

图神经网络导论

节点表征学习

授课教师：周晟

浙江大学 软件学院

2023.11.14



课程内容

- 1 节点表征学习
- 2 基于矩阵分解的节点表征学习
- 3 从 Word2Vec 到 Node2Vec
- 4 随机游走与矩阵分解的等价性



1 节点表征学习

2 基于矩阵分解的节点表征学习

3 从 Word2Vec 到 Node2Vec

4 随机游走与矩阵分解的等价性



节点表征学习的研究动机

研究动机：

① 直接使用原始网络的问题

- ① 结构稀疏（邻接矩阵稀疏）
- ② 信息多样（网络中除了结构信息，往往包含属性、边等信息）
- ③ 下游任务难以试用

② 将节点表示为特征向量的优势

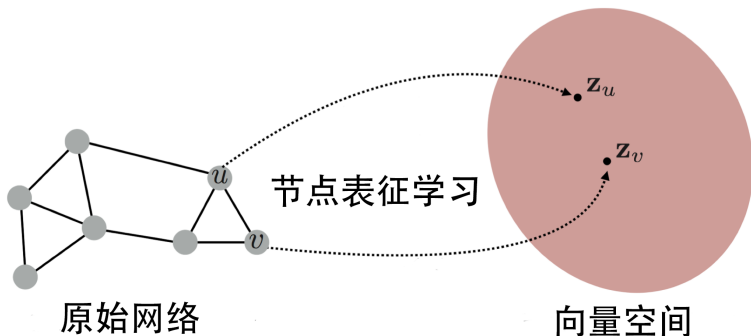
- ① 低维稠密向量便于存储
- ② 多种信息使用同一模式表示
- ③ 可以广泛应用于下游任务



节点表征学习

节点表征学习

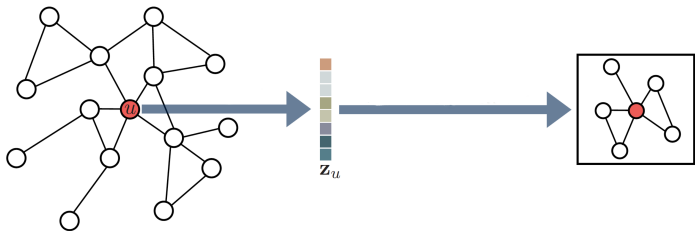
节点表征学习 (Network/Node Embedding) 是将网络的节点表示成低维向量空间中的一个点, 使得原始网络中的信息可以最大程度地保留到低维向量空间中。



节点表征学习

节点的低维稠密向量希望包含如下信息：

- ① 邻近性 (Proximity)
- ② 结构特征 (中心性等)
- ③ 节点属性
- ④ 边信息
- ⑤ ...



- 1 节点表征学习
- 2 基于矩阵分解的节点表征学习
- 3 从 Word2Vec 到 Node2Vec
- 4 随机游走与矩阵分解的等价性



矩阵分解

矩阵分解

矩阵分解是将一个矩阵拆解为数个矩阵的乘积的运算，其目标是使得矩阵乘积与原始矩阵尽量接近。

The diagram shows the matrix equation $W \times H \approx V$. Matrix W is a 4x2 grid, Matrix H is a 2x6 grid, and Matrix V is a 4x6 grid. The multiplication is represented by a large 'X' and the approximation by a tilde symbol.

- V: 图特征
- W,H: 节点的特征



基于矩阵分解的节点表征学习

研究思路

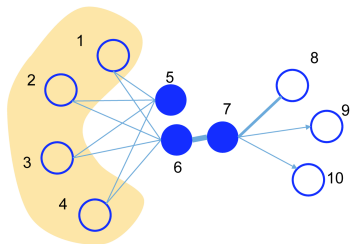
将节点之间的关系表示为关系矩阵 $V \in \mathcal{R}^{N \times N}$ ，然后使用矩阵分解得到节点的表征 $W, H^T \in \mathcal{R}^{N \times d}$ 。

节点之间的关系：

- ① 邻居关系
- ② 二阶邻近关系
- ③ 高阶邻近关系
- ④ 非对称邻近关系
- ⑤ ...



