GraphSAGE 学习笔记

大纲

目的与动机

GraphSAGE模型的目的是对图中节点的低维度嵌入表达进行归纳式学习 *(inductive learning)* 。

图节点的低维度嵌入向量学习已经在很多预测任务中被证明非常有效，如内容推荐与蛋白质功能识别等等。

图节点的嵌入，根本上的思想是将节点高维度的信息提取出来，嵌入到低维度稠密的向量中，再利用这些嵌入的低维度向量进行下一步任务。

在实际应用中，经常需要为全新的节点甚至是全新的子图生成嵌入向量。这种归纳能力对于高效的生产环境模型而言是至关重要的。因为他们需要在不断增长的图上工作，比如Reddit上的帖子。模型也需要有更好的泛化能力，为具有相同形式特征的节点生成嵌入向量。比如在蛋白质作用网络中，学习了在人体的一个组织中蛋白质作用关系的模型，也需要能应用到人体的其他组织中。

但是，目前已有的大多数工作，都从根本上是直推式 *(transductive)* 的，并不能自然地泛化到之前未见过的节点上，他们都关注于在一个固定的图上的应用，没有较好的泛化能力。

事实上，与直推式相比，具有归纳能力的模型是特别难实现的。因为对未知节点的泛化要求模型能够把新的子图对齐到已经学习的特征向量空间里，这需要模型能够学习能同时表现节点局部与全局特征的相邻节点属性。

GraphSAGE希望找到对图中节点的低维度表达进行归纳式学习的方法。

已有工作

在图节点嵌入领域，已有了很多相关工作。但如之前提到的，他们大多都不能解决，至少不能很好的解决图节点嵌入归纳式学习的问题。根据论文，已有的工作可以大体分为这些方向。

1. 基于矩阵分解的方式。

   这种方式基于在图中的随机探索与矩阵分解。他们都是根本上就属于直推式的模型，因为很多都需要完整的图才能应用。所以他们都不能很好的进行归纳式学习，就算能，也需要大量的计算。

2. 图上的监督式学习。

   他们大多关注于图分类(或者子图分类)，而本模型更加关注于单独节点的嵌入表示生成。

3. 图卷积网络。

   在这一方向上的大多数模型都不能很好的应用到大规模图上，或者只适合于整张图的分类。

   而与GraphSAGE密切相关的GCN，也依赖于已知整张图的Laplacian矩阵。

模型引入

GraphSAGE事实上是一个具有通用性的框架，目的是对图节点的嵌入表示进行归纳式学习。GraphSAGE中的SAGE是SAmple与aggreGAte的缩写。

GraphSAGE其实是对GCN的一种拓展，将GCN拓展到归纳式无监督学习的任务上，并且用可训练的代理函数替代确定不变而较为简单的图卷积。他利用了图中节点的特征来学习出一个嵌入函数 *(embedding function)* 生成节点的嵌入表达，这个函数可以泛化到未知节点上。

事实证明，GraphSAGE不仅能利用节点的特征信息，也可以利用节点邻居的拓扑结构、相邻节点的特征信息，乃至于整张图的结构信息。所以，GraphSAGE不仅能被应用到具有大量特征信息的图节点上，也能被应用到不包含节点特征的图上。

GraphSAGE，正如名字所表现的那样，最主要的过程就是sample与aggregate。为了生成每个节点的嵌入特征表达，模型先从模型的相邻节点中取样固定个数的节点，再将抽样出的节点特征信息输入代理函数中，得到节点新的特征表达。

所以，GraphSAGE与先前直推式方法的不同之处在于，他学习的是一组能生成节点特征表达的代理函数，而不是单独的学习每个节点的不同特征表达。这也是GraphSAGE归纳与泛化能力的根源所在。

模型介绍

嵌入向量的计算

首先，介绍GraphSAGE的前向传播部分，也就是嵌入特征向量的计算部分。

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
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

如上是论文中的GraphSAGE节点嵌入特征生成算法。

可以简单地认为，在图、节点特征、各项参数等信息给定的情况下，算法的输入$\mathbf{x}_v$为节点的初始特征向量，输出$\mathbf{z}_v$为节点的嵌入特征向量。

