1 引入：图机器学习

1.1 先修知识点

1. 机器学习
2. 算法和图论
3. 概率论与数理统计

1.2 为什么是图

图机器学习中的常用工具：NetworkX, PyTorch Geometric, DeepSNAP, GraphGym, SNAP.PY

选择图的原因：图是用于描述并分析有关联/互动的实体的一种普适语言。它不将实体视为一系列孤立的点，而认为其互相之间有关系。它是一种很好的描述领域知识的方式。

网络与图的分类：

1. **networks / natural graphs**：自然表示为图
    - **社交网络**：社会是七十亿个体的网络。
    - **交流与交易**：电子设备、电话，金融交易。
    - **生物制药**：基因或蛋白质之间互动从而调节生理活动的过程。
    - **神经连接**：我们的想法隐藏于神经元的连接之中。
2. **graphs**：作为一种表示方法
    - 信息/知识被组织或连接。
    - 软件可以被图的方式表达出来。
    - 相似网络，数据点之间的连接相似。
    - 关系结构，分子、场景图、3D形状、基于粒子的物理模拟。

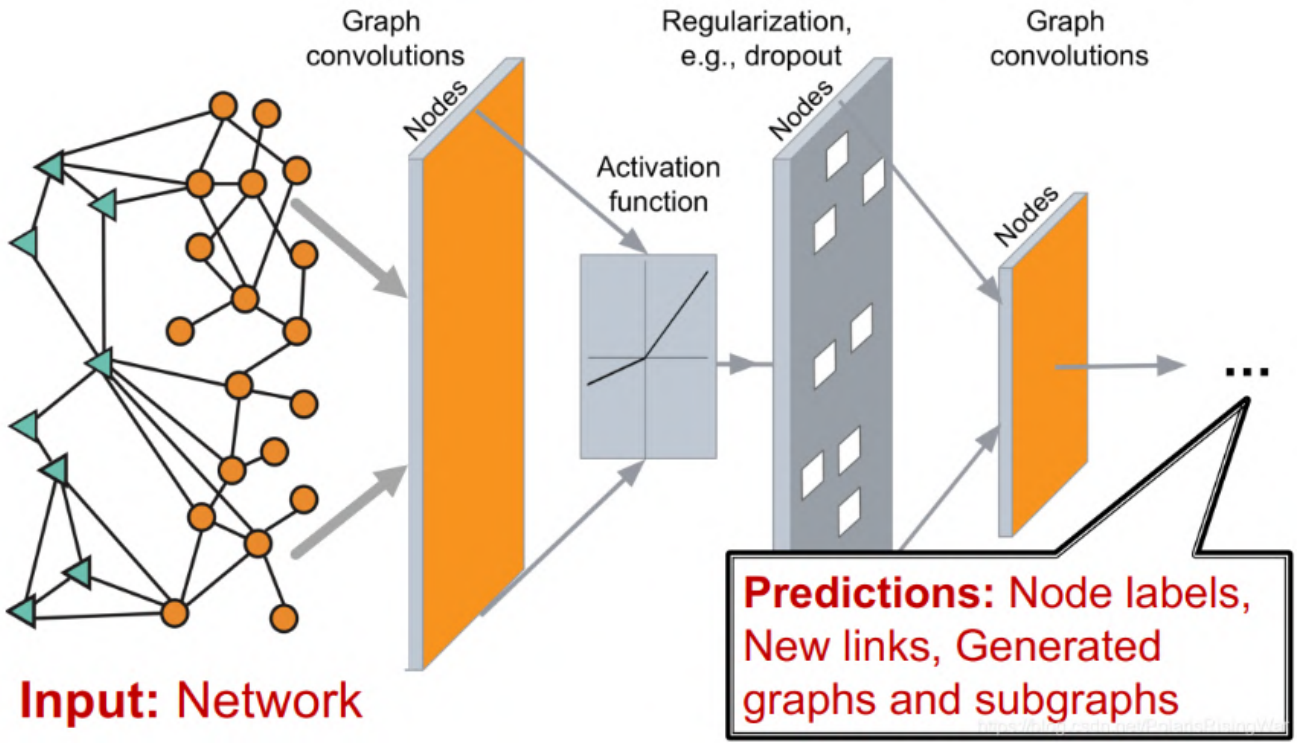
复杂领域会有丰富的关系结构，可以被表示为**关系图**relational graph，通过显式地建模关系，可以获得更好的表现。

但是现代深度学习工具常用于建模简单的序列sequence（如文本、语音等具有线性结构的数据）和grid（图片具有平移不变性，可以被表示为fixed size grids或fixed size standards）。

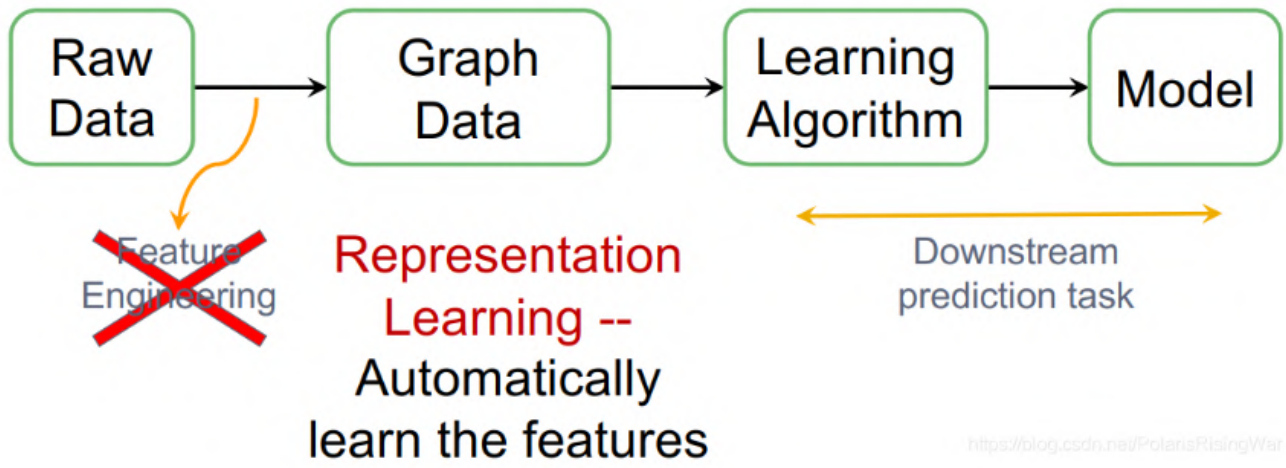
这些传统工具很难用于图的建模，其难点在于网络的复杂：

- 任意的大小和复杂的拓扑结构。
- 没有基准点，没有节点固定的顺序。没有那种上下左右的方向。
- 经常出现动态的图，而且会有多模态的特征。

本课程中讲述如何将神经网络模型适用范围拓展到图上：

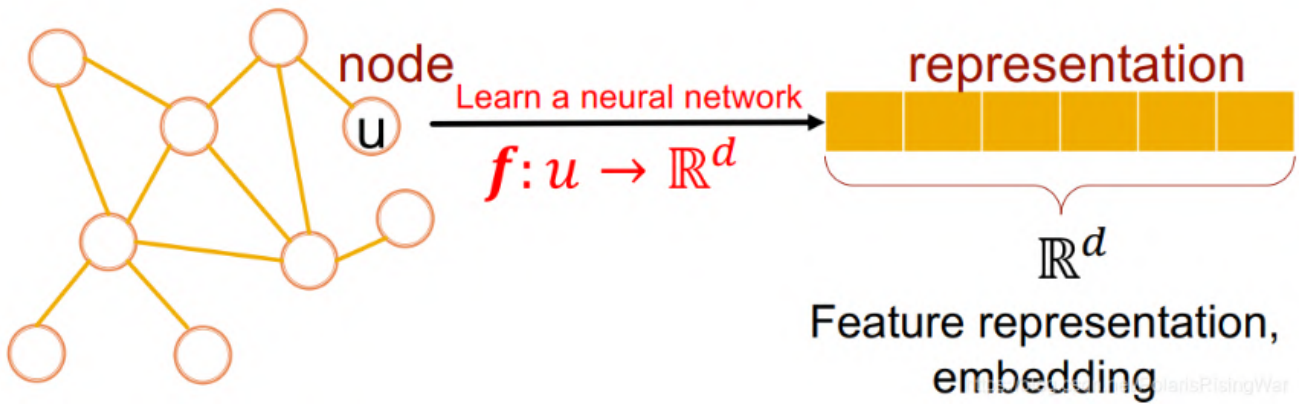


有监督机器学习全流程图:



