An Introductive

1. 引言：

作者针对图分类（节点分类问题）提出了一种低纬度下的inductive算法，核心点在于其采样和聚集思想。采样方式在本文中采用的是均匀采样，聚集算子包含：均值聚集算子、LSTM聚集算子和池化算法。此外，作者提出了batch的训练方式，通过上述方式，一方面，避免对拉普拉斯算子进行运算，另一方面，我们可以通过更换相应的算子来实现算法的改进。

2. 文章核心：

首先我们通过浏览其目录，忽略相关的数学证明，核心点归纳如下：①嵌入生成算法（算法1，算法2），②GraphSage的参数学习（监督学习和非监督学习的损失函数设定），③聚集结构（三个聚集算子）。对此，我只针对这个三个方面发表自己的见解，至于数学推理，这个有兴趣的可以参考其附录内容。

2.1 嵌入生生成算法

首先，介绍算法一，这里直接上原文截图。

算法一

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

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

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

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

下面对算法一进行详细解释。

输入参数： 输入特征 x_v,深度 K，这里的K，既是深度，也是层数，也是迭代数，一定要注意这个K的含义，在图中表示为 "K-hop"；W^k表示权重，可以为每层设置不同的权重系数；非线性激活函数σ；聚集算子$AGGREGATE_k$，也可以在每层设置不同的函数，也可以实现自己的聚集算子；邻居映射函数 $\mathcal{N}:v\rightarrow 2^{\mathcal{V}}$，这里的概念为幂集运算，我们可以这么理解，对$v$的邻居节点进行采样的后节点的集合，隶属于$2^{\mathcal{V}}$集合。

输出参数： 获取的第K层特征，这里用z_v表示。

算法步骤：

第一行：对所有节点的状态初始化，作为节点的第0层特征，用h_v^0表示。

第四行：对v的邻居节点采样（文中采用均匀采样），并采用聚集算子运算其$k-1$层，即相对于本次运算的上一次特征，得到的结果用$h_{\mathcal{N}(v)}^k$表示

第五行：将节点v的$k-1$层特征（上一次特征，在这里也可以理解为，暂时还没有改变的特征值）与$h_{\mathcal{N}(v)}^k$进行拼接，并得到第k层特征。

到第七行：对节点v的第k层特征进行标准化处理。

第三到八行：重复采样、聚集操作，直到获取到第K层特征。

第九行：将特征输出。

核心： 算法一的核心在于，采用采样、聚集更新节点状态，随着深度增加，采样节点可以获取外层更多邻居节点的信息。

接着对算法二进行分析，算法二相当于先同过运算获取要采样节点的k阶子图。

算法 二

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**
  $B^{k-1} \leftarrow \mathcal{B}^k$;
  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
$\mathbf{z}_u \leftarrow \mathbf{h}_u^K, \forall u \in \mathcal{B}$