算法中的K为迭代次数，也是代理函数的个数，也可以被理解为网络的"层数"。下面，只要弄明白每一"层"做了什么，就能明白整个算法做了什么。

每一层的输入$\mathbf{h}^{k-1}$为上一层中节点的嵌入特征表达，而输出$\mathbf{h}^k$即为新一轮迭代后生成的节点表达。每一层的计算过程可以被概括为，对$\forall v \in \mathcal{V}$:

1. 对v的相邻节点抽样，得到$\mathcal{N}(v)$。
2. 将抽样出的节点特征输入代理方程，得到中间值

$$\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$$

3. 接着，计算新的节点特征表达：

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}_k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$$

   在这一步中，首先将综合了邻居节点特征表达的中间值与上一层中本节点的嵌入特征表达拼接起来，乘以权重矩阵$\mathbf{W}_k$，再输入到激活函数σ中，就得到了新的节点特征表达。

4. 最后，对新的特征表达进行L2 normalization：

$$\mathbf{h}_v^k \leftarrow \frac{\mathbf{h}_v^k}{||\mathbf{h}_v^k||_2}$$

至此，模型就完成了一次迭代的过程。将这一过程重复K次，就能得到模型最终输出的节点嵌入表达。

直观来说，每个节点都会在模型的迭代过程中获取更远、更广泛、更抽象的信息。

事实上，迭代的轮数k也就是搜索深度：第一轮迭代中，每个节点只能提取一阶邻居中的信息，而第二轮迭代中，二阶邻居的信息也借助一阶邻居的连接传播过来……以此类推。

适用于mini batch的版本

为了能够在模型上应用SGD，GraphSAGE对前向传播算法进行了微调，来让他可以适用于mini batch，同时计算多个节点的特征表达：

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, $\mathcal{N}_k : v \to 2^{\mathcal{V}}, \forall k \in \{1, ..., K\}$

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
$\mathbf{z}_u \leftarrow \mathbf{h}_u^K, \forall u \in \mathcal{B}$

简单来说，改进后算法的主要思想就是，在迭代计算所有节点的特征表达之前，先对所有节点进行多轮采样，得到每一轮中所有需要计算的节点集合，而不是在迭代过程中进行采样。

虽然论文中并未提及，但这么做的目的应当是，在完成随机取样的操作后，可以将剩下的所有操作放到GPU上进行加速。提前完成取样过程可以大大提升SGD的效率。

模型训练

无监督学习

GraphSAGE采用了如下的损失函数来应用到无监督学习中。这个损失函数鼓励相邻节点具有相似的表示，而截然不同的节点具有完全不同的表示。

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n}))$$

其中，节点v是在定长随机行走中在节点u附近一起出现的节点，σ为Sigmoid函数，P_n是一个负采样分布，而Q定义了负样例的个数。

> 由于输出向量在前向传播算法中进行了L2 normalization，故$\mathbf{z}_u^\top \mathbf{z}_v$计算的就是两个向量的余弦相似度。

重要的是，GraphSAGE与之前很多嵌入方法的区别在于，输入损失函数的嵌入特征表达$\mathbf{z}_u$并非是对每个节点训练出来的各自的特征表达，而是从各个节点相邻节点的特征中生成的。

当GraphSAGE被应用到无监督学习场景下时，他就像一个上游的"特征向量生成器"，或"嵌入表达仓库"，能为多个下游任务提供节点嵌入特征向量。

监督学习

GraphSAGE也能很容易地应用到监督学习中。当下游任务只有一个时，GraphSAGE能与下游任务连接起来，直接使用下游任务的损失函数，如交叉熵等。

代理函数

与许多传统任务不同，节点的相邻节点并没有天然的顺序可言。所以，代理函数必须不仅可训练、具有很强的表达能力，在理想上，他们最好是对称的，也即可以被应用到无序的节点集合上。

论文中共列出了如下三种代理函数：

Mean aggregator

将伪代码中的第三、第四行替换为：

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}_k \cdot \mathrm{MEAN}(\{\mathbf{h}_u^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$$

