2.1 内置函数和消息传递API

(English Version)

在DGL中，**消息函数** 接受一个参数 `edges`，这是一个 `EdgeBatch` 的实例，在消息传递时，它被DGL在内部生成以表示一批边。`edges` 有 `src`、`dst` 和 `data` 共3个成员属性，分别用于访问源节点、目标节点和边的特征。

聚合函数 接受一个参数 `nodes`，这是一个 `NodeBatch` 的实例，在消息传递时，它被DGL在内部生成以表示一批节点。`nodes` 的成员属性 `mailbox` 可以用来访问节点收到的消息。一些最常见的聚合操作包括 `sum`、`max`、`min` 等。

更新函数 接受一个如上所述的参数 `nodes`。此函数对 `聚合函数` 的聚合结果进行操作，通常在消息传递的最后一步将其与节点的特征相结合，并将输出作为节点的新特征。

DGL在命名空间 `dgl.function` 中实现了常用的消息函数和聚合函数作为 **内置函数**。一般来说，DGL建议 **尽可能** 使用内置函数，因为它们经过了大量优化，并且可以自动处理维度广播。

如果用户的消息传递函数无法用内置函数实现，则可以实现自己的消息或聚合函数(也称为 **用户定义函数**)。

内置消息函数可以是一元函数或二元函数。对于一元函数，DGL支持 `copy` 函数。对于二元函数，DGL现在支持 `add`、`sub`、`mul`、`div`、`dot` 函数。消息的内置函数的命名约定是 `u` 表示 `源` 节点，`v` 表示 `目标` 节点，`e` 表示 `边`。这些函数的参数是字符串，指示相应节点和边的输入和输出特征字段名。关于内置函数的列表，请参见 DGL Built-in Function。例如，要对源节点的 `hu` 特征和目标节点的 `hv` 特征求和，然后将结果保存在边的 `he` 特征上，用户可以使用内置函数 `dgl.function.u_add_v('hu', 'hv', 'he')`。而以下用户定义消息函数与此内置函数等价。

```
def message_func(edges):
    return {'he': edges.src['hu'] + edges.dst['hv']}
```

DGL支持内置的聚合函数 `sum`、`max`、`min` 和 `mean` 操作。聚合函数通常有两个参数，它们的类型都是字符串。一个用于指定 `mailbox` 中的字段名，一个用于指示目标节点特征的字段名，例如，`dgl.function.sum('m', 'h')` 等价于如下所示的对接收到消息求和的用户定义函数：

```
import torch
def reduce_func(nodes):
     return {'h': torch.sum(nodes.mailbox['m'], dim=1)}
```

关于用户定义函数的进阶用法，参见 User-defined Functions。

在DGL中，也可以在不涉及消息传递的情况下，通过 `apply_edges()` 单独调用逐边计算。`apply_edges()` 的参数是一个消息函数。并且在默认情况下，这个接口将更新所有的边。例如：

```
import dgl.function as fn
graph.apply_edges(fn.u_add_v('el', 'er', 'e'))
```

对于消息传递，`update_all()` 是一个高级API。它在单个API调用里合并了消息生成、 消息聚合和节点特征更新，这为从整体上进行系统优化提供了空间。

`update_all()` 的参数是一个消息函数、一个聚合函数和一个更新函数。 更新函数是一个可选择的参数，用户也可以不使用它，而是在 `update_all` 执行完后直接对节点特征进行操作。 由于更新函数通常可以用纯张量操作实现，所以DGL不推荐在 `update_all` 中指定更新函数。例如：

```
def update_all_example(graph):
    # 在graph.ndata['ft']中存储结果
    graph.update_all(fn.u_mul_e('ft', 'a', 'm'),
                     fn.sum('m', 'ft'))
    # 在update_all外调用更新函数
    final_ft = graph.ndata['ft'] * 2
    return final_ft
```

此调用通过将源节点特征 `ft` 与边特征 `a` 相乘生成消息 `m`， 然后对所有消息求和来更新节点特征 `ft`，再将 `ft` 乘以2得到最终结果 `final_ft`。

调用后，中间消息 `m` 将被清除。上述函数的数学公式为：

$$final\_ft_i = 2 * \sum_{j \in \mathcal{N}(i)} (ft_j * a_{ij})$$