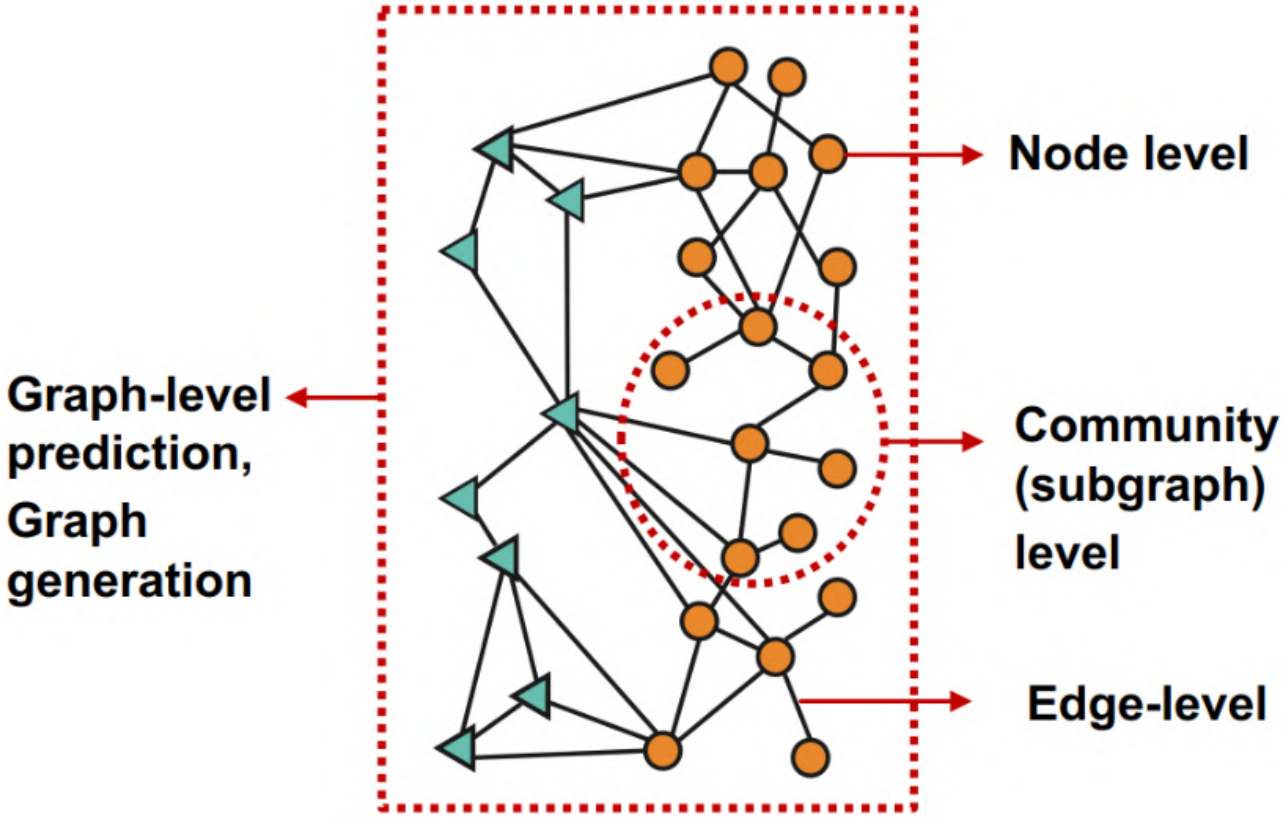
在传统机器学习流程中，我们需要对原始数据进行**特征工程**（feature engineering）（比如手动提取特征等），但是现在我们使用**表示学习**（representation learning）的方式来自动学习到数据的特征，直接应用于下流预测任务。

图的表示学习：大致来说就是将原始的节点（或链接、或图）表示为向量（嵌入embedding），图中相似的节点会被embed得靠近（指同一实体，在节点空间上相似，在向量空间上就也应当相似）



1.3 图机器学习的应用

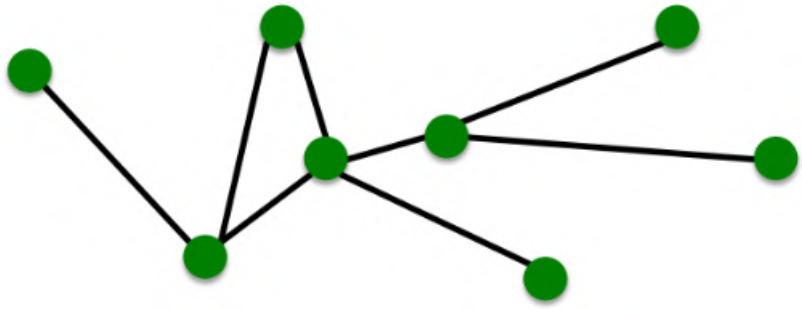
图机器学习任务分成四类：



- **节点级别（node level）**：预测结点属性，对结点进行聚类。例如，对联机用户/项目进行分类。
- **边级别（edge level）**：预测两个结点之间是否存在连接。例如，知识图谱构建。
- **社区 / 子图级别（community/subgraph level）**：对不同图进行分类。例如，分子属性预测。
- **图级别，包括预测任务（graph-level prediction）和图生成任务（graph generation）**：例如，药物发现、物理模拟。

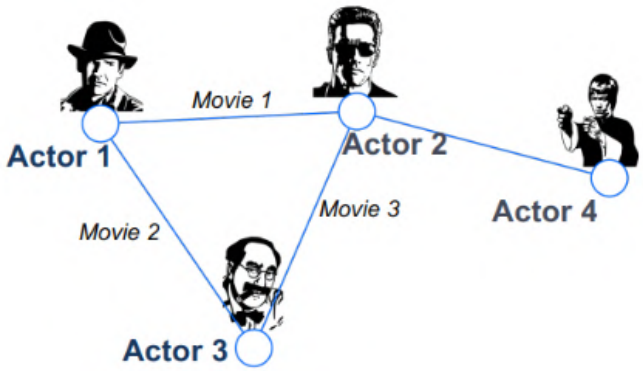
1.4 表示图的选择

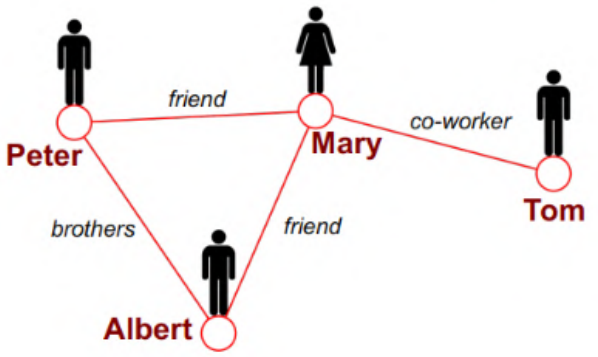
图的组成成分：

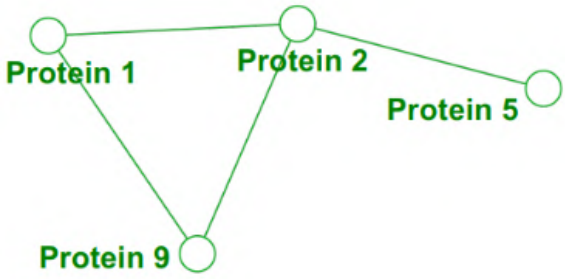


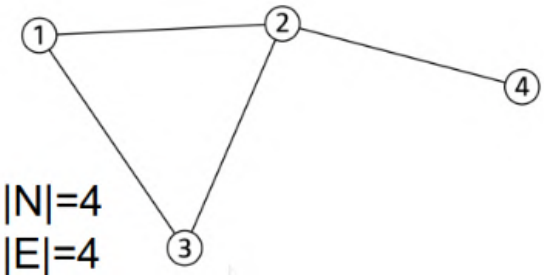
- **节点**（N/V）
- **连接/边**（E）
- **网络/图**（G）

图是一种解决关系问题时的通用语言，各种情况下的统一数学表示。将问题抽象成图，可以用同一种机器学习算法解决所有问题。









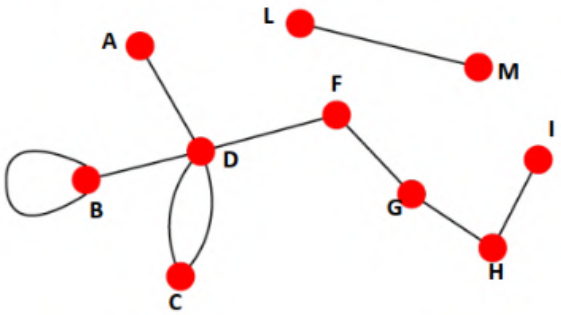
建图时需要考虑以什么作为节点，以什么作为边。对某一领域或问题选择合适的网络表示方法会决定我们能不能成功使用网络：

