图神经网络基础

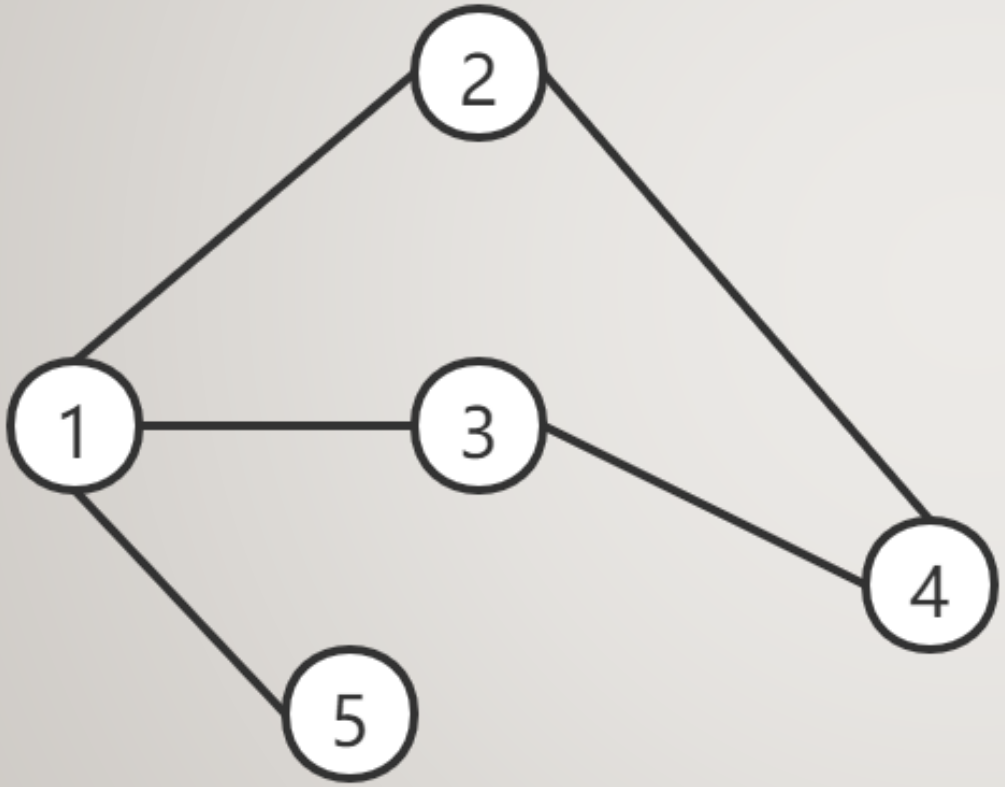
作者: 雷克斯掷骰子

2021-2

大纲

1. 图论基础 (3 – 11)
2. 图游走算法 (12 – 15)
3. 图神经网络层 (16 – 23)
4. 图采样 (24 – 26)
5. 图处理Python API 推荐(27)

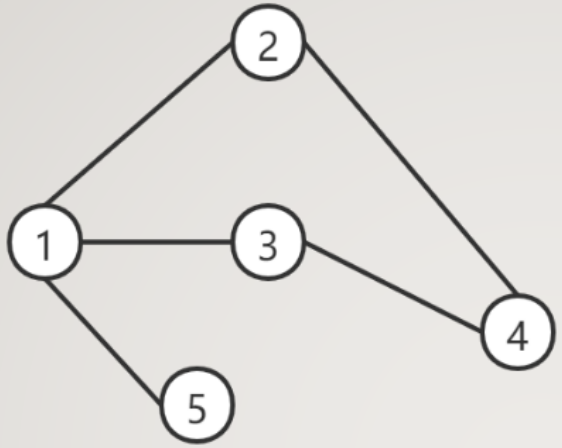
图论基础 – 什么是图



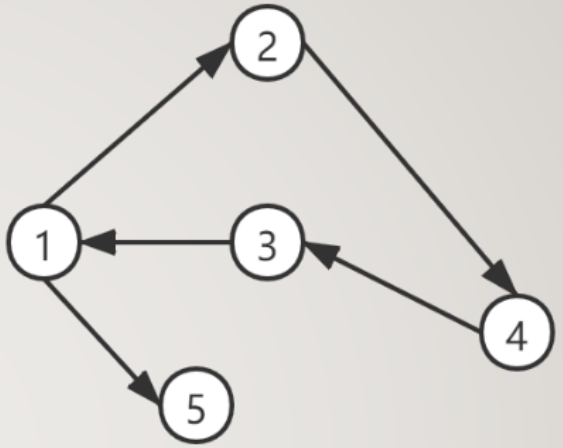
- 图的定义：图是描述复杂事务的数据表示形式，由节点和边组成，数学上一般表述为图 G = (V, E)。其中的V (vertical) 代表节点，可被理解为事物。而E (edge) 代表边，描述的是两个事物之间的关系。
- 图的优势：可轻松描述不规则数据，充分利用数据间关系信息。

