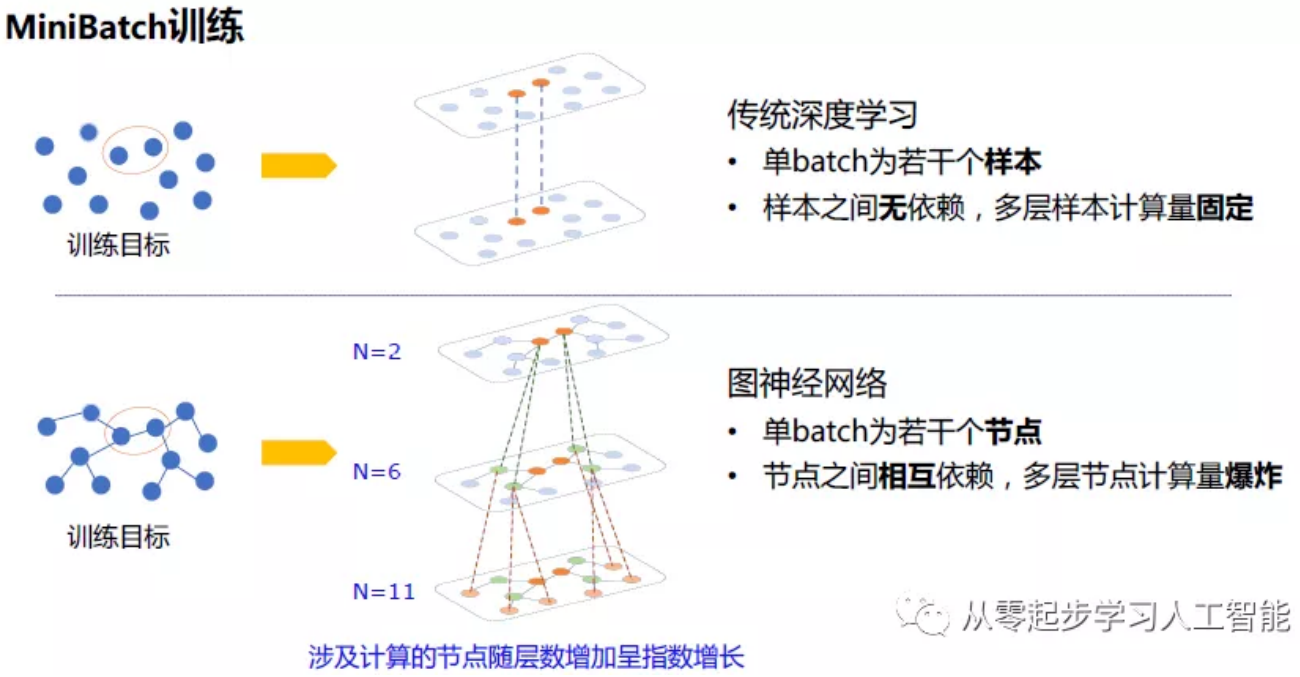
图神经网络-图采样Graphsage代码实现

原创 段智华 从零起步学习人工智能 2月15日

一：为什么要图采样?



二 Graphsage 采样代码实践

GraphSage 的 PGL 完整代码实现位于 https://github.com/PaddlePaddle/PGL/tree/main/examples/graphsage，本文实现一个简单的graphsage 采样代码 。

安装依赖

```
# !pip install paddlepaddle==1.8.4
!pip install pgl -q
```

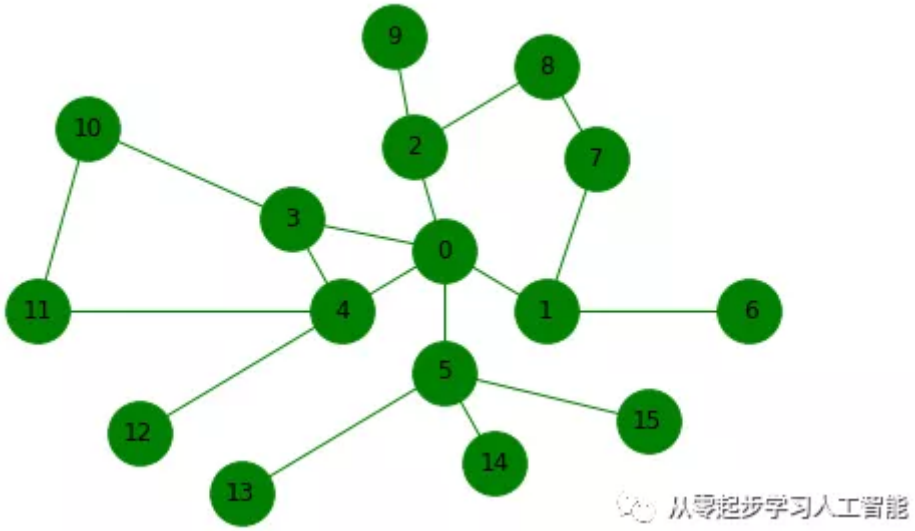
1. 构建graph

图网络的构建使用Graph类， Graph类的具体实现可以参考 https://github.com/PaddlePaddle/PGL/blob/main/pgl/graph.py

```
import random
import numpy as np
import pgl
import display

def build_graph():
```

