浅谈图表征学习-从word2vec 到 DeepWalk

原创 tau Tau的学习笔记 2020-05-02

今天来分享下图表征学习领域早期一篇很有名的工作**DeepWalk**，*"DeepWalk: online learning of social representations"*， 这篇文章发表在14年KDD上，目前谷歌显示引用达到3058。在**DeepWalk**之后诞生了很多类似的方法， 比如node2vec, LINE, BiNE等等。

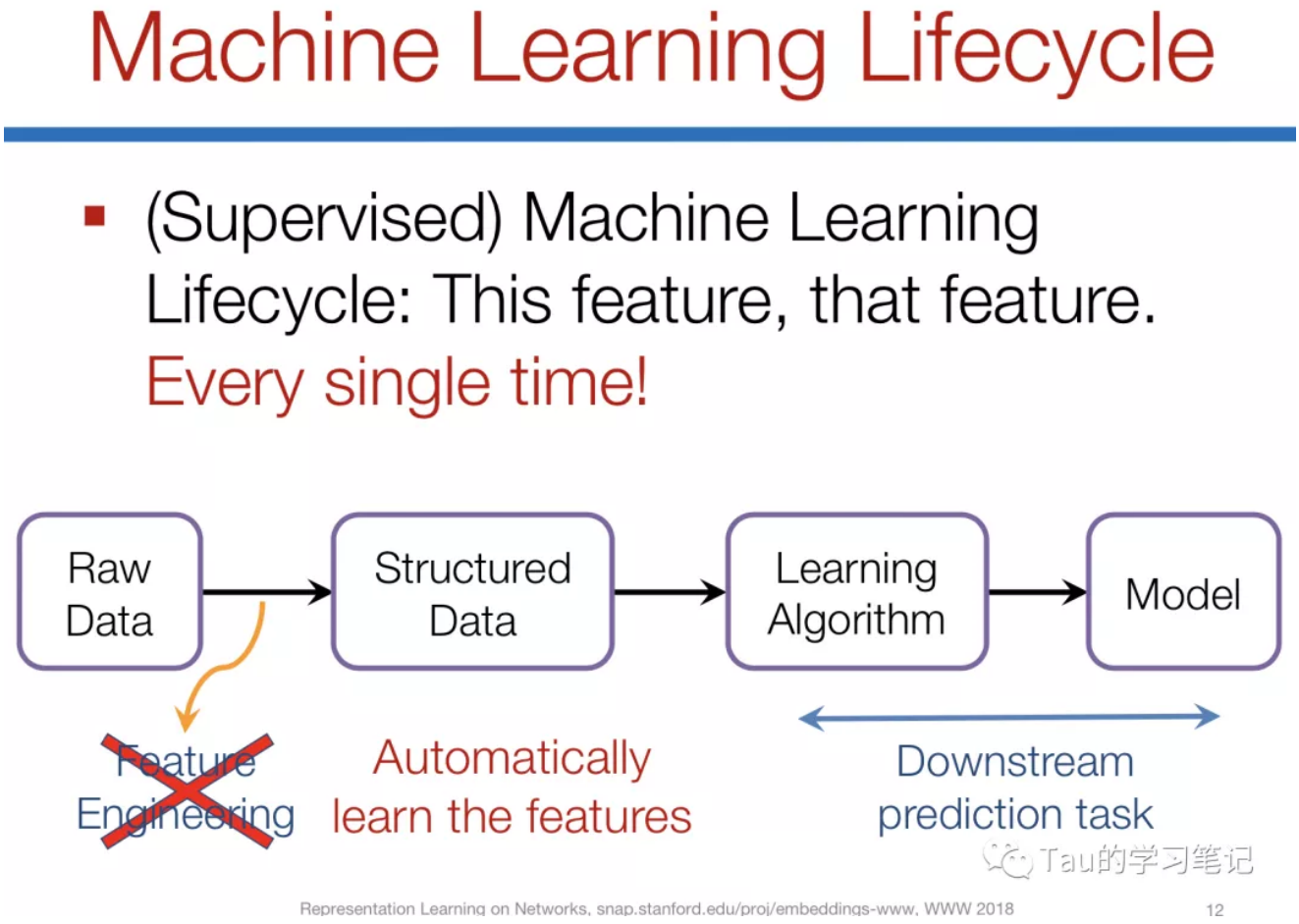
为什么要图表征

常见机器学习模型比如神经网络，SVM 等，输入的都是规则的数据，比如图像，文本，但是却难以直接处理图数据。为了解决这个问题，之前采用的方法主要有：

1. 使用基于矩阵分解的方法, 比如 SVD, PCA分解连接矩阵.
2. 使用手工构造特征的方法, 比如将pagerank, degree等值拼接起来作为特征向量。

第二种方法可以看这篇文章作为例子，*"Collective Spammer Detection in Evolving Multi-Relational Social Networks"*。

但是这两种方法有明显的不足，第一种方法时间复杂度至少是二次的，可拓展性差。第二种方法是特征工程，需要经验且耗时间。如下图，我们希望能够自动的学习到图的特征，比如节点的特征，边的特征，然后用于下游的任务。



www 2018 Representation Leanrning on Networks, Jure

从word2vec 到 node embedding