低阶邻近关系



- ① 一阶邻近关系 (First Order Proximity) 是指节点之间的邻居关系，按照取值可分为离散和连续。
- ② 二阶邻近关系 (Second Order Proximity) 是指节点的邻居之间的相似性。

LINE: Large-scale Information Network Embedding. (WWW 2015)[Tang et al., 2015]



低阶邻近关系

LINE: 一阶邻近关系驱动的特征学习

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)},$$

$$\mathcal{O}_1 = - \sum_{(i,j) \in E} \omega_{ij} \log p_1(v_i, v_j),$$

LINE: 二阶邻近关系驱动的特征学习

$$p_2(v_j | v_i) = \frac{\exp(\vec{u}_j^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k^T \cdot \vec{u}_i)},$$

$$\mathcal{O}_2 = - \sum_{(i,j) \in E} w_{ij} \log p_2(v_j | v_i).$$

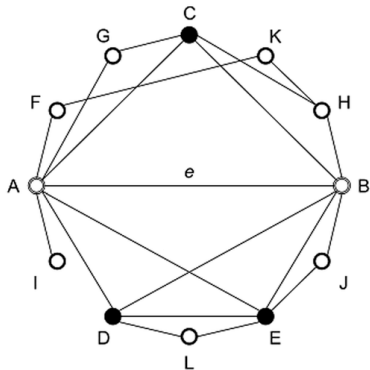


高阶邻近关系：Common Neighbors

Common Neighbors

Common Neighbors 定义为两个节点的共同邻居（无向/有向邻居）的个数：

$$S^{CN} = A^2$$



● Common neighbors

○ Non-common neighbors



高阶邻近关系：Adamic-Adar

Adamic-Adar

Adamic-Adar 是 Common Neighbor 的一种扩展，在计算邻居的过程中，每个节点被赋予一个反比于节点度的权重：

$$A(x, y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$

矩阵形式为：

$$\mathbf{S}^{AA} = \mathbf{A} \cdot \mathbf{D} \cdot \mathbf{A}$$

$$\mathbf{D}_{ii} = 1 / \sum_j (A_{ij} + A_{ji})$$

高阶邻近关系：Katz

Katz Centrality

Katz Centrality 描述了节点在网络中的中心度，在节点邻居的基础上计算所有经过两点之间的路径的个数：

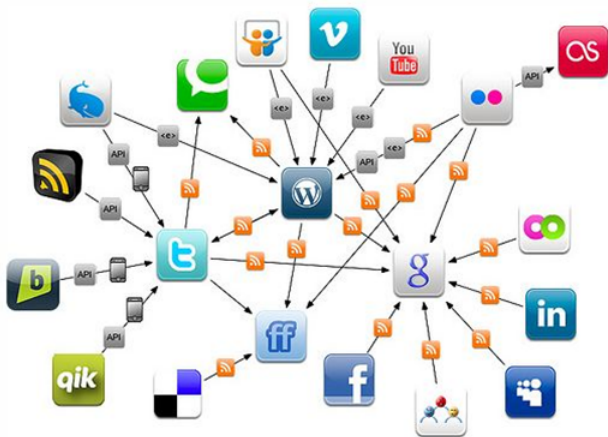
$$C_{\text{Katz}}(i) = \sum_{k=1}^{\infty} \sum_{j=1}^n \alpha^k (A^k)_{ji}$$

Katz Distance

Katz Distance 是两个节点间所有路径的加权和，其中每条路径的权重是路径长度的指数函数：

$$\mathbf{S}^{\text{Katz}} = \sum_{l=1}^{\infty} \beta \cdot \mathbf{A}^l$$

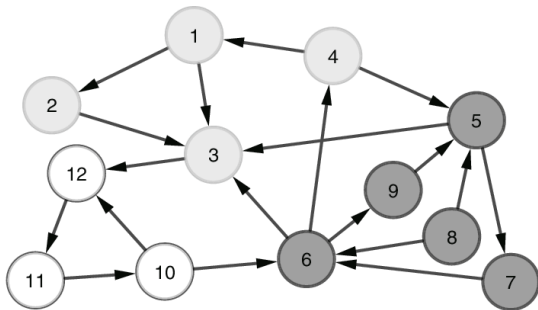
高阶邻近关系: Rooted PageRank



如何分析网页的相关性和重要性?