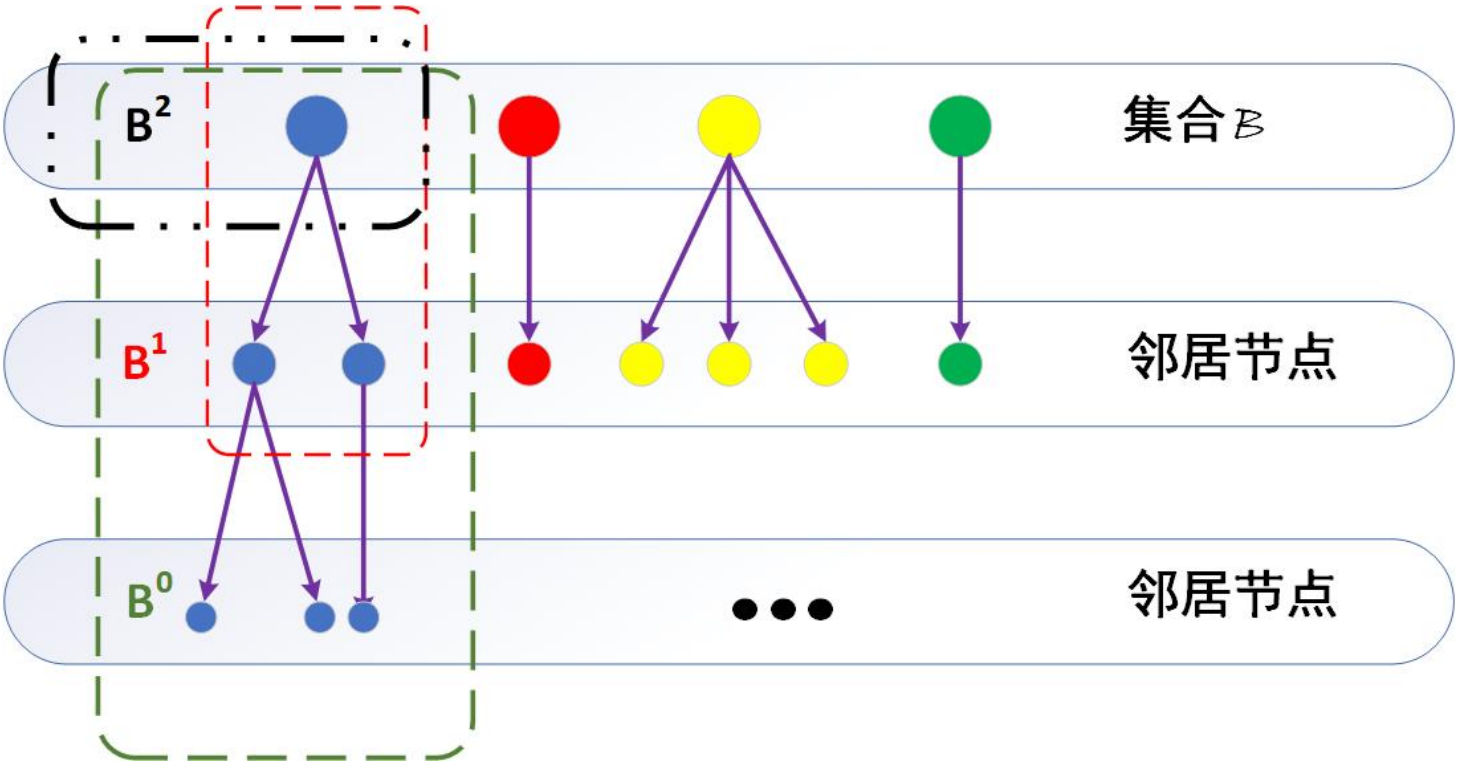
下面对算法二进行详细解释，只介绍和算法一有所区别的地方。

输入参数： 相比于算法一，这里v是属于batch中的点，可以这么考虑一个batch我们对图中v_0, v_1, v_3进行采样，那么$\mathcal{B}$就是包含这些点的集合

输出参数： 这个也是输出一个batch中的点的特征集合。

算法步骤：

第一到七行是获取子图过程，为了方便大家理解，绘制了如下树形图。第一行的两个B不一致，可能是作者出错了。



在这里我们假定$K=2$，也就是要获取batch中每个节点的第2层特征。

执行算法第一行，$B^2=\mathcal{B}$，注意这里的B^2是个集合，其中的每一项如图中**黑色框**所示。
执行算法第二到七行，首先$B^1=B^2$，接着对在B^2中的点进行采样邻居操作，将集合中的某一项绘制如**红色框**所示，也就是说B^1的点包含B^2层的点和B^2的邻居节点。类推最后得到B^0,选取其中一项如**绿色图框**所示，即包含从batch中某点所有元素的节点。

理解这一点，剩下的就比较容易了。

第八行是初始化所有节点的特征为h_v^0。

第九到十五行是进行采样聚合的过程，首先需要明白一点，B^0是包含$B1$邻居节点的初始特征。以图为例，红色B^1所在行的节点要采样的邻居节点在B^0中。根据这个原理，首先操作B^1中的点，包含蓝色根节点和其邻居节点，再次操作B^2中的点，包含根节点。

虽然有所重复，但是根据某清华大佬的实验，所占用的显存明显降低，据此我也推测占用的内存可能会有所增加。

第十六行是batch输出的特征。

为了降低运算量，作者对固定数目的邻居节点采样。我们依然假定$K=2$，那么采用 **算法1**，$k=1$时，采样S_1数目节点，$k=2$时，采样S_2数目节点，选取某个目标节点（根节点）。从 **算法2** 角度来看，$k=2$时（注意算法2是倒序）采样S_2节点，$k=1$时，采样S_1*S_2数目节点