DeepWalk是一种节点embedding 的方法。在聊这个之前，先提一下word2vec。word2vec 是自然语言处理里的一个技术。关于word2vec，有篇谷歌的文章 *"Distributed Representations of Words and Phrases and their Compositionality"*。word2vec的思想就是，通过在大量的文本上训练，得到word在潜空间的表示即一个向量，然后这些向量用于下游的任务，而不是使用one-hot编码word。按照表征学习的理论，直接用one-hot编码训练的过程中，模型会学得word的表征。但是这个表征因为数据集的限制，没有在大规模的语料上训练得到的表征好。word2vec 就相当于是一个预训练模型。

回过头来看，word2vec的思想和上面提到的图表征学习的思想是很相近的，都是希望捕获原始数据信息(图的结构信息，文本中word之间的信息)，编码成向量，用于下游的任务。

DeepWalk的作者，利用了巧妙的变换，然后用到了word2vec的技术。即在图上随机游走，获取到很多条节点的序列，比如（node1, node3, node5),(node1, node5, node6)...然后将这些序列看成sentence，node看成word，然后用word2vec的技术，为这些节点生成表征向量。

具体细节

文章中的伪代码如下所示：

Algorithm 1 DEEPWALK(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, \text{t})$
    SkipGram$(\Phi, \mathcal{W}_{v_i}, w)$
  end for
end for

