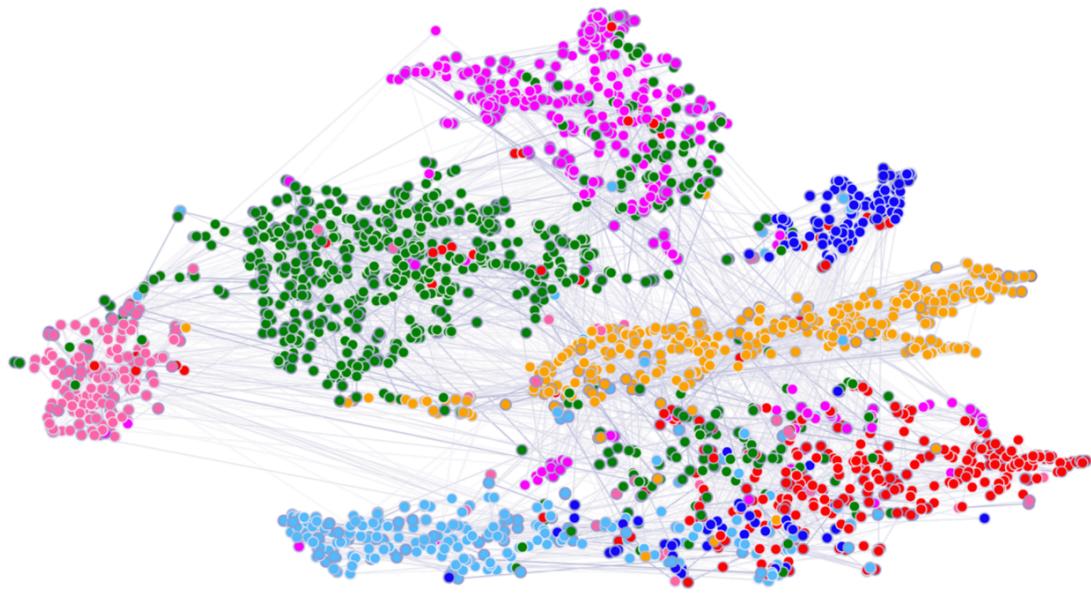


Graph Neural Network Review

By 吴天龙

2018-09-04

Email: wutianlong@didichuxing.com



Cora dataset

图(graph)是一个非常常用的数据结构，现实世界中很多很多任务可以描述为图问题，比如社交网络，蛋白体结构，交通路网数据，以及很火的知识图谱等，甚至规则网格结构数据(如图像，视频等)也是图数据的一种特殊形式，因此图是一个很值得研究的领域。

针对graph的研究可以分为三类：

1. 经典的graph算法，如生成树算法，最短路算法，复杂一点的二分图匹配，费用流问题等等；
2. 概率图模型，将条件概率表达为图结构，并进一步挖掘，典型的有条件的随机场等
3. 图神经网络，研究图结构数据挖掘的问题，典型的有graph embedding, graph CNN等

本文主要针对图神经网络，介绍一些最近几年该领域的一些研究进展。由于应用很广泛(主要是社交网络发展和知识图谱的推动)，以及受到深度学习在其他领域成功的启示，这个方向是目前机器学习领域最火的方向之一了，今年kdd2018中31篇tutorials里面有9篇是关于graph的，bestpaper也是关于graph的，论文名字叫做：adversarial attacks on classification models for graphs. 可见学术界和工业界的热情。

本文首先介绍graph Embedding，为结构化的graph生成分布式表示；然后介绍graph convolutional network(图卷积)，最后简单介绍基于图的序列建模。

【目录】

1.Graph Embedding

- 1.1.WHAT & WHY
- 1.2.Word2vec
- 1.3.古典方法
- 1.4.Deepwalk
- 1.5.LINE
- 1.6.GraRep
- 1.7.Node2vec
- 1.8.Struc2vec
- 1.9.GraphSAGE
- 1.10.CANE
- 1.11.SDNE
- 1.12.GraphGAN
- 1.13.总结

2.Graph CNN

- 2.1. Spectral Convolution
 - 2.1.1. Graph上的傅里叶变换
 - 2.1.2. Graph上的傅里叶逆变换
 - 2.1.3. Graph上的谱图卷积
 - 2.1.4. **1st** Spectral Convolution
 - 2.1.5. **2nd** Spectral Convolution
 - 2.1.6. **3rd** Spectral Convolution
- 2.2. Spatial Convolution
 - 2.2.1. DCNNs
 - 2.2.2. CNN4G
 - 2.2.3. GATs
- 2.3. 总结

3.基于Graph的序列建模

- 3.1. DCRNNs
- 3.2. GAT-LSTM
- 3.3. 总结

1.Graph Embedding

Graph embedding(GE) 也叫做network embedding(NE)也叫做Graph representation learning(GRL), 或者network representation learning(NRL), 最近有篇文章把graph和network区分开来了, 说graph一般表示抽象的图比如知识图谱, network表示实体构成的图例如社交网络, 我觉得有点过分区分了。图1.1是整个GE大家族, 本文只介绍绿色的, 蓝色模型就不一一介绍了。严格来说, 图卷积也属于GE的一部分, 由于目标不同以及其关注度较大, 我们单独划分出来(文章第二部分)

➤ Graph Embedding!

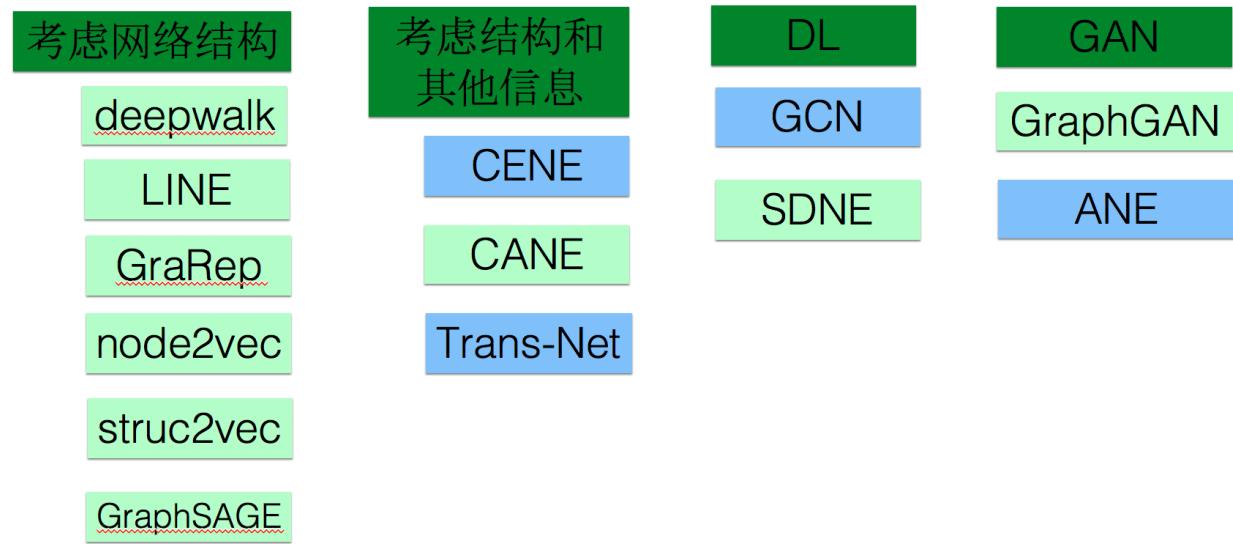


图1.1 Graph Embedding大家族

1.1.WHAT & WHY

首先我们回答什么是embedding，以及为什么要做embedding

WHAT?

- Embedding在数学上是一个函数， $f : X \rightarrow Y$ ，即将一个空间的点映射到另一个空间，通常是从高维抽象的空间映射到低维具象的空间；
- 一般映射到低维空间的表示具有分布式稠密表示的特性

WHY?

- 抽象的事物应该有一个低维的表示；比如我们看到一张含有一张小狗的图片，它底层的表示就是像素值。同样的我们看到“dog”这个单词时，它也应该有更低维的表示
- 计算机和神经网络善于处理低纬度信息
- 解决one-hot编码问题；one-hot编码是一种特殊的高维到低维的映射，具有稀疏性，且向量长度较长并随着样本集变化而变化。one-hot编码是一种“讨巧”的编码方式，一般无法表示两个实体之间的相关性，embedding可以一定程度上解决这些问题。

1.2.Word2vec

word2vec[2013]: [Distributed Representations of Words and Phrases and their Compositionality](#)

首先简要介绍word2vec，这是自然语言神经网络模型的基础，同时对GE领域也有深远的影响，详细内容请参考paper或者该[博客](#)。

word2vec是根据语料库中单词的共现关系求出每个单词的embedding。常用word2vec模型有cbow和skip-gram两种，cbow根据上下文预测中心词，skip-gram根据中心词预测上下文，由于两种模型相似，本节只介绍skip-gram模型。

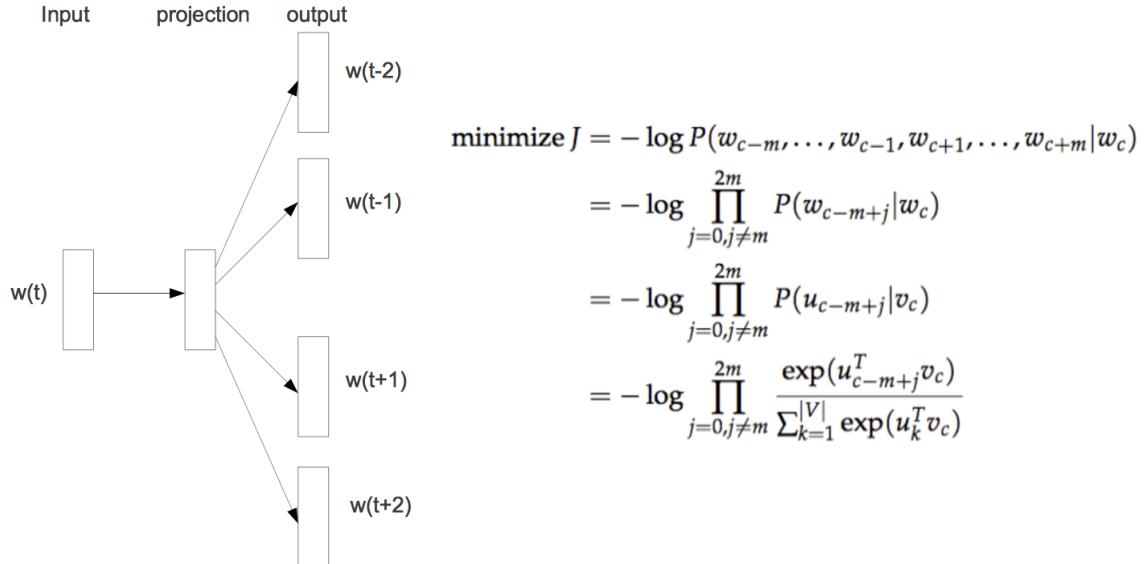


图1.2 skip-gram模型

Word2vec本质是一个输入不确定的softmax回归，普通的softmax回归输入是确定的，我们需要求出参数矩阵，而word2vec中，输入的向量和参数矩阵都是不确定的，并且输入向量才是我们真正需要的，参数矩阵则是副产品。普通softmax回归是一个凸优化问题，然而word2vec由于输入不确定不是一个凸优化问题，需要采取类似于坐标上升的优化方法。