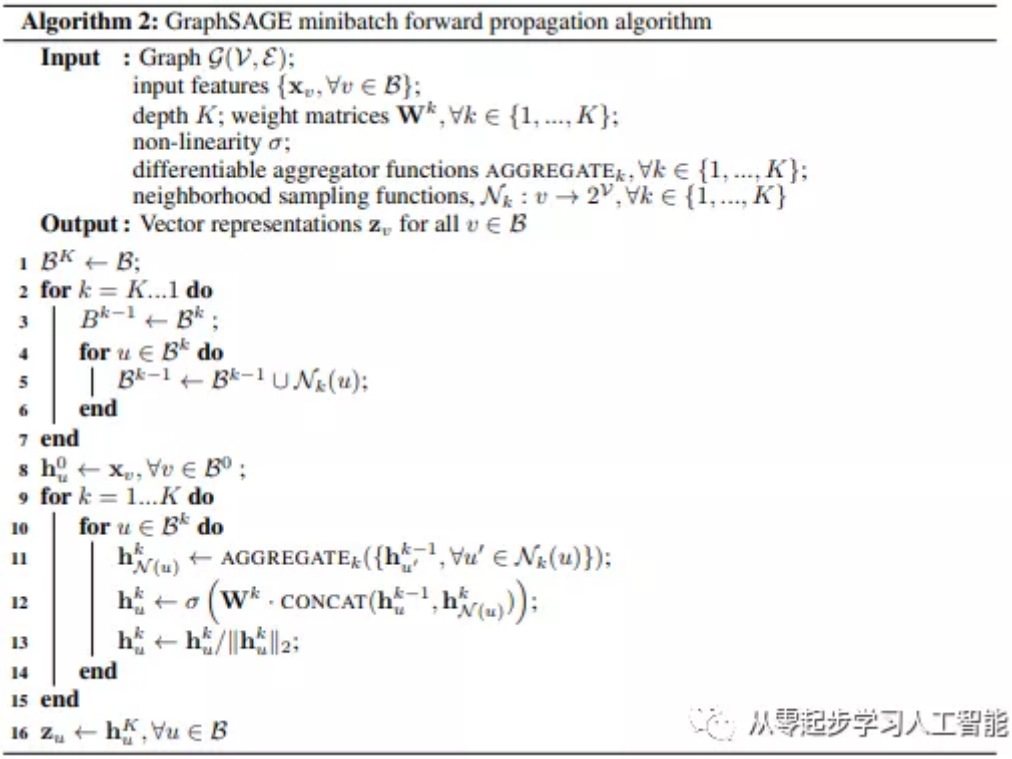
```
    # 定义节点的个数；每个节点用一个数字表示，即从0~9
    num_node = 16
    # 添加节点之间的边，每条边用一个tuple表示为: (src, dst)
    edge_list = [(2, 0), (1, 0), (3, 0),(4, 0), (5, 0),
                 (6, 1), (7, 1), (8, 2), (9, 2), (8, 7),
                 (10, 3), (4, 3), (11, 10), (11, 4), (12, 4),
                 (13, 5), (14, 5), (15, 5)]

    g = pgl.graph.Graph(num_nodes = num_node, edges = edge_list)

    return g

# 创建一个图对象，用于保存图网络的各种数据。
g = build_graph()
display.display_graph(g)
```

运行结果：



2. GraphSage采样函数实现

GraphSage的作者提出采样算法来使得模型能够以Mini-batch的方式进行训练，算法代码见论文附录A https://cs.stanford.edu/people/jure/pubs/graphsage-nips17.pdf。



Algorithm 2: GraphSAGE minibatch forward propagation algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$;
input features $\{\mathbf{x}_v, \forall v \in \mathcal{B}\}$;
depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$;
non-linearity σ;
differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$;
neighborhood sampling functions, $\mathcal{N}_k : v \rightarrow 2^{\mathcal{V}}, \forall k \in \{1, ..., K\}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{B}$

