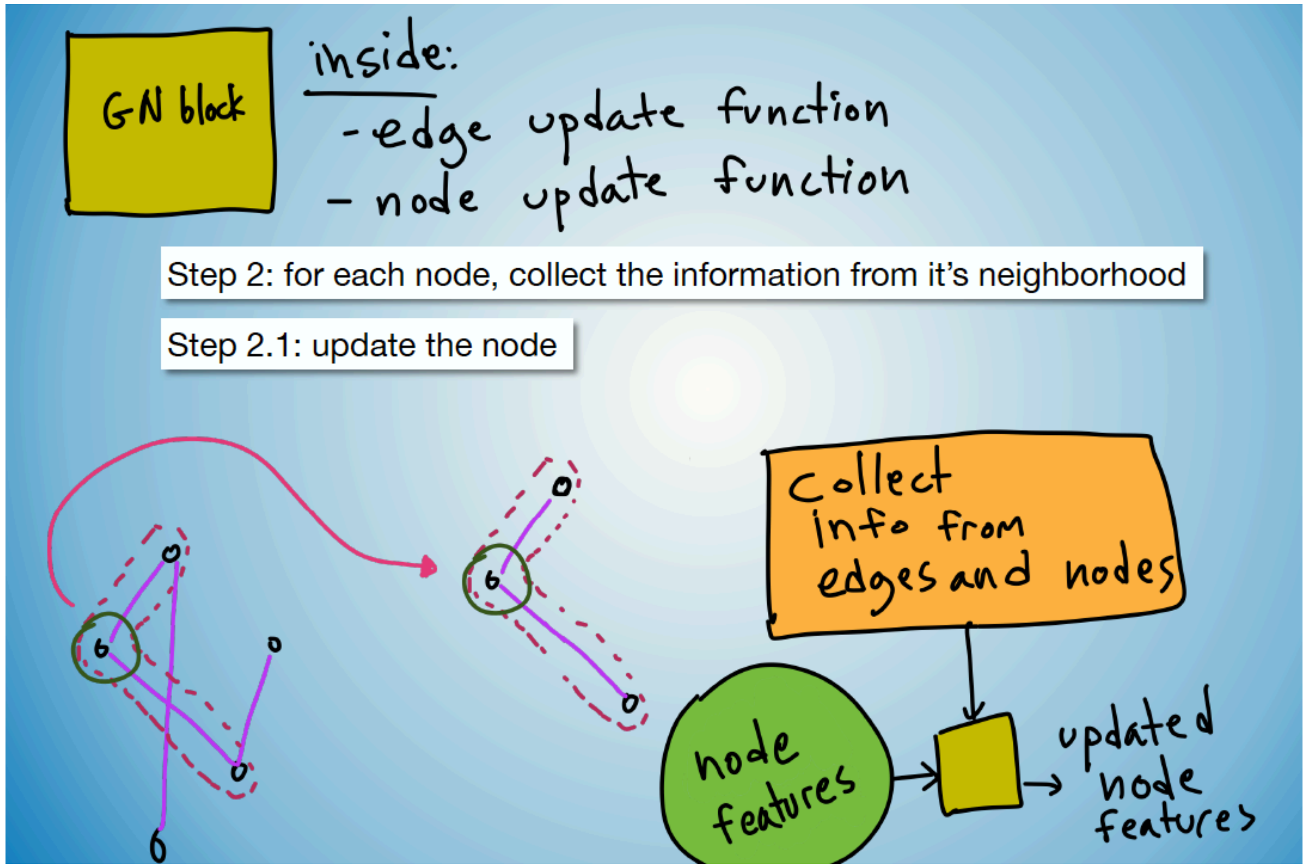


Each edge has a starting and an ending node

Different kind of graphs



Message passing : updating nodes



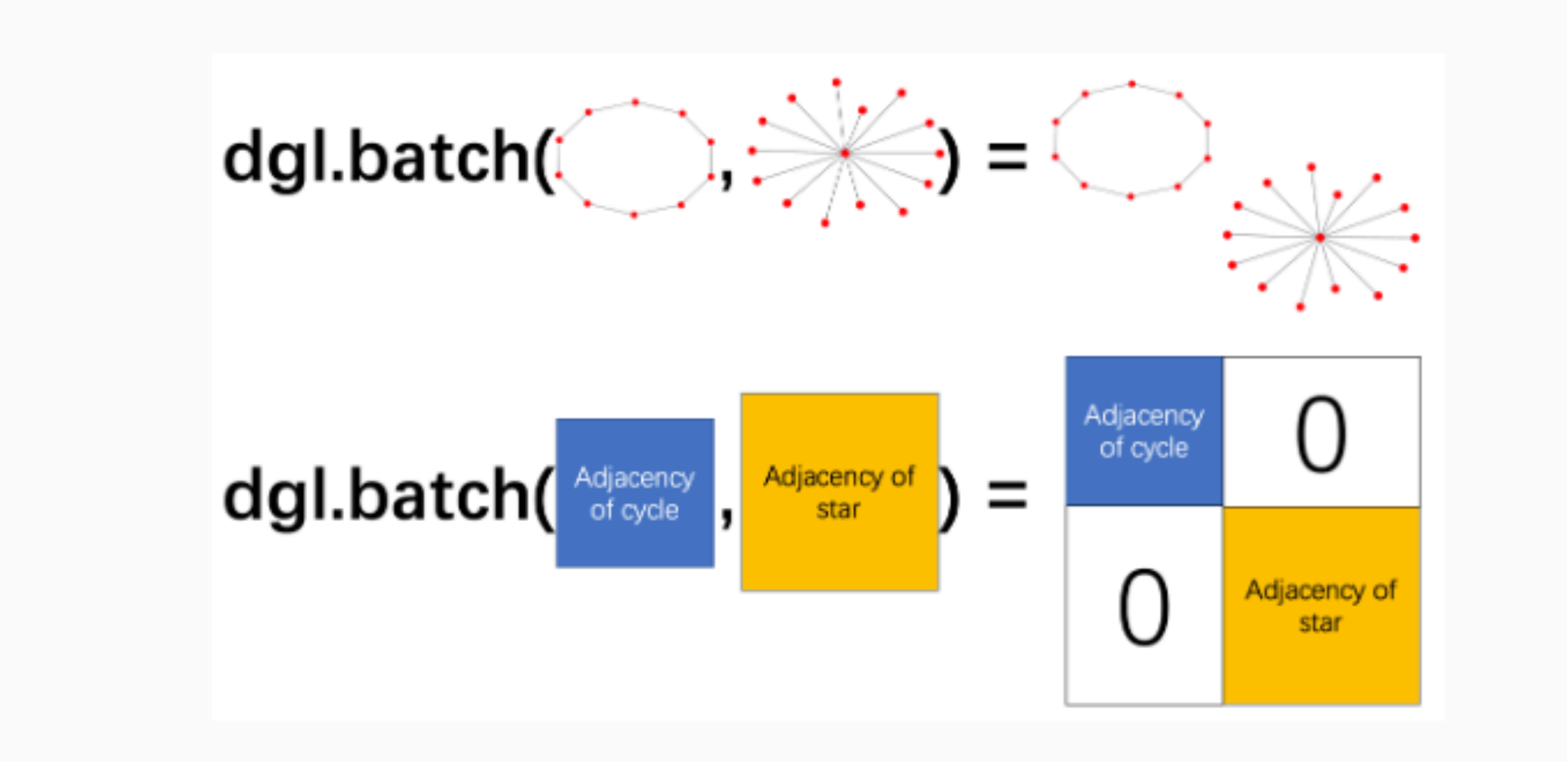

```
[41] def simple_reduce(nodes):

         print(nodes.mailbox['message'])

         mean_over_mailbox = torch.mean(nodes.mailbox['message'].type(torch.FloatTensor), 1)

         return {'new_feature': mean_over_mailbox}


     graph.register_reduce_func(simple_reduce)

     for node_i in range(len(graph.nodes())):
         graph.recv(v=node_i)

     print(graph.ndata)
```