RandomWalk

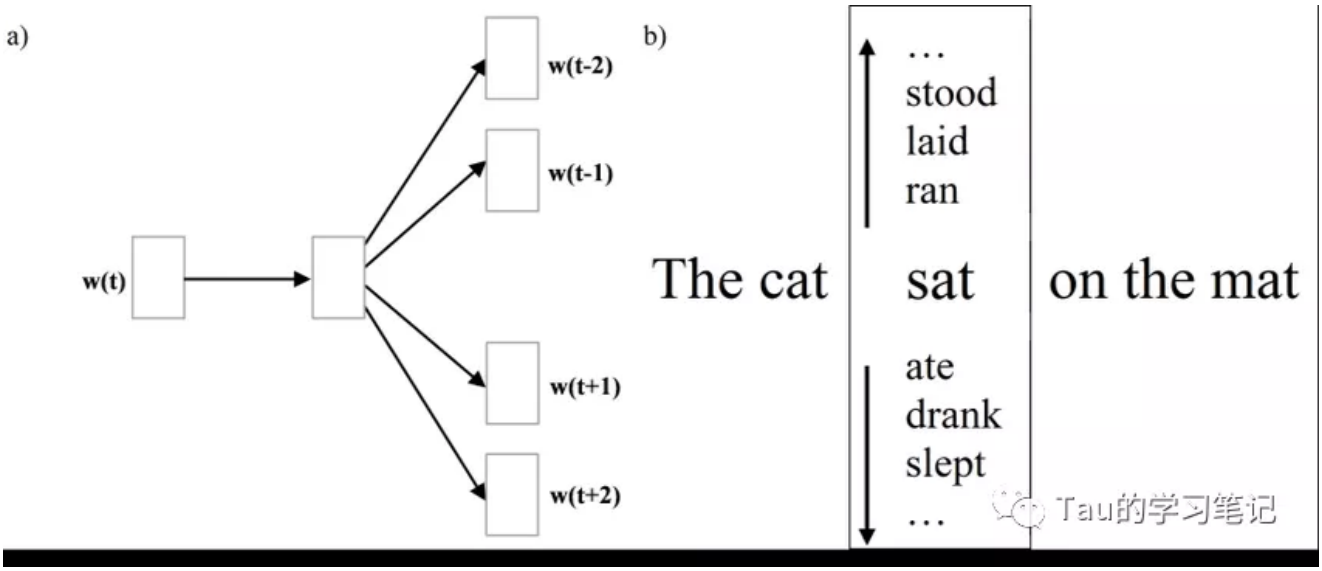
DeepWalk 采取的随机游走的策略比较简单，对于初始节点v_i, 无偏的随机选择其一个邻居节点作为下一个节点, 不断重复上面的过程，直到序列W_{v_i}长度达到 t. 官方给出的代码RandomWalk如下：有一个alpha参数，即返回初始节点的概率

```python
def random_walk(self, path_length, alpha=0, rand=random.Random(), start=None):
  """ Returns a truncated random walk.

      path_length: Length of the random walk.
      alpha: probability of restarts.
      start: the start node of the random walk.
  """
  G = self
  if start:
    path = [start]
  else:
    # Sampling is uniform w.r.t V, and not w.r.t E
    path = [rand.choice(list(G.keys()))]

  while len(path) < path_length:
    cur = path[-1]
    if len(G[cur]) > 0:
      if rand.random() >= alpha:
        path.append(rand.choice(G[cur]))
      else:
        path.append(path[0])
    else:
      break
  return [str(node) for node in path]
```


https://github.com/phanein/deepwalk/blob/master/deepwalk/graph.py

Skip-gram & CBOW

skip-gram 和 CBOW 是两种 word2vec 算法，前者是通过中间的词预测两边的词，后者是通过周边的词预测中间的词。就实际效果而言，skip-gram 效果更好。skip-gram 的结构如下图所示。



https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c

这两种方法的优化目标，如下图所示：

Skip-gram:

$$\text{minimize } J = -\log P(w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m} | w_c)$$

CBOW:

$$\text{minimize } J = -\log P(w_c | w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m})$$

两种方法有相似之处，都是使得语料里面一起出现的word co-occur的概率最大。DeepWalk 采用了 Skip-gram。

negative sampling & hierarchical Softmax

negative sampling 和 hierarchical Softmax 都是为了解决上面的方法优化遇到的算法复杂度的问题。

上面两个方法在优化的过程中，都会遇到一个问题，就是利用softmax 计算概率需要遍历整个词袋，对图而言就是所有节点集合。

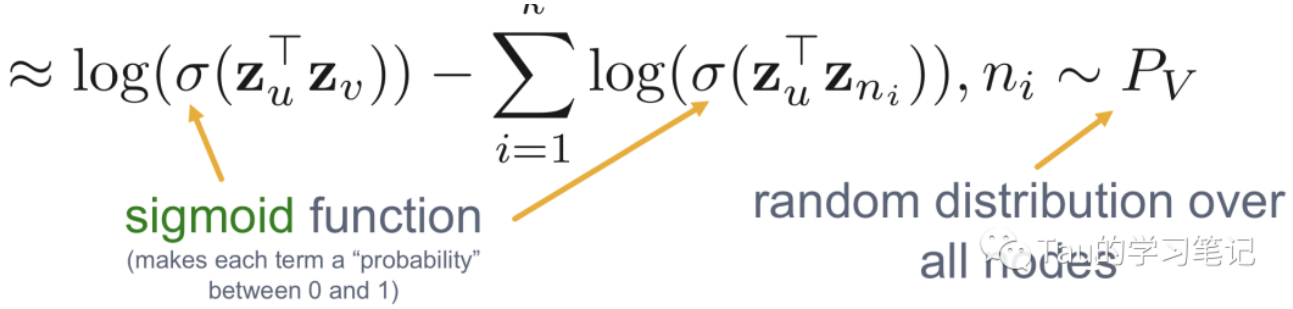
举例来说，将word或节点embedding 的向量点乘当作两个word或者节点的相似度，有的学者称这个近似是两个word或节点 co-occur的概率，那么计算word或者节点u出现的情况下，word或者节点v出现的概率$P(v|u)$计算公式如下所示：Z_u代表u对应的embedding 向量。