$\mathcal{B}^K \leftarrow \mathcal{B}$;
for $k = K...1$ **do**
  $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^k$;
  for $u \in \mathcal{B}^k$ **do**
    $\mathcal{B}^{k-1} \leftarrow \mathcal{B}^{k-1} \cup \mathcal{N}_k(u)$;
  end
end
$\mathbf{h}_u^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{B}^0$;
for $k = 1...K$ **do**
  for $u \in \mathcal{B}^k$ **do**
    $\mathbf{h}_{\mathcal{N}(u)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_{u'}^{k-1}, \forall u' \in \mathcal{N}_k(u)\})$;
    $\mathbf{h}_u^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_u^{k-1}, \mathbf{h}_{\mathcal{N}(u)}^k)\right)$;
    $\mathbf{h}_u^k \leftarrow \mathbf{h}_u^k / \|\mathbf{h}_u^k\|_2$;
  end
end
$\mathbf{z}_u \leftarrow \mathbf{h}_u^K, \forall u \in \mathcal{B}$

- 假设要利用中心节点的k阶邻居信息，则在聚合的时候，需要从第k阶邻居传递信息到k-1阶邻居，并依次传递到中心节点。
- 采样的过程与此相反，在构造第t轮训练的Mini-batch时，从中心节点出发，在前序节点集合中采样Nt个邻居节点加入采样集合。
- 将邻居节点作为新的中心节点继续进行第t-1轮训练的节点采样，以此类推。
- 将采样到的节点和边一起构造得到子图。

```python
def traverse(item):
    """traverse
    """
    if isinstance(item, list) or isinstance(item, np.ndarray):
        for i in iter(item):
            for j in traverse(i):
                yield j
    else:
        yield item

def flat_node_and_edge(nodes):
    """这个函数的目的是为了将 list of numpy array 扁平化成一个list
    例如: [array([7, 8, 9]), array([11, 12]), array([13, 15])] --> [7, 8, 9, 1
    """
    nodes = list(set(traverse(nodes)))
    return nodes
```

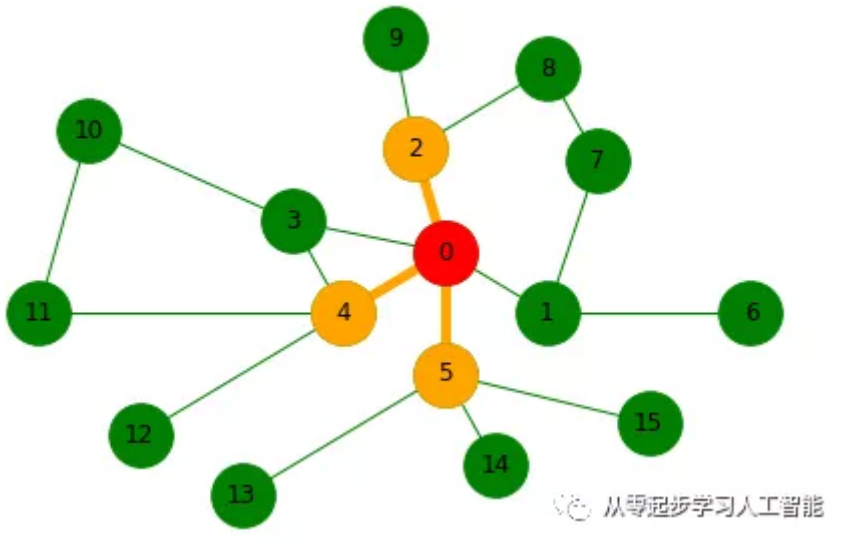
```

def graphsage_sample(graph, start_nodes, sample_num):
    subgraph_edges = []
    # pre_nodes: a list of numpy array,
    pre_nodes = graph.sample_predecessor(start_nodes, sample_num)

    # 根据采样的子节点， 恢复边
    for dst_node, src_nodes in zip(start_nodes, pre_nodes):
        for node in src_nodes:
            subgraph_edges.append((node, dst_node))


    subgraph_nodes = flat_node_and_edge(pre_nodes)

    return subgraph_nodes, subgraph_edges

```

随机获取一阶邻居信息

```
seed = 458
np.random.seed(seed)
random.seed(seed)

start_nodes = [0]

layer1_nodes, layer1_edges = graphsage_sample(g, start_nodes, sample_num=3)
print('layer1_nodes: ', layer1_nodes)
print('layer1_edges: ', layer1_edges)
display.display_subgraph(g, {'orange': layer1_nodes}, {'orange': layer1_edges}
```