Forming the graph mini-batches

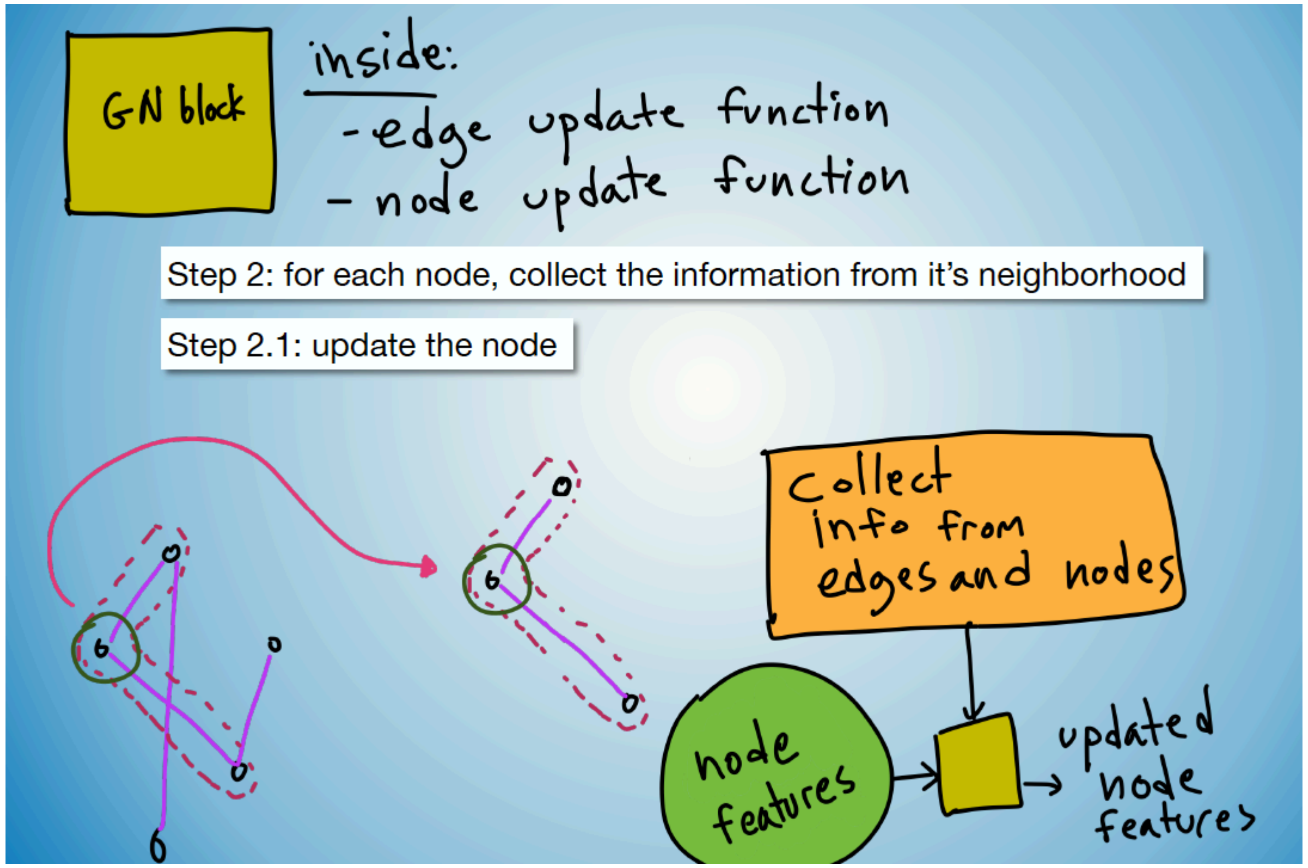
Graphs are sparse objects (different graphs have different shapes) .
We need a collate function to form the batches of graphs

```
[43] def collate(samples):
         # The input `samples` is a list of pairs
         #  (graph, label).
         graphs, labels = map(list, zip(*samples))
         batched_graph = dgl.batch(graphs) # --- the batch function creates the minibatch out of sparse graphs --- #
         return batched_graph, torch.tensor(labels)
```