由于两个原因：

1. 一般来说语料库单词都是很多的，softmax计算(主要是配分函数)会非常耗时；
2. 正负样本极度不均衡

一般不直接优化图1.1中的表达式，有一些优化，比如分层softmax和负采样，一般来说现在最常用的是skip-gram+负采样。

这里提示一下，一般来说遇到配分函数的计算都需要优化，优化方法一般采用噪声对比估计(Noise Contrastive Estimation, NCE)，负采样(Negative Sampling)技术是NCE的一种特殊形式，后面我们会经常用到这个trick，负采样的一般表达式如公式1.1，其中 $\tilde{p}(x)$ 是真实样本分布， $U(x)$ 是某个“均匀”分布或者其他、确定的、方便采样的分布，说白了，NCE的做法就是将它转化为二分类问题，将真实样本判为1，从另一个分布采样的样本判为0。公式1.2是word2vec应用的负采样公式，即语料出现的 Skip Gram 视为正样本，随机采样的词作为负样本。关于NCE优化等价于直接优化softmax回归的数学证明详见[NCE等价于softmax回归的证明](#)

$$\mathbb{E}_{x \sim \tilde{p}(x)} \log(p(1|x)) + \mathbb{E}_{x \sim U(x)} \log(p(0|x)) \quad (1.1)$$

$$\mathbb{E}_{w_j \sim \tilde{p}(w_j|w_i)} \log(\langle u_i, v_j \rangle) + \mathbb{E}_{x \sim \tilde{p}(w_j)} \log(1 - \langle u_i, v_j \rangle) \quad (1.2)$$

1.3. 古典方法

在word2vec出现之前[2013]，已经有一些方法试图对graph进行embedding，这里简要介绍三个

1. Locally Linear Embedding

该方法假设一个节点可以用它邻居节点的线性组合表示，目标函数为 $\phi(Y) = \frac{1}{2} \sum_i |Y_i - \sum_j W_{ij} Y_j|^2$

2.Laplacian Eigenmaps

该方法认为在原始空间越相似的节点(使用边权衡量), 映射到低维空间以后也会越相似, 目标函数为
 $\phi(Y) = \frac{1}{2} \sum_{ij} (Y_i - Y_j)^2 W_{ij}$

3.Graph Factorization

该方法通过矩阵分解求得embedding表示, 目标函数为
 $\phi(Y, \lambda) = \frac{1}{2} \sum_{(i,j) \in E} (W_{ij} - \langle Y_i, Y_j \rangle)^2 + \frac{\lambda}{2} \sum_i \|Y_i\|^2$

1.4.Deepwalk

Deepwalk[2014] : [DeepWalk: online learning of social representations](#)

思考一个问题, 既然自然语言中的单词(word)可以通过共现关系进行embedding, 那graph类似于整个语料库, 图中节点(node)类似于单词, 我们可不可以通过node之间的共现关系对graph中的node进行embedding呢? 答案的关键在于怎么描述node的共现关系。对于word2vec来说, 语料库中每一个句子都可以描述单词之间的共现, 那么我们可不可以构造一个类似于“句子”的序列呢, 把这些序列作为node的“句子”来得到其embedding呢? 实际上, deepwalk的工作就是在graph上构造这些“句子”, 然后直接使用word2vec模型训练得到graph中每个node的embedding表示!

deepwalk是基础的network embedding算法, 是将word2vec直接用于graph的算法, 只是增加了随机游走以构造序列(类似于word2vec中的语料库)。公式1.3是random walk的公式, 根据边权来完全随机的选择下一个节点, 实际上**random walk是一钟可回头的深度优先搜索**

$$p(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z}, & \text{if } (v, x) \in E. \\ 0, & \text{otherwise} \end{cases} \quad (1.3)$$

1.5.LINE

LINE[2015] : [Large scale information network embedding](#)

作者认为高纬度相似的节点, 映射到低维空间也应该是相似的。就两个节点的相似性, 作者有两个维度的定义:

1. **1st order proximity**: 在高维层次即原始图中, 表示为节点之间的边权; 在低位层次即两个向量要是比较近似的, 用一个sigmoid函数表示
2. **2nd order proximity**: 用于有向图, 在高维层次, 表示为两个节点之间有多少公共一度节点; 在低维层次表示为一个条件概率。

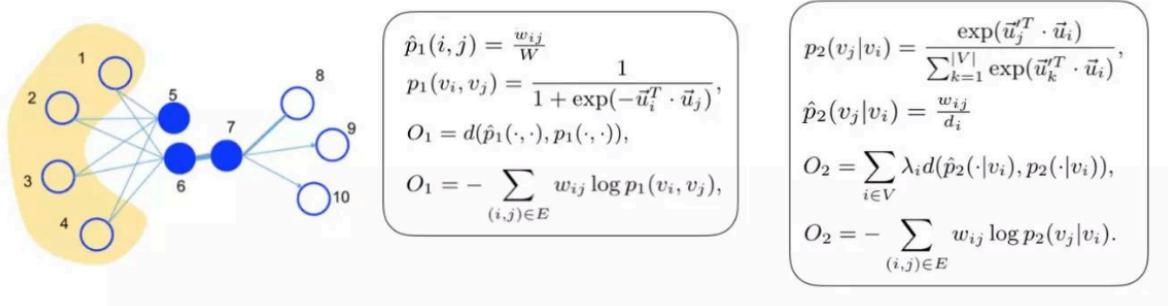


图1.3 LINE模型

1^{st} order proximity比较好理解, 2^{nd} order proximity是比较迷惑人的, 包括网上很多博客解释都是含糊不清或者是错的。其实最主要的是要理解每个节点都有两个向量, 一个是环境向量(\vec{u}_j'), 一个是自身向量(\vec{u}_j)。环境向量处于被动地位, 用于描述该节点充当context角色时的表示, 自身向量即为最终embedding得到的向量, 表示为条件概率可以理解为从节点*i*到节点*j*的概率, 这表征着该节点作为环境时跟其他节点的关系。

需要注意的是 1^{st} order proximity和 2^{nd} order proximity是分别优化的, 当然对于 $p_2(v_j|v_i)$ 又是一个softmax, 我们同样使用负采样优化技术:

$$\log\sigma(\langle \vec{u}_j', \vec{u}_i \rangle) + \sum_{i=1}^K E_{v_n \sim P_n(v)} [\log\sigma(-\langle \vec{u}_j', \vec{u}_i \rangle)] \quad (1.4)$$

最后注意一点, LINE基于邻域相似的假设, 这其实是一种基于BFS方法

1.6.GraRep

GraRep[2015] : [Learning Graph Representations with Global Structural](#)

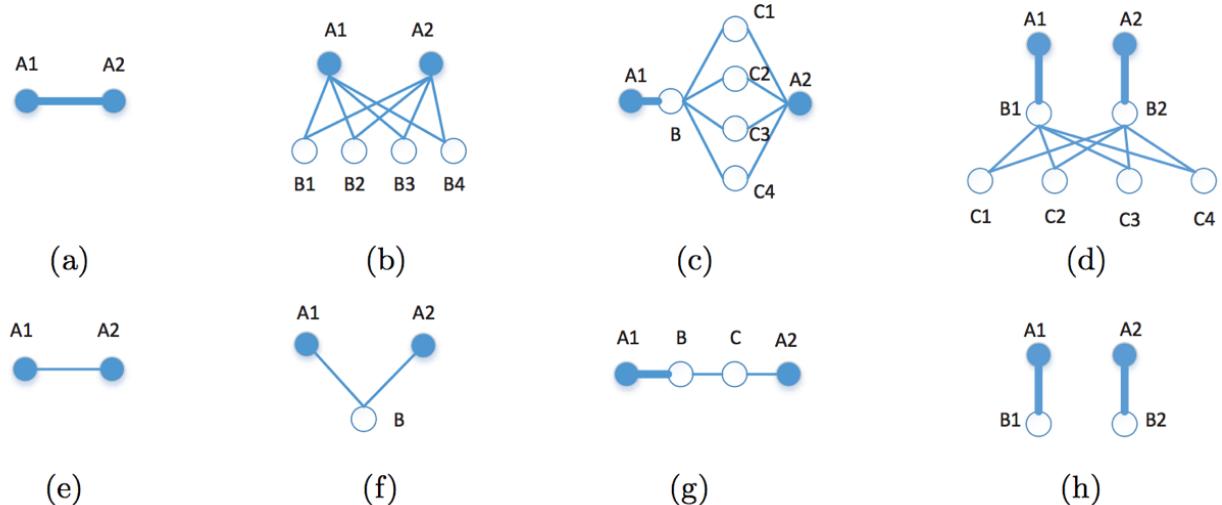


图1.4 节点之间K(1,2,3,4)阶相似性

LINE只考虑2-order相似性好吗? 为什么不考虑3-order, 4-order..., 如图1.4中, 我们很明显知道上面一行A1与A2的关系强于下面一行, 那么怎么描述这种关系呢? GraRep就是解决这个问题, 和LINE思路一样, 每个节点两个向量, 一个自身向量, 一个环境向量。

首先定义k-order相似性:

$$L_k = \sum_{w \in V} L_k(w) \quad (1.5)$$

其中 $L_k(w)$ 表示对于特定节点 w 的k-order相似性, $L_k(w)$ 表示为公式1.6(已经使用了负采样优化):

$$L_k(w) = (\sum_{c \in V} p_k(c|w) \log\sigma(\vec{w} \cdot \vec{c})) + \lambda \mathbb{E}_{c' \in p_k(V)} [\log\sigma(-\vec{w} \cdot \vec{c}')] \quad (1.6)$$

其中 $p_k(c|w)$ 表示从 w 节点经过 k 跳以后到达 c 的概率, 即 P_k 为节点 k 阶转移矩阵。一阶转移矩阵可以用归一化邻接矩阵 $A = D^{-1}W$ 表示, 那么 k 阶转移矩阵为 $P_k = A^k$, 即 $p_k(c|w) = A_{w,c}^k$

$p_k(c)$ 表示图中 k 跳的边缘分布, 即随机选择开始位置, 经过 k 跳以后到达 c 的概率:

$$p_k(c) = \sum_{w'} q(w') p_k(c|w') = \frac{1}{N} \sum_{w'} A_{w',c}^k \quad (1.7)$$

其中 $q(w')$ 表示 w' 节点的先验分布，这里假设为均匀分布，即 $q(w') = \frac{1}{N}$

将1.6, 1.7代入???并令导数为0，可以得到下式：

$$W_i^k \cdot C_j^k = \log\left(\frac{A_{i,j}^k}{\sum_t A_{t,j}^k}\right) - \log\left(\frac{\lambda}{N}\right) \quad (1.8)$$

即求一个矩阵分解即可，丢弃 C 矩阵(环境向量组成的矩阵)， W 矩阵即为所求。

1.7.Node2vec

Node2vec[2016] : [Scalable Feature Learning for Networks](#)

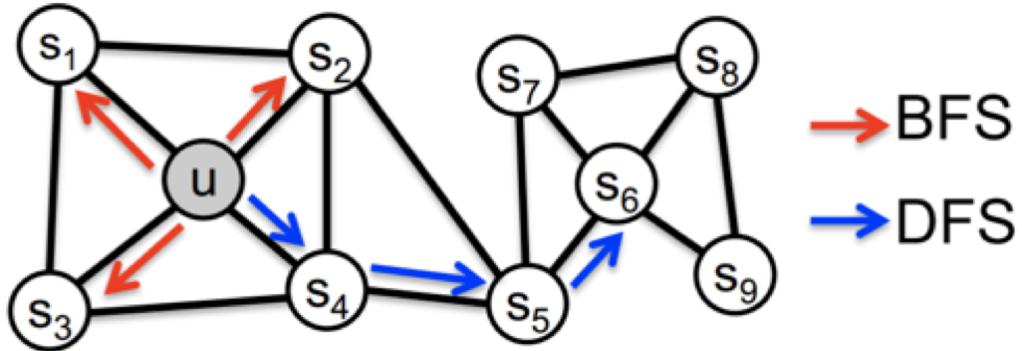


图1.5 BFS和DFS搜索

前面说到Deepwalk其实是基于DFS模型，LINE和GraRep是基于DFS的模型，一般来说基于BFS的模型倾向于获取图中的内容相似性(即局部相似性)，基于DFS的模型更倾向于获取结构相似性。那么能否设计一种可以平衡BFS和DFS的模型呢？Node2vec给出了一种方案，它设计了一个二阶随机游走，通过 p, q 两个参数来平衡BFS和DFS两种搜索策略。