2.2 参数学习

作者在此针对无监督和有监督方式提出两种损失函数。

无监督学习： 该损失函数原理是临近的节点应该具有相似的特征，反之则区别明显。该损失函数如下：

$$\mathcal{J}_{\mathcal{G}}(\mathbf{z}_u) = -log(\sigma(\mathbf{z}_u^T \mathbf{z}_v)) - \mathcal{Q} * E_{v_n \sim \mathcal{P}_n(v)} log(\sigma(-\mathbf{z}_u^T \mathbf{z}_{v_n})$$

其中，v是临近u节点经过固定长度的随机游走获取的节点，σ为激活函数，$\mathcal{P}_n$是一个负采样分布概率，$\mathcal{Q}$为定义的负样本数目，注意这里的$\mathbf{z}_u$是由其的邻居节点所包含的特征生成，但是至于如何生成，作者在文中并未详细说明。

监督学习： 这里将损失函数替换为交叉熵，这个函数在一般的深度学习框架中都有现成的函数。

2.3 聚集算子

这里采用的都是无序、非位置图，所以其一个重要特性就是置换不变性，因此许多的算子可以被应用。

均值算子： 原作中括号不对称

$$h_v^k \leftarrow \sigma(\mathbf{W} * MEAN(\{h_v^{k-1}\} \cup \{h_v^{k-1}, \forall u \in \mathcal{N}(v)\}))$$

这个很容易理解了吧。值得注意的是，这里的 ∪操作相当于是一次"skip connection"，学过神经网络的都懂。

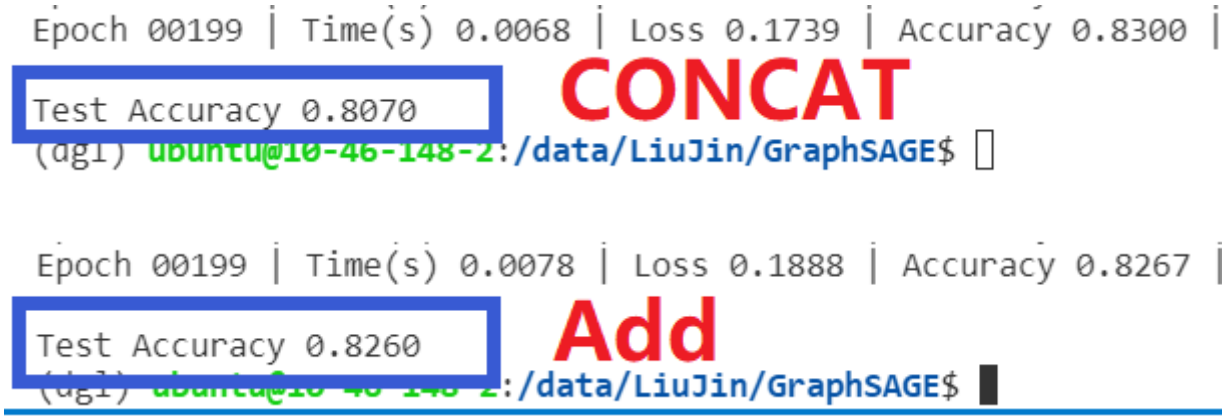
LSTM算子：
LSTM具备更强的表达能力，但是不具备扰动不变性，作者使用加扰动的邻居节点后生成的无序节点集合来操作LSTM。至于LSTM的公式，这个网上一大堆。这里就不写了。

池化算子： 通过一个全连接网络来操作邻居节点特征。这里采用的是最大值池化，其实本质上这个公式和均值、最小化之类的没什么区别，当然，除了效果，具体公式如下：

$$AGG_k = max(\sigma(\mathbf{W}_{pool} h_{u_i^k} + b), \forall u_i \in \mathcal{N}(v))$$

3. 算法复现

我对dgl提供的一些例子算法进行测试，在cora 数据集下，发现算法在采用CONCAT即拼接方式的时候，准确率80%左右，而采用相加运算时，有83%的准确率，也可能是某些参数设置的问题，不过这两种方式应该差距还是有的。如下图所示，实验环境： ubuntu18.04，P40显卡，但是我估计影响不大。


Epoch 00199 | Time(s) 0.0068 | Loss 0.1739 | Accuracy 0.8300 |
Test Accuracy 0.8070
CONCAT
(dgl) ubuntu@10-46-148-2:/data/LiuJin/GraphSAGE$
Epoch 00199 | Time(s) 0.0078 | Loss 0.1888 | Accuracy 0.8267 |
Test Accuracy 0.8260
Add
:/data/LiuJin/GraphSAGE$