- 有些情况下只有唯一的明确做法
- 有些情况下可以选择很多种做法
- 设置连接的方式将决定研究问题的本质

有向图与无向图：

Undirected

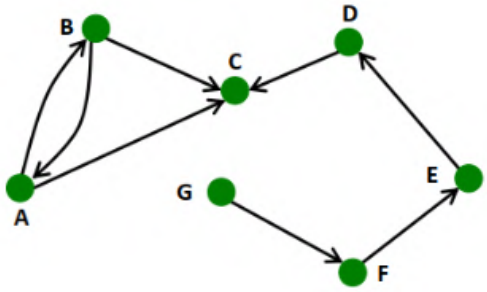
- **Links:** undirected (symmetrical, reciprocal)



- **Examples:**
  - Collaborations
  - Friendship on Facebook

Directed

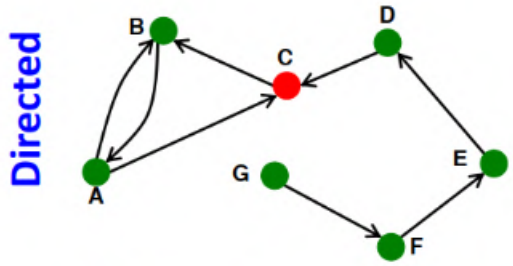
- **Links:** directed (arcs)



- **Examples:**
  - Phone calls
  - Following on Twitter

度（degree）：与结点相连边的个数。

- **无向图**：$Avg.\,degree = \bar{k} =< K >= \frac{1}{N}\sum_{i=1}^{N} k_i = \frac{2E}{N}$
- **有向图**：分为入度（in-degree）与出度（out-degree），度是两者之和。



Source: Node with $k^{in} = 0$
Sink: Node with $k^{out} = 0$

In directed networks we define an **in-degree** and **out-degree.**
The (total) degree of a node is the sum of in- and out-degrees.

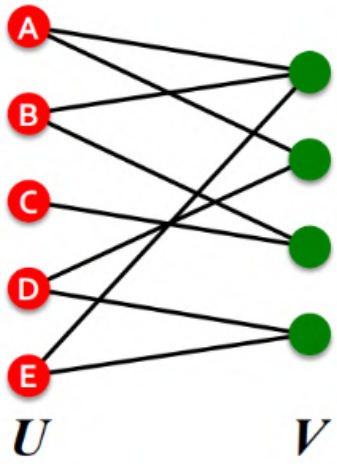
$$k_C^{in} = 2 \qquad k_C^{out} = 1 \qquad k_C = 3$$

$$\bar{k} = \frac{E}{N}$$

$$\overline{k^{in}} = \overline{k^{out}}$$

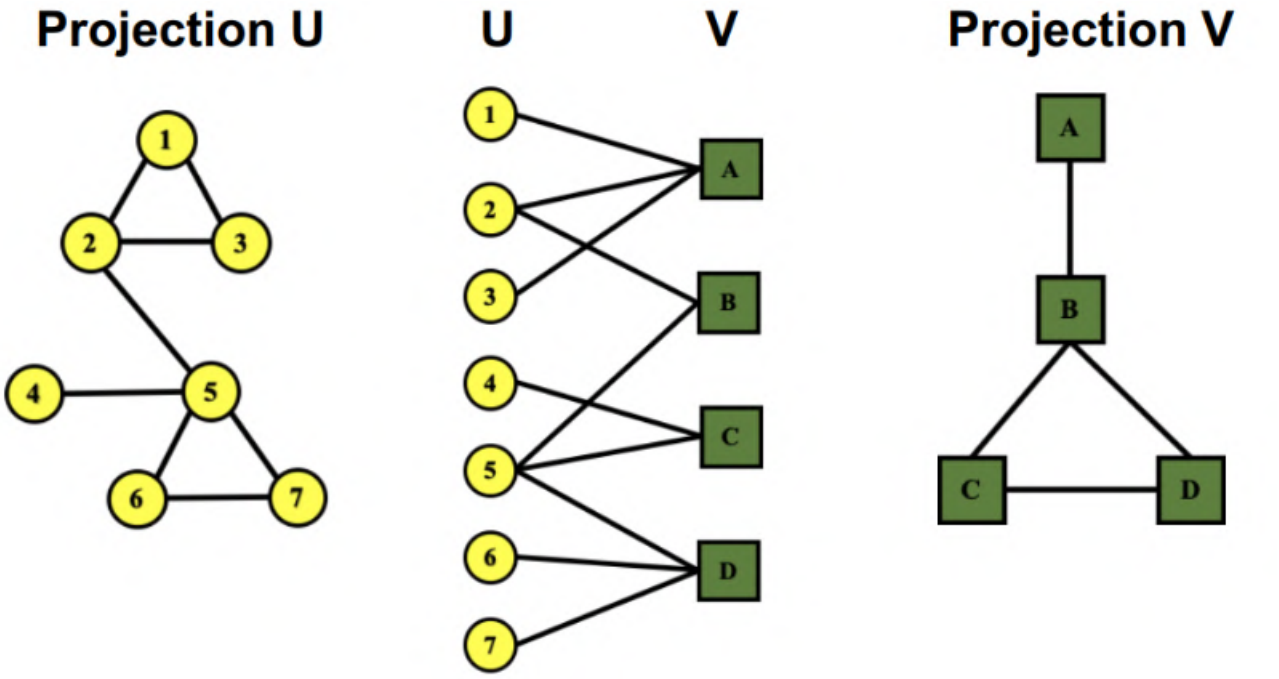
二部图（Bipartite Graph）：

- **Bipartite graph** is a graph whose nodes can be divided into two disjoint sets ***U*** and ***V*** such that every link connects a node in ***U*** to one in ***V***; that is, ***U*** and ***V*** are **independent sets**



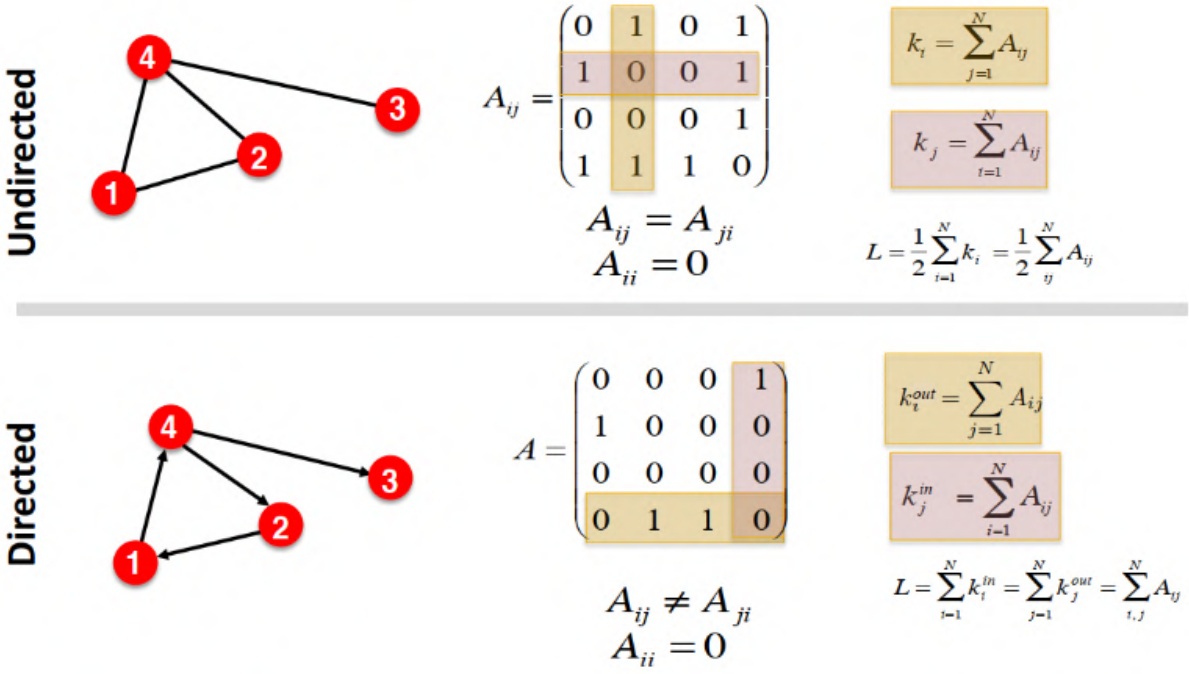
- **Examples:**
  - Authors-to-Papers (they authored)
  - Actors-to-Movies (they appeared in)
  - Users-to-Movies (they rated)
  - Recipes-to-Ingredients (they contain)
- **"Folded" networks:**
  - Author collaboration networks
  - Movie co-rating networks