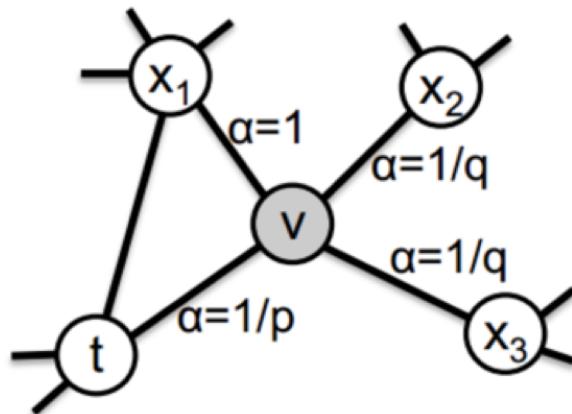


图1.6 p, q 控制的二阶随机游走

图1.6是 p, q 控制的二阶随机游走的示意图，其中 α 是关于当前节点 v 搜索下一个节点 x 的函数， p, q 为超参数，表达式如下：

$$\alpha(t, x) = \begin{cases} \frac{1}{p}, & if d_{tx} = 0. \\ 1, & if d_{tx} = 1. \\ \frac{1}{q}, & if d_{tx} = 2. \end{cases} \quad (1.9)$$

那么从当前节点 v 随机游走到下一个节点 x 的概率就是其归一化形式：

$$p(x|v) = \frac{\alpha(t, x)}{\sum_{(y,v) \in E} \alpha(t, y)} \quad (1.10)$$

从公式1.9和1.10可以看出， p 控制着随机游走以多大的概率“回头”， q 控制着随机游走偏向于DFS还是BFS，当 q 较大时，倾向于BFS，当 q 较小时，倾向于DFS。特别地，当 p, q 都为1时，等价于random walk。

node2vec在工业界也十分成功，这里分享两个案例，一个是[Facebook在广告领域的定制化受众的应用](#)，另一个是我们更加熟悉的[腾讯的微信朋友圈广告策略](#)

1.8. Struc2vec

Struc2vec[2017]：[struc2vec: Learning Node Representations from Structural Identity](#)

Struc2vec认为node的embedding不应该考虑任何相邻性，而是只考虑节点的空间结构相似性。它认为deepwalk和node2vec等模型之所以有效是因为，现实世界中相邻的节点往往有比较相似的空间环境，因此这种方法会有效，但是有些时候这种方法会失效，例如图1.7。

结构的相似并不一定能保证两个社区相似，因此网络表示学习的任务在于，1) 把结构相似的团体找出来？2) 把相似节点找出来？
1和社区发现有何区别？

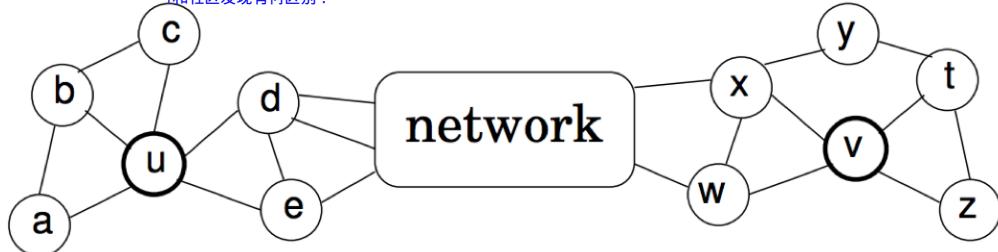


图1.7 graph结构相似形

图1.7中，节点 u 和 v 之间可能相隔很远，但是结构上他们却是十分相似的，GE算法应该考虑这种远距离的结构相似形。那么怎么用数学语言描述这种结构相似形呢？一个直观的想法是，如果两个节点的所有邻居组成的两个序列相似，那么这两个节点就比较相似，实际上，论文正是使用这种方式分层定义结构相似形的。

考虑节点 u ，令 $R_k(u)$ 表示与 u 最短距离为 k 的节点集合， $s(S)$ 表示节点集合 $S \subset V$ 的有序度(degree)序列，定义 $f_k(u, v)$ ：

$$f_k(u, v) = f_{k-1}(u, v) + g(s(R_k(u)), s(R_k(v))) \quad (1.11)$$

g 函数是比较两个序列相似度的函数，可以是Dynamic Time Warping(动态时间规整算法)，用来计算两个可能长度不等的序列的相似度， $f_k(u, v)$ 表征节点 u, v 之间的结构不相似性，可以看出，随着 k 增大， $f_k(u, v)$ 的感受野越来越大，因此我们需要先在底层 $k = 1$ 比较节点之间相似性，如果两个节点之间关系较紧密，我们需要在更高层次(更大感受野)来度量两个节点的相似性，公式1.13反应出这一点(倾向于向上游走)。

利用 $f_k(u, v)$ 构造分层图，同层之间节点之间的权值如公式1.12，相邻层对应节点的权值为公式1.13

$$w_k(u, v) = e^{-f_k(u, v)} \quad (1.12)$$

$$\begin{cases} w(u_k, u_{k+1}) = \log(\Gamma_k(u) + e), & \Gamma_k(u) = \sum_{v \in V} \mathbf{1}(w_k(u, v) > \bar{w}_k). \\ w(u_k, u_{k-1}) = 1 \end{cases} \quad (1.13)$$

接下来就是随机游走构建sequence了，首先每一步有一个概率 p 在本层， $1 - p$ 跳出本层，如果在本层按照公式1.14采样下一个节点，如果跳出本层，按照公式1.15来选择上一层(u_{k+1})还是下一层(u_{k-1})：

$$p_k(v|u) = \frac{e^{-f_k(u, v)}}{Z_k(u)} \quad (1.14)$$

$$\begin{cases} p_k(u_{k+1}|u_k) = \frac{w(u_k, u_{k+1})}{w(u_k, u_{k+1}) + w(u_k, u_{k-1})} \\ p_k(u_{k-1}|u_k) = 1 - p_k(u_{k+1}|u_k) \end{cases} \quad (1.15)$$

关于struc2vec有一个很成功的工业应用，[蚂蚁金服风控模型](#)应用了struc2vec后，相较与之前的node2vec有质的提升。

1.9.GraphSAGE

GraphSAGE[2017] : [Inductive representation learning on large graph](#)

介绍一组概念：

- 直推式学习：直接根据graph学习到一个 $N \times F$ 的矩阵，graph结构变化是要重新学习的
- 归纳式学习：利用节点邻域信息直接学习出新增节点（unseen node）的embedding表示

前面介绍的模型全部都是直推式学习，换句话说当graph变化时，需要重新学习。GraphSAGE是归纳式学习的一个代表，它不直接学习每个节点的表示，而是学习聚合函数，学到了聚合函数以后，对于新增的节点我直接生成它的embedding表示，而不需要重头学习。下面算法就是GraphSAGE的核心内容了，图1.8更形象的表示这一算法。

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 $\mathbf{W}^k, \forall k \in \{1, \dots, K\}$; non-linearity σ ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, \dots, K\}$; neighborhood function $\mathcal{N}: v \rightarrow 2^{\mathcal{V}}$

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

```

1  $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$ ;
2 for  $k = 1 \dots K$  do K为分层采样的层数
3   for  $v \in \mathcal{V}$  do 采样方法为层次遍历
4      $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\});$ 
5      $\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k))$ 
6   end
7    $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$ 
8 end
9  $\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$ 

```

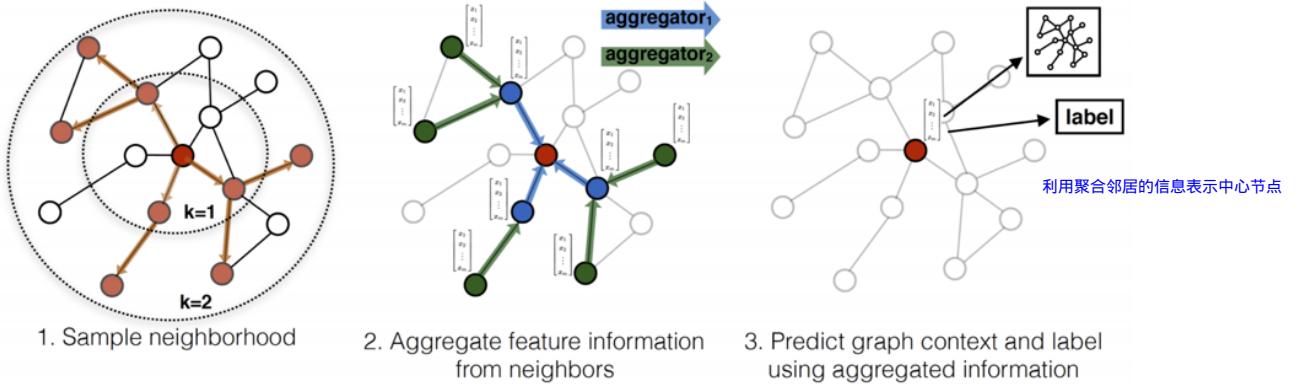


Figure 1: Visual illustration of the GraphSAGE sample and aggregate approach.

图1.8 GraphSAGE模型示意图

该算法分为两步：

1. 邻居采样：因为每个节点的度是不一致的，为了计算高效，为每个节点分层(K层)采样固定数量的邻居。
2. 邻居特征聚集：通过聚集采样到的邻居特征，更新当前节点的特征，网络第k层聚集到的邻居即为BFS过程第k层的邻居采样。

Aggregator function 要求要对顺序不敏感，因为random选择的node本身是无序的，可以有如下几种选择，当然也可以自己设计：

1. Mean aggregator: $h_v^k \leftarrow \sigma(W \cdot MEAN(\{h_v^{k-1}\} \cup \{h_u^{k-1}, \forall u \in \mathcal{N}(v)\}))$
2. LSTM aggregator: LSTM输入是有序的，每次采样的node需要shuffle一下
3. Pooling aggregator: $Aggregator_k^{pool} \leftarrow max(\{\sigma(W_{pool} h_{u_i}^k + b), \forall u_i \in \mathcal{N}(v)\})$
4. GCN aggregator: 图卷积聚合器，后文再表

1.10.CANE

CANE[2017] : [Context-AwareNetwork Embedding for Relation Modeling](#)

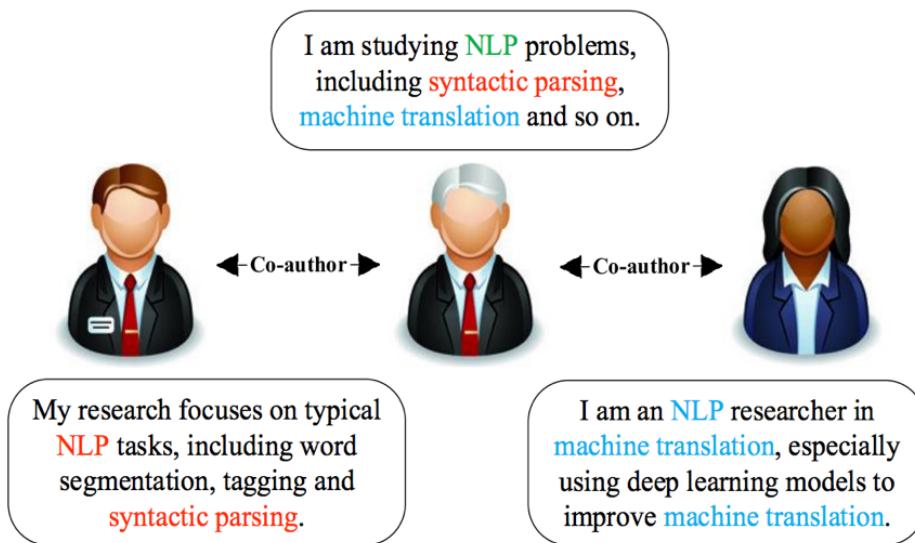


图1.9 附带文本信息的社交网络

前面所有模型都是在图结构上做文章，但是现实世界中节点和边上有丰富的信息(如图1.9)，包括我们组现在遇到问题(我们在对拼车的规模与成本之间关系建模)，节点和边的信息都是不可忽略的。我们应该使用这些信息，由于graph embedding是伴随社交网络发展的，所以大多数文章把节点上的信息当做文本信息处理，类似的模型还有CANE，将文本转化为特殊的节点，这样就有两种连边，(节点-文档)以及(节点-节点)，对两种边一起建模，trans-net将机器翻译的一些东西引进来，这些都是和nlp相关的，就不都说了，这里只简单介绍CANE，**CANE兼顾结构相似性和文本内容相似性，应用attention机制，可以自适应地计算文本之间的相似性。**