图的分类

- 无向图与有向图

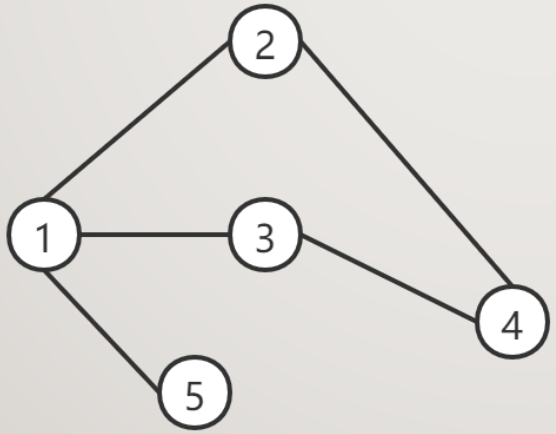


无向图

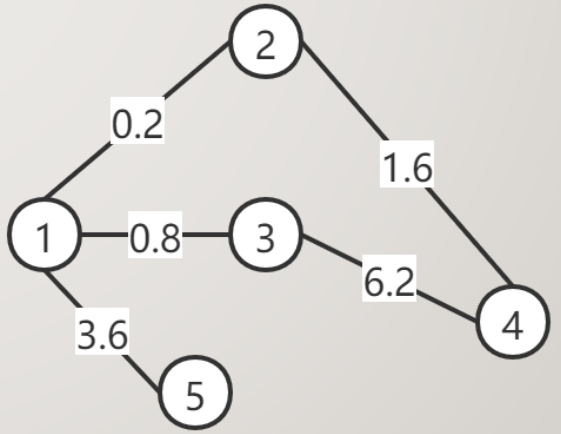


有向图

- 无权图与有权图

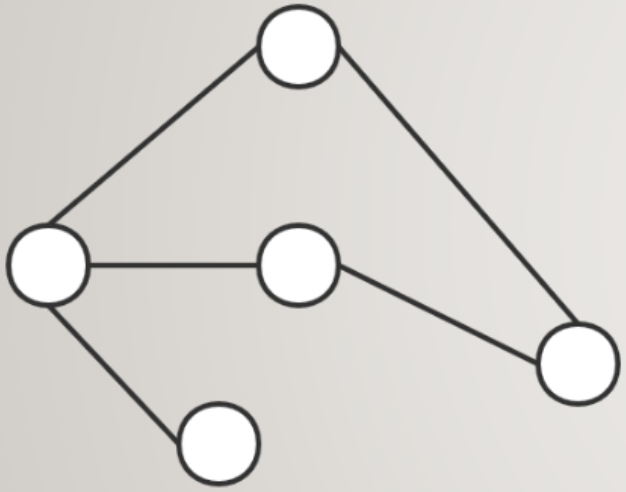


无权图

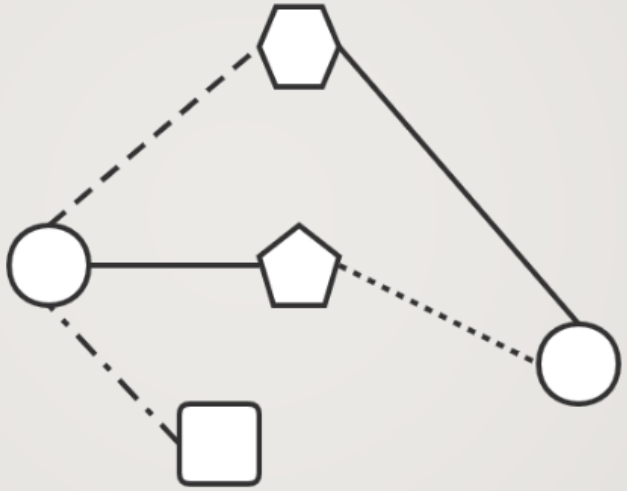


有权图

图的分类



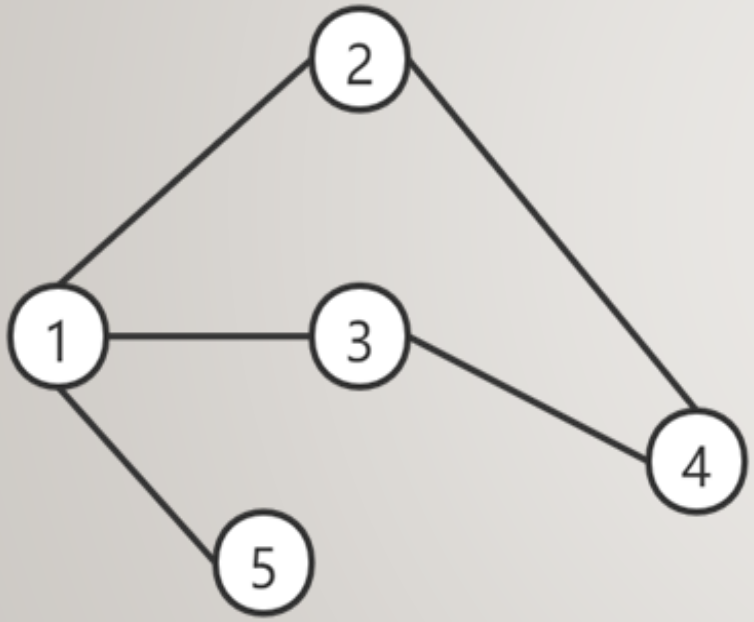
同构图



异构图

- 同构图与异构图
  - 同构图:节点类型和边的类型只有一种的图。
  - 异构图:节点类型 + 边类型 > 2的图。

图的表示 – 无向图



1. 邻接矩阵

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2. 邻接列表

1: 2, 3, 5

2: 1, 4

3: 1, 4

4: 2, 3

5: 1

3. 边集

(1, 2),

(2, 1),

(1, 5),

(5, 1),

(2, 4),

(4, 2),

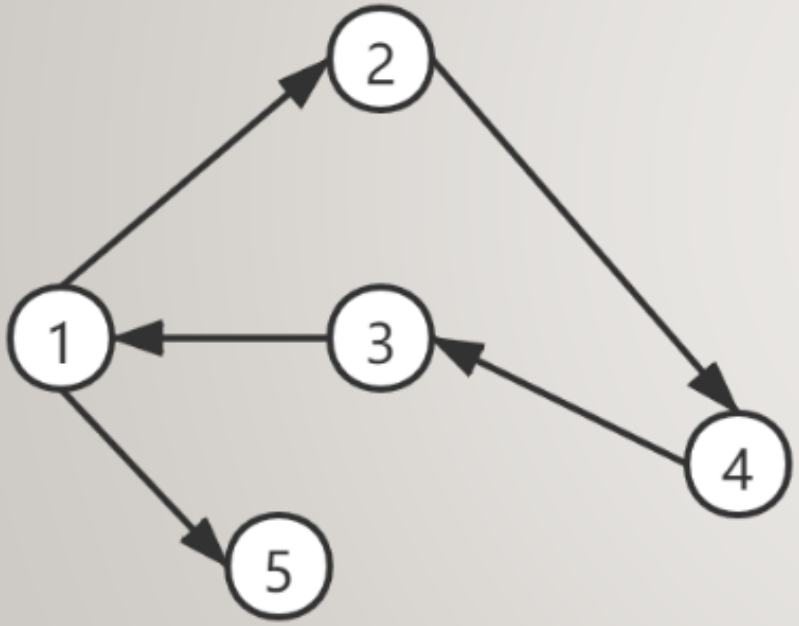
(4, 3),

(3, 4),

(3, 1),

(1, 3)

图的表示 – 有向图



1. 邻接矩阵

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2. 邻接列表

1: 2, 5

2: 4

3: 1

4: 3

3. 边集

(1, 2),

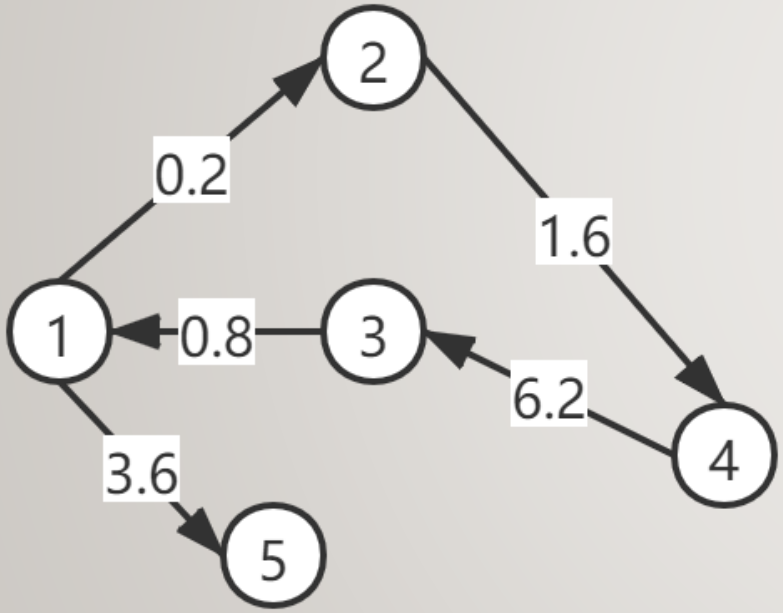
(1, 5),

(2, 4),

(4, 3),

(3, 1)

图的表示 – 有权图



1. 邻接矩阵

$$A = \begin{bmatrix} 0 & 0.2 & 0 & 0 & 3.6 \\ 0 & 0 & 0 & 1.6 & 0 \\ 0.8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2. 邻接列表

1: (2, 0.2) , (5, 3.6)

2: (4, 1.6)

3: (1, 0,8)

4: (3, 6.2)

3. 边集

(1, 0.2, 2),

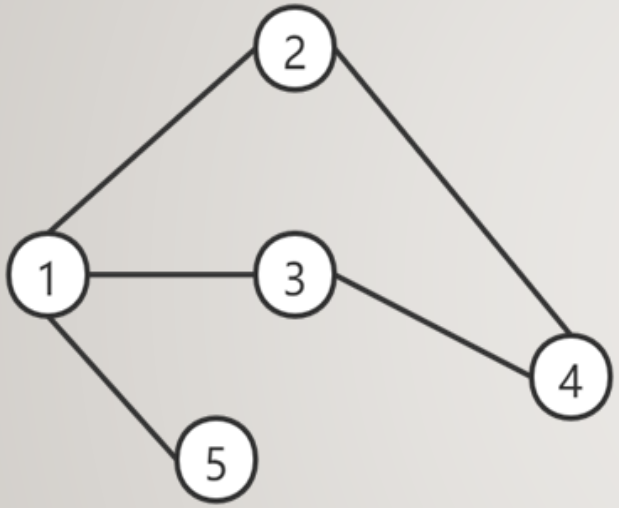
(1, 3.6, 5),

(2, 1.6, 4),

(4, 6.2, 3),

(3, 0.8, 1)

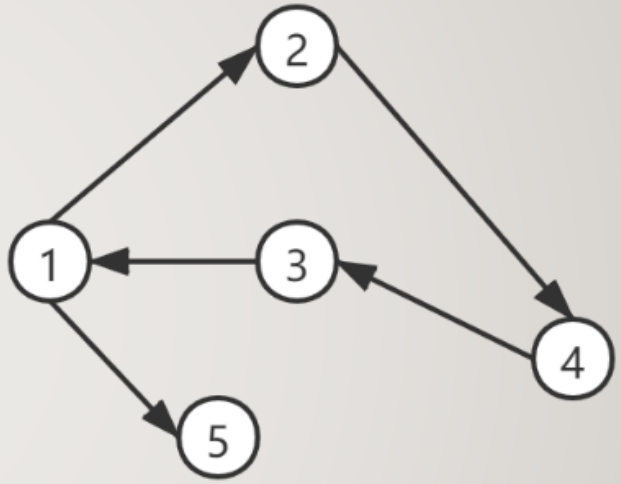
图论基础 - 邻居与度



- 无向图中节点1:

邻居 (neighbor) = 2, 3, 5

度 (degree) = 3



- 有向图中节点1:

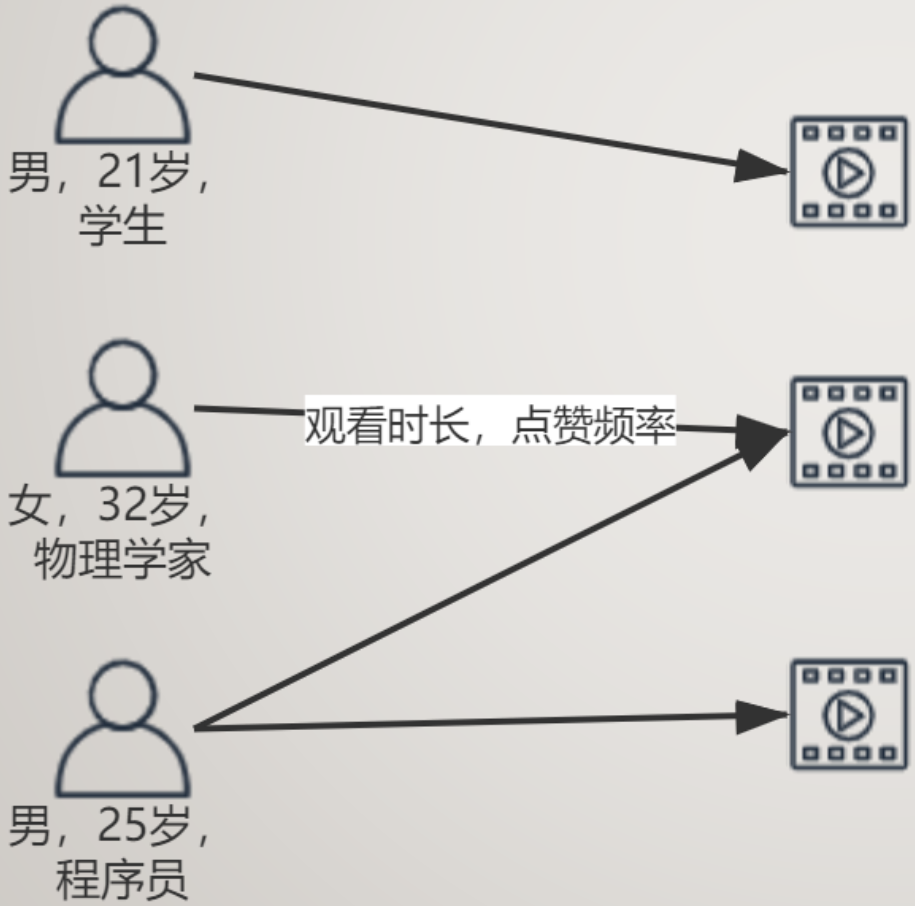
前继邻居 (predecessor) = 3

后继邻居 (successor) = 2, 5

入度 (indegree) = 1

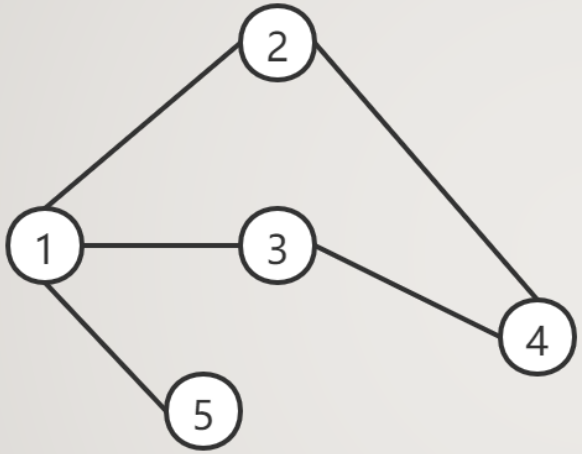
出度 (outdegree) = 2

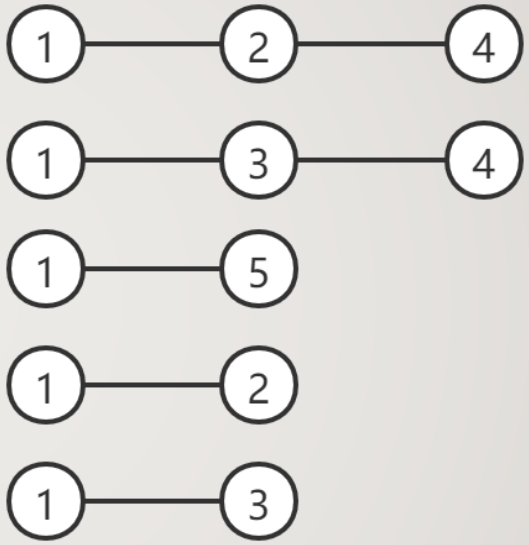
图论基础 – 结构特征、节点特征、边特征



- 结构特征：整个图即代表结构特征。
- 节点特征：如用户画像特征，或由嵌入操作得到的节点向量特征。
- 边特征：又称关系特征，如用户对物品的观看时长，点赞频率等。或由前途得到的边向量特征，边的权重也可视为特征。

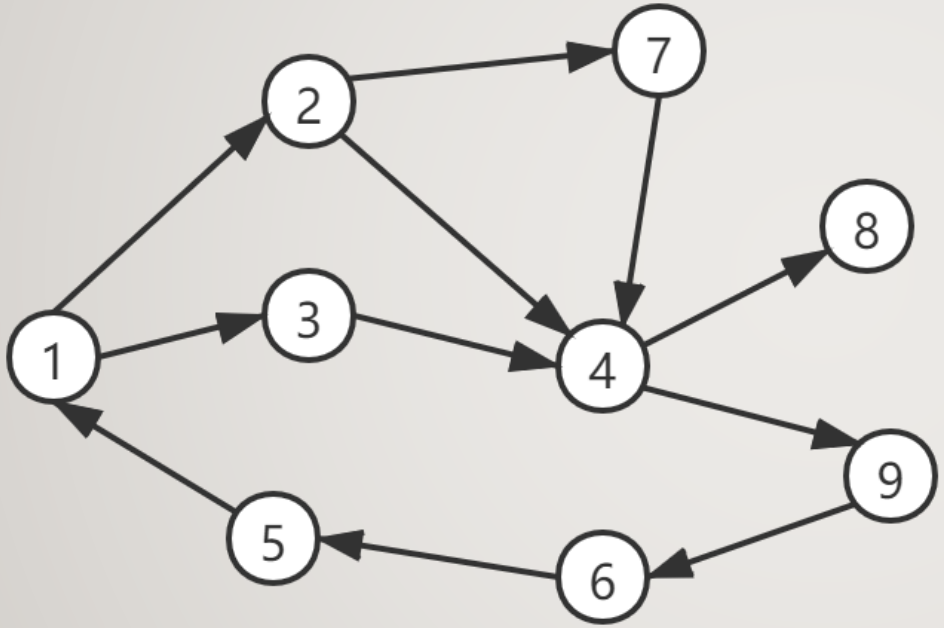
图论基础 – 路径





- 路径的定义：路径是从某一个节点到另一个节点之间经过的边与节点组成的子图，包含头尾节点。
- 路径的阶数：一条路径上的边数被称为路径的阶数，例如图中124或134属于二阶路径。12，13，15属于一阶路径。所以又可把节点2，3，5称为节点1的一阶邻居，节点4称为节点1的二阶邻居。

