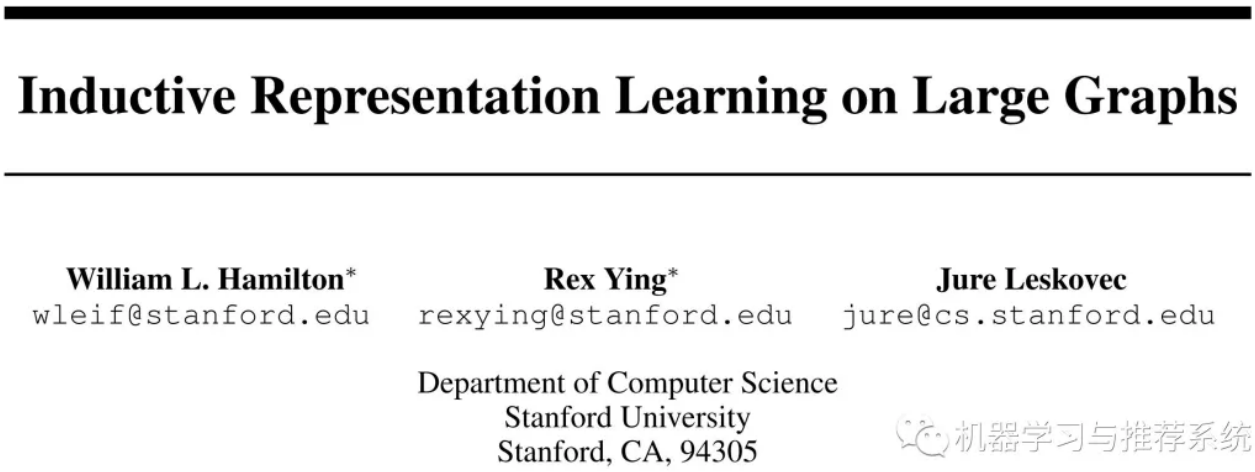
【Embedding】GraphSAGE：不得不学的图网络

原创 silver 机器学习与推荐系统 2020-08-29

收录于话题
#embedding 7 #GNN 28



Inductive Representation Learning on Large Graphs

William L. Hamilton* wleif@stanford.edu
Rex Ying* rexying@stanford.edu
Jure Leskovec jure@cs.stanford.edu

Department of Computer Science
Stanford University
Stanford, CA, 94305

GraphSAGE 是 17 年的文章了，但是一直在工业界受到重视，最主要的就是它论文名字中的两个关键词：inductive 和 large graph。今天我们就梳理一下这篇文章的核心思路，和一些容易被忽视的细节。

为什么要用 GraphSAGE

大家先想想图为什么这么火，主要有这么几点原因，图的数据来源丰富，图包含的信息多。所以现在都在考虑如何更好的使用图的信息。

那么我们用图需要做到什么呢？最核心的就是利用图的结构信息，为每个 node 学到一个合适的 embedding vector。只要有了合适的 embedding 的结果，接下来无论做什么工作，我们就可以直接拿去套模型了。

在 GraphSAGE 之前，主要的方法有 DeepWalk，GCN 这些，但是不足在于需要对全图进行学习。而且是以 transductive learning 为主，也就是说需要在训练的时候，图就已经包含了要预测的节点。

考虑到实际应用中，图的结构会频繁变化，在最终的预测阶段，可能会往图中新添加一些节点。那么该怎么办呢？GraphSAGE 就是为此而提出的，它的核心思路其实就是它的名字 GraphSAGE = Graph Sample Aggregate。也就是说对图进行 sample 和 aggregate。