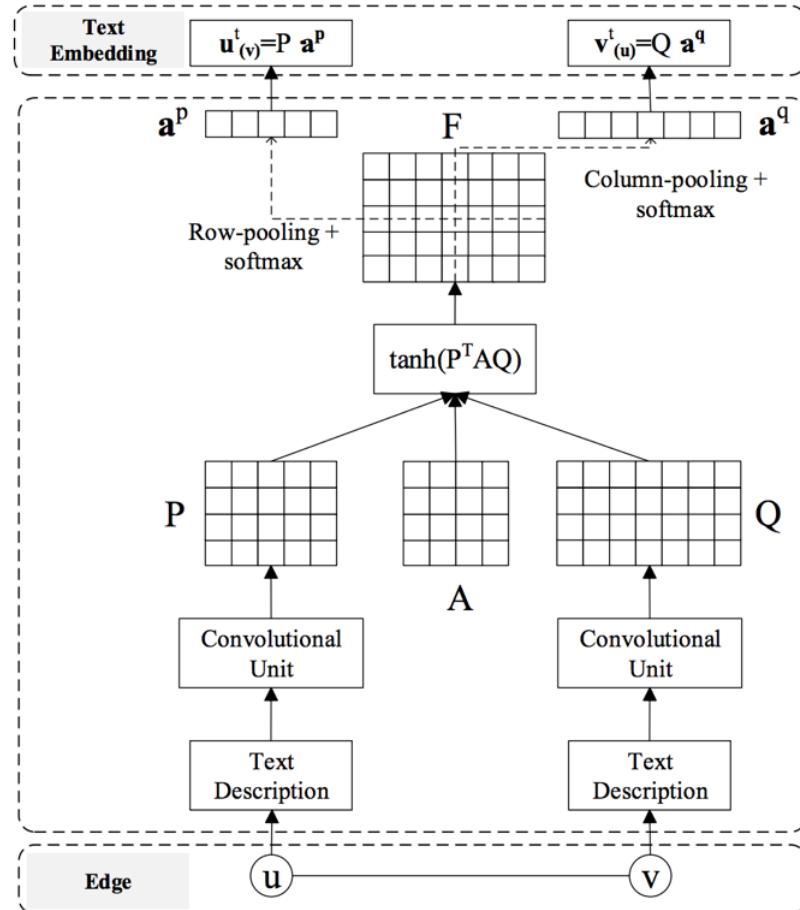


图1.10 CANE模型框架示意图

图1.9是CANE的框架示意图，从最底层看，使用训练好的词向量(就是一开始我们提到的word2vec)先把每个节点的文本信息表示为一个矩阵，然后使用行卷积变换；接着 P 和 Q 和一个Attention系数矩阵执行 $\tanh(P^T A Q)$ 运算， F_{ij} 表示着 P_i 对 Q_j 的重要性，然后分别采用的row-pooling和column-pooling + softmax产生一个注意力向量，比如 a^p 就表示 v 对 P 每列的重要性；最后使用注意力向量和原词向量矩阵相乘得到最终的 v 对 u 和 u 对 v 的text向量，分别表示为 $v^t(u)$ 和 $u^t(v)$

损失函数由两部分表示，一部分代表结构相似性 $L_s(e)$ ，另一部分代表文本信息相关性 $L_t(e)$ ：

$$L(e) = L_s(e) + L_t(e) \quad (1.16)$$

考虑结构相似性，和LINE相似：

$$p(v^s | u^s) = \frac{\exp(\langle u^s, v^s \rangle)}{\sum_{z \in V} \exp(\langle u^s, z \rangle)} \quad (1.17)$$

$$L_s(e) = w_{u,v} \log p(v^s | u^s) \quad (1.18)$$

文本相似性由三部分组成，分别为text-text相似性，text-structure相似性和structure-text相似性：

$$L_t(e) = \alpha L_{tt}(e) + \beta L_{ts}(e) + \gamma L_{st}(e) \quad (1.19)$$

$$L_{tt}(e) = w_{u,v} \log p(v^t | u^t) \quad (1.20)$$

$$L_{ts}(e) = w_{u,v} \log p(v^t | u^s) \quad (1.21)$$

$$L_{st}(e) = w_{u,v} \log p(v^s | u^t) \quad (1.22)$$

当然，对于公式中所有的softmax，都需要使用负采样加速的。

1.11.SDNE

SDNE[2016] : Structural Deep Network Embedding

严格来说，前面所有模型都是浅层模型，SDNE是一个基于自编码器的深度模型，图1.11是该模型的架构示意图

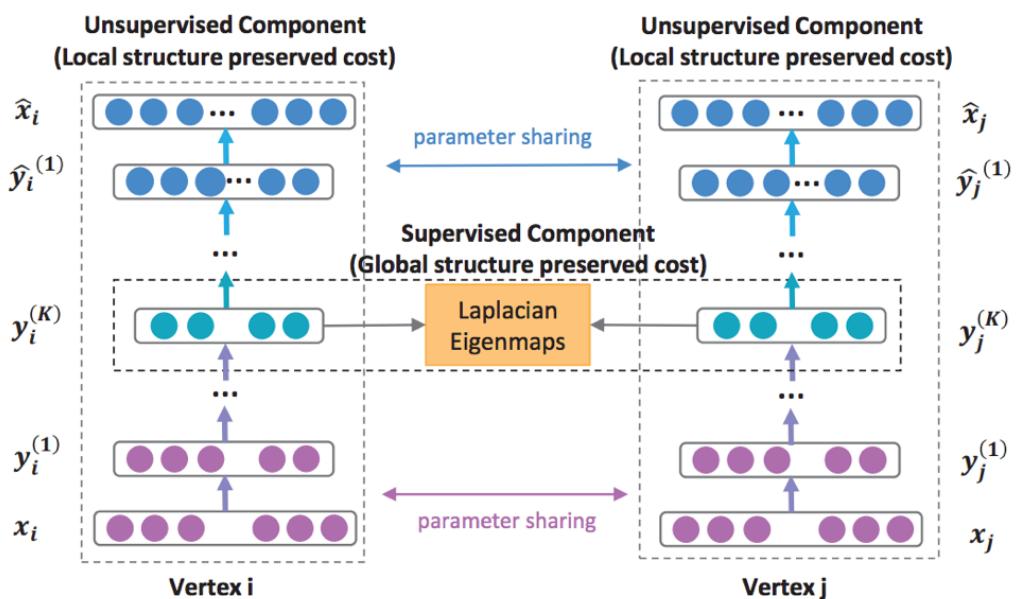


图1.11 SDNE模型示意图

SDNE基于深度自编码器，它的输入是graph的邻接矩阵，每个node对应一行，该行表示该node与其他所有node是否有边。它的损失函数有两部分组成，第一部分是基于“如果两个node有边，那么他们应该是相似”的事实，第二部分是重建所示，公式如下：

$$\mathcal{L}_{mix} = \mathcal{L}_{2nd} + \alpha \mathcal{L}_{1st} + v \mathcal{L}_{reg} \quad (1.23)$$

注意 \mathcal{L}_{reg} 是正则化项。

\mathcal{L}_{2nd} 是重建损失，表达式为公式1.24：

$$\begin{aligned} \mathcal{L}_{2nd} &= \sum_{i=1}^n \| (\hat{x}_i - x_i) \odot b_i \|_2^2 \\ &= \| (\hat{X} - X) \odot B \|_F^2 \end{aligned} \quad (1.24)$$

这里面有一个小trick，就是公式里的 B ，它会增大对0的惩罚，即对于 $x_i = 0$, b_i 较大，对于 $x_i \neq 0$, b_i 较小。这里是基于两个考虑：

1. 一般来说邻接矩阵都比较稀疏，如果不对向量中的0做特殊惩罚，容易得到 $\vec{0}$ 的平凡解。
2. 原始图中两个node之间没有边(邻接矩阵中对应值为0)，并不代表他们真的没有关系，可能因为图不完整或者两个节点不是直接相连而是 $K - hop$ 邻域($K > 1$)。

\mathcal{L}_{1st} 控制有边的两个node，embedding得到的向量接近，表达式为公式1.25：

$$\mathcal{L}_{2nd} = \sum_{i,j}^n s_{i,j} \| (\hat{y}_i^{(K)} - y_i^{(K)}) \|_2^2 \quad (1.25)$$

其中 $s_{i,j}$ 表示两个节点的边权。

1.12.GraphGAN

GraphGAN[2018] : [GraphGAN: Graph Representation Learning with Generative Adversarial Nets](#)

如果评出最近两年最火的模型是什么？那么GAN一定在候选名单中，GAN和VAE是目前最优秀也是最经常拿来比较的两个深度生成模型，顺便提一句，最近有一篇paper: [Variational Inference: A Unified Framework of Generative Models and Some Revelations](#)将GAN和VAE统一到变分推断的框架下，作者大喊一句：你们别打了，都是自家兄弟，相煎何太急😊

回归正题，最火的模型当然也要在最火的方向参合一脚，这就是GraphGAN，类似的模型还有ANE(Adversarial Network Embedding)，这里只介绍GraphGAN。

GraphGAN思想是，生成器生成一条图中不存在的边，判别器判断一条边是否为图中存在的边，当训练完全以后，生成器生成的边和图中真实存在的边无法分辨，即embedding成功，换句话说**生成器逼近网络一阶信息**。

先看GraphGAN的目标函数：

$$\min_{\theta_G} \max_{\theta_D} V(G, D) = \sum_{c=1}^V (\mathbb{E}_{v \sim p_{true}(\cdot | v_c)} [\log D(v, v_c; \theta_D)]) + (\mathbb{E}_{v \sim G(\cdot | v_c; \theta_G)} [\log (1 - D(v, v_c; \theta_D))]) \quad (1.26)$$

公式1.26可以这样理解：

1. 固定 θ_D 看 \min_{θ_G} ，只有后面一项和 θ_G 有关，最小化后面一项相当于最大化 $D(v, v_c; \theta_D)$ ，其中 v 是从 $G(\cdot | v_c; \theta_G)$ 采样的，换句话说 v 是生成器 G 生成的样本。也就是说 \min_{θ_G} 的目的是让判别器 D 认为生成器 G 产生的样本为真，这是优化生成器。
2. 固定 θ_G 看 \max_{θ_D} ，两项都和 θ_D 有关，对于第一项，最大化第一项相当于最大化 $D(v, v_c; \theta_D)$ ，其中 v 是从 $p_{true}(\cdot | v_c)$ 采样的，换句话说是让判别器 D 认为真实的样本为真；对于第二项，最大化第二项相当于最小化 $D(v, v_c; \theta_D)$ ，其中 v 是从 $G(\cdot | v_c; \theta_G)$ 采样的，换句话说是让判别器 D 认为生成器 G 产生的样本为假。这是优化判别器。