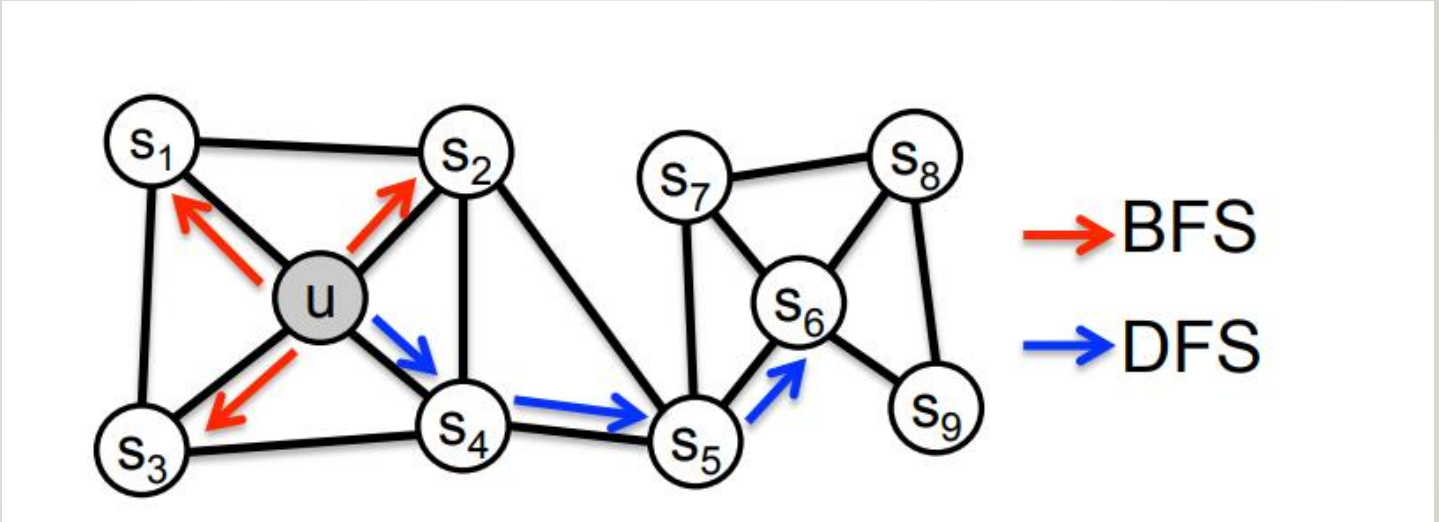
图游走算法 DEEPWALK




```
[1, 2, 4, 9, 6]
[1, 2, 4, 8]
[1, 3, 4, 9]
[2, 7, 4, 9, 6, 5]
[6, 5, 1, 2, 7]
[4, 8]
[4, 9, 6, 5, 1, 3]
[5, 1, 2, 4, 9, 6]
```

- 中心思想：在图中随机游走生成节点序列，之后用Word2Vec的方式得到节点embedding。

图游走算法 NODE2VEC



- 中心思想：与DeepWalk不同的是Node2Vec可通过调整方向的参数来控制模型更倾向宽度优先的游走(BFS), 还是深度优先的游走(DFS)。
- BFS (Breadth-first Sampling)，宽度优先游走，生成的序列往往是由起始节点周边的组成的网络结构。这就能让最终生成的embedding具备更多结构化的特征。
- DFS (Depth-first Sampling)，深度优先游走，更有可能游走到当前节点远方的节点。所以生成的序列会具备更纵深的远端信息。

图游走算法 NODE2VEC

- Node2Vec公式组:

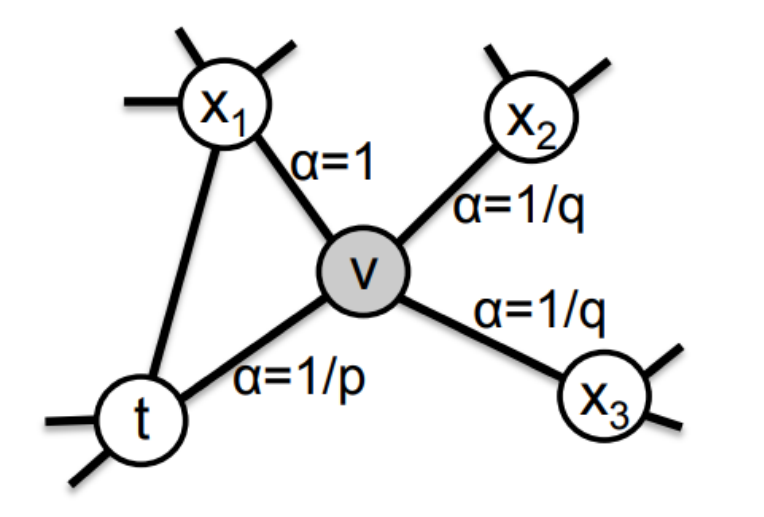
$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & if(v,x) \in E \\ 0 & otherwise \end{cases} \quad (1)$$

等号左边的表达是指从当前节点v走到下一节点x的概率。E表示当前节点v所有的后继邻居节点集。Z是归一化常量，π_{vx}是转换概率，例如在一个有向无权图的DeepWalk中，Z就是当前节点后继邻居的数量，π_{vx}则等于1。

$$\pi_{vx} = \alpha_{pq}(t,x) \cdot w_{vx} \quad (2)$$

w_{vx}是考虑有权图中边的权重，而$\alpha_{pq}(t,x)$ 在这里可被认为是元转移概率，计算方式请看下一页。

图游走算法 NODE2VEC



$$\alpha_{pq}(t,x)=\begin{cases}\frac{1}{p} \ if \ d_{tx}=0\\ 1 \ if \ d_{tx}=1\\ \frac{1}{q} \ if \ d_{tx}=2\end{cases} \quad (3)$$

- d_{tx}指t时刻也就是下一游走的候选节点x的节点类型，有0，1，2三个枚举类型。
- 如果 $d_{tx}=0$，则代表该候选节点是前一时刻游走时的起始节点，就是上图中的节点t，往这个方向游走就代表走回头路。而α 就等于1/p。由此可见超参数p就是控制游走以多大概率回头。
- 如果 $d_{tx}=1$， 则代表该候选节点x与前一时刻的起始节点t以及当前节点v是等距的，就是上图中的x1节点。此时α 等于1 。往这个方向游走就是BFS宽度优先游走。
- 如果 $d_{tx}=2$，则代表其他，α 等于1/q。往此方向游走就是DFS深度优先游走。所以q就是控制游走更偏向BFS还是DFS。
- 当q<1时，更倾向于DFS。当q>1时，更倾向于BFS。当q=1时，Node2Vec则退化为DeepWalk。

图卷积网络GCN

- GCN (Graph Convolutional Networks)，图卷积网络。
- 计算过程：

$$H^{l+1} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l}w^{l}\right) \tag{1}$$

其中H^{l}指第l层的输入特征，H^{l+1}自然就是指输出特征。w^{l}指线性变换矩阵。$\sigma(\cdot)$是非线性激活函数。

- $\tilde{A}$叫做“有自连的邻接矩阵”。定义如下：

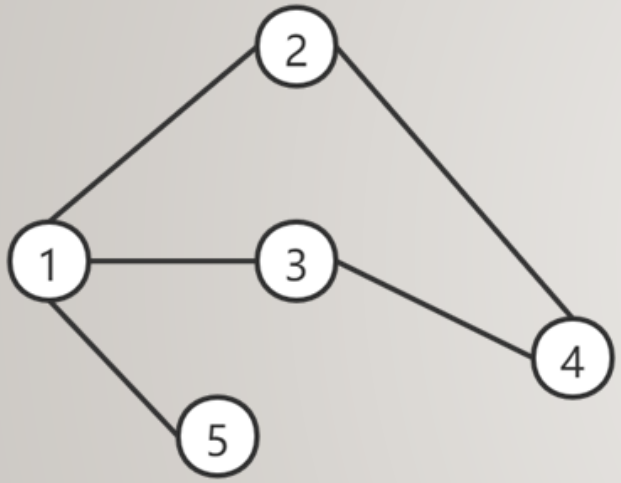
$$\tilde{A} = A + I \tag{2}$$

其中I是单位矩阵，A是邻接矩阵

- $\tilde{D}$是自连矩阵的度矩阵，定义如下:

$$\tilde{D}_{ii} = \sum_{j} \tilde{A}\, ij \tag{3}$$

图卷积网络GCN 计算过程示例



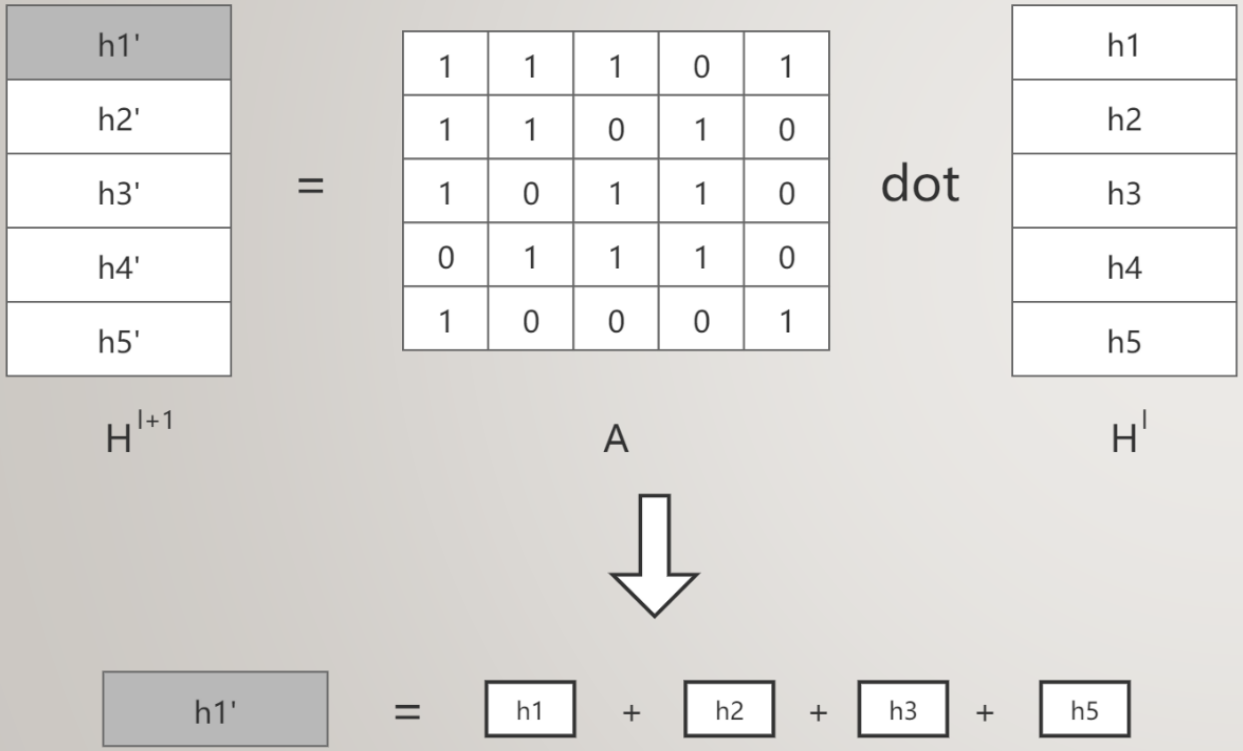
$$A=\begin{bmatrix}0&1&1&0&1\\1&0&0&1&0\\1&0&0&1&0\\0&1&1&0&0\\1&0&0&0&0\end{bmatrix}$$

$$\tilde{A}=\begin{bmatrix}1&1&1&0&1\\1&1&0&1&0\\1&0&1&1&0\\0&1&1&1&0\\1&0&0&0&1\end{bmatrix}$$

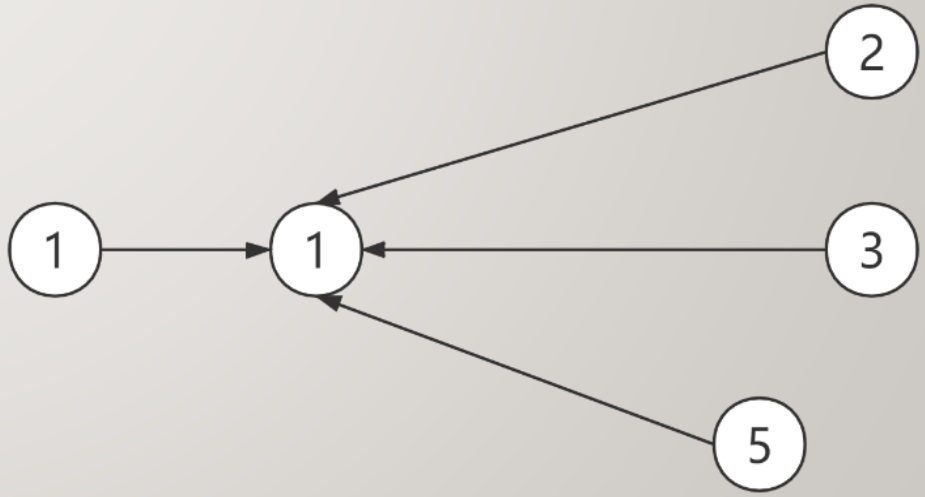
$$\tilde{D}=\begin{bmatrix}4&0&0&0&0\\0&3&0&0&0\\0&0&3&0&0\\0&0&0&3&0\\0&0&0&0&2\end{bmatrix}$$

$$\tilde{D}^{-\frac{1}{2}}=\begin{bmatrix}\frac{1}{\sqrt{4}}&0&0&0&0\\0&\frac{1}{\sqrt{3}}&0&0&0\\0&0&\frac{1}{\sqrt{3}}&0&0\\0&0&0&\frac{1}{\sqrt{3}}&0\\0&0&0&0&\frac{1}{\sqrt{2}}\end{bmatrix}$$

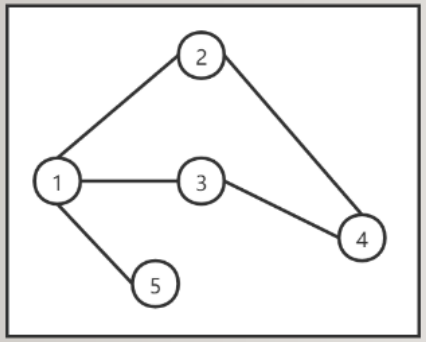
图卷积网络GCN 计算原理图解

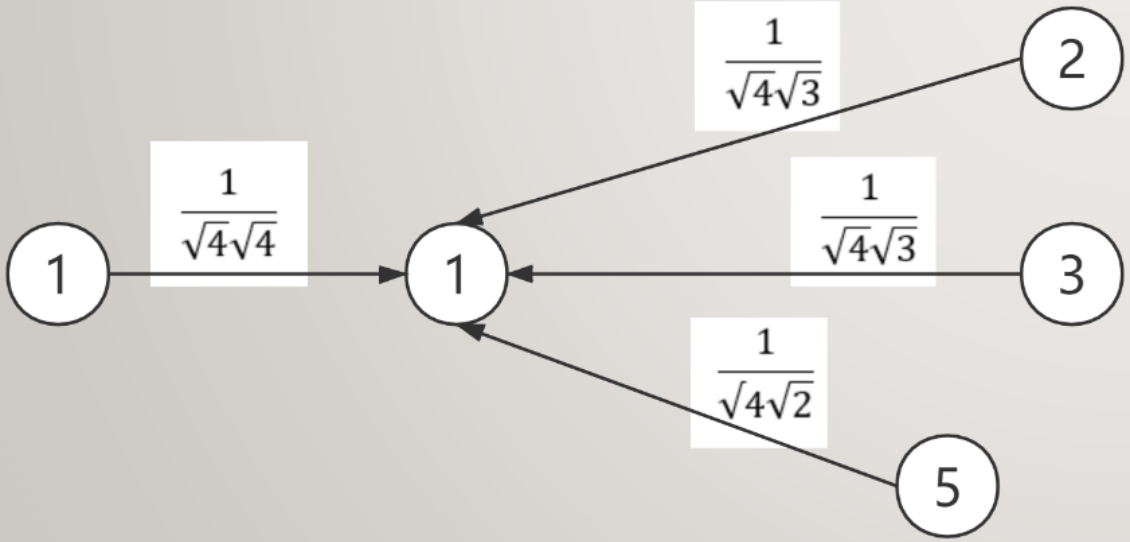


- 如果不考虑度矩阵，则$\tilde{A}H^{l}$的计算过程如左图所示。下一层第一个节点的向量表示就是当前层节点h1, h2, h3, h5这些节点向量表示的和。
- 这其实是一个消息传递的过程，如下图所示，节点1,2,3,5将自身的特征向量传递给节点1，然后在节点1的位置进行求和的消息聚合操作。



