6.3 针对链接预测任务的邻居采样训练方法

(English Version)

结合负采样来定义邻居采样器和数据加载器

用户仍然可以使用与节点/边分类中相同的邻居采样器。

```
sampler = dgl.dataloading.MultiLayerFullNeighborSampler(2)
```

DGL中的 `EdgeDataLoader` 还支持生成用于链接预测的负样本。为此，用户需要定义负采样函数。例如， `Uniform` 函数是基于均匀分布的采样函数，它对于每个边的源节点，采样 `k` 个负样本的目标节点。

以下数据加载器将为每个边的源节点均匀采样5个负样本的目标节点。

```
dataloader = dgl.dataloading.EdgeDataLoader(
    g, train_seeds, sampler,
    negative_sampler=dgl.dataloading.negative_sampler.Uniform(5),
    batch_size=args.batch_size,
    shuffle=True,
    drop_last=False,
    pin_memory=True,
    num_workers=args.num_workers)
```

关于内置的负采样方法，用户可以参考 Negative Samplers for Link Prediction。

用户还可以自定义负采样函数，它应当以原图 `g` 和小批量的边ID数组 `eid` 作为入参，并返回源节点ID数组和目标节点ID数组。

下面给出了一个自定义的负采样方法的示例，该采样方法根据与节点的度的幂成正比的概率分布对负样本目标节点进行采样。

```
class NegativeSampler(object):
    def __init__(self, g, k):
        # 缓存概率分布
        self.weights = g.in_degrees().float() ** 0.75
        self.k = k

    def __call__(self, g, eids):
        src, _ = g.find_edges(eids)
        src = src.repeat_interleave(self.k)
        dst = self.weights.multinomial(len(src), replacement=True)
        return src, dst

dataloader = dgl.dataloading.EdgeDataLoader(
    g, train_seeds, sampler,
    negative_sampler=NegativeSampler(g, 5),
    batch_size=args.batch_size,
    shuffle=True,
    drop_last=False,
    pin_memory=True,
    num_workers=args.num_workers)
```

调整模型以进行小批次训练

如 5.3 链接预测 中所介绍的， 用户可以通过比较边(正样本)与不存在的边(负样本)的得分来训练链路模型。用户可以重用在边分类/回归中的节点表示模型， 来计算边的分数。

```
class StochasticTwoLayerGCN(nn.Module):
    def __init__(self, in_features, hidden_features, out_features):
        super().__init__()
        self.conv1 = dgl.nn.GraphConv(in_features, hidden_features)
        self.conv2 = dgl.nn.GraphConv(hidden_features, out_features)

    def forward(self, blocks, x):
        x = F.relu(self.conv1(blocks[0], x))
        x = F.relu(self.conv2(blocks[1], x))
        return x
```

对于得分的预测，只需要预测每个边的标量分数而不是类别的概率分布， 因此本示例说明了如何使用边的两个端点的向量的点积来计算分数。

```
class ScorePredictor(nn.Module):
    def forward(self, edge_subgraph, x):
        with edge_subgraph.local_scope():
            edge_subgraph.ndata['x'] = x
            edge_subgraph.apply_edges(dgl.function.u_dot_v('x', 'x', 'score'))
            return edge_subgraph.edata['score']
```

使用负采样方法后，DGL的数据加载器将为每个小批次生成三项：

- 一个正样本图，其中包含采样得到的小批次内所有的边。
- 一个负样本图，其中包含由负采样方法生成的所有不存在的边。

- 邻居采样方法生成的块的列表。

因此，可以如下定义链接预测模型，该模型的输入包括上述三项以及输入的特征。

```
class Model(nn.Module):
    def __init__(self, in_features, hidden_features, out_features):
        super().__init__()
        self.gcn = StochasticTwoLayerGCN(
            in_features, hidden_features, out_features)

    def forward(self, positive_graph, negative_graph, blocks, x):
        x = self.gcn(blocks, x)
        pos_score = self.predictor(positive_graph, x)
        neg_score = self.predictor(negative_graph, x)
        return pos_score, neg_score
```

模型的训练

训练循环通过数据加载器去遍历数据，将得到的图和输入特征传入上述模型。

```
model = Model(in_features, hidden_features, out_features)
model = model.cuda()
opt = torch.optim.Adam(model.parameters())

for input_nodes, positive_graph, negative_graph, blocks in dataloader:
    blocks = [b.to(torch.device('cuda')) for b in blocks]
    positive_graph = positive_graph.to(torch.device('cuda'))
    negative_graph = negative_graph.to(torch.device('cuda'))
    input_features = blocks[0].srcdata['features']
    pos_score, neg_score = model(positive_graph, negative_graph, blocks, input_features)
    loss = compute_loss(pos_score, neg_score)
    opt.zero_grad()
    loss.backward()
    opt.step()
```