所有的GAN的loss函数(最小最大游戏)都可以这么理解。现在看看生成器和判别器的形式：

$$D(v, v_c) = \sigma(d_v^T \cdot d_{v_c}) = \frac{1}{1 + \exp(-d_v^T \cdot d_{v_c})} \quad (1.27)$$

$$G(v|v_c) = \frac{\exp(g_v^T \cdot g_{v_c})}{\sum_{v \neq v_c} \exp(g_v^T \cdot g_{v_c})} \quad (1.28)$$

如上，判别器是一个sigmoid函数，生成器是一个softmax函数，实际上你可以设计任何合理的生成器与判别器。正如我们所见，式1.27是一个softmax，我们需要优化它，论文中作者设计了一种叫做Graph Softmax的优化方法，这里不详细介绍了，有兴趣的同学移步论文。

1.13. 总结

前面说了很多GE的模型，从古典方法，到只考虑结构的模型，再到考虑节点和边extra info的模型，再到深度模型，大家可能觉得眼花缭乱，并不知道实际中应该用哪个模型。

这里我提示一句，你选择的模型一定要和你的实际问题相关：

- 比如如果你的问题更加关注内容相似性(局部邻域相似性)那么你可以选择node2vec, LINE, GraRep等；
- 如果你的问题更加关注结构相似性，你可以选择struc2vec，这里可以简单说一下为什么蚂蚁金服风控模型使用struc2vec相比node2vec有质的提升，这是因为在风控领域，你可信并不能代表你的邻居可信(有些“大V”节点邻居众多)，但是一个直观的感觉是，如果两个人在图中处于相似的地位(比如两个“大V”)，那么这两个人应该都可信或都不可信，并且一般来说这样的两个人(节点)都相聚比较远；
- 再者如果你的模型需要考虑节点和边的额外信息，那么你可选择CANE, CENE, Trans-net等；
- 如果你想处理大规模易变图，你可以采用GraphSAGE，或者先使用其他GE方法，再使用GraphSAGE归纳学习；
- 如果你想微调你的模型，你可选择GraphGAN；
- 甚至你可以选择很多GE方法，并将得到的embedding向量进行聚合，比如concat等方式；

总之，选用什么模型取决于你的问题！

2. Graph CNN

前面是GE领域的概述了，现在要说的是另一个事情就是图卷积(Graph Convolutional Neural Network, GCN)，其实这相对于GE是另一个思路，GE基于高维相似性映射到低维以后也是相似的，我们想使用深度学习应该先学习图嵌入（借鉴nlp中的word2vec），而GCN就是直接端到端分类或回归，当然也可以先使用进行图嵌入，拿到嵌入向量以后进行gcn，另外gcn很多时候会附带的产生图嵌入。

一般来说GCN分为两大类，一种是谱图卷积，是在傅里叶域进行卷积变换；另一种是非谱图卷积(也叫做“空间域卷积”)，是直接在Graph上进行卷积。我们分别介绍这两类，图2.1是GCN的大家族。

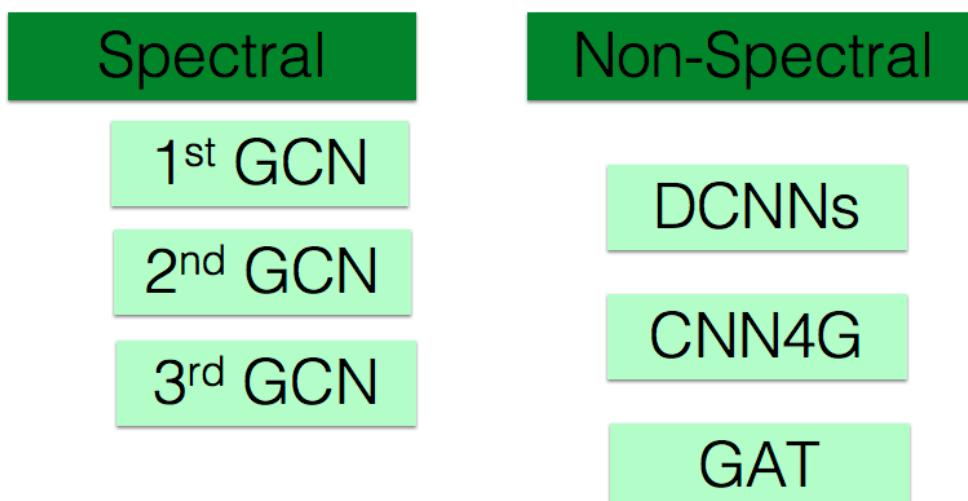


图2.1 Graph CNN大家族

2.1. Spectral Convolution

谱图卷积需要一些额外的知识，我们先回忆一些知识😊

2.1.1. Graph上的傅里叶变换

傅里叶变换是将时域上的函数转换为频域上函数，是对于同一个函数的不同视角，它的数学公式如下：

$$F(w) = \mathcal{F}(f(t)) = \int f(t)e^{-iwt} dt \quad (2.1)$$

公式2.1意义就是傅里叶变换是时域信号 $f(t)$ 与基函数 e^{-iwt} 的积分，这个基函数可不是随便选的，它实际上是拉普拉斯算子(Δ)的特征函数，那什么是拉普拉斯算子呢？拉普拉斯算子是笛卡尔坐标系 x_i 中的所有非混合二阶偏导数之和，对于 n 维笛卡尔坐标系，其表达式为：

$$\Delta = \sum_{i=1}^n \frac{\partial^2}{\partial x_i^2} \quad (2.2)$$

拉普拉斯算子的特征方程为：

$$\Delta g = \lambda g \quad (2.3)$$

而基函数 e^{-iwt} 正是方程2.3的一个解！

$$\Delta e^{-iwt} = \frac{\partial^2}{\partial x_i^2} e^{-iwt} = -w^2 e^{-iwt} \quad (2.4)$$

其中特征值 $\lambda = -w^2$ ，也就是 w 和特征值 λ 密切相关。

最终我们的结论是：傅里叶变换就是时域信号与拉普拉斯算子特征函数的积分！

那么，我们是否可以把以上的结论推广到Graph上呢，即我们能否得到：Graph上的傅里叶变换就是时域信号与Graph拉普拉斯算子特征向量的求和？

显然可以，因为这是从整体推特殊的逻辑，这里我们将拉普拉斯算子换为Graph拉普拉斯算子(这里先不要管Graph拉普拉斯算子是什么，后面会说)，将特征函数换为特征向量(特征函数可以认为无限维的特征向量)，将积分换为求和，这是合理的。那么有 N 个节点的Graph上的傅里叶变换表达式便为：

$$F(\lambda_l) = \hat{f}(\lambda_l) = \sum_{i=1}^N f(i) * u_l(i) \quad (2.5)$$

其中， λ_l 为Graph拉普拉斯算子第 l 个特征值， u_l 为第 l 个特征向量，式2.5是 λ_l 对应的傅里叶变换，对于 Graph拉普拉斯算子，它有 N 个特征值，那么全部写出来便是：

$$\begin{pmatrix} \hat{f}(\lambda_1) \\ \hat{f}(\lambda_2) \\ \dots \\ \hat{f}(\lambda_N) \end{pmatrix} = \begin{pmatrix} u_1(1) & u_1(2) & \dots & u_1(N) \\ u_1(1) & u_1(2) & \dots & u_1(N) \\ \dots & \dots & \dots & \dots \\ u_1(1) & u_1(2) & \dots & u_1(N) \end{pmatrix} \begin{pmatrix} f(\lambda_1) \\ f(\lambda_2) \\ \dots \\ f(\lambda_N) \end{pmatrix} \quad (2.6)$$

将式2.6写成矩阵形式即为：

$$\hat{f} = Uf \quad (2.7)$$

那么现在还有一个隐藏的问题，什么是Graph的拉普拉斯算子？图的拉普拉斯算子定义如下：

$$L = D - W \quad (2.8)$$

其中 W 是邻接矩阵， D 是度矩阵，即 $D_{i,i}$ 等于 W 矩阵第*i*行的和， $D_{i,j,i\neq j} = 0$ ，即非对角线上元素为0.至于Graph拉普拉斯算子为什么这么定义，一两段文字说不清楚，感兴趣的大家可以搜搜图谱理论的知识，我们姑且认为这样定义是合理的。

2.1.2. Graph上的傅里叶逆变换

有了傅里叶变换，那么怎么将频率函数变换为时域函数呢？这就是傅里叶逆变换，表达式为：

$$\mathcal{F}^{-1}[F(w)] = \frac{1}{2\pi} \int F(w) e^{-iwt} dw \quad (2.9)$$

观察式2.9和式2.1很相似，差一个常数系数，所以类似的，图上傅里叶逆变换为：

$$\begin{pmatrix} f(\lambda_1) \\ f(\lambda_2) \\ \dots \\ f(\lambda_N) \end{pmatrix} = \begin{pmatrix} u_1(1) & u_1(2) & \dots & u_1(N) \\ u_1(1) & u_1(2) & \dots & u_1(N) \\ \dots & \dots & \dots & \dots \\ u_1(1) & u_1(2) & \dots & u_1(N) \end{pmatrix} \begin{pmatrix} \hat{f}(\lambda_1) \\ \hat{f}(\lambda_2) \\ \dots \\ \hat{f}(\lambda_N) \end{pmatrix} \quad (2.10)$$

写成矩阵形式为：

$$f = U^T \hat{f} \quad (2.11)$$

2.1.3. Graph上的谱图卷积

前面说了这么多傅里叶变换的东西，这跟我们要说的卷积有什么关系呢？卷积定理可以回答这个问题

卷积定理：函数卷积的傅立叶变换是函数傅立叶变换的乘积。

听起来很拗口，但是用公式表达一目了然：

$$\mathcal{F}(f * h) = \hat{f}(w) \cdot \hat{h}(w) \quad (2.12)$$

其中*表示卷积，式2.12和式2.9可以得到：

$$f * h = \mathcal{F}^{-1}(\hat{f}(w) \cdot \hat{h}(w)) = \frac{1}{2\pi} \int \hat{f}(w) \cdot \hat{h}(w) e^{-iwt} dw \quad (2.13)$$

式2.13表示了卷积与傅里叶变换的关系。我们知道机器学习中普通的卷积是处理网格数据的，例如图像等，对于不规则的Graph数据，直接推广CNN是不容易的，那么我们可以换个思路，我们可以利用式2.13在傅里叶域先做乘积，然后做傅里叶逆变换，这就等价于在原空间直接做卷积！

Graph上的谱图卷积表达式为：

$$(f * h)_G = U \begin{pmatrix} \hat{h}(\lambda_1) & & & \\ & \hat{h}(\lambda_2) & & \\ & & \dots & \\ & & & \hat{h}(\lambda_N) \end{pmatrix} U^T f \quad (2.14)$$

写成矩阵形式为：

$$(f * h)_G = U((U^T f) \odot (U^T h)) \quad (2.15)$$

2.1.4. 1st Spectral Convolution

1st Spectral Convolution: [Spectral Networks and Locally Connected Networks on Graphs](#)

公式2.15是我们推导半天的最终成果了， $U^T f$ 和 $U^T h$ 分别为Graph原特征的傅里叶变换和卷积核的傅里叶变换，两者点乘，再左乘 U 即为原空间的卷积。

那么第一代谱图卷积就呼之欲出了，只需要把 $\hat{h}(\lambda_l)$ 作为卷积核参数即可！表达式如下：

$$y_{output} = \sigma(U \begin{pmatrix} \theta_1 & & & \\ & \theta_2 & & \\ & & \ddots & \\ & & & \theta_N \end{pmatrix} U^T x) \quad (2.16)$$