图卷积网络GCN 计算原理图解





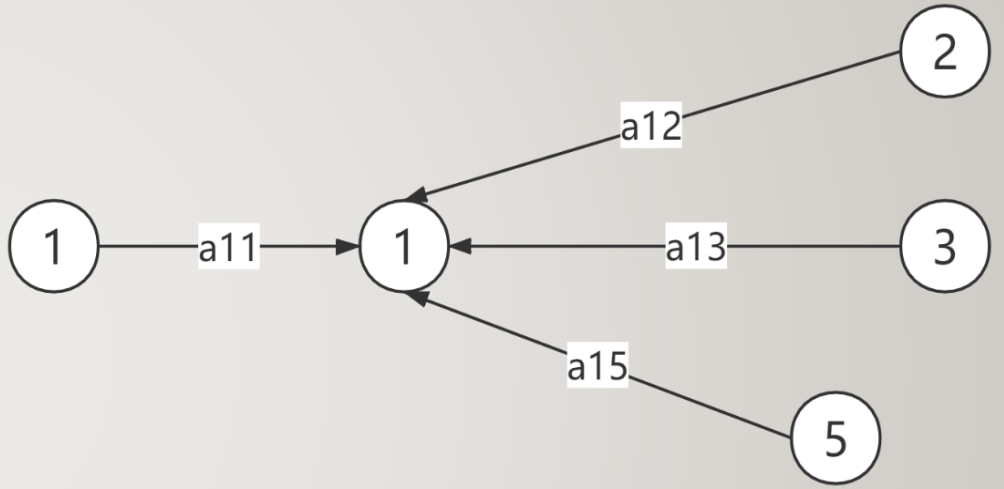
- 考虑度矩阵后 $\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{l}$ 这一计算的意义如左图所示。就是一个加权求和操作，度越大权重就越低。

GAT 图注意力网络

- GAT (Graph Attention Networks)，加入了注意力机制的图神经网络，其消息传递的权重是通过注意力机制得到。
- GAT计算过程:

$$a_{ij} = softmax_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in N_i} \exp(e_{ik})} \quad (1)$$

$$e_{ij} = LeakyReLU\left(a^T [Wh_i || Wh_j]\right) \quad (2)$$

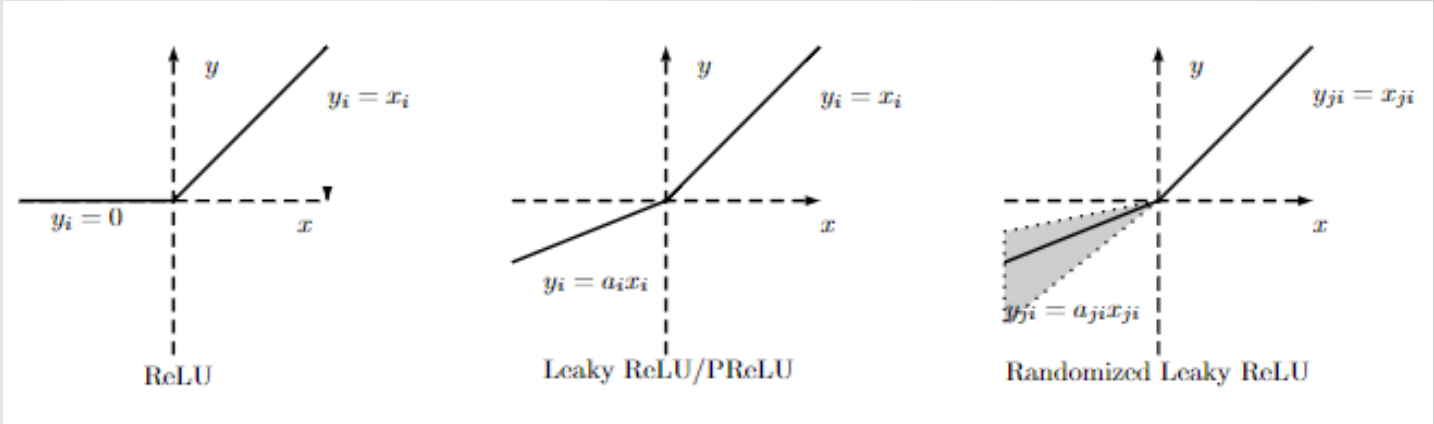


- h_i和h_j是当前输入层的节点i与节点j的特征表示，W是线性变换矩阵，形状是$W \in R^{F \times F'}$，其中F就是输入特征的维度。F'是输出特征的维度。
- ||是向量拼接操作，原本维度为F的h_i与h_j经过W线性变换后维度均变为F'，经过拼接后得到维度为$2F'$的向量。此时再点乘一个维度为$2F'$的单层矩阵α的转置，然后经LeakyReLU激活后得到1维的e_{ij}。
- 得到所有e_{ij}后，再进行softmax操作，得到注意力权重 a_{ij}。

基础知识——LEAKYRELU

$$ReLU(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$LeakyReLU(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$



- $ReLU$函数属于“非饱和激活函数”，由公式可见$ReLU$就是将所有负值都设为0。如果大多数的参数都为负值，那么显然$ReLU$的激活能力会大大折扣。
- LeakyReLU在负值部分赋予了一个负值斜率α。如此一来负值是会根据α的值变化，而不会都为0。
- LeakyReLU又衍生出了：
  - PReLU (parametric rectified linear 参数化线性修正)，负值斜率由训练数据决定的
  - 与RReLU (randomized rectified linear 随机线性修正) ，负值斜率是在一个范围内随机取值，且会在训练过程中随机变化。

GAT 图注意力网络

- 计算节点i的在当前GAT网络层的输出向量h_i'即可描述为:

(3)

其中$\sigma(\cdot)$代表任意激活函数，N_i代表节点i的一阶邻居集，W与注意力计算中的W是一样的。到这就是一个消息传递，并用加权求和的方式进行消息聚合的计算过程。

- 在GAT中，我们可以进行多次消息传递操作，然后将每次得到的向量拼接或者求平均。这称之为多头注意力(Multi-Head Attention)，请看如下公式：
- (GAT的论文中建议在GAT网络中间的隐藏层采取拼接操作，而最后一层采取平均操作。)
  - 拼接每一层单头消息传递得到的向量：

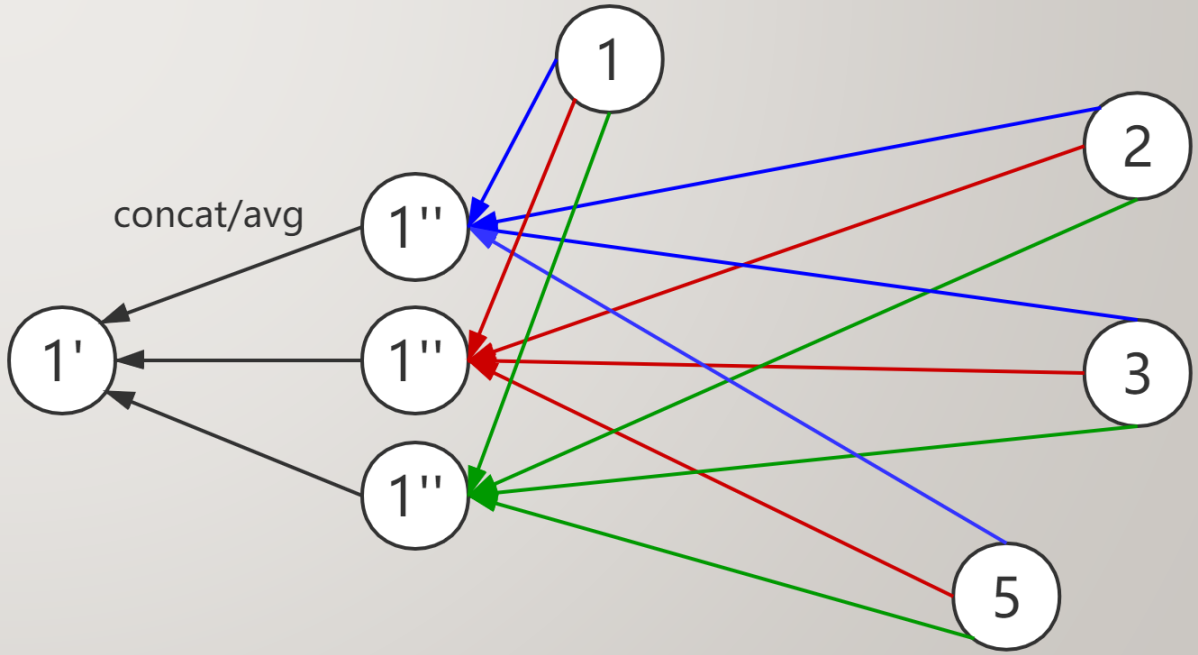
$$h_i' = \Big\Vert_{k=1}^{K} \sigma\Big(\sum_{j \in N_i} \alpha_{ij}^k W^k h_j\Big)$$

(4)

  - 平均每一层单头消息传递得到的向量：

$$h_i' = \sigma\Big(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in N_i} \alpha_{ij}^k W^k h_j\Big)$$

