

# GraphSAGE 源码分析报告

——袁宇箭 2018K8009908017 2020.11.11

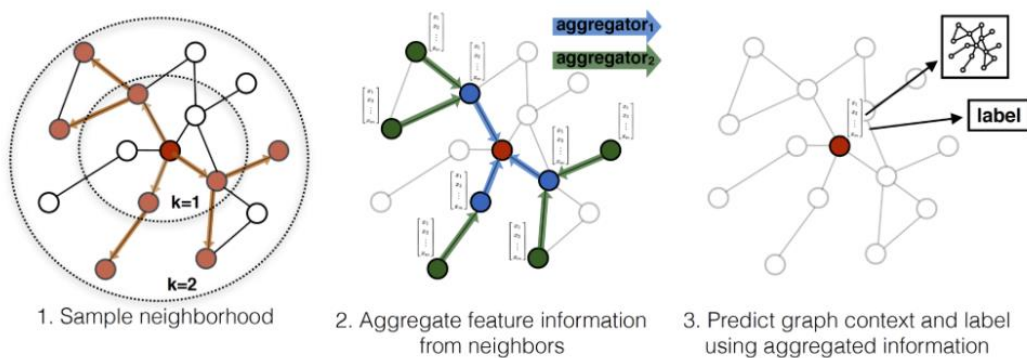
## 一、 GraphSAGE 介绍

GraphSAGE 即 Graph Sample and aggreGatE, 类似于传统的图卷积神经网络 GCN, 它也是一种图的深度学习算法, 它的特点在于引入了 Inductive 和 sample 这两个特点。GraphSage 的出现完成了机器学习从 Transductive (直推式学习) 到 inductive (归纳式学习) 的转变。

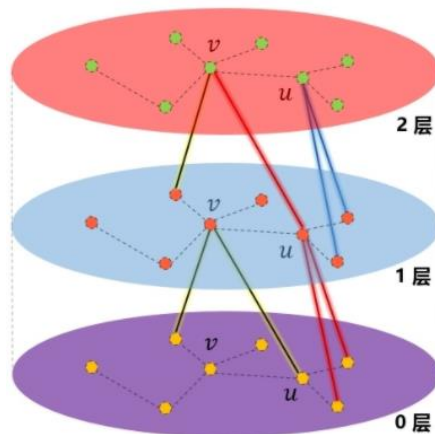
以往 GCN 算法是典型的直推式学习方法, 它所学习到的参数很大程度上与图的结构有关, 一旦图发生了变化则需要重新学习参数; 而 GraphSAGE 便是采用归纳式学习方法, 它学习节点之间的聚合模式, 利用结点领域的聚合模型直接学习出新节点的嵌入特征, 只要图不发生太大的变化则无需重新学习参数, 大大提高了算法的鲁棒性。

同时 GraphSAGE 采用了不一样的采样方法, 也就是 sample 也是它的一个突出特点。

## 二、 GraphSAGE 工作过程简介
















如上图, 可以观察到图中每个结点都有一个或者多个邻居, 在图二中展示了聚合的过程, 蓝色结点将它的邻居 (绿色结点) 的特征集合在自己身上, 红色结点便将其蓝色结点邻居的特征集中在自己身上, 通过这样一层一层的聚合便可以计算得出结点的特征, 从而应用于新节点上。更直观的工作过程图如下:



它的核心算法就是对每一层将每一个点的邻居特征都聚合起来, 所以需要使用两层循环来完成这个功能, 具体伪代码如下:

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm	
<b>Input</b> : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ ; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$ ; depth $K$ ; weight matrices $\mathbf{W}^k, \forall k \in \{1, \dots, K\}$ ; non-linearity $\sigma$ ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, \dots, K\}$ ; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$	
<b>Output</b> : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$	
1	$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V};$
2	<b>for</b> $k = 1 \dots K$ <b>do</b>
3	<b>for</b> $v \in \mathcal{V}$ <b>do</b>
4	$\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\});$
5	$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$
6	<b>end</b>
7	$\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \ \mathbf{h}_v^k\ _2, \forall v \in \mathcal{V}$
8	<b>end</b>
9	$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