大功告成！我们现在可以按照式2.16进行谱图卷积了。但是1st Spectral Convolution还有一些不完美的地方：

- 计算量大，每次卷积运算都要进行矩阵相乘，计算复杂度 $O(N^3)$ (即使使用优化的斯坦福算法复杂度也是 $O(N^{2.37})$)，且需要特征分解，特征分解复杂度为 $O(N^3)$
- 没有spatial localization(空间局部性)，传统的CNN受控于感受野，每次卷积只需考虑一小部分邻域，但是GCN是每一次卷积都要考虑所有的节点。
- 参数个数 $O(N)$ ，传统的CNN参数个数是常数，例如3*3的卷积核有9个参数。

2.1.5. 2nd Spectral Convolution

2nd Spectral Convolution[2016] : [Convolutional neural networks on graphs with fast localized spectral filtering](#)

1st Spectral Convolution有一些不完美的地方，2nd Spectral Convolution就是解决这些问题的！我们先把图卷积重写：

$$y = \sigma(U g_\theta(\Lambda) U^T x) \quad (2.17)$$

其中 Λ 是特征值组成的对角矩阵，那么对于1st Spectral Convolution，

$$g_\theta(\Lambda) = diag(\theta) \quad (2.18)$$

对于2nd Spectral Convolution，把 $g_\theta(\Lambda)$ 改为如下形式：

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k \quad (2.19)$$

我们把式2.19代入到2.17得到第二代卷积的公式：

$$\begin{aligned}
y &= \sigma(U \sum_{k=0}^{K-1} \theta_k \Lambda^k U^T x) \\
&= \sigma(\sum_{k=0}^{K-1} \theta_k U \Lambda^k U^T x) \\
&= \sigma(\sum_{k=0}^{K-1} \theta_k L^k x)
\end{aligned} \tag{2.20}$$

式2.20推导过程中应用了特征分解的性质 $U \Lambda^k U^T = L^k$.

公式2.20即为第二代卷积公式，当我们事先把 L^k 计算好，每一步只需要向量与矩阵相乘，复杂度降低为 $O(KN^2)$ ，如果使用稀疏矩阵算法，复杂度为 $O(K|E|)$ ，但是第二代卷积怎么体现 spatial localization 呢？

这里介绍一个拉普拉斯算子的性质：

Lemma 2.1: 如果 $d_G(m, n) > s$, 则 $L_{m,n}^s = 0$, $d_G(m, n)$ 表示图 G 中节点 m 和 n 的最短距离

证明：首先意识到对于 L 来说，如果两个节点 i, j 没有边相连，那么 $L_{i,j} = 0$ 。

$(L^s)_{m,n} = \sum L_{m,k_1} L_{k_1,k_2} \dots L_{k_{s-1},n}$, 我们意识到 L 中除了对角线为正数，其他地方都是负数，也就是对于所有的 k_1, k_2, \dots, k_{s-1} , $L_{m,k_1} L_{k_1,k_2} \dots L_{k_{s-1},n}$ 是同号的，如果 $(L^s)_{m,n} \neq 0$, 那么至少存在一组 k_1, k_2, \dots, k_{s-1} 使得 $L_{m,k_1} L_{k_1,k_2} \dots L_{k_{s-1},n}$ 不为 0, 即所有的 $L_{m,k_1}, L_{k_1,k_2}, \dots, L_{k_{s-1},n}$ 都不为 0。也就是说存在一条通路 $m \rightarrow k_1 \rightarrow k_2 \rightarrow \dots \rightarrow k_{s-1} \rightarrow n$, 这条通路长度为 s , 即 $d_G(m, n) = s$, 这与条件 $d_G(m, n) > s$ 冲突，故原命题正确，证毕。

由 Lemma 2.1 可知，式2.20的卷积公式仅使用 $K - hop$ 邻域，即感受野半径是 K 。

论文中提到另一种优化方式，将 L^k 利用切比雪夫展开来优化计算复杂度，优化后计算复杂度也为 $O(K|E|)$ ，我本身没有看出有什么加速。

切比雪夫展开：任何 k 次多项式都可以使用切比雪夫多项式展开

切比雪夫展开公式为：

$$\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_k(x) &= 2xT_{k-1}(x) - T_{k-2}(x)
\end{aligned} \tag{2.21}$$

将2.20中的 L^k 使用切比雪夫展开可得：

$$y == \sigma(\sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})) \tag{2.22}$$

其中 $\tilde{L} = 2 \frac{L}{\lambda_{max}} - I_N$, 这是为了将 L 的谱半径约束到 $[-1, 1]$, 使得在连续相乘的时候不会爆炸。

总结 2nd Spectral Convolution:

- 计算复杂度降低为 $O(K|E|)$
- 只考虑 $K - hop$ 邻域，有一定的空间局部性

2.1.6. 3rd Spectral Convolution

3rd Spectral Convolution[2017] : [Semi-supervised classification with graph convolutional networks](#)

到第二代，谱图卷积已经基本成型，第三代GCN改动不大了，概括两个点：

1. 令 $K=1$ ，即每层卷积只考虑直接邻域，这类似于CNN中 3×3 的kernel。
2. 深度加深，宽度减小(深度学习的经验，深度>宽度)。

对于公式2.22，令 $\lambda_{max} \approx 2, K = 1$ 可得：

$$\begin{aligned} g_{\theta'} \star x &\approx \theta'_0 x + \theta'_1 (L - I_N) x \\ &= \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \end{aligned} \quad (2.23)$$

公式2.23推导应用了归一化的 $L = D^{-\frac{1}{2}} (D - A) D^{-\frac{1}{2}} = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ 。

再进一步简化，假设 $\theta'_0 = -\theta'_1$ ，则式2.23可以表示为：

$$g_{\theta'} \star x \approx \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) x \quad (2.24)$$

由于 $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ 的谱半径范围是 $[0, 2]$ ，进一步约束为：

$$I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} \quad (2.25)$$

其中 $\tilde{A} = A + I_N, D_{ii} = \sum_j \tilde{A}_{ij}$ 。

式2.25应用多通道卷积，并表达为矩阵形式如下：

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \quad (2.26)$$

其中 $X \in \mathbb{R}^{N \times C}, \Theta \in \mathbb{R}^{C \times F}, Z \in \mathbb{R}^{N \times F}$ ， N, C, F 分别代表节点个数，通道数和卷积核数。

总结3rd Spectral Convolution：

- 计算复杂度降低为 $O(|E|)$
- 只考虑 $1 - hop$ 邻域，通过堆叠多层，获得更大的感受野

2.2. Spatial Convolution

前面讲的都是基于谱图理论的卷积，思想是在原空间卷积不好做，那就在图拉普拉斯算子的傅里叶域做。那么是否可以直接在原空间做卷积呢？答案是可以的，Spatial Convolution(空间域卷积)就是直接在图上做卷积，我们知道卷积核大小是固定的，也就是我们需要选择固定大小的邻域来做卷积，但是相比于规则网格结构，graph中的节点通常具有不同个数的邻域，这是在原空间做卷积的主要困难。

2.2.1. DCNNs

DCNNs[2016] : [Diffusion-convolutional neural networks](#)

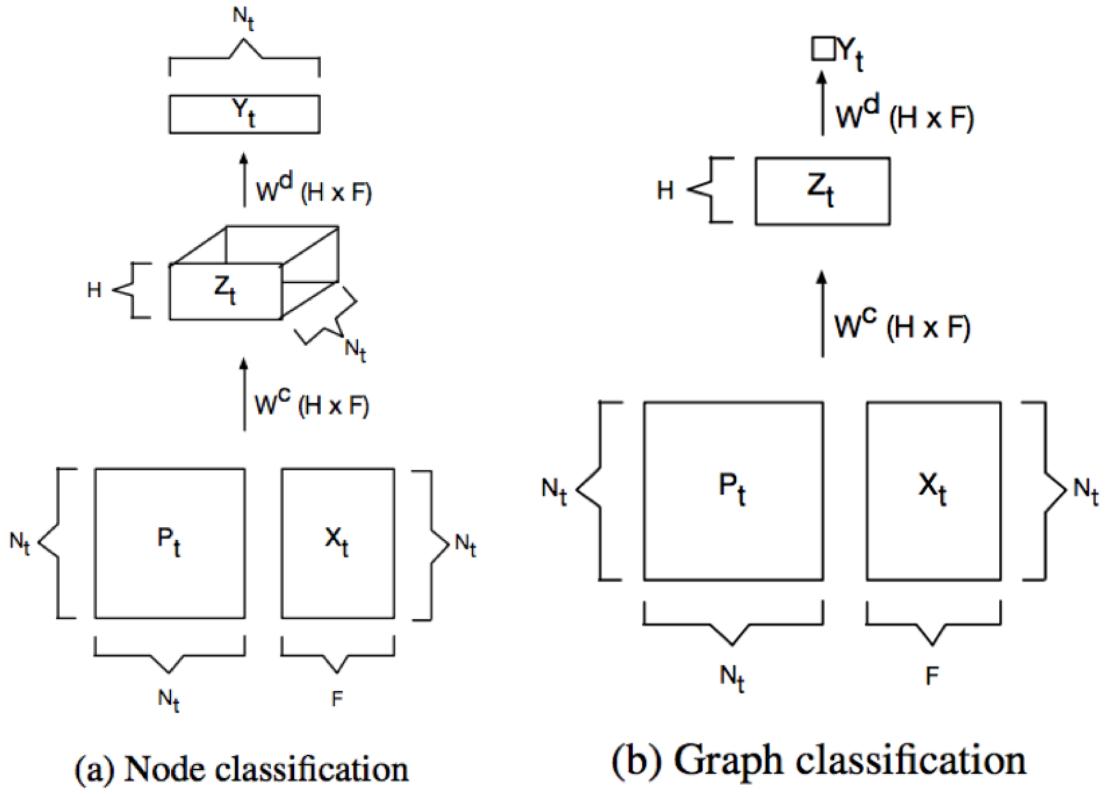


图2.2 DCNNs用于Node分类和Graph分类的示意图

DCNNs可以应用于两种任务，一个是节点分类，一个Graph分类，我们先说节点分类任务。

对于节点分类任务，卷积公式为：

$$Z_{tijk} = f(W_{jk}^c \cdot \sum_{l=1}^{N_t} P_{tijl}^* X_{tlk}) \quad (2.27)$$

其中 X_{tlk} 表示第 t 个 Graph G_t ，第 l 节点的第 k 个特征； P_{tijl}^* 表示 G_t 的 i 节点经过 j 跳以后到达 l 节点的概率， $P_{tijl}^* = P_{tik}^j$ ， P_t 为 G_t 的转移矩阵， P_t^j 为 G_t 的 j 跳转移矩阵； f 为非线性激活函数； W^c 为卷积核。

将式2.27表达为张量形式为：

$$Z_t = f(W^c \odot P_t^* X_t) \quad (2.28)$$

对于Graph分类任务，对于 G_t 上的每个节点 i ，取 H 跳的所有邻居的第 F 个特征求均值，乘以权值 W^c 即可，表达式如下：

$$Z_t = f(W^c \odot 1_{N_t}^T P_t^* X_t / N_t) \quad (2.29)$$

得到 Z 以后，按照图图2.2架构网络即可。

DCNNs总结：

- 卷积核参数 $O(H * F)$
- 考虑 H 跳邻域

2.2.2. CNN4G

CNN4G[2016] : [Learning convolutional neural networks for graphs](#)