(5)



消息传递

- 消息传递Message Passing表示在一个图网络中节点间传导信息的通用操作.
- 消息传递的范式：

$$h'_v = \varphi(v) = f(h_v, g(h_u|u \in N_v) \qquad (1)$$

h'_v是当前节点v的在当前层的输出特征，h_v是输入特征。

$\varphi(\cdot)$即表达对某个节点进行消息传递动作。

N_v是节点v的邻居集。

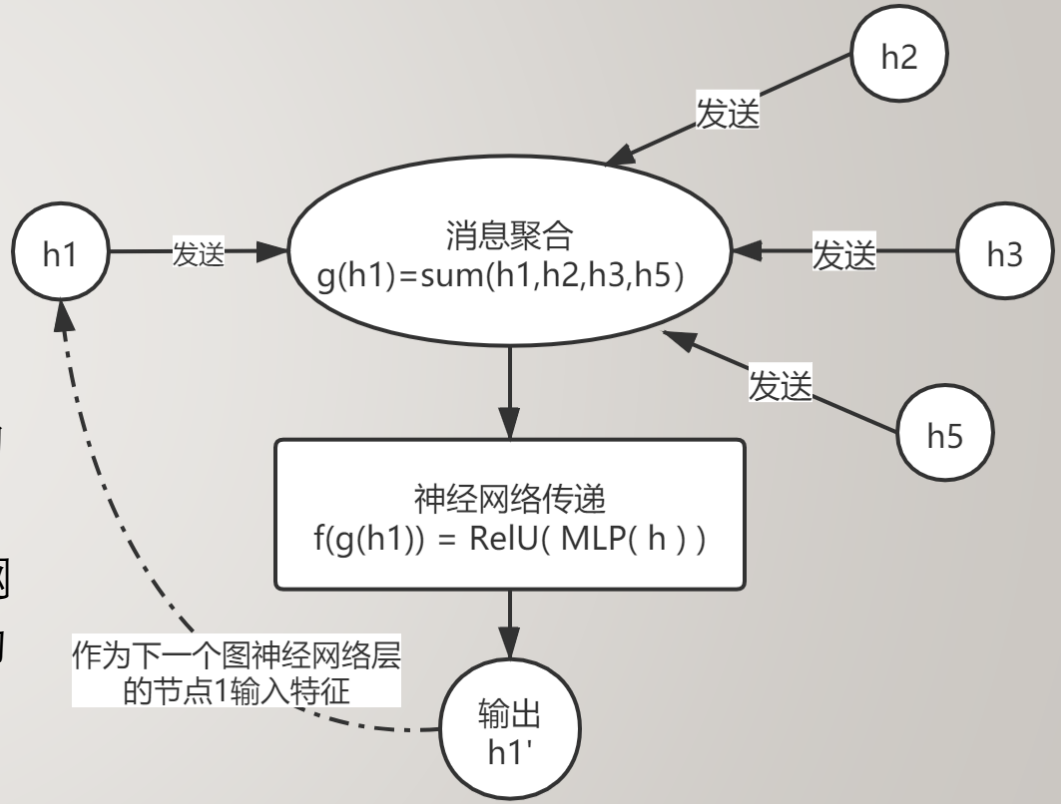
$h_u|u \in N_v$代表遍历节点v的邻居集，相当于邻居节点消息发送的动作。

$g(\cdot)$是一个消息聚合的函数，例如Sum, Avg, Max。$g(\cdot)$在GCN的网络层中，就是一个基于度的加权求和，而在GAT中就是基于注意力的加权求和。

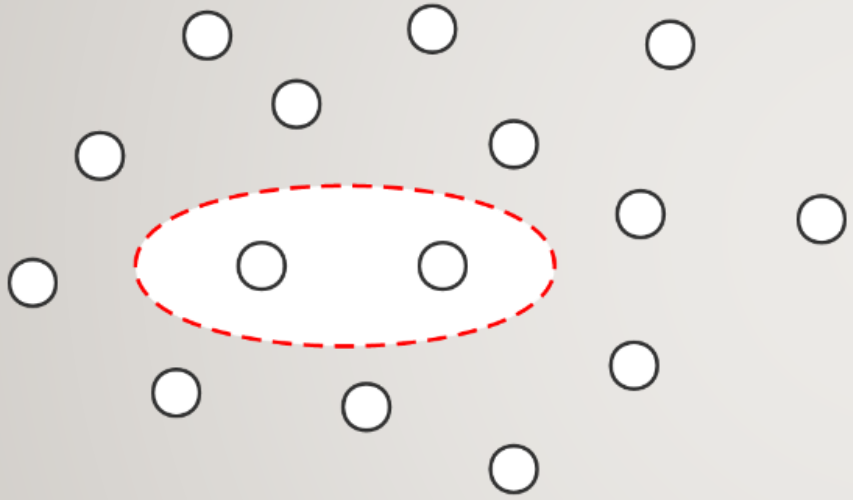
$f(\cdot)$表示对消息聚合后的节点特征进行神经网络的通常操作。

- 一个GNN层的计算范式可表达为：

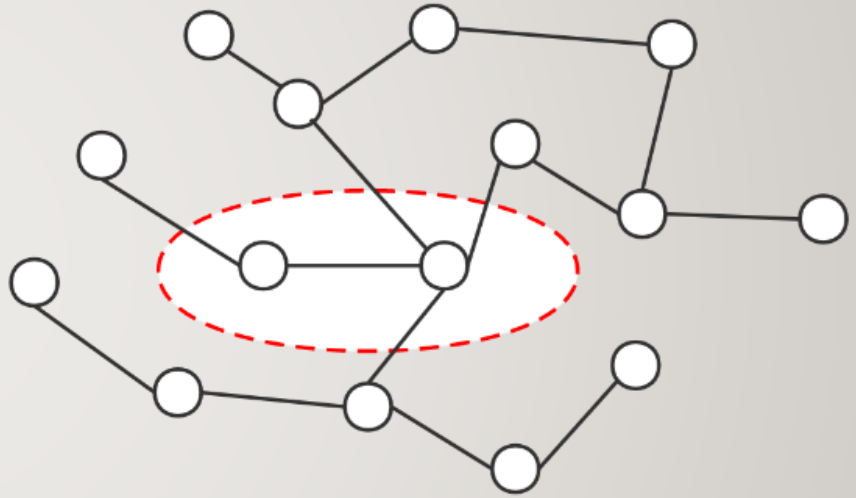
$$H^{l+1} = H^l : \{\varphi(v)|v \in V\} \qquad (2)$$