折叠/投影二部分图（Folded/Projected Bipartite Graphs）：就是将一个bipartite图的两个节点子集分别投影，projection图上两个节点之间有连接，这两个节点在folded/projected bipartite graphs上至少有一个共同邻居。



表示图（Representing Graphs）：

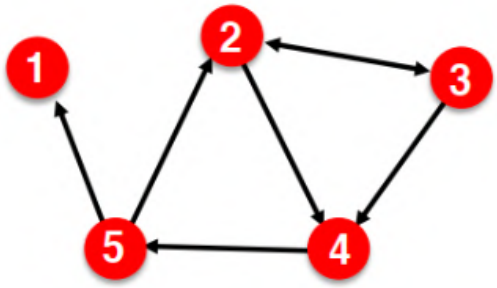
- **邻接矩阵（Adjacency Matrix）**：每一行/列代表一个节点，如果节点之间有边就是1，没有就是0。



无向图的邻接矩阵天然对称。

网络的邻接矩阵往往是稀疏矩阵，矩阵的密度为 $\frac{E}{N^2}$ 。

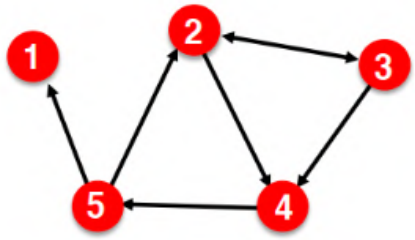
- **边表（Edge List）**：
  - (2, 3)
  - (2, 4)
  - (3, 2)
  - (3, 4)
  - (4, 5)
  - (5, 2)
  - (5, 1)



这种方式常用于深度学习框架中，因为可以将图直接表示成一个二维矩阵。这种表示方法的问题在于很难进行图的操作和分析，就算只是计算图中点的度数都会很难。

- **邻接表（Adjacency List）**：

- Easier to work with if network is
  - Large
  - Sparse
- Allows us to quickly retrieve all neighbors of a given node
  - 1:
  - 2: 3, 4
  - 3: 2, 4
  - 4: 5
  - 5: 1, 2



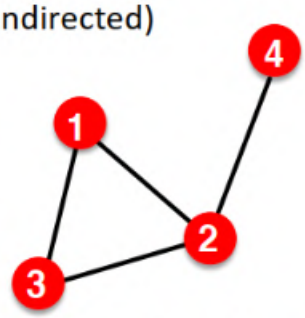
对图的分析和操作更方便。

节点和边的属性，可选项：

1. Weight (e.g., frequency of communication)
2. Ranking (best friend, second best friend…)
3. Type (friend, relative, co-worker)
4. Sign: Friend vs. Foe, Trust vs. Distrust
5. Properties depending on the structure of the rest of the graph: Number of common friends

有权/无权：

Unweighted
(undirected)



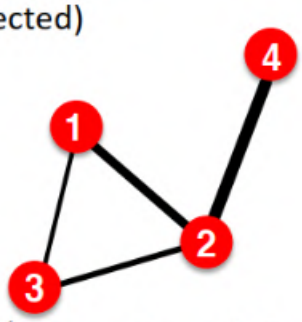
$$A_{ij}=\begin{pmatrix}0&1&1&0\\1&0&1&1\\1&1&0&0\\0&1&0&0\end{pmatrix}$$

$$A_{ii}=0 \qquad A_{ij}=A_{ji}$$

$$E=\frac{1}{2}\sum_{i,j-1}^{N}A_{ij} \qquad \bar{k}=\frac{2E}{N}$$

Examples: Friendship, Hyperlink

Weighted
(undirected)



$$A_{ij}=\begin{pmatrix}0&2&0.5&0\\2&0&1&4\\0.5&1&0&0\\0&4&0&0\end{pmatrix}$$

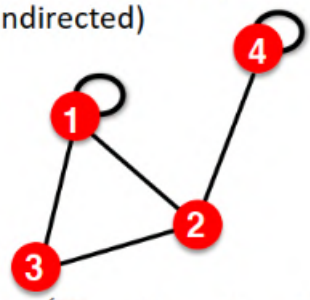
$$A_{ii}=0 \qquad A_{ij}=A_{ji}$$

$$E=\frac{1}{2}\sum_{i,j=1}^{N}nonzero(A_{ij}) \qquad \bar{k}=\frac{2E}{N}$$

Examples: Collaboration, Internet, Roads

自环/多重图:

Self-edges (self-loops)
(undirected)



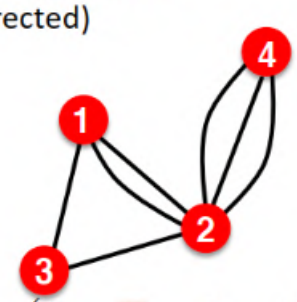
$$A_{ij}=\begin{pmatrix}1&1&1&0\\1&0&1&1\\1&1&0&0\\0&1&0&1\end{pmatrix}$$

$$A_{ii}\neq 0 \qquad A_{ij}=A_{ji}$$

$$E=\frac{1}{2}\sum_{i,j=1,i\neq j}^{N}A_{ij}+\sum_{i=1}^{N}A_{ii}$$

Examples: Proteins, Hyperlinks

Multigraph
(undirected)



$$A_{ij}=\begin{pmatrix}0&2&1&0\\2&0&1&3\\1&1&0&0\\0&3&0&0\end{pmatrix}$$

$$A_{ii}=0 \qquad A_{ij}=A_{ji}$$

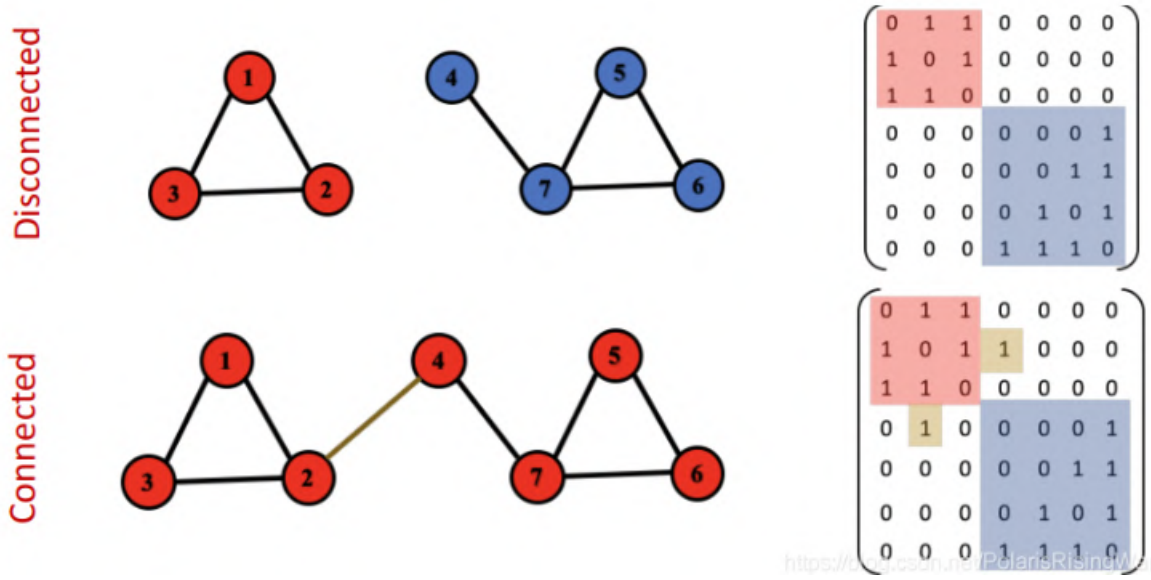
$$E=\frac{1}{2}\sum_{i,j=1}^{N}nonzero(A_{ij}) \qquad \bar{k}=\frac{2E}{N}$$