PageRank 背景



由页面构建的有向图

节点表示页面，边表示页面之间的跳转关系



PageRank 基本假设

PageRank 基本假设

网页中的每一次超链接（跳转）表示源页面（Source Page）到目标页面（Target Page）的一次投票（Vote），一个重要的页面往往能收到更多的投票。

直接计算每个页面的输入链接？



PageRank 数学原理

PageRank 数学原理

PageRank 本质上是一种稳态传播模型，它假设有向图上的每一条边，都表示节点重要性的流动，最终的节点重要性由稳态决定。

边的重要性

如果一个节点的重要性是 r_i ，它有 d_i 条出度边，则每一条边上传播的重要性为 $\frac{r_i}{d_i}$ ，可以用矩阵 $M \in \mathcal{R}^{N \times N}$ 表示，其中 $M_{ij} = \frac{1}{d_j}$

节点的重要性

每个节点的重要性 r_i 是所有入度边上传播的重要性的和，归一化后满足 $\sum_i r_i = 1$

网络的稳态

当网络满足稳态时，节点的重要性满足：

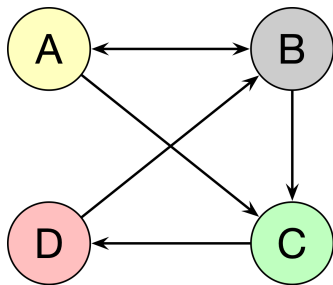
$$r = M \cdot r$$

排序向量 r 是节点之间转移概率矩阵 M 的特征向量（对应的特征值为 1）。

M 矩阵也称为随机邻接矩阵（Stochastic Adjacency Matrix）



PageRank 数学原理



$$r_A = r_B/2$$

$$r_B = r_A/2 + r_D$$

$$r_C = r_A/2 + r_B/2$$

$$r_D = r_C$$

	A	B	C	D
A	0	$r_A/2$	$r_A/2$	0
B	$r_B/2$	0	$r_B/2$	0
C	0	0	0	r_C
D	0	r_D	0	0



随机游走的网络爬虫：

- ① t 时刻，爬虫处于第 i 个页面
- ② $t+1$ 时刻，爬虫从第 i 个页面的所有出边中随机选择一条边
- ③ $t+1$ 时刻，爬虫从选择的这条边进行跳转，达到第 j 个页面
- ④ 爬虫进行无限次的上述操作

页面停留概率

给定随机游走的网络爬虫，定义 t 时刻 N 维页面停留概率 $p(t)$ ，第 i 维表示 t 时刻停留在第 i 个页面的概率。

在稳定状态下，概率分布 $p(t)$ 需满足：

$$p(t+1) = M \cdot p(t) = p(t)$$

$p(t)$ 也称为随机游走的稳态分布 (Stationary Distribution of random walk)



PageRank 缺陷

PageRank 是否一定会收敛?

PageRank 存在的缺陷:

- ① 部分节点没有出边
- ② 部分节点形成局部子图, 且与网络的其他部分没有连接

解决方案

在每一次随机游走的过程中, 有一定的概率 β 继续从当前节点的出边中选择一条进行跳转, 还有 $1 - \beta$ 的概率随机跳转到任意页面。这种条件下节点的重要性定义为:

$$r_j = \sum_{i \rightarrow j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

从 PageRank 到 Rooted PageRank

Rooted PageRank

Rooted PageRank 是 PageRank 算法稳态收敛后从节点随机游走到另一个节点的概率：

$$\mathbf{S}^{RPR} = \alpha \cdot \mathbf{S}_{ij}^{RPR} \cdot \mathbf{P} + (1 - \alpha) \cdot \mathbf{I}$$



基于非对称邻近性拟合的节点表征学习

- 为了应对有向图的非对称性，作者将节点 v 的 embedding 表示为 E_{source} 和 E_{target} 两个不同的向量
- 将所有节点的两种 embedding 列成矩阵，可以表示为：

$$\mathbf{U} = [\mathbf{U}^s, \mathbf{U}^t]$$

- 假设存在一个表示节点之间两两相似性的矩阵 $S_{[n \times n]}$
- 那么以下目标函数应当被最小化：

$$\left\| \mathbf{S} - \mathbf{U}^s \cdot \mathbf{U}^{t\top} \right\|_F^2$$



基于非对称邻近性拟合的节点表征学习

非对称邻近性度量的统一范式

$$\min \left\| \mathbf{S} - \mathbf{U}^s \cdot \mathbf{U}^{t^\top} \right\|_F^2$$

$$\mathbf{S} = \mathbf{M}_g^{-1} \cdot \mathbf{M}_l$$

已有非对称邻近性度量

Proximity Measurement	\mathbf{M}_g	\mathbf{M}_l
Katz	$\mathbf{I} - \beta \cdot \mathbf{A}$	$\beta \cdot \mathbf{A}$
Personalized Pagerank	$\mathbf{I} - \alpha \mathbf{P}$	$(1 - \alpha) \cdot \mathbf{I}$
Common neighbors	\mathbf{I}	\mathbf{A}^2
Adamic-Adar	\mathbf{I}	$\mathbf{A} \cdot \mathbf{D} \cdot \mathbf{A}$