该模型是针对Graph分类任务的，主要思路是选出一些节点代表整个Graph，并为每个节点选出特定个数的邻域，然后在每个节点和其邻域节点组成的矩阵上做卷积。

算法步骤：

1. 找出 w 个节点，这 w 个节点可以代表整个Graph，文章使用的是centrality的方法，即选出 w 个最中心的节点。具体的计算该node与所有node距离和，距离和越小越中心，如图2.3中，左边红色节点与其他所有节点距离和为23，右边红色节点为25，所以左边红色节点更中心，应该优先选它。
2. 为每个节点找出 K 个邻域，作为该节点的感受野，如图2.4
 - 2.1 先选出一阶邻域，一阶邻域不够就选二阶邻域，直到至少 K 个节点，或者没有节点可选。
 - 2.2 选出的节点如果不够，补充0向量，如果多了就按照中心rank排名删除后面的节点。
3. 选出的 w 个节点，每个节点和其邻域节点都可以组成规则的 $(K + 1) * F$ 的矩阵，而所有的边也可以组成一个 $(K + 1) * (K + 1) * F$ 的张量，在这些张量上做卷积，如图2.5。

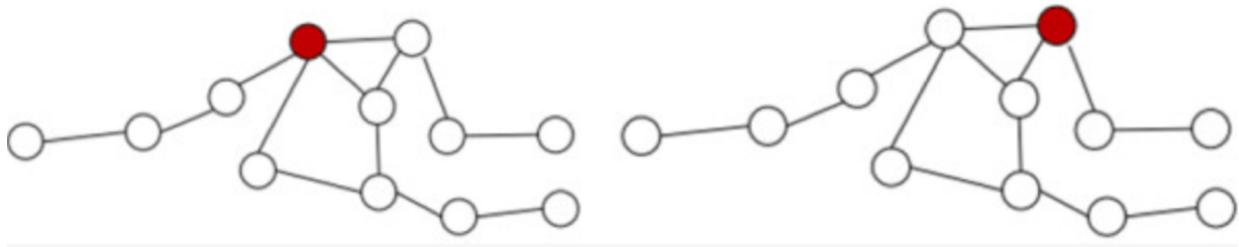


图2.3 中心rank排名计算方式

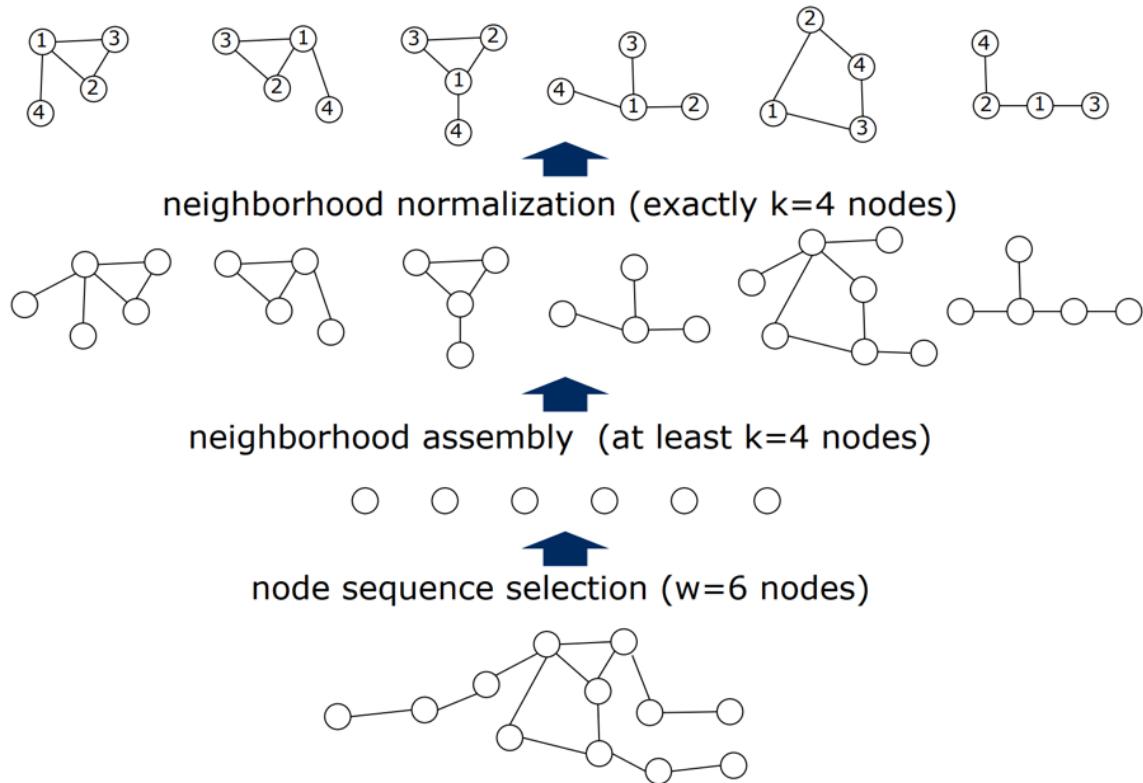


图2.4 邻域的选择示意图

Convolutional architecture

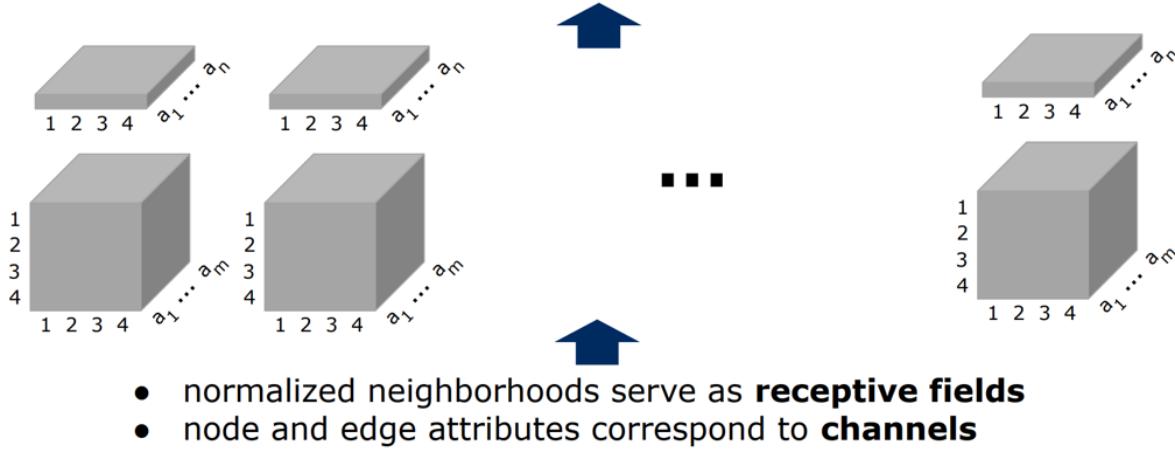


图2.5 对每组节点卷积示意图

2.2.3. GATs

GATs[2018] :[Graph Attention Networks](#)

该模型结合了注意力机制，可以动态计算节点之间的相关性，并且该模型可以直推式学习，也可以归纳式学习，是非谱图卷积领域里state-of-the-art的方法，在很多公开数据集取得了非常好的效果。

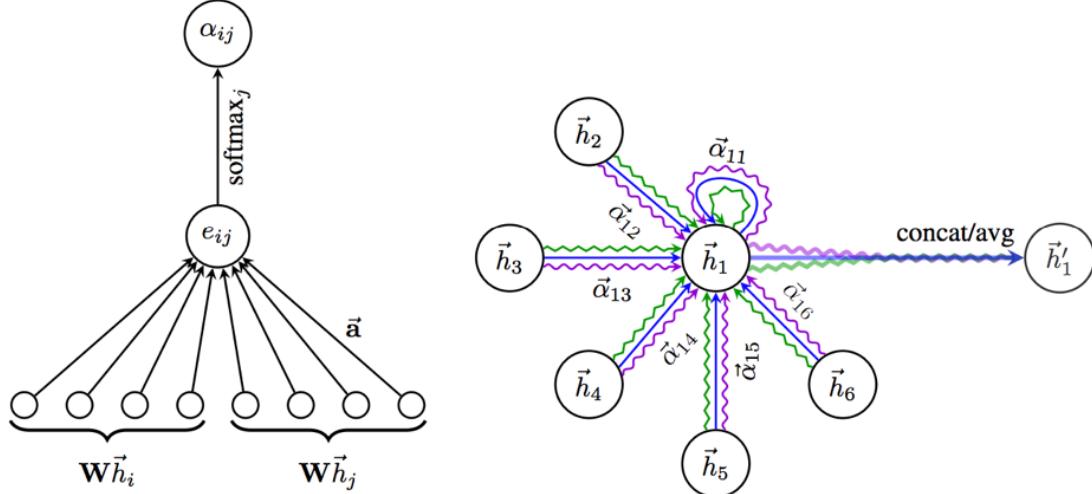


Figure 1: **Left:** The attention mechanism $a(\vec{W}\vec{h}_i, \vec{W}\vec{h}_j)$ employed by our model, parametrized by a weight vector $\vec{a} \in \mathbb{R}^{2F'}$. **Right:** An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain \vec{h}'_1 .

图2.6 注意力机制与multi-head注意力机制

这个模型最最重要的就是attention，即动态计算邻域节点和该节点的强度关系，我们将看到只要用到attention的思路都是一样的，论文[Attention is all you need](#)总结的很精准。

GATs的输入是节点特征的集合 $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \dots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^F$ ，其输出是经过变换后的节点特征集合 $\mathbf{h}' = \{\vec{h}'_1, \vec{h}'_2, \dots, \vec{h}'_N\}, \vec{h}'_i \in \mathbb{R}^{F'}$ 。

注意力系数计算如下：

$$e_{ij} = a(\vec{W}\vec{h}_i, \vec{W}\vec{h}_j) \quad (2.30)$$

其中 a 表示注意力机制函数，将上式归一化得到：

$$\alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \quad (2.31)$$

其中 \mathcal{N}_i 表示节点*i*的邻域节点集合。

将注意力机制函数用单独一层神经网络来表示，参数为 $\vec{a} \in \mathbb{R}^{2F'}$ ，则式2.31可写为：

$$\alpha_{ij} = \frac{\exp(\vec{a}^T [\vec{W}\vec{h}_i \parallel \vec{W}\vec{h}_j])}{\sum_{k \in \mathcal{N}_i} \exp(\vec{a}^T [\vec{W}\vec{h}_i \parallel \vec{W}\vec{h}_j])} \quad (2.32)$$

其中 \parallel 表示concat运算。一旦得到注意力系数，则输出即为变换后的特征与注意力系数的线性组合：

$$\vec{h}'_i = \sigma \left(\sum_{k \in \mathcal{N}_i} \alpha_{ij} \vec{W}\vec{h}_j \right) \quad (2.33)$$

如图2.6所示，我们可以设计 K 注意力系数，并将它们的结果聚合起来，聚合函数可以选择concat或者avg等：

$$\begin{aligned} \vec{h}'_i &= \left\|_{k=1}^K \sigma \left(\sum_{k \in \mathcal{N}_i} \alpha_{ij}^k W^k \vec{h}_j \right) \right\| \\ \vec{h}'_i &= \sigma \left(\frac{1}{K} \sum_{k=1}^K \sum_{k \in \mathcal{N}_i} \alpha_{ij}^k W^k \vec{h}_j \right) \end{aligned} \quad (2.34)$$

我们可以用多个GAT layer嵌套起来，已获得更大感受野。

2.3. 总结

一般来说GCN分为两大类，一种是谱图卷积，是在傅里叶域进行卷积变换；另一种是非谱图卷积(也叫做“空间域卷积”)，是直接在Graph上进行卷积。谱图卷积有一套统一的理论支持，但是有时候会受到图拉普拉斯算子的限制；空间域卷积更加灵活，主要困难在于选择定量邻域上，很多模型千奇百怪，没有统一理论，更多是一种trick。

3. 基于Graph的序列建模

最后一块说说图序列建模吧，这类问题相对较少，比较典型的例子是交通路网数据，短期内路网结构可以认为是不变的，但是里面的数据是流动的，换句话说Graph是固定的，但是里面的数据是时间序列的，类比于视频预测一样，预测接下来一段时间，图中数据的变化，例如路网eta，路网流量预测等问题。