图采样



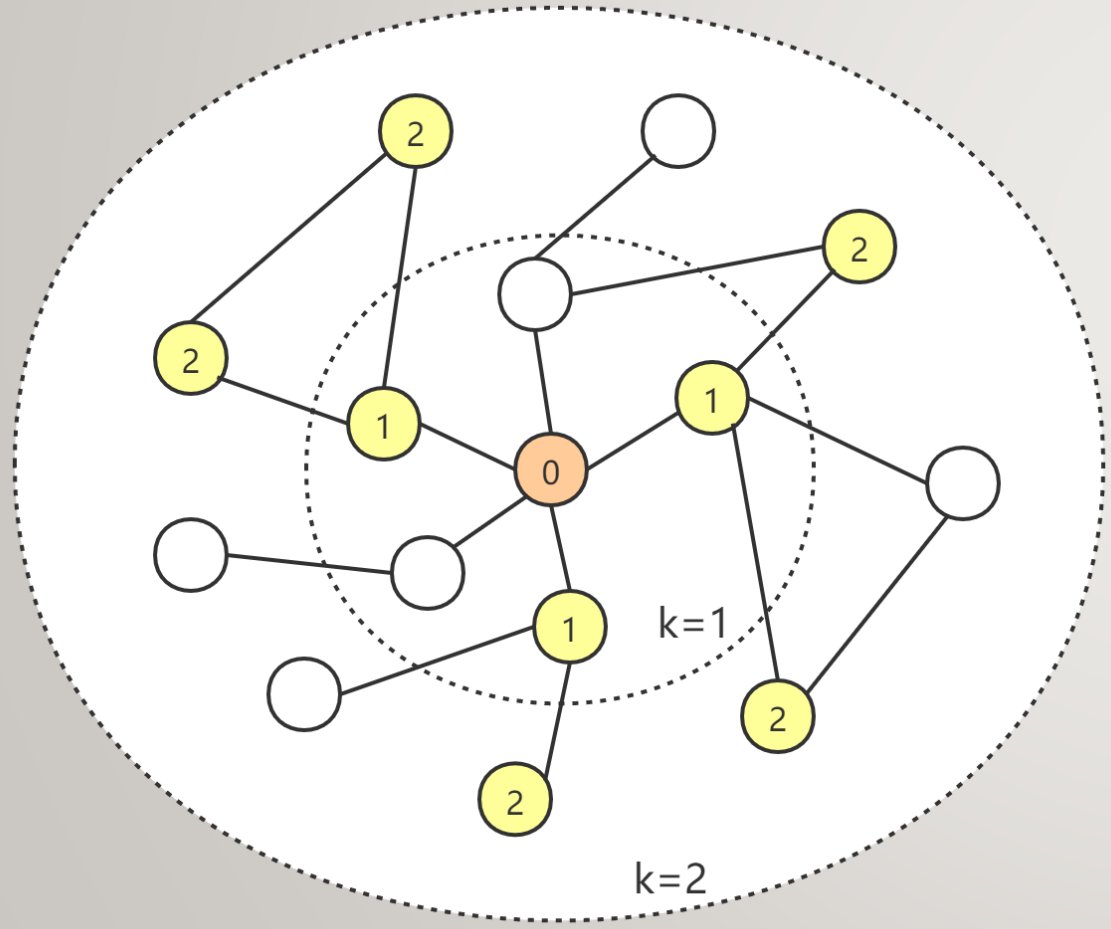
普通深度学习的训练样本



图神经网络的训练样本

- 如果图数据量过大，不可以直接仿照传统深度学习的一样的小批量训练方式。因为普通深度学习中训练样本之间并无依赖，但是图结构的数据中，节点与节点之间有依赖关系。如果随意采样的话，则破坏了样本之间的关系信息。所以需要专门的图采样方法。

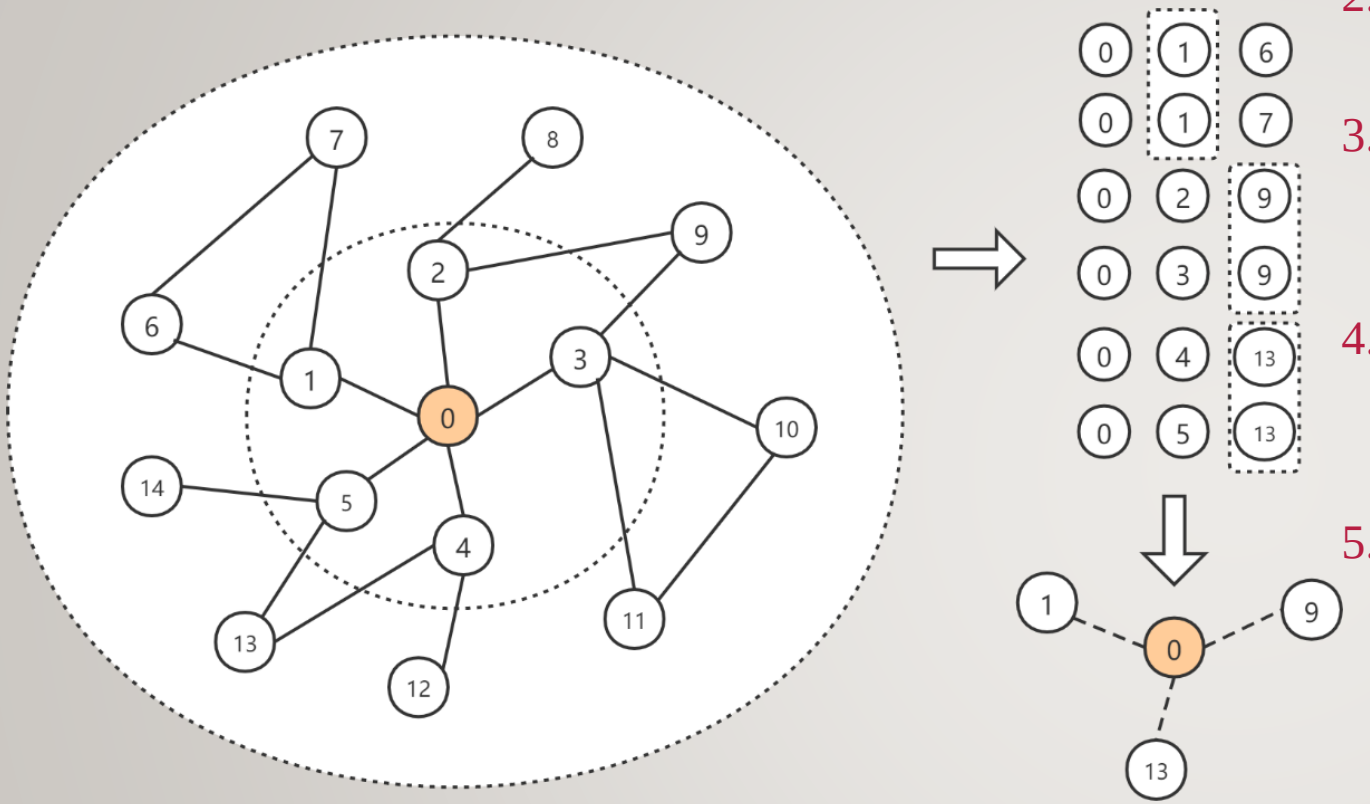
图采样算法GRAPHSAGE



- 中心思想：小批量采样原有大图的子图。

1. 步骤1: 随机选取一个或若干个节点作为0号节点。
2. 步骤2: 在0号节点的一阶邻居中随机选取若干个节点作为1号节点。
3. 步骤3: 在刚刚1号节点的一阶邻居中，不回头的随机选取若干个节点作为2号节点，不回头指的是不在回头取0号节点。该步骤亦可认为是随机选取0号节点通过1号节点连接这的二阶邻居。
4. 步骤4: 以此类推，图中的k是GraphSAGE的超参，可认为是0号节点的邻居阶层数，若k设定为5。则代表我们总共可以取到0号节点的第5阶邻居。
5. 步骤5: 将采样获得的所有节点保留边的信息组成子图作为一次小批量样本输入到图神经网络中进行下游任务；或者输出经过自外而内的消息传递聚合了子图所有信息的0号节点特征向量。

图采样算法PINSAGE



- 中心思想：采样通过随机游走经过的高频节点生成的子图。
- PinSAGE的优势在于可以很快速的收集到远端节点，且生成的子图已经经过一次频率删选所获得的样本表达能力更强也更具泛化能力。

- 采样过程：

1. 步骤1: 随机选取一个或若干个节点作为0号节点。
2. 步骤2: 以0号节点作为起始节点开始随机游走生成序列，游走方式可以采取DeepWalk或者Node2Vec。
3. 步骤3: 统计随机游走中高频出现的节点作为0号节点的邻居生成一个新的子图。出现的频率可作为超参设置。
4. 步骤4: 将新子图中的边界节点(如在上图中的节点1，9，13)作为新的起始节点，重复步骤2开始随机游走。
5. 步骤5: 统计新一轮随机游走的高频节点，作为新节点在原来子图中接上。注意每个新高频节点仅接在它们原有的起始节点中(如节点1作为起始节点随机游走生成节点序列中的高频节点仅作为节点1的邻居接在新子图中)。
6. 步骤6: 重复上述过程k次，k为超参。将生成的新子图作为一次小批量样本输入到图神经网络中进行下游任务。或者输出经过自外而内的消息传递聚合了子图所有信息的0号节点特征向量。

图处理PYTHON API推荐

1. Networks。最老牌的图处理库，早在2002年就发布了第一版。没有图神经网络的算法，但其地位就像是图算法领域的numpy。官方文档：NetworkX — NetworkX documentation

2. DGL (Deep Graph Library)，亚马逊的图神经网络框架，第一版发布于2018年。基于深度学习框架的图神经网络库，实现了主流的图神经网络算法。兼容MxNet, PyTorch, Tensorflow。是目前最主流的处理图的python API。官方文档： Overview of DGL — DGL 0.5.3 documentation

3. PGL (Paddle Graph Learning) 。第一版发布于2020 年，百度的框架，号称速度是DGL的13倍。缺点是只兼容百度自家的飞浆 (PaddlePaddle) 深度学习框架。官方文档： Paddle Graph Learning (PGL) — pgl 2.1 documentation

结束