Asymmetric Transitivity Preserving Graph Embedding (KDD 2016)[Ou et al., 2016]

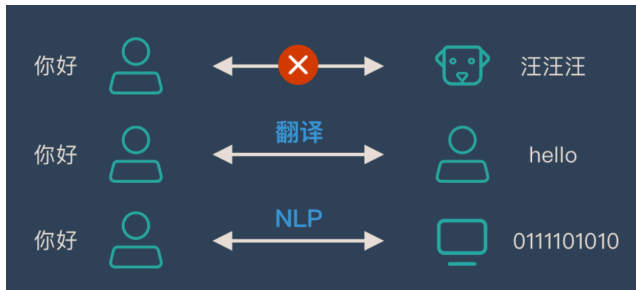


- 1 节点表征学习
- 2 基于矩阵分解的节点表征学习
- 3 从 Word2Vec 到 Node2Vec
- 4 随机游走与矩阵分解的等价性



词向量生成——Word2Vec

- 对于机器而言，它无法直接理解人类的自然语言。通常需要人先对语言进行抽象和数学化之后再交给机器处理。



- Word2Vec[Mikolov et al., 2013] 是 NLP 中的经典方法，它将“不可计算”、“非结构化”的词转化为“可计算”、“结构化”的向量。

词向量

- 词向量就是将语言中的词进行数学化的一种方式，即将一个词表示成一个向量。
- 常见的编码方式有两种：
 - One-Hot Representation
 - Distributed Representation



One-Hot Representation

- One-Hot 编码是一种最简单的词向量表示方式。
- 统计句子中所有出现过的词，构造词典。其中每个词用长度为词典大小的向量表示，每个向量中只有一位是 1，其余都是 0。1 的位置对应该词在词典中的位置。

“I think I can make it”

词典：{ “I”，“think”，
“can”，“make”，“it” }

词	One-Hot编码
I	10000
think	01000
I	10000
can	00100
make	00010
it	00001



One-hot 编码的主要缺陷

- 容易受维度灾难的困扰。每个词都需要用词典大小维度的向量来表示，若词汇量达到千万甚至上亿级别，内存容易爆炸。
- 无法刻画词与词之间的相似性。任意两个词之间都是孤立或者说正交的，One-Hot 编码无法表现单词之间的关系。



基本思想

将每一个词都映射为一个固定长度（不随词典大小改变）的较短向量。所有向量构成一个词向量空间，而每个向量为空间中的一个点。由此就能引入“距离”，从而根据词向量之间的距离来判断其语法和语义上的相似性。

经典的例子：

$$\overrightarrow{King} - \overrightarrow{Man} + \overrightarrow{Woman} \approx \overrightarrow{Queen}$$

优秀的词向量应该能够包含语义信息！



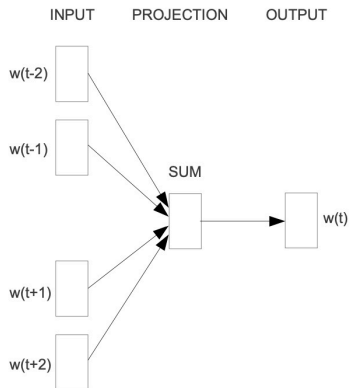
研究动机

词的语义往往随着上下文 (context) 的变化而变化, 因此需要在上下文环境中学习词向量。

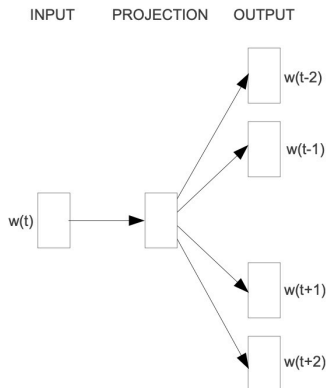
模型假设

- CBOW(Continuous Bags-of-Words): 已知词 w_t 的上下文 $w_{t-l}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+l}$ (l 为上下文窗口大小) 的情况下, 预测当前词 w_t 。
- Skip-gram: 已知词 w_t 的情况下, 对其上下文 $w_{t-l}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+l}$ 进行预测。