DGL提供了在同构图上做链路预测的一个示例： 无监督学习GraphSAGE。

异构图上的随机批次训练

计算异构图上的节点表示的模型也可以用于计算边分类/回归中的边两端节点的表示。

```
class StochasticTwoLayerRGCN(nn.Module):
    def __init__(self, in_feat, hidden_feat, out_feat, rel_names):
        super().__init__()
        self.conv1 = dglnn.HeteroGraphConv({
                rel : dglnn.GraphConv(in_feat, hidden_feat, norm='right')
                for rel in rel_names
            })
        self.conv2 = dglnn.HeteroGraphConv({
                rel : dglnn.GraphConv(hidden_feat, out_feat, norm='right')
                for rel in rel_names
            })

    def forward(self, blocks, x):
        x = self.conv1(blocks[0], x)
        x = self.conv2(blocks[1], x)
        return x
```

对于得分的预测，同构图和异构图之间唯一的实现差异是后者需要用`dgl.DGLGraph.apply_edges()`来遍历所有的边类型。

```
class ScorePredictor(nn.Module):
    def forward(self, edge_subgraph, x):
        with edge_subgraph.local_scope():
            edge_subgraph.ndata['x'] = x
            for etype in edge_subgraph.canonical_etypes:
                edge_subgraph.apply_edges(
                    dgl.function.u_dot_v('x', 'x', 'score'), etype=etype)
            return edge_subgraph.edata['score']

class Model(nn.Module):
    def __init__(self, in_features, hidden_features, out_features, num_classes,
                 etypes):
        super().__init__()
        self.rgcn = StochasticTwoLayerRGCN(
            in_features, hidden_features, out_features, etypes)
        self.pred = ScorePredictor()

    def forward(self, positive_graph, negative_graph, blocks, x):
        x = self.rgcn(blocks, x)
        pos_score = self.pred(positive_graph, x)
        neg_score = self.pred(negative_graph, x)
        return pos_score, neg_score
```

数据加载器的定义也与边分类/回归里的定义非常相似。唯一的区别是用户需要提供负采样方法， 并且提供边类型和边ID张量的字典，而不是节点类型和节点ID张量的字典。

```
sampler = dgl.dataloading.MultiLayerFullNeighborSampler(2)
dataloader = dgl.dataloading.EdgeDataLoader(
    g, train_eid_dict, sampler,
    negative_sampler=dgl.dataloading.negative_sampler.Uniform(5),
    batch_size=1024,
    shuffle=True,
    drop_last=False,
    num_workers=4)
```

如果用户想自定义负采样函数，那么该函数应以初始图以及由边类型和边ID张量构成的字典作为输入。 它返回以边类型为键、源节点-目标节点数组对为值的字典。示例如下所示：

```
class NegativeSampler(object):
    def __init__(self, g, k):
        # 缓存概率分布
        self.weights = {
            etype: g.in_degrees(etype=etype).float() ** 0.75
            for _, etype, _ in g.canonical_etypes
        }
        self.k = k

    def __call__(self, g, eids_dict):
        result_dict = {}
        for etype, eids in eids_dict.items():
            src, _ = g.find_edges(eids, etype=etype)
            src = src.repeat_interleave(self.k)
            dst = self.weights[etype].multinomial(len(src), replacement=True)
            result_dict[etype] = (src, dst)
        return result_dict
```

随后，需要向数据载入器提供边类型和对应边ID的字典，以及负采样器。示例如下所示：

```
train_eid_dict = {
    g.edges(etype=etype, form='eid')
    for etype in g.etypes}

dataloader = dgl.dataloading.EdgeDataLoader(
    g, train_eid_dict, sampler,
    negative_sampler=NegativeSampler(g, 5),
    batch_size=1024,
    shuffle=True,
    drop_last=False,
    num_workers=4)
```

异构图上的随机批次模型训练与同构图中的训练几乎相同，不同之处在于， `compute_loss` 是以边类型字典和预测结果字典作为输入。

```
model = Model(in_features, hidden_features, out_features, num_classes, etypes)
model = model.cuda()
opt = torch.optim.Adam(model.parameters())

for input_nodes, positive_graph, negative_graph, blocks in dataloader:
    blocks = [b.to(torch.device('cuda')) for b in blocks]
    positive_graph = positive_graph.to(torch.device('cuda'))
    negative_graph = negative_graph.to(torch.device('cuda'))
    input_features = blocks[0].srcdata['features']
    pos_score, neg_score = model(positive_graph, negative_graph, blocks, input_features)
    loss = compute_loss(pos_score, neg_score)
    opt.zero_grad()
    loss.backward()
    opt.step()
```