GraphSAGE 的思路

我们提到了 sample 和 aggregate，具体指的是什么呢？这个步骤如何进行？为什么它可以应用到大规模的图上？接下来就为大家用通俗易懂的语言描述清楚。

顾名思义，sample 就是选一些点出来，aggregate 就是再把它们的信息聚合起来。那么整个流程怎么走？看下面这张图：

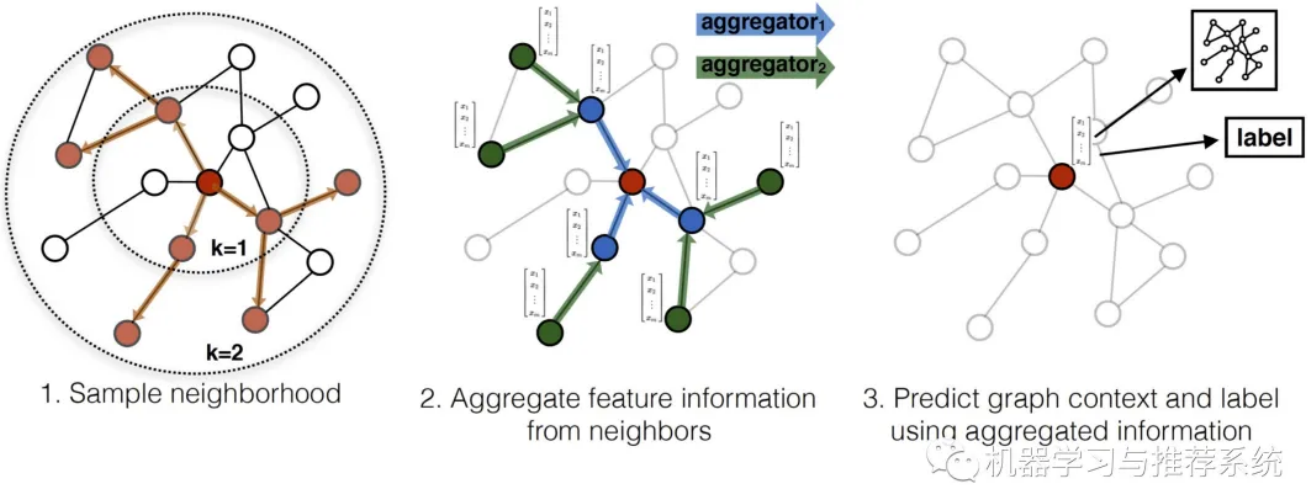


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

我们在第一幅图上先学习 sample 的过程。假如我有一张这样的图，需要对最中心的节点进行 mebedding 的更新，先从它的邻居中选择 S1 个（这里的例子中是选择 3 个）节点，假如 K=2，那么我们对第 2 层再进行采样，也就是对刚才选择的 S1 个邻居再选择它们的邻居。

在第二幅图上，我们就可以看到对于聚合的操作，也就是说先拿邻居的邻居来更新邻居的信息，再用更新后的邻居的信息来更新目标节点（也就是中间的红色点）的信息。听起来可能稍微有点啰嗦，但是思路上并不绕，大家仔细梳理一下就明白了。

第三幅图中，如果我们要预测一个未知节点的信息，只需要用它的邻居们来进行预测就可以了。

我们再梳理一下这个思路：如果我想知道小明是一个什么性格的人，我去找几个他关系好的小伙伴观察一下，然后我为了进一步确认，我再去选择他的小伙伴们的其他小伙伴，再观察一

下。也就是说，通过小明的小伙伴们的小伙伴，来判断小明的小伙伴们是哪一类人，然后再根据他的小伙伴们，我就可以粗略的得知，小明是哪一类性格的人了。

GraphSAGE 思路补充

现在我们知道了 GraphSAGE 的基本思路，可能小伙伴们还有一些困惑：单个节点的思路是这样子，那么整体的训练过程该怎么进行呢？至今也没有告诉我们 GraphSAGE 为什么可以应用在大规模的图上，为什么是 inductive 的呢？

接下来我们就补充一下 GraphSAGE 的训练过程，以及在这个过程它有哪些优势。

首先是考虑到我们要从初始特征开始，一层一层的做 embedding 的更新，我们该如何知道自己需要对哪些点进行聚合呢？应用前面提到的 sample 的思路，具体的方法来看一看算法：

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

首先看算法的第 2-7 行，其实就是一个 sample 的过程，并且将 sample 的结果保存到 B 中。接下来的 9-15 行，就是一个 aggregate 的过程，按照前面 sample 的结果，将对应的邻居信息 aggregate 到目标节点上来。

细心的小伙伴肯定发现了 sample 的过程是从 K 到 1 的（看第 2 行），而 aggregate 的过程是从 1 到 K 的（第 9 行）。这个道理很明显，采样的时候，我们先从整张图选择自己要给哪些节点 embedding，然后对这些节点的邻居进行采样，并且逐渐采样到远一点的邻居上。