Word2Vec



CBOW

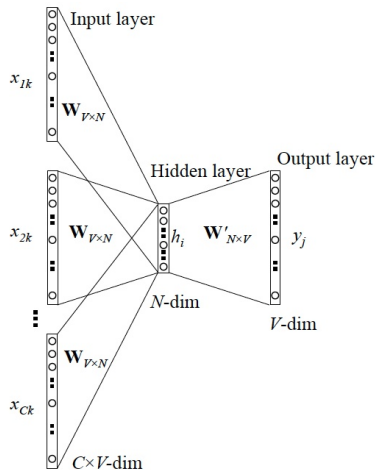


Skip-gram

CBOW 和 Skip-gram

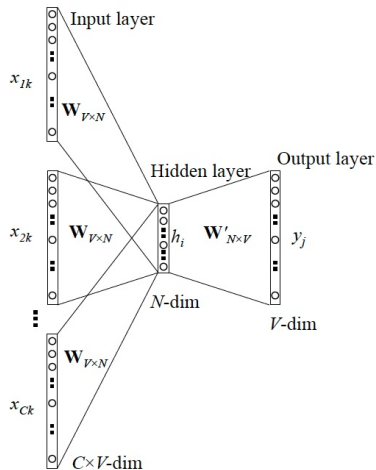
Word2Vec——CBOW

- 输入层：上下文单词的 One-Hot 向量（设词典大小为 V ，上下文单词数为 C ）
- 所有的 One-Hot 向量分别乘以共享的输入权重矩阵 $W_{V \times N}$ ，得到 C 个 N 维向量（ N 为自己定义的维度数）



Word2Vec——CBOW

- 对这 C 个向量求平均，得到隐含层向量 $h_i \in \mathbb{R}^{1 \times N}$
- 再将 h_i 乘以输出权重矩阵 $W'_{N \times V}$ ，通过 softmax 后获得预测的每个单词的概率 $y_j \in \mathbb{R}^{1 \times V}$
- 其中概率最大处所对应的词就是所预测的结果。



模型训练

给定文档，每一个单词都可以作为测试数据用于验证模型是否能准确预测该词汇，因此 CBOW 模式可以用有监督的方式进行训练，即最大化词汇出现的对数似然：

$$\min \mathcal{L}_{CBOW} = -\log(y_j)_p$$

其中 p 为真实值在词典中的对应位置。

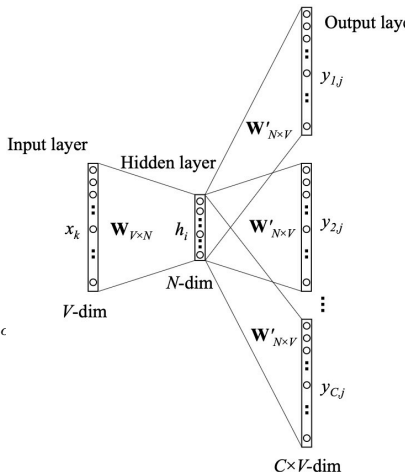
输入权重矩阵 $W \in \mathbb{R}^{V \times N}$ 可以被看做“词典”，每个词的词向量为它的 One-Hot 向量与 W 相乘的结果，即 W 中 One-Hot 的“1”所对应的行。

Word2Vec——Skip-gram

- Skip-gram 的核心思想是通过中心词来预测周围词。由此，输入只有中心词的 One-Hot 向量，就不需要求平均了。
- 需要预测的周围词有 C 个，目标函数为：

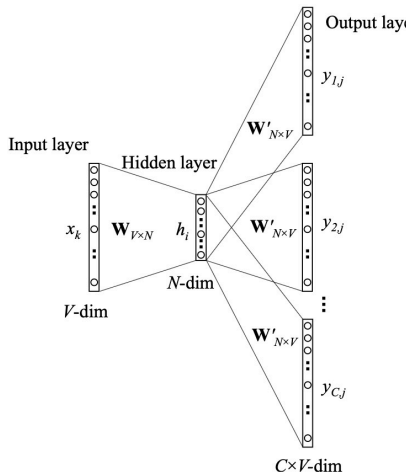
$$\min \quad \mathcal{L}_{Skip-gram} = -\log \prod_{c=1}^C (y_{c,j})_{p_c}$$

p_c 为第 c 个词的真实值在词典中的对应位置。



Word2Vec——Skip-gram

- 对于不同的周围词，隐含层向量 h_i 相同，权重矩阵 W' 共享，那理论上预测出来的词都是相同的，与实际不符。
- Skip-gram 预测周围词的准确率并不是我们真正的目标——我们希望的是学到语义尽可能准确的词向量。因此，尽管预测上可能不准确，但从语义上理解是可行的。