最容易想到的，当然就是Mean aggregator。Mean aggregator对抽样出的邻居节点取逐元素的均值。他几乎和GCN中的卷积计算规则是完全等价的，用线性的方式综合了邻居节点的信息。

论文中把这种基于平均的代理函数称为"卷积"的。因为他事实上就是对局部谱图卷积的一种粗略而线性地拟合。

值得一提的是，这种代理函数与论文中介绍的其他代理函数最显著的区别在于，他缺少伪代码中第五行的"拼接"操作：

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}_k \cdot \mathrm{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$$

这种拼接操作，事实上可以被看作是层与层之间简单的"残差连接"——这被证明能很大地提升模型性能。

LSTM aggregator

论文中也尝试了一种更为复杂的代理函数：LSTM aggregator。

相较于mean aggregator，LSTM aggregator具有更强大的表达能力。

然而，值得一提的是，LSTM从本质上并不是对称的，他们对数据输入的先后是敏感的，因为他们本质上是用来处理序列的。为了能让LSTM在无序集合上工作，GraphSAGE简单地把LSTM作用到节点邻居的随机排列上。

Pooling aggregator

Pooling aggregator兼具可训练性与对称性。Pooling aggregator的主要过程是这样的：被抽样的节点首先经过一层全连接层的变换，随后进行逐元素的最大值池化：

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

从原则上来说，在池化之前进行的变换可以是任意深层感知机网络，但在论文中只关注于单层网络的简单情况。

直观地看，代理函数中的多层感知机可以被看做为邻居节点表达计算计算特征值的一组函数，而对每一特征值的池化操作让模型很有效地提取出节点邻居不同方面的信息。

值得一提的是，从原则上来说，所有对称的向量值函数都是可行的，但是在实验过程中，最大值池化与平均值池化并没有表现出明显差异，所以在接下来的实验中，作者专注于最大支池化。

模型评估

表现

文章中，GraphSAGE共在三个数据集上进行了测试，从两个方面来评估模型性能：增长的图上的归纳式学习与图之间的泛化。GraphSAGE在这些测试中都取得了很好的表现。

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

在实验的表现中，总的来说，基于LSTM与池化的代理函数表现得更好。而在LSTM与池化代理函数的比较中，他们的性能没有显著差别，但池化代理函数的计算速度要明显高于LSTM代理函数。

理论分析

在论文中，GraphSAGE被证明，即使他从本质上是基于节点特征的，但他的确拥有学习图的结构特征的能力。

在论文中，作者证明了GraphSAGE有能力预测节点的集聚系数 *(clustering coefficient)* 。论文中给出了相关的定理，表明只要每个节点的特征向量不同，且模型的维度足够高，总存在一组系数，使得GraphSAGE能够以任意的精确度拟合节点的群聚系数。

不足与改进

在论文中，作者提及了GraphSAGE的不足与未来可以改进之处：

1. 拓展模型，使其可以作用到无向图或是multimodal graph。
2. 尝试非均匀的邻居节点采样函数 *(non-uniform neighbourhood sampling functions)* ，甚至在训练的过程中学习这样的采样函数。

感想

GraphSAGE是一个很优秀的模型，也是一个很好的平台与框架。他利用取样 *(sampling)* 的方法，很好地解决了图结构的灵活多变带来的难题：节点度数的任意性。利用取样，取出固定个数的相邻节点后，原本由于输入大小不确定而难以应用到图节点上的代理函数现在可以很好的在图上发挥作用。

相比于先前已有的很多嵌入模型，GraphSAGE的独特与出彩之处在于他关注重心的转变。原有的模型大多专注于得到节点的嵌入向量，因而不具有很好的泛化能力，也很难应用于增长中的图，在现实中作用甚微。而GraphSAGE关注于嵌入向量的生成函数——只要得到了能够从邻居节点特征向量中提取出节点嵌入表达的函数，那么得到节点嵌入向量只是轻而易举的事情。因为这样的转变，GraphSAGE具有了更好的泛化能力，可以被应用到归纳式学习中，很好地生成从未见过的节点的嵌入表达。这其中，颇有些"授人以鱼不如授人以渔"的思想。