但是在聚合时，肯定先从最远处的邻居上开始进行聚合，最后第 K 层的时候，才能聚合到目标节点上来。这就是 GraphSAGE 的完整思路。

那么需要思考一下的是，这么简单的思路其中有哪些奥妙呢？

GraphSAGE 的精妙之处

首先是为什么要提出 GraphSAGE 呢？其实最主要的是 inductive learning 这一点。这两天在几个讨论群同时看到有同学对 transductive learning 和 inductive learning 有一些讨论，总体来说，inductive learning 无疑是可以在测试时，对新加入的内容进行推理的。

因此，GraphSAGE 的一大优点就是，训练好了以后，可以对新加入图网络中的节点也进行推理，这在实际场景的应用中是非常重要的。

另一方面，在图网络的运用中，往往是数据集都非常大，因此 mini batch 的能力就非常重要了。但是正因为 GraphSAGE 的思路，我们只需要对自己采样的数据进行聚合，无需考虑其它节点。每个 batch 可以是一批 sample 结果的组合。

再考虑一下聚合函数的部分，这里训练的结果中，聚合函数占很大的重要性。关于聚合函数的选择有两个条件：

- 首先要可导，因为要反向传递来训练目标的聚合函数参数；
- 其次是对称，这里的对称指的是对输入不敏感，因为我们在聚合的时候，图中的节点关系并没有顺序上的特征。

所以在作者原文中选择的都是诸如 Mean，max pooling 之类的聚合器，虽然作者也使用了 LSTM，但是在输入前会将节点进行 shuffle 操作，也就是说 LSTM 从序列顺序中并不能学到什么知识。

此外在论文中还有一个小细节，我初次看的时候没有细读论文，被一位朋友指出后才发现果然如此，先贴一下原文：

Mean aggregator. Our first candidate aggregator function is the mean operator, where we simply take the elementwise mean of the vectors in $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$. The mean aggregator is nearly equivalent to the convolutional propagation rule used in the transductive GCN framework [17]. In particular, we can derive an inductive variant of the GCN approach by replacing lines 4 and 5 in Algorithm 1 with the following:[4]

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}). \quad (2)$$

这里的 lines 4 and 5 in Algorithm 1，也就是我们前面给出的算法中的第 11 和 12 行。

也就是说，作者在文中提到的 GraphSAGE-GCN 其实就是用上面这个聚合函数，替代掉其它方法中先聚合，再 concat 的操作，并且作者指出这种方法是局部谱卷积的线性近似，因此将其称为 GCN 聚合器。

来点善后工作

最后我们就简单的补充一些喜闻乐见，且比较简单的东西吧。用 GraphSAGE 一般用来做什么？

首先作者提出，它既可以用来做无监督学习，也可以用来做有监督学习，有监督学习我们就可以直接使用最终预测的损失函数为目标，反向传播来训练。那么无监督学习呢？

其实无论是哪种用途，需要注意的是图本身，我们还是主要用它来完成 embedding 的操作。也就是得到一个节点的 embedding 后比较有效的 feature vector。那么做无监督时，如何知道它的 embedding 结果是对是错呢？

作者选择了一个很容易理解的思路，就是邻居的关系。默认当两个节点距离相近时，就会让它们的 embedding 结果比较相似，如果距离远，那 embedding 的结果自然应该区别较大。这样一来下面的损失函数就很容易理解了：

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

z_v 表示是目标节点 u 的邻居，而 v_n 则表示不是，P_n(v) 是负样本的分布，Q 是负样本的数量。

那么现在剩下唯一的问题就是邻居怎么定义？

作者选择了一个很简单的思路：直接使用 DeepWalk 进行随机游走，步长为 5，测试 50 次，走得到的都是邻居。

总结

实验结果我们就不展示了，其实可以看到作者在很多地方都用了一些比较 baseline 的思路，大家可以在对应的地方进行更换和调整，以适应自己的业务需求。

后面我们也会继续分享 GNN 和 embedding 方面比较经典和启发性的一些 paper，欢迎大家持续关注~~~

PS：看到这里的都是真爱了，悄悄说一句谢谢大家的信任，我这个懒人都这么久没更了，每天还有新小伙伴们持续关注，实在是太感动了，哭花脸o(╥﹏╥)o

点个再看嘛，不点就点个赞嘛，实在不点那我也只能算了嘛，再哭花脸(╥╯^╰╥)

往期回顾

一文搞懂 PyTorch 内部机制
一篇长文学懂 pytorch
一个例子告诉你，在 pytorch 中应该如何并行生成数据