Examples: Communication, Collaboration

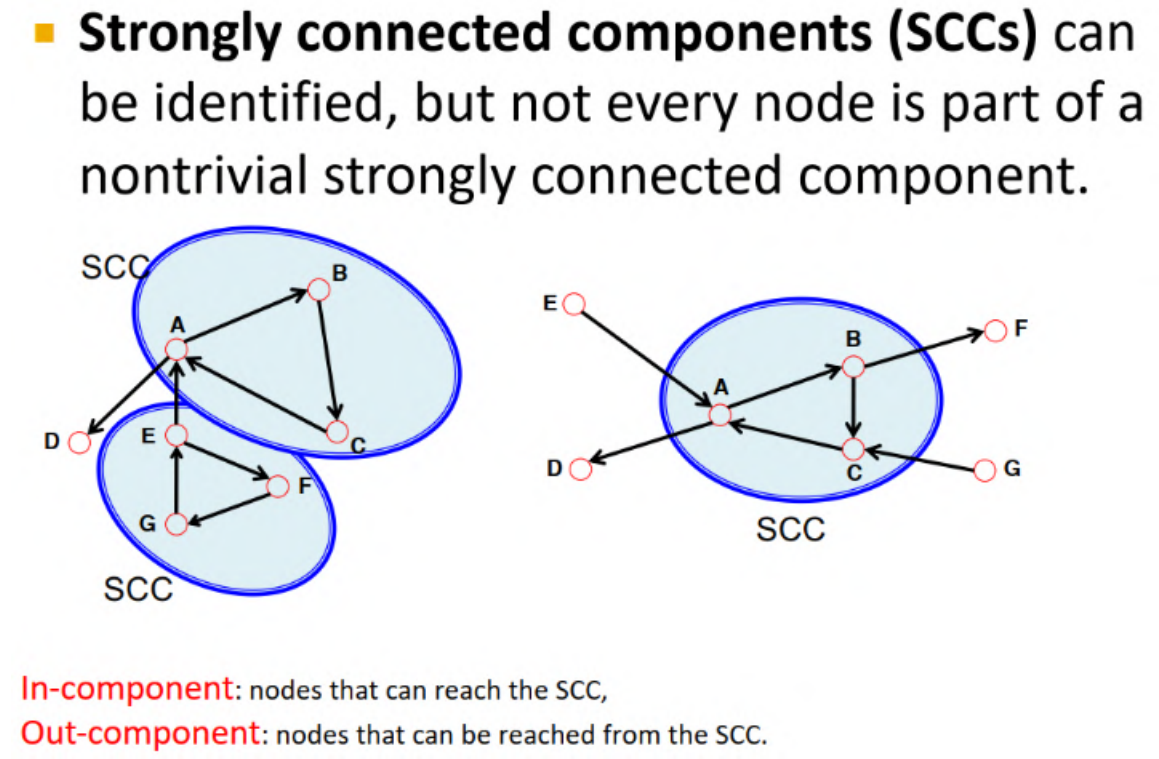
这个multigraph有时也可被视作是weighted graph，就是说将多边的地方视作一条边的权重（在邻接矩阵上可看出效果是一样的）。但有时也可能就是想要分别处理每一条边，这些边上可能有不同的property和attribute。

连通性（Connectivity）：

- **无向图的连通性**：
    1. 连通：任意两个节点都有路径相通。
    2. 不连通：由2至多个连通分量（connected components）构成。



- **有向图的连通性**：
    1. **强连通有向图**：具有从每个节点到每个其他节点的路径，反之亦然（例如，A-B路径和B-A路径）
    2. **弱连通有向图**：如果忽视边的方向则是连通的。
    3. **强连通分量**：