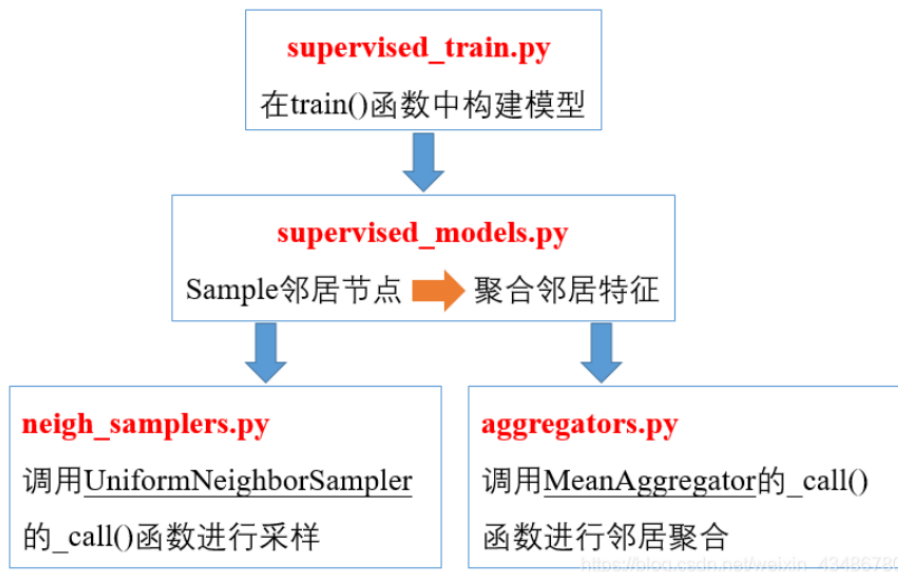
### 三、 代码整体框架

 _init_.py	Initial commit of cleaned repo.
 aggregators.py	Added mean pooling.
 inits.py	Cleaning up comments etc.
 layers.py	Create layers.py
 metrics.py	Cleaning up comments etc.
 minibatch.py	modify minibatch iterator end condition so that the last batch can be...
 models.py	fix node2vec neg affinity calculation
 neigh_samplers.py	Cleaning up comments etc.
 prediction.py	fix loss computation
 supervised_models.py	Added mean pooling.
 supervised_train.py	make adjacency matrix a placeholder to make saved graphdef smaller
 unsupervised_train.py	make adjacency matrix a placeholder to make saved graphdef smaller
 utils.py	Make networkx version check Python3 friendly

可以发现 GraphSAGE 算法实现上总体来说分为有监督学习和无监督学习方法, 本报告暂时分析有监督学习算法

### 四、 有监督学习方法

整体代码的框架如下:



大致过程具体为：在 supervised\_train.py 中

```
1 """supervised_train.py """
2 minibatch = NodeMinibatchIterator(G,
3                                     id_map,
4                                     placeholders,
5                                     class_map,
6                                     num_classes,
7                                     batch_size=FLAGS.batch_size,
8                                     max_degree=FLAGS.max_degree,
9                                     context_pairs=context_pairs)
```

首先通过 load\_data 函数输入数据集，得到图 G 等信息，并将其打包赋值给 minibatch。之后在 minibatch.py 中调用 construct\_adj 函数：

```
def construct_adj(self):
    adj = len(self.id2idx)*np.ones((len(self.id2idx)+1, self.max_degree))
    deg = np.zeros((len(self.id2idx),))

    for nodeid in self.G.nodes():
        if self.G.node[nodeid]['test'] or self.G.node[nodeid]['val']:
            continue
        neighbors = np.array([self.id2idx[neighbor]
                               for neighbor in self.G.neighbors(nodeid)
                               if (not self.G[nodeid][neighbor]['train_removed'])])
        deg[self.id2idx[nodeid]] = len(neighbors)
        if len(neighbors) == 0:
            continue
        if len(neighbors) > self.max_degree:
            neighbors = np.random.choice(neighbors, self.max_degree, replace=False)
        elif len(neighbors) < self.max_degree:
            neighbors = np.random.choice(neighbors, self.max_degree, replace=True)
        adj[self.id2idx[nodeid], :] = neighbors
    return adj, deg
```

对 batch 进行划分，并得到邻接表（deg）和每个顶点的度（adj），之后返回到 supervised\_train.py 文件进行采样：

```

if FLAGS.model == 'graphsage_mean':
    # Create model
    sampler = UniformNeighborSampler(adj_info)
    if FLAGS.samples_3 != 0:
        layer_infos = [SAGEInfo("node", sampler, FLAGS.samples_1, FLAGS.dim_1),
                        SAGEInfo("node", sampler, FLAGS.samples_2, FLAGS.dim_2),
                        SAGEInfo("node", sampler, FLAGS.samples_3, FLAGS.dim_2)]
    elif FLAGS.samples_2 != 0:
        layer_infos = [SAGEInfo("node", sampler, FLAGS.samples_1, FLAGS.dim_1),
                        SAGEInfo("node", sampler, FLAGS.samples_2, FLAGS.dim_2)]

```

它根据 construct\_adj 的返回值来调用函数进行采样得到 sampler，具体的采样过程是在 models.py 中首先进入父类的 sample 函数，得到 samples 和 support\_sizes，并将其输入 aggregate 函数进行聚合，实际上采样的过程就是一个递归的过程。

```

def aggregate(self, samples, input_features, dims, num_samples, support_sizes, batch_size=None,
              aggregators=None, name=None, concat=False, model_size="small"):
    """ At each layer, aggregate hidden representations of neighbors to compute the hidden representations
        at next layer.
    Args:
        samples: a list of samples of variable hops away for convolving at each layer of the
                 network. Length is the number of layers + 1. Each is a vector of node indices.
        .....

        h = aggregator((hidden[hop],
                        tf.reshape(hidden[hop + 1], neigh_dims)))
        next_hidden.append(h)
        hidden = next_hidden
    return hidden[0], aggregators

```

最后调用 meanagregator 的 \_call () 函数进行聚合。聚合就是利用最新采样到的邻居的特征更新当前节点的特征，而节点的特征便是学习的目的。

## 五、 优缺点分析

优点：1 它改进了 GCN 训练方法的缺点，可以提升模型的灵活性和泛化能力。

2.可以分批训练，提升收敛速度。

缺点：算法的递归采样导致采样个数随层数指数增长，这会导致程序运行时间相比 GCN 要长