$$\log\left(\frac{\exp\left(\mathbf{z}_u^\top \mathbf{z}_v\right)}{\sum_{n \in V} \exp\left(\mathbf{z}_u^\top \mathbf{z}_n\right)}\right)$$

计算这个值需要遍历整个V, 复杂度为$O(V)$

negative sampling

利用negative-sampling 上面的公式可以用下面的公式近似：



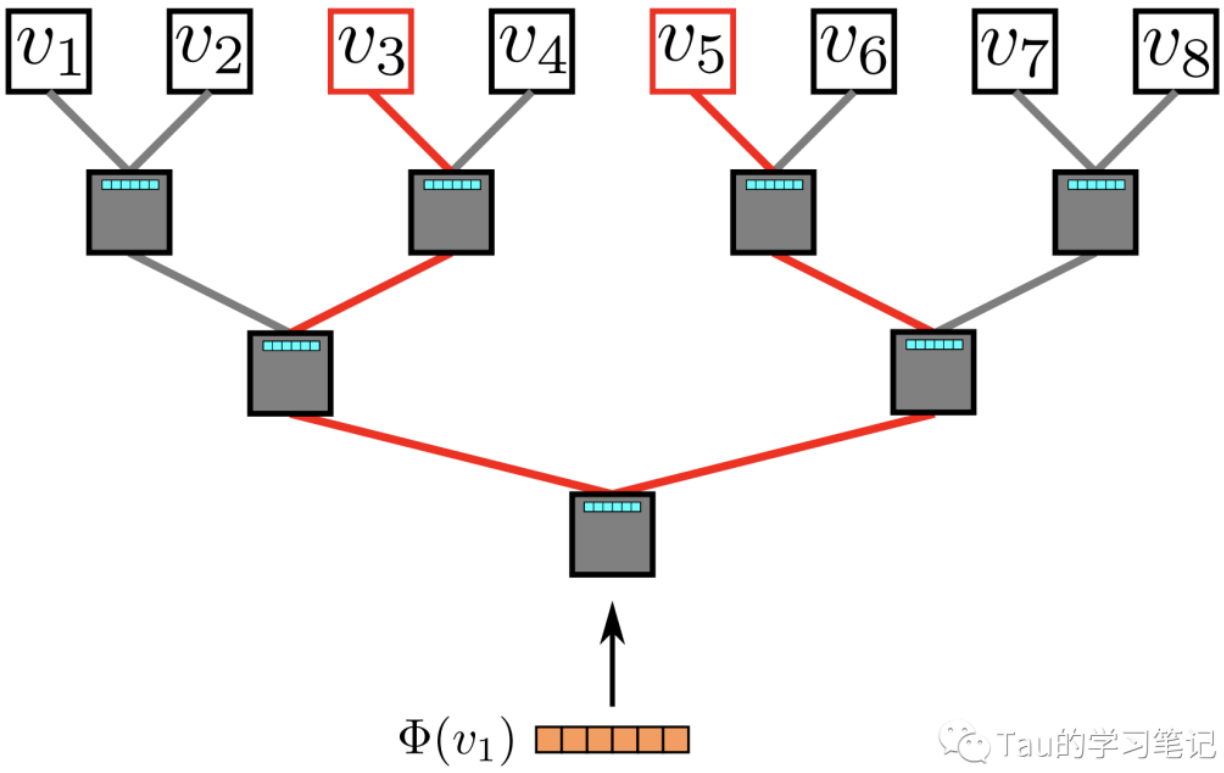
图片来自cs224w的slide

negative-sampling 的基本思想是，对于整个词袋，和某个词一起出现的词的个数是很少的，那么不需要区分整个词袋，只要能把正例即希望预测出的词从另外少数随机选择的negative samples 分出来就行了。

hierarchical Softmax

hierarchical Softmax 是一种方法，代替之前的softmax 计算。理论上能够将之前的softmax 函数计算复杂度$O(V)->O(log_2V)$