而事实上，在与图相关的任务中，这种转变也是符合直觉的。很多图节点嵌入模型都从词嵌入模型得到了灵感，而词嵌入模型中之所以可以只关注于得到词的嵌入向量(GloVe)，是因为自然语言处理任务中的最小单位——词，很少会需要拓展与增加。不管是什么语言，词或者字的总量都是一定的，只要学习出每一个单词的表达，就可以很好地应用到大多数任务中。然而，在与图相关的任务中，图的增长与拓展是非常常见的，我们很少会拥有所有的节点，很多时候节点的数量甚至是无穷的。所以，图上的模型**必须拥有很好地处理新节点的能力**。在很多应用场景中，学习出固定的图中各节点固定的嵌入表达甚至没有很大意义。

而关于GraphSAGE未来的改进与深入研究方向，有一个便是尝试探究更多类型的代理函数。如作者在论文中所提到的，GraphSAGE事实上是一种框架，一个平台，在其建立的框架(sample and aggregate)之内可以应用不同的代理函数。在论文中，作者已提出了三种，而更多的可能也亟待探寻。比如，原本是应用于序列任务的LSTM作为代理函数取得了很好的效果，那么在序列处理任务中同样表现出色的RNN+Attention机制能否也被应用在GraphSAGE中，取得更好的表现呢?

除此以外，正如作者在论文中两次提到的，另一个未来研究方向是对采样函数的探索。一方面，可以尝试non-uniform sampling function对模型表现的影响。另一方面也非常直观：既然GraphSAGE中，将原本固定的代理函数拓展为可训练的代理函数，而采样过程作为模型最主要的两个过程之一，采样函数是否也可以拓展为可训练的呢?

如果继续分析一下这个问题。我们需要一个怎样的采样函数呢？直观来说，我们希望采样得到的样本更能体现出节点的特征，能够提取出更多与节点有关的信息，换言之，他们与节点的关联度更大。因而，如果从这个角度来看，在理想情况下，我们的采样函数应当更倾向于取出与节点关联度更大的相邻节点。实现这样的采样函数的一个最直观的方法是根据关联程度为邻居节点分配不同的权重，并基于权重进行采样——这事实上是另一形式的Attention机制。

然而，若直接将权重作为取样概率的话，采样函数将会是不可训练的，至少并不容易训练。为了便于训练，权重系数必须保留。

如果我们直接放弃"采样"的概念，对邻居节点直接做加权平均，那采用这样的采样函数的GraphSAGE模型就基本与GAT一致了。**为了保留"采样"的概念，我构想了这样一种采样函数：**

设要采样的节点为u，与u相邻节点的集合为$\mathcal{V}_u$。首先，计算相邻节点的注意力系数：

$$e_v = a(\mathbf{h}_u^{k-1}, \mathbf{h}_v^{k-1}), v \in \mathcal{V}_u,$$

随后，对注意力系数进行排序，选取系数最大的前S个节点为$\mathcal{N}(u)$。对每个节点的特征向量乘以$softmax$后的注意力系数，得到采样函数的最终输出：

$$\mathcal{N}^{'}(u) = \{\alpha_v \mathbf{h}_v^{k-1} : v \in \mathcal{N}(u)\}$$

其中，

$$\alpha_v = \mathrm{softmax}(e_v) = \frac{\exp(e_v)}{\sum_{i \in \mathcal{N}(u)} \exp(e_k)}$$

从某种意义上来说，这样的采样函数其实是修改的Attention机制，不作加权平均，并丢弃了注意力系数较低的量。这可以被看做某种意义上的，注意力机制中的"skip connection"。

然而，直观来说，由于采样函数对一些特征向量的丢弃，这代表这他们的注意力系数同样也不再被考虑，这可能使采样函数很难被训练，模型收敛速度变慢甚至不收敛。为了保证采样函数的训练，采样个数S应当基于节点度数选取更合适的值。

另一方面，输出的向量集合中，各个特征向量乘以了规范后的注意力系数α_v，这导致向量不再为单位向量，向量模长可能会被压缩得很小，这很可能会导致代理函数有较大的性能损失。