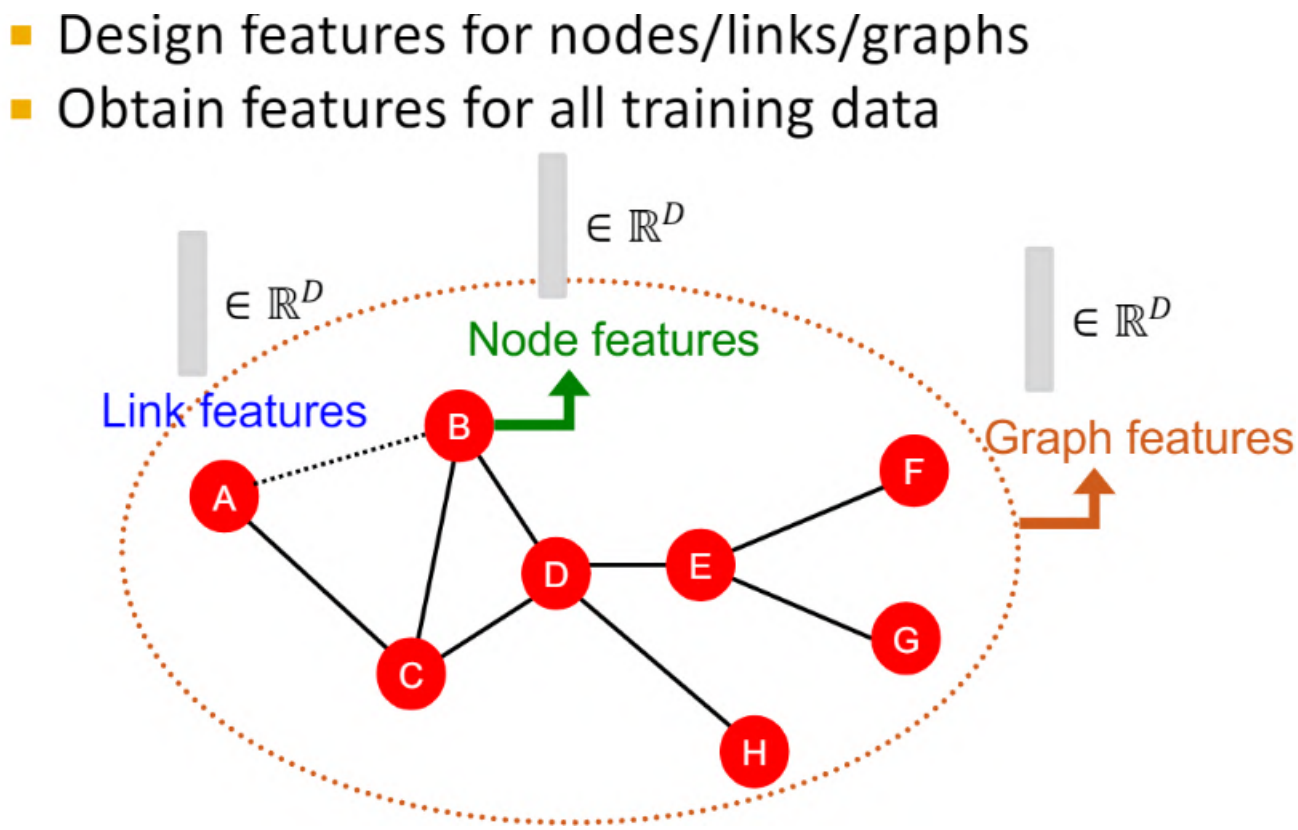


2 图上的传统机器学习方法

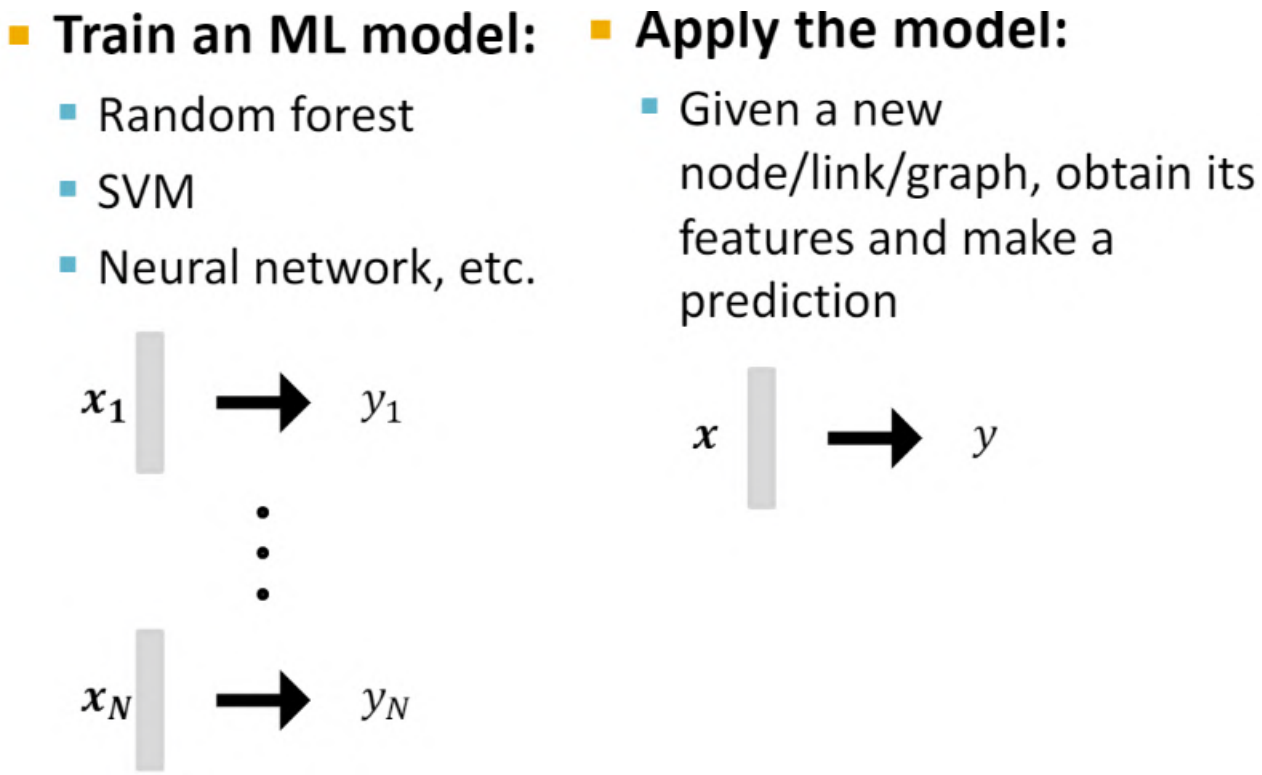
2.1 章节前言

传统图机器学习流程可分为以下四步：

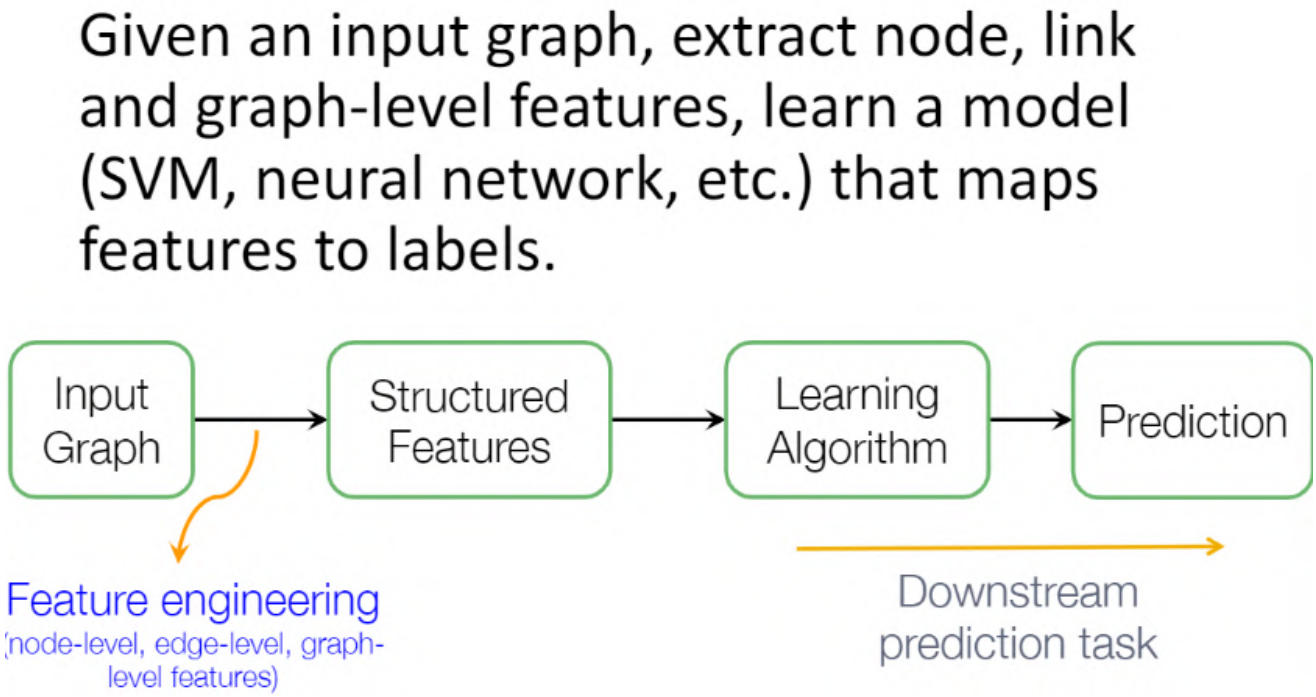
1. 第一步是根据不同的下游任务为节点/链接/图的人工设计特征（hand-designed features）
2. 第二步是将我们设计特征构建为训练数据集



3. 第三步是使用训练数据集训练一个机器学习模型，常见的有随机森林，SVM和神经网络等。
4. 第四步是使用训练好的的模型完成新样本的预测任务。

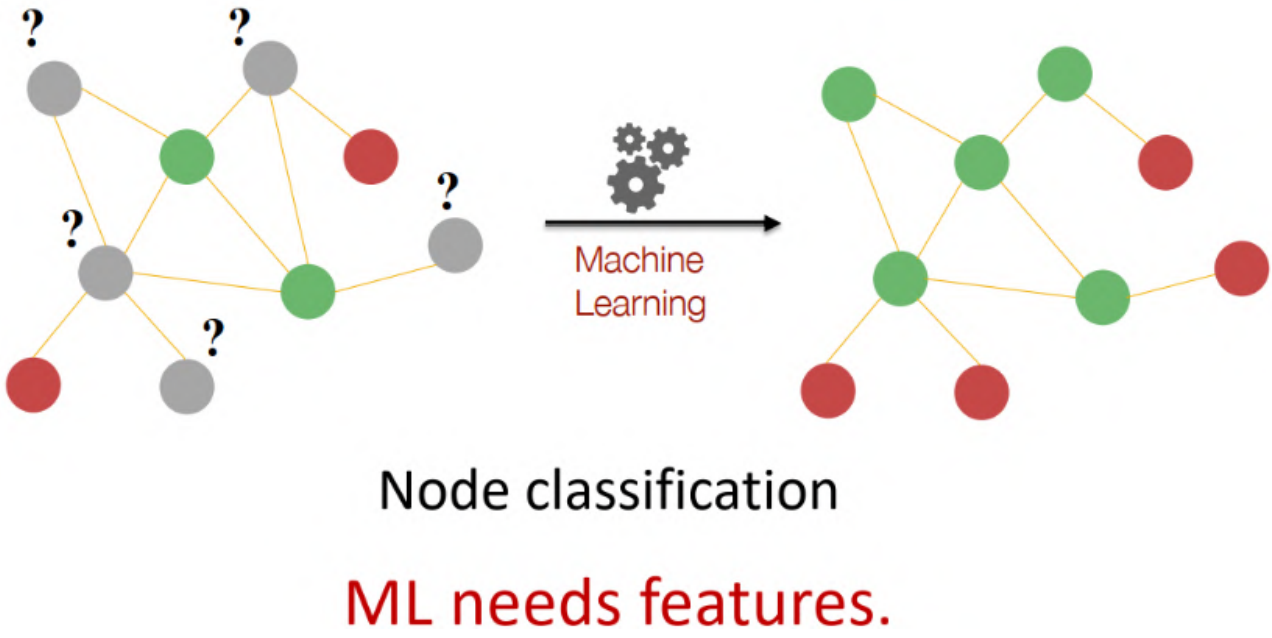


从上面的步骤中我们可以发现，在传统图机器学习中**模型性能好坏很大程度上取决于人工设计的图数据特征（hand-designed features）**



2.2 传统基于特征的方法：节点

半监督学习任务：

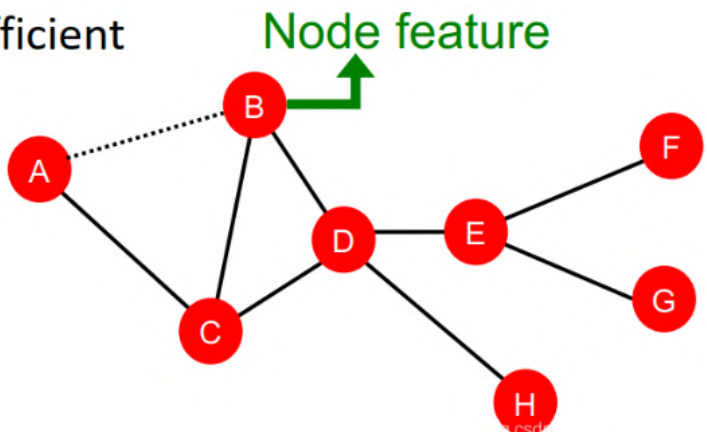


任务是预测灰点属于红点还是绿点。区分特征是度数（红点度数是1，绿点度数是2）。

特征抽取目标：找到能够描述节点在网络中结构与位置的特征。

Goal: Characterize the structure and position of a node in the network:

- Node degree
- Node centrality
- Clustering coefficient
- Graphlets



节点度数（node degree）：缺点在于将节点的所有邻居视为同等重要的。

节点中心性（node centrality） c_v：考虑了节点的重要性。

- **特征向量中心性（eigenvector centrality）**：认为如果节点邻居重要，那么节点本身也重要。因此节点 v 的centrality是邻居centrality的加总：$c_v = \frac{1}{\lambda}\sum_{u\in N(v)} c_u$，其中 λ 是一个正的常值。

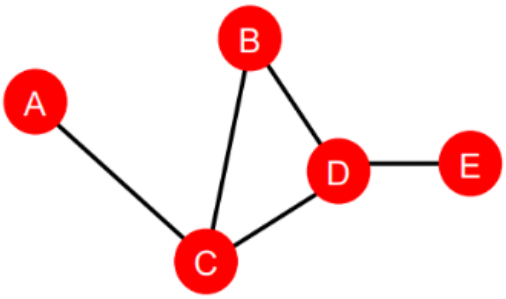
  这是个递归式，解法是将其转换为矩阵形式 $\lambda \mathbf{c} = \mathbf{A}\mathbf{c}$，其中 $\mathbf{A}$ 是邻接矩阵，$\mathbf{c}$ 是centrality向量。

  从而发现centrality就是特征向量。根据Perron-Frobenius Theorem可知最大的特征值 λ_{max} 总为正且唯一，对应的leading eigenvector c_{max} 就是centrality向量。

- **中介中心性（betweenness cent）**：认为如果一个节点处在很多节点对的最短路径上，那么这个节点是重要的。（衡量一个节点作为bridge或transit hub的能力。就对我而言直觉上感觉就像是新加坡的马六甲海峡啊，巴拿马运河啊，埃及的苏伊士运河啊，什么君士坦丁堡，上海，香港……之类的商业要冲的感觉）

$$c_v = \sum_{s\neq v\neq t} \frac{\#(s\text{和}t\text{之间包含}v\text{的最短路径})}{\#(s\text{和}t\text{之间的最短路径})}$$

- Example:

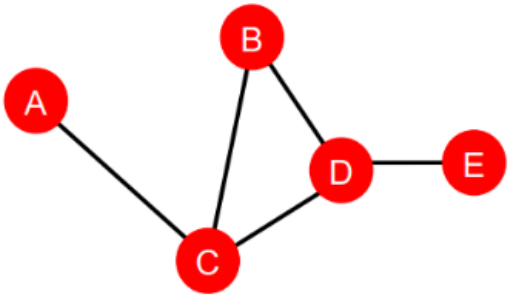


$$c_A = c_B = c_E = 0$$
$$c_C = 3$$
(A-C-B, A-C-D, A-C-D-E)
$$c_D = 3$$
(A-C-D-E, B-D-E, C-D-E)

- **紧密中心性（closeness centrality）**：认为如果一个节点距其他节点之间距离最短，那么认为这个节点是重要的。

$$c_v = \frac{1}{\sum_{u \neq v} u和v之间的最短距离}$$

- **Example:**



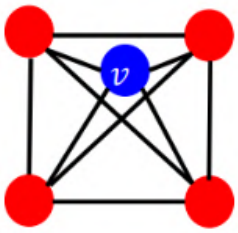
$c_A = 1/(2+1+2+3) = 1/8$
(A-C-B, A-C, A-C-D, A-C-D-E)

$c_D = 1/(2+1+1+1) = 1/5$
(D-C-A, D-B, D-C, D-E)

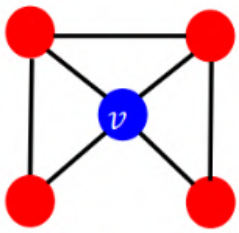
聚类系数（clustering coefficient）：衡量节点邻居的连接程度，描述节点的局部结构信息。

$$e_v = \frac{\#(相邻结点之间的边数)}{\binom{k_v}{2}} \in [0,1]$$

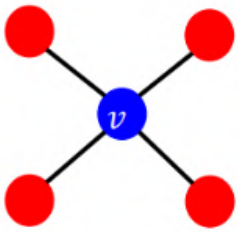
- **Examples:**



$e_v = 1$



$e_v = 0.5$

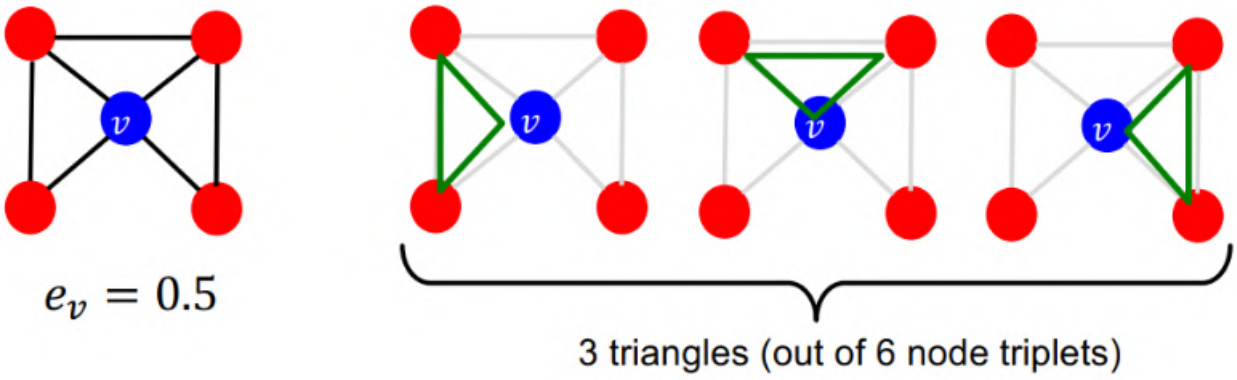


$e_v = 0$

第1个例子：$e_v = 6/6$
第2个例子：$e_v = 3/6$
第3个例子：$e_v = 0/6$

所以这个式子代表 v 邻居所构成的节点对，即潜在的连接数。整个公式衡量节点邻居的连接有多紧密。

- **Observation:** Clustering coefficient counts the #(triangles) in the ego-network

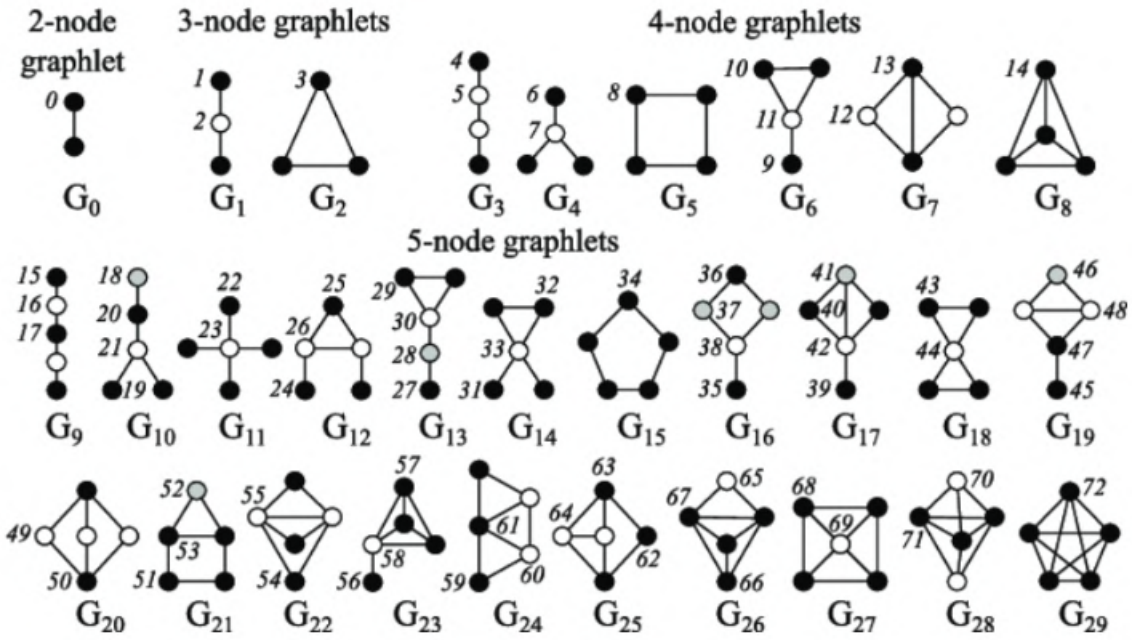


- We can generalize the above by counting #(pre-specified subgraphs, i.e., graphlets).