如下图所示，构建一个二叉树，每一个词或者分类结果都分布在二叉树的叶子节点上，在做实际分类的时候，从根节点一直走到对应的叶子节点，在每一个节点都做一个二分类（需要通过学习）。假设这是一颗均衡二叉树，并且词袋的大小是$|V|$，那么从根走到叶子节点只需要进行log_2V次计算. 具体的计算方法就是将每层的概率累乘起来，即得到概率值。

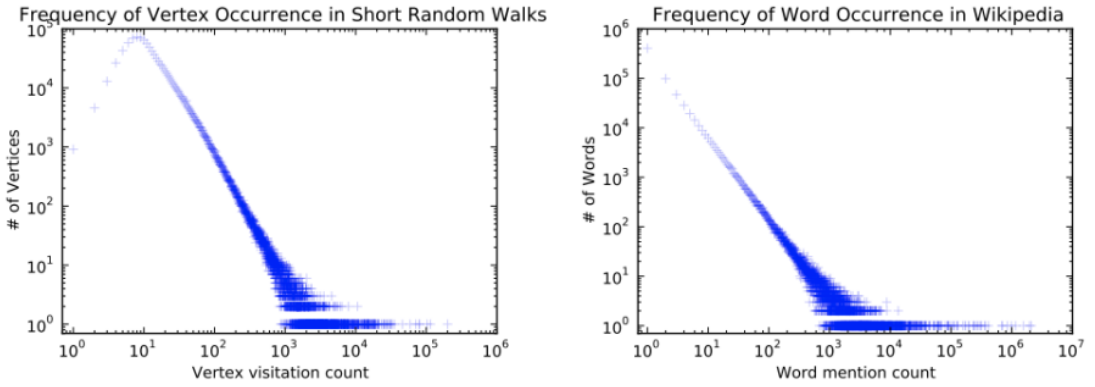


图片来自DeepWalk论文

hierarchy sotfmax 还可以进一步优化，高频词用较短的路径到达，低频词用较长的路径到达，可以进一步降低整个训练过程的计算量。DeepWalk 采用了hierarchical Softmax。

总结

DeepWalk作者指出，在社交网络图中随机游走，节点出现的规律和单词在语料中出现的频率规律相似即power-law 或者叫2-8定律，然后将word2vec 技术应用于节点表征学习。



(a) YouTube Social Graph (b) Wikipedia Article Text

Figure 2: The power-law distribution of vertices appearing in short random walks (2a) follows a power-law, much like the distribution of words in natural language (2b).

图片来自DeepWalk论文

整个算法具有良好的可拓展性，在随机游走采样阶段，虽然在图上随机游走会有missing cache 的问题, 但是可以并行的去做, 整体效率是可以的。同时节点的出现频次符合指数分布，大部分低频节点都分布在长尾。多台机机器同时训练也不太会发生冲突，文中提出可以采用异步的随机梯度下降 (ASGD)。笔者试过在8百万节点的社交网络图上跑过DeepWalk，训练完成花费十几个小时。

但是其也存在模型参数量大，随机游走策略较简单，和难以应对动态图的缺点。

参考资料：

1. http://web.stanford.edu/class/cs224w/slides/07-noderepr.pdf cs224w：Machine Learning with Graphs 的 slide
2. https://arxiv.org/pdf/1403.6652.pdf DeepWalk论文
3. http://web.stanford.edu/class/cs224n/slides/ cs224n-2020-lecture01-wordvecs1.pdf cs224n 第一节课的slide, 谈到了word2vec