从成对出现的词的角度，word2vec 的目标是最大化成对出现的概率：

$$P(D = 1 \mid w, c) = \sigma(\vec{w} \cdot \vec{c}) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{c}}}$$

$$P(D = 0 \mid w, c) = \sigma(-\vec{w} \cdot \vec{c}) = \frac{1}{1 + e^{\vec{w} \cdot \vec{c}}}$$

如果词汇量非常大，那么网络最后的 softmax 将会带来很大的时间开销。Word2Vec 提出了两种加快训练速度的方式：

- Hierarchical softmax
- Negative Sampling



Word2Vec 加速

Word2Vec 实际训练时通常采用 skip-gram with negative-sampling (SGNS) 方法进行加速:

$$\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbb{E}_{c_N \sim P_D} [\log \sigma(-\vec{w} \cdot \vec{c}_N)]$$

$$P_D(c) = \frac{\#(c)}{|D|}$$

最终的目标函数定义为:

$$\ell = \sum_{w \in V_W} \sum_{c \in V_C} \#(w, c) (\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbb{E}_{c_N \sim P_D} [\log \sigma(-\vec{w} \cdot \vec{c}_N)])$$



Word2Vec——实验结果

- 相较于 baseline, CBOW 和 Skip-gram 都表现出了更好的性能。
- CBOW 在语法方面表现得更好, 而 Skip-gram 在语义上表现得更好。

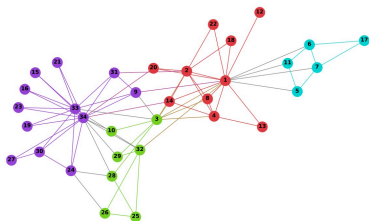
Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

Word2Vec 实验结果

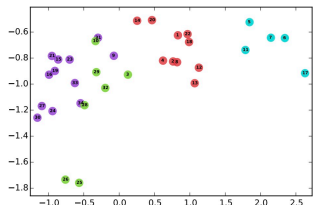


图嵌入方法——DeepWalk

- DeepWalk 是第一个将深度学习（无监督学习）应用于图嵌入的方法。
- 输入一个 Graph，最终目标是获得其中每个节点的向量表示。



(a) Input: Karate Graph



(b) Output: Representation



DeepWalk——主要思想

- 使用随机游走 (Random Walk), 在 Graph 上生成一个所经过节点的序列。
- 该序列可以被看作是语言模型中的一个句子, 而其中的每一个节点是单词。
- 由此, 就可以用 Word2Vec 来学习节点的表示。

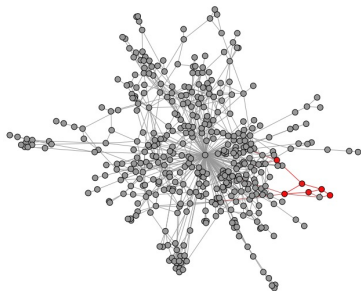


DeepWalk——Random Walk

- 随机游走，顾名思义，就是从输入的 Graph 中任意一个节点 v_i 开始，每次从当前节点的邻居中随机选取一个作为下一个节点，直到达到截断长度 t 。

生成的序列 $\mathcal{W}_{v_i} = (\mathcal{W}_{v_i}^1, \dots, \mathcal{W}_{v_i}^k, \dots, \mathcal{W}_{v_i}^t)$ 。

- 在 DeepWalk 中，对于每一个节点 v_i ，都会随机游走出 γ 条序列。



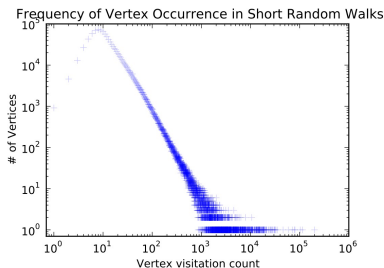
使用随机游走还会带来两个好处：

- 利于并行化：随机游走可以同时从不同的节点开始。多个随机游走同时进行，可以加快整个 Graph 的处理速度。
- 较强的适应性：对于网络的局部细微变化，可以通过一部分较短的随机游走来获取信息，而不需要全局更新。