这部分研究相对也少，主要是参照ConvLSTM的做法，将RNN和GCN融合。

先简单说一下ConvLSTM的思路，假设我想做视频预测的任务，那么我需要获取每一帧图像的空间(或几帧得空间)信息，同时视频的连续帧之间具有时间相关性，也就是说我需要同时获取空间和时间相关性。但是传统的LSTM输入是一维向量，如果我将视频每一帧图像展开为一维向量，那么将失去他们的空间信息。如果我可以让LSTM可以处理输入为二维的图像，那该多好啊，我们就可以同时获取其空间和时间信息，这就是ConvLSTM解决的问题。简单来说ConvLSTM将LSTM中所有的矩阵相乘运算换为卷积网络，详细内容大家可以参看论文[Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting](#)

3.1. DCRNNs

DCRNNs[2018] : [Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting](#)

Graph上的时间序列建模问题可以描述为：

$$[X^{(t-T'+1)}, \dots, X^{(t)}; \mathcal{G}] \rightarrow h(\cdot) \rightarrow [X^{(t+1)}, \dots, X^{(t+T)}] \quad (3.1)$$

即Graph固定下，根据前一段序列预测后一段序列，是一个seq2seq问题。

有了ConvLSTM的思路，我们很自然可以想到，将GCN和RNN结合起来，改造出新的适合处理Graph输入的RNN模型即可，DCRNNs就是将DCNNs和GRU结合起来的例子，为了可以抓取上游和下游的信息，作者对DCNNs增加了双向扩散，双向扩散卷积公式如下：

$$X_{:,p} \star_{\mathcal{G}} f_{\theta} = \sum_{k=0}^{K-1} (\theta_{k,1}(D_o^{-1}W^T)^k + \theta_{k,2}(D_I^{-1}W^T)^k) \quad (3.2)$$

其中 D_o 和 D_I 分别表示出度矩阵和入度矩阵，式3.2计算的是第 p 的特征(通道)的卷积公式，对于有 Q 个卷积核的卷积运算表达式如下：

$$H_{:,q} = a \left(\sum_{p=1}^P X_{:,p} \star_{\mathcal{G}} f_{\Theta_{q,p},:} \right) \quad (3.3)$$

其中 $q = 1, 2, \dots, Q$ ，那么将式3.3应用到GRU中，可以得到DCGRU表达式：

$$\begin{aligned} r^{(t)} &= \sigma(\Theta_r \star_{\mathcal{G}} [X^{(t)}, H^{(t-1)}] + b_r) \\ u^{(t)} &= \sigma(\Theta_u \star_{\mathcal{G}} [X^{(t)}, H^{(t-1)}] + b_u) \\ C^{(t)} &= \tanh(\Theta_C \star_{\mathcal{G}} [X^{(t)}, (r^{(t)} \odot H^{(t-1)})] + b_C) \\ H^{(t)} &= u^{(t)} \odot H^{t-1} + (1 - u^{(t)}) \odot C^{(t)} \end{aligned} \quad (3.4)$$

实际上该问题是一个seq2seq建模问题，因此我们可以使用常用的encoder-decoder框架，如图3.1

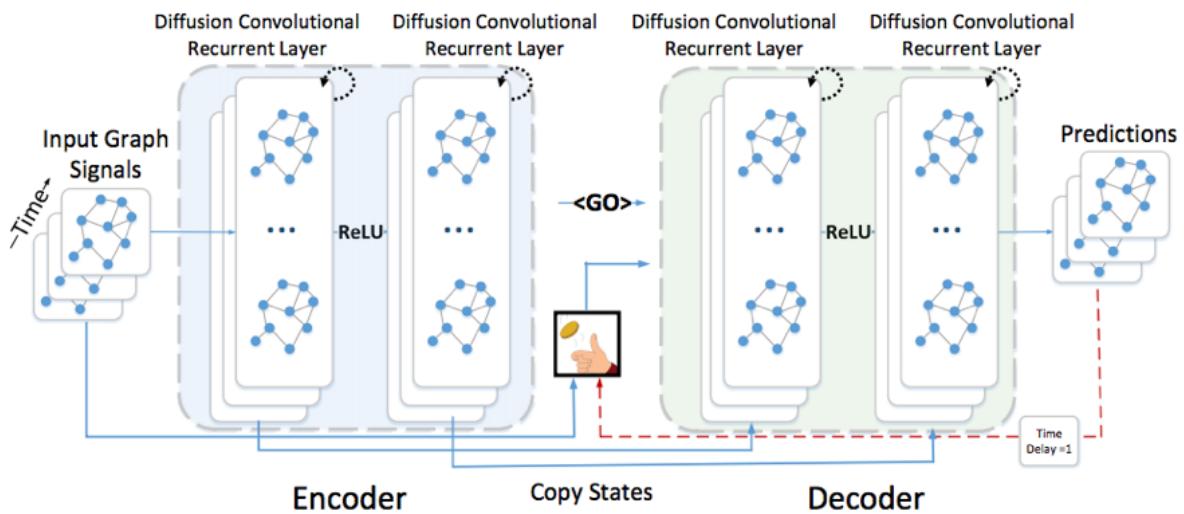


图3.1 DCRNNs示意图

3.2. GAT-LSTM

GAT-LSTM[2019] : Graph Attention LSTM(文章还未出版)

这篇论文是鄙人的一点小小贡献，和上面DCRNNs一样，只不过我将DCRNNs换成更加强大的GATs。我解决的是城市路网的交通流预测，城市路网是有潮汐现象的，所以其实路网之前的权值是不固定的，应该是变化的，DCRNNs路网权值是固定的，是一个距离的函数。

GAT-LSTM公式如下：

$$\begin{aligned}
 i^{(t)} &= \sigma(g(X^{(t)}; W_{xi}, a_{xi}) + g(H^{(t-1)}; W_{hi}, a_{hi}) + b_i) \\
 f^{(t)} &= \sigma(g(X^{(t)}; W_{xf}, a_{xf}) + g(H^{(t-1)}; W_{hf}, a_{hf}) + b_f) \\
 o^{(t)} &= \sigma(g(X^{(t)}; W_{xo}, a_{xo}) + g(H^{(t-1)}; W_{ho}, a_{ho}) + b_o) \\
 \mathcal{C}^{(t)} &= f^{(t)} \odot \mathcal{C}^{(t-1)} + i^{(t)} \odot \tanh(g(X^{(t)}; W_{xc}, a_{xc}) + g(H^{(t)}; W_{hc}, a_{hc}) + b_c) \\
 \mathcal{H}^{(t)} &= o^{(t)} \odot \tanh(\mathcal{C}^{(t)})
 \end{aligned} \tag{3.5}$$

图3.2是GAT-LSTM变换的示意图，图3.3是我使用的encoder-forecaster框架，其中虚线代表状态拷贝，使用encoder-forecaster框架是基于如下考虑：

- 编码-解码网络的解码器一般还需要输入，训练期间使用标准值，推断期间使用预测值；而编码-预测网络的预测网络是不需要额外输入的，这可以避免推断期间的误差传递
- 编码-解码网络的编码器和解码器可以是完全不相关的模型，但是编码-预测网络的编码器与预测器是对称的，这是因为编码器中高层次的状态已经获得了全局的时空关系，可以指导和更新预测器低层状态

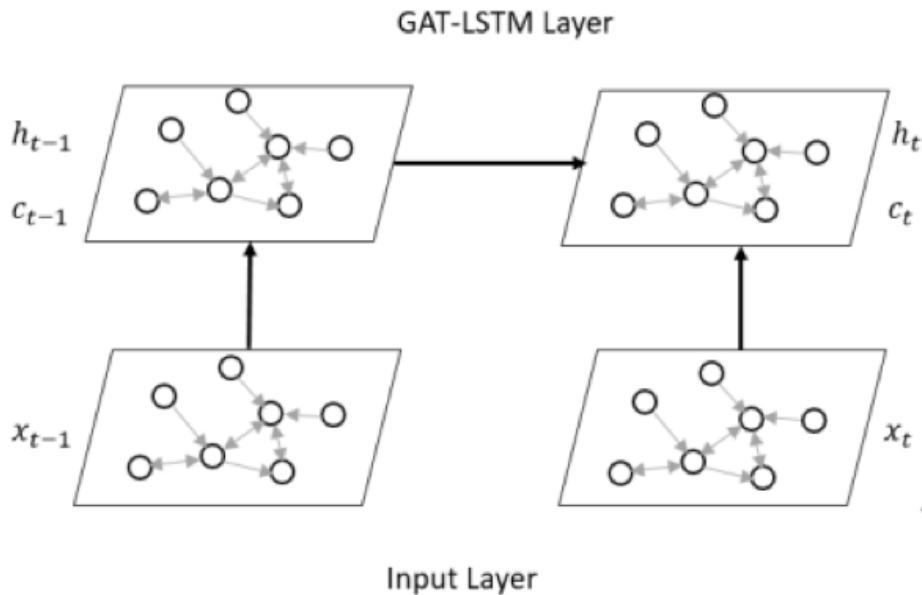


图3.2 GAT-LSTM特征变换示意图

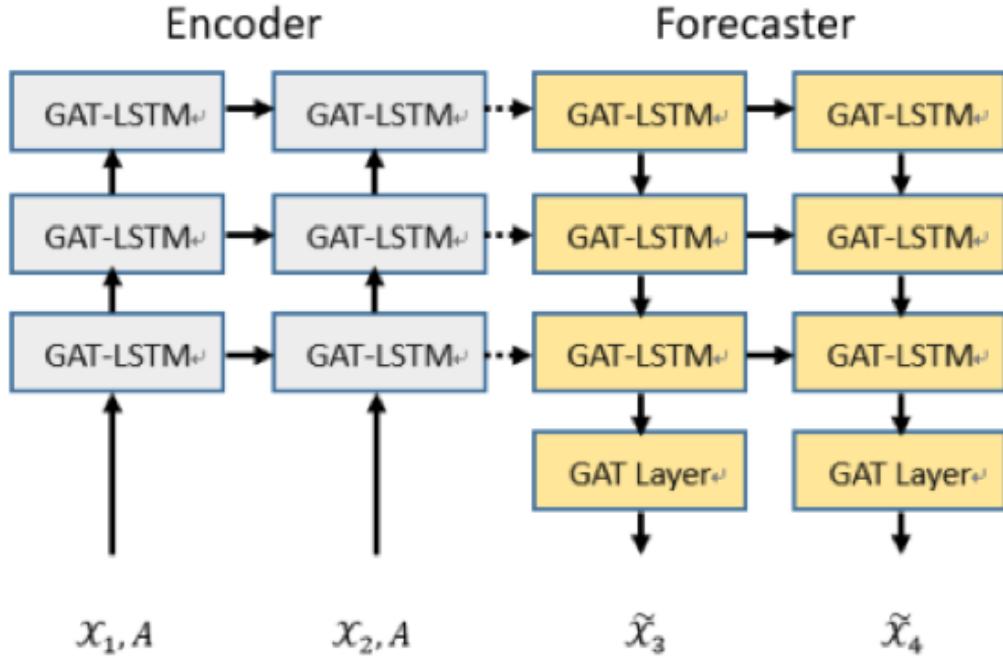


图3.3 encoder-forecaster框架示意图

3.3. 总结

基于Graph的序列建模研究的不多，目前主流方法是使用GCN改造RNN，是的模型可以同时获取Graph的空间相关性和时间相关性。

文章到此就算结束了，本文对当前主流的Graph神经网络进行了综述，最后总结一句话：如果你的问题适合建模为**Graph**，那么你应该把80%的经历花在构图上，只要你的**Graph**足够准确，数据足够精确且丰富，那么你可以有一万种方法解决你的问题。

以上全当是抛砖引玉，希望对大家有所帮助😊