运行结果

```
layer1_nodes:  [2, 4, 5]
layer1_edges:  [(4, 0), (2, 0), (5, 0)]
```

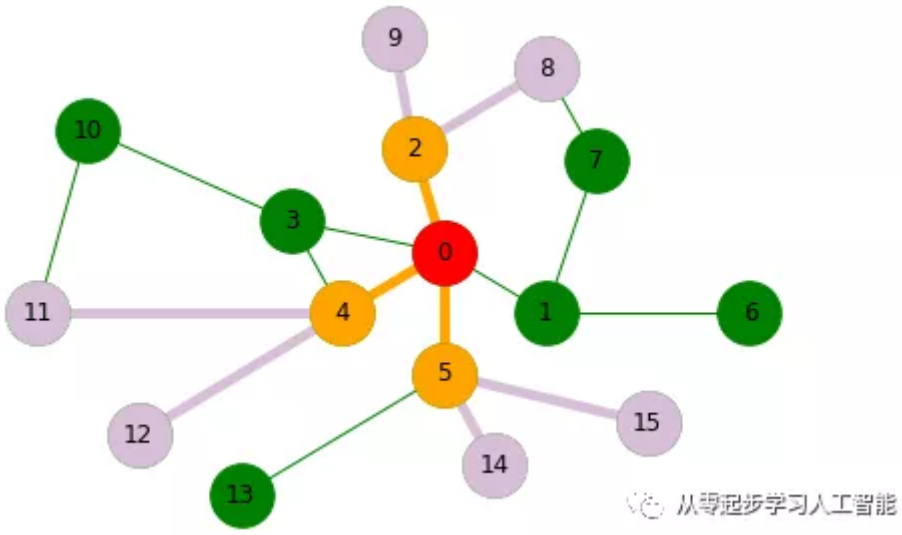


继续获取二阶邻居节点信息

```
layer2_nodes, layer2_edges = graphsage_sample(g, layer1_nodes, sample_num=2)
print('layer2_nodes: ', layer2_nodes)
print('layer2_edges: ', layer2_edges)
display.display_subgraph(g, {'orange': layer1_nodes, 'Thistle': layer2_nodes},
```

运行结果

```
layer2_nodes:  [8, 9, 11, 12, 14, 15]
layer2_edges:  [(8, 2), (9, 2), (11, 4), (12, 4), (14, 5), (15, 5)]
```



图节点可视化代码

```
#%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

```
import networkx as nx # networkx是一个常用的绘制复杂图形的Python包。

def display_graph(g):
    nx_G = nx.Graph()
    nx_G.add_nodes_from(range(g.num_nodes))
    nx_G.add_edges_from(g.edges)

    pos = {0: [0.5, 0.5], 1:[0.6, 0.4], 2:[0.47, 0.67], 3: [0.35, 0.55], 4:[0.
           6: [0.8, 0.4], 7:[0.65, 0.65], 8:[0.6, 0.8], 9:[0.45, 0.85], 10:[0.
           12:[0.2, 0.2], 13:[0.3, 0.1], 14:[0.55, 0.15], 15:[0.7, 0.22]}
    nx.draw(nx_G,
            pos,
            with_labels=True,
            node_color='green',
            edge_color='green',
            node_size=1000)

    plt.show()

#display_graph(g)# 创建一个GraphWrapper作为图数据的容器，用于构建图神经网络。

def display_subgraph(g, sub_nodes, sub_edges):
    nx_G = nx.Graph()
    nx_G.add_nodes_from(range(g.num_nodes))
    nx_G.add_edges_from(g.edges)

    pos = {0: [0.5, 0.5], 1:[0.6, 0.4], 2:[0.47, 0.67], 3: [0.35, 0.55], 4:[0.
           6: [0.8, 0.4], 7:[0.65, 0.65], 8:[0.6, 0.8], 9:[0.45, 0.85], 10:[0.
           12:[0.2, 0.2], 13:[0.3, 0.1], 14:[0.55, 0.15], 15:[0.7, 0.22]}
    nx.draw(nx_G,
            pos,
            with_labels=True,
            node_color='green',
            edge_color='green',
            node_size=1000,
            width=1)

    nx.draw_networkx_nodes(nx_G, pos, nodelist=[0], node_color='red', node_siz

    for color, nodes in sub_nodes.items():
```

```
        nx.draw_networkx_nodes(nx_G, pos, nodelist=nodes, node_color=color, nc

    for color, edges in sub_edges.items():
        nx.draw_networkx_edges(nx_G, pos, edgelist=edges, edge_color=color, wi

    plt.show()
```

注：本文图文资料来源于 AIStudio-人工智能学习与实训社区