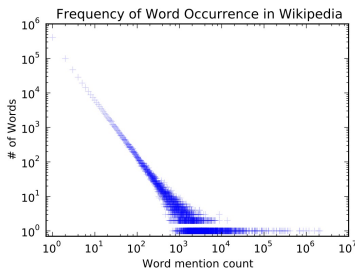


DeepWalk——Why Word2Vec

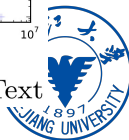
- Word2Vec 为什么可以被用在图嵌入中?
- Graph 中随机游走的节点分布规律与 NLP 中单词在语料库中出现的规律有着类似的幂律分布特征。



(a) YouTube Social Graph



(b) Wikipedia Article Text



DeepWalk——Skip-gram

- 既然 Graph 的特性与 NLP 的特性十分类似，那么将 NLP 中词向量的模型用在图嵌入中也是自然而然的想法。
- 由此，DeepWalk 选用 Word2Vec 的 Skip-gram 模型来学习节点表示。
 - 相较于 CBOW，Skip-gram 在语义上表现得更好，而图嵌入更关注的是节点的“语义”。
 - Skip-gram 只需要计算中心节点的向量，计算量较小。



DeepWalk——Skip-gram

- 对于节点 v_i ，以它为 root 生成的随机游走序列为 \mathcal{W}_{v_i} 。不过， \mathcal{W}_{v_i} 并不是 v_i 所对应的“上下文”，它是一个独立的句子。
- \mathcal{W}_{v_i} 中的每一个节点 v_j 都有其对应的上下文 $\mathcal{W}_{v_i}[j - w : j + w]$ ， w 为上下文窗口大小。

Algorithm 2 SkipGram(Φ , \mathcal{W}_{v_i} , w)

```
1: for each  $v_j \in \mathcal{W}_{v_i}$  do
2:   for each  $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$  do
3:      $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$ 
4:      $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$ 
5:   end for
6: end for
```

- 总结一下，DeepWalk 主要就是包括两个步骤：
 - 对于每一个节点进行随机游走采样获得 $|V|$ 个节点序列。
 - 使用 Skip-gram 模型更新参数，学习节点表示。
- 实际操作中，上述步骤需要重复 γ 次以获得更稳定和准确的结果。



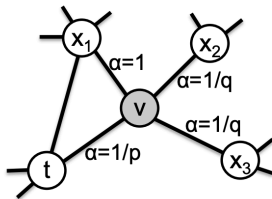
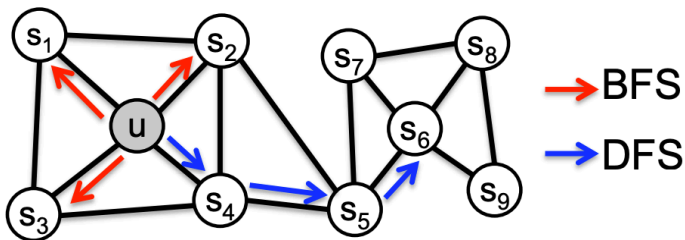
DeepWalk——实验结果

	% Labeled Nodes	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1(%)	DEEPWALK	36.00	38.20	39.60	40.30	41.00	41.30	41.50	41.50	42.00
	SpectralClustering	31.06	34.95	37.27	38.93	39.97	40.99	41.66	42.42	42.62
	EdgeCluster	27.94	30.76	31.85	32.99	34.12	35.00	34.63	35.99	36.29
	Modularity	27.35	30.74	31.77	32.97	34.09	36.13	36.08	37.23	38.18
	wvRN	19.51	24.34	25.62	28.82	30.37	31.81	32.19	33.33	34.28
	Majority	16.51	16.66	16.61	16.70	16.91	16.99	16.92	16.49	17.26
Macro-F1(%)	DEEPWALK	21.30	23.80	25.30	26.30	27.30	27.60	27.90	28.20	28.90
	SpectralClustering	19.14	23.57	25.97	27.46	28.31	29.46	30.13	31.38	31.78
	EdgeCluster	16.16	19.16	20.48	22.00	23.00	23.64	23.82	24.61	24.92
	Modularity	17.36	20.00	20.80	21.85	22.65	23.41	23.89	24.20	24.97
	wvRN	6.25	10.13	11.64	14.24	15.86	17.18	17.98	18.86	19.57
	Majority	2.52	2.55	2.52	2.58	2.58	2.63	2.61	2.48	2.62