dgl.batch() function puts the sparse graphs in one mini-batch



The GN block




```python
class GN_block(nn.Module):
    def __init__(self, in_feats, out_feats, activation):
        super(GN_block, self).__init__()
        self.apply_mod = NodeApplyModule(in_feats, out_feats, activation)

    def forward(self, g, feature):
        # Initialize the node features with h.
        g.ndata['h'] = feature
        g.update_all(msg, reduce)
        g.apply_nodes(func=self.apply_mod)
        return g.ndata.pop('h')
```

The GN block individual steps

Step1 : Replace a node feature by average of neighboring node features

```
def reduce(nodes):
    """Take an average over all neighbor node features hu and use it to
    overwrite the original node feature."""
    accum = torch.mean(nodes.mailbox['m'], 1)
    return {'h': accum}
```

Step2 : Update the node features by passing through a linear layer + activation

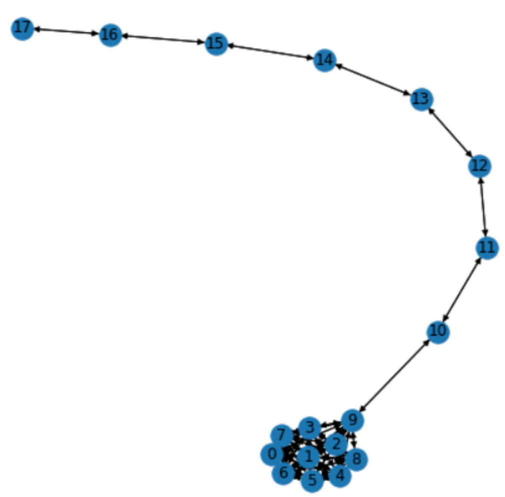
```
class NodeApplyModule(nn.Module):
    """Update the node feature hv with ReLU(Whv+b)."""
    def __init__(self, in_feats, out_feats, activation):
        super(NodeApplyModule, self).__init__()
        self.linear = nn.Linear(in_feats, out_feats)
        self.activation = activation

    def forward(self, node):
        h = self.linear(node.data['h'])
        h = self.activation(h)
        return {'h' : h}
```

Step3 : GN block = assigning features —> updating nodes + edges —> apply activation

```
class GN_block(nn.Module):
    def __init__(self, in_feats, out_feats, activation):
        super(GN_block, self).__init__()
        self.apply_mod = NodeApplyModule(in_feats, out_feats, activation)

    def forward(self, g, feature):
        # Initialize the node features with h.
        g.ndata['h'] = feature
        g.update_all(msg, reduce)
        g.apply_nodes(func=self.apply_mod)
        return g.ndata.pop('h')
```

The classifier

```python
import torch.nn.functional as F


class Classifier(nn.Module):
    def __init__(self, in_dim, hidden_dim, n_classes):
        super(Classifier, self).__init__()

        self.layers = nn.ModuleList([
            GN_block(in_dim, hidden_dim, F.relu),
            GN_block(hidden_dim, hidden_dim, F.relu)])
        self.classify = nn.Linear(hidden_dim, n_classes)

    def forward(self, g):
        # For undirected graphs, in_degree is the same as
        # out_degree.
        h = g.in_degrees().view(-1, 1).float()
        h = h.cuda() # --- converting to gpu ---- #

        for conv in self.layers:
            h = conv(g, h)
        g.ndata['h'] = h
        hg = dgl.mean_nodes(g, 'h')
        return self.classify(hg)
```



→ GN Block 1 → GN Block 2 → Average of nodes → Classify