BlogCatalog 数据集上的实验结果



改进的 RandomWalk 策略:



- 1 节点表征学习
- 2 基于矩阵分解的节点表征学习
- 3 从 Word2Vec 到 Node2Vec
- 4 随机游走与矩阵分解的等价性



随机游走与矩阵分解的等价性

SGNS 算法的目标函数为：

$$\ell = \sum_{w \in V_W} \sum_{c \in V_C} \#(w, c) (\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbb{E}_{c_N \sim P_D} [\log \sigma(-\vec{w} \cdot \vec{c}_N)])$$

可以拆解为：

$$\begin{aligned} \ell &= \sum_{w \in V_W} \sum_{c \in V_C} \#(w, c) (\log \sigma(\vec{w} \cdot \vec{c})) \\ &+ \sum_{w \in V_W} \sum_{c \in V_C} \#(w, c) (k \cdot \mathbb{E}_{c_N \sim P_D} [\log \sigma(-\vec{w} \cdot \vec{c}_N)]) \\ &= \sum_{w \in V_W} \sum_{c \in V_C} \#(w, c) (\log \sigma(\vec{w} \cdot \vec{c})) \\ &+ \sum_{w \in V_W} \#(w) (k \cdot \mathbb{E}_{c_N \sim P_D} [\log \sigma(-\vec{w} \cdot \vec{c}_N)]) \end{aligned}$$



随机游走与矩阵分解的等价性

其中数学期望可以通过观测样本进行估计：

$$\begin{aligned}\mathbb{E}_{c_N \sim P_D} [\log \sigma(-\vec{w} \cdot \vec{c}_N)] &= \sum_{c_N \in V_C} \frac{\#(c_N)}{|D|} \log \sigma(-\vec{w} \cdot \vec{c}_N) \\ &= \frac{\#(c)}{|D|} \log \sigma(-\vec{w} \cdot \vec{c}) + \sum_{c_N \in V_C \setminus \{c\}} \frac{\#(c_N)}{|D|} \log \sigma(-\vec{w} \cdot \vec{c}_N)\end{aligned}$$

对于观测到的一组词，其在优化过程中的目标为：

$$\ell(w, c) = \#(w, c) \log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \#(w) \cdot \frac{\#(c)}{|D|} \log \sigma(-\vec{w} \cdot \vec{c})$$



随机游走与矩阵分解的等价性

为了计算方便, 令 $x = \vec{w} \cdot \vec{c}$, 对于观测到的一组词, 其优化的导数为:

$$\frac{\partial \ell}{\partial x} = \#(w, c) \cdot \sigma(-x) - k \cdot \#(w) \cdot \frac{\#(c)}{|D|} \cdot \sigma(x)$$

另导数等于 0 可得:

$$e^{2x} - \left(\frac{\#(w, c)}{k \cdot \#(w) \cdot \frac{\#(c)}{|D|}} - 1 \right) e^x - \frac{\#(w, c)}{k \cdot \#(w) \cdot \frac{\#(c)}{|D|}} = 0$$

另 $y = e^{2x}$ 将式子转化为 y 的二项式, 易得 y 的解为 -1 和:

$$y = \frac{\#(w, c)}{k \cdot \#(w) \cdot \frac{\#(c)}{|D|}} = \frac{\#(w, c) \cdot |D|}{\#w \cdot \#(c)} \cdot \frac{1}{k}$$



随机游走与矩阵分解的等价性

将 x, y 代入之后可得：

$$\vec{w} \cdot \vec{c} = \log \left(\frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \cdot \frac{1}{k} \right) = \log \left(\frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \right) - \log k$$




其中 $\log \left(\frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \right)$ 是 pointwise mutual information (PMI) ，最终词向量的计算等价于：

$$M_{ij}^{\text{SGNS}} = W_i \cdot C_j = \vec{w}_i \cdot \vec{c}_j = \text{PMI}(w_i, c_j) - \log k$$

即对 PMI 矩阵进行矩阵分解。



References I

-  Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.
-  Ou, M., Cui, P., Pei, J., Zhang, Z., and Zhu, W. (2016). Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114.
-  Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015). Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077.