在社交网络之中会有很多这种三角形，因为可以想象你的朋友可能会经由你的介绍而认识，从而构建出一个这样的三角形/三元组。

有根连通异构子图（graphlet）：

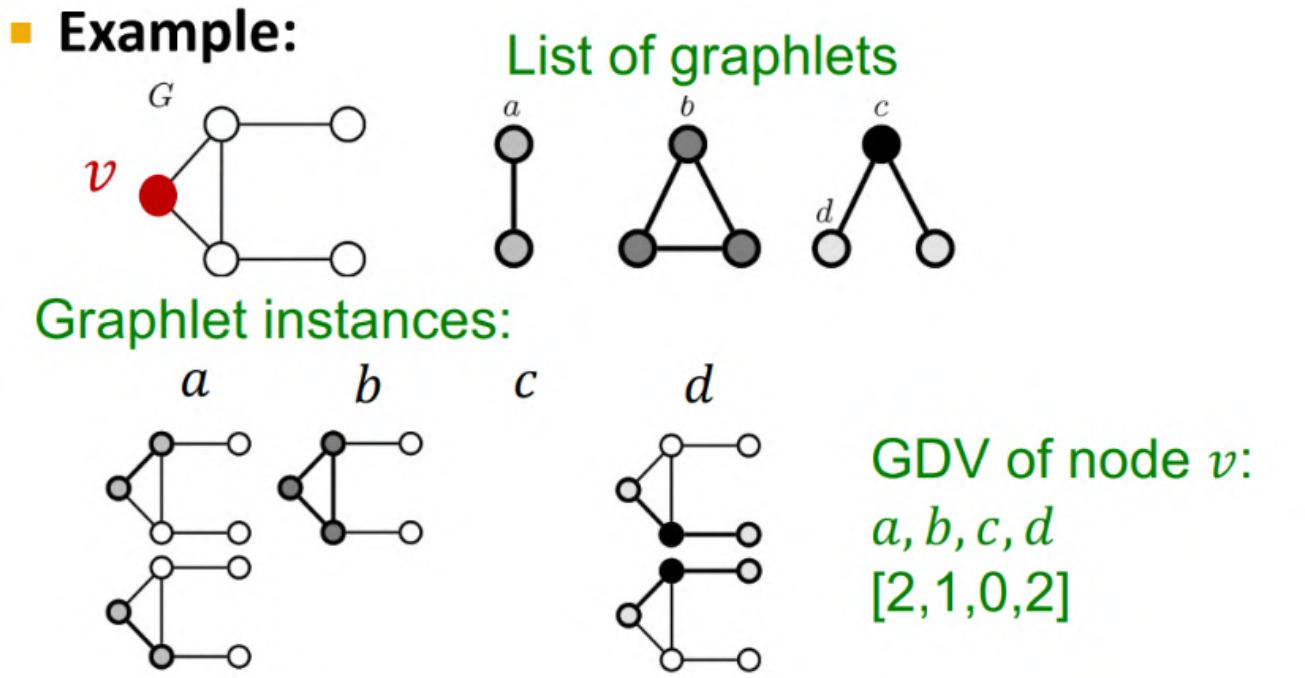
Graphlets: Rooted connected non-isomorphic subgraphs:



对于某一给定节点数 k，会有 n_k 个连通的异构子图。就是说，这些图首先是connected的，其次这些图有 k 个节点，第三它们异构。节点数为2-5情况下一共能产生如图所示73种graphlet。这73个graphlet的核心概念就是不同的形状，不同的位置。

（注意这里的graphlet概念和后文图的graphlet kernel的概念不太一样）

Graphlet Degree Vector（GDV）：节点基于graphlet的特征，是以给定节点为根的graphlet的计数向量。



（如图所示，对四种graphlet，v 的每一种graphlet的数量作为向量的一个元素。注意：graphlet c 的情况不存在，是因为像graphlet b 那样中间那条线连上了。这是因为graphlet是induced subgraph，所以那个边也存在，所以 c 情况不存在）

GDV与其他两种描述节点结构的特征的区别：

- **节点的度**：计算节点连接出去的边的个数。
- **聚类系数**：计算节点连接出去三角形的个数。
- **GDV**：计算节点连接出去的graphlet的个数。

考虑2-5个节点的graphlets，我们得到一个长度为73个坐标coordinate（就前图所示一共73种graphlet）的向量GDV，描述该点的局部拓扑结构topology of node's neighborhood，可以捕获距离为4 hops的互联性interconnectivities。相比节点度数或clustering coefficient，GDV能够描述两个节点之间更详细的节点局部拓扑结构相似性local topological similarity。

节点的特征可以分为两类：

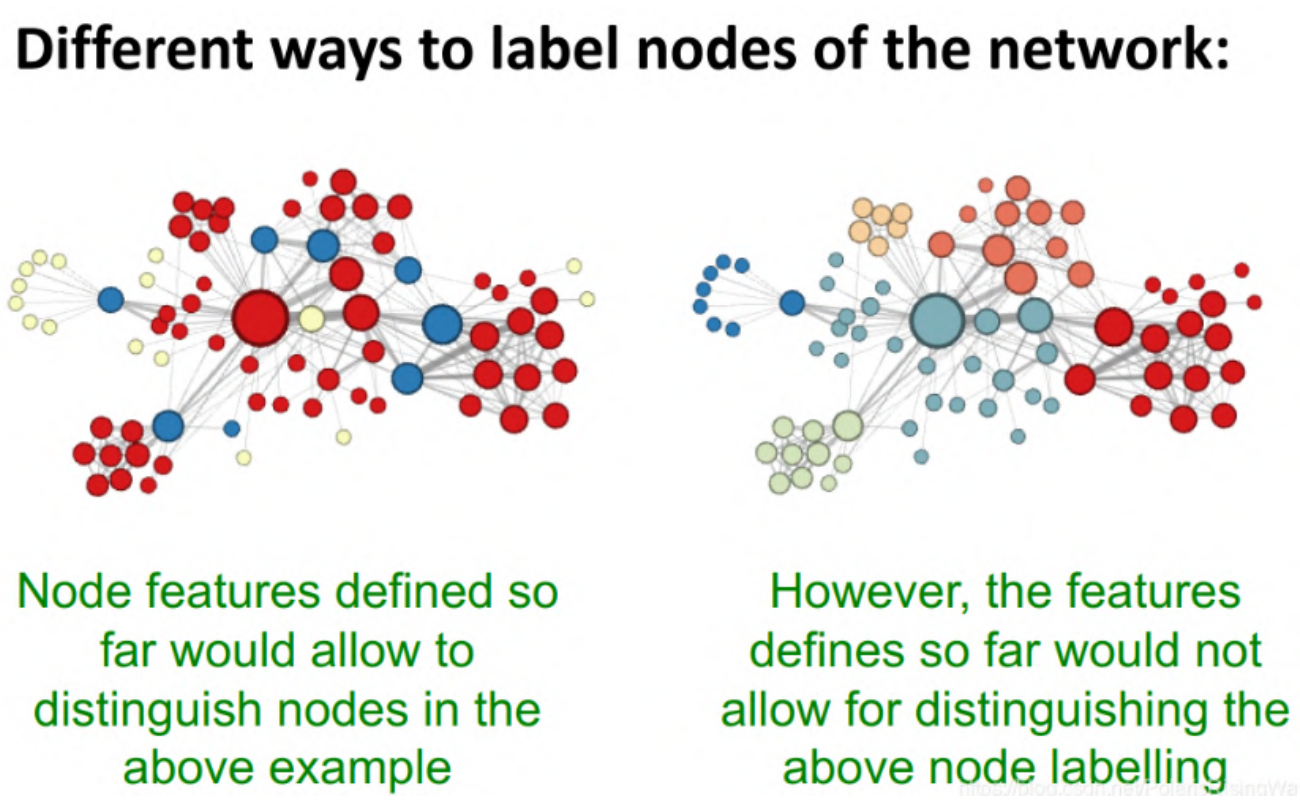
1. **Importance-based features**：捕获节点在图中的重要性。

- **Importance-based features**: capture the importance of a node is in a graph
  - Node degree:
    - Simply counts the number of neighboring nodes
  - Node centrality:
    - Models importance of neighboring nodes in a graph
    - Different modeling choices: eigenvector centrality, betweenness centrality, closeness centrality
- Useful for predicting influential nodes in a graph
  - **Example:** predicting celebrity users in a social network

2. **Structure-based features**：捕获节点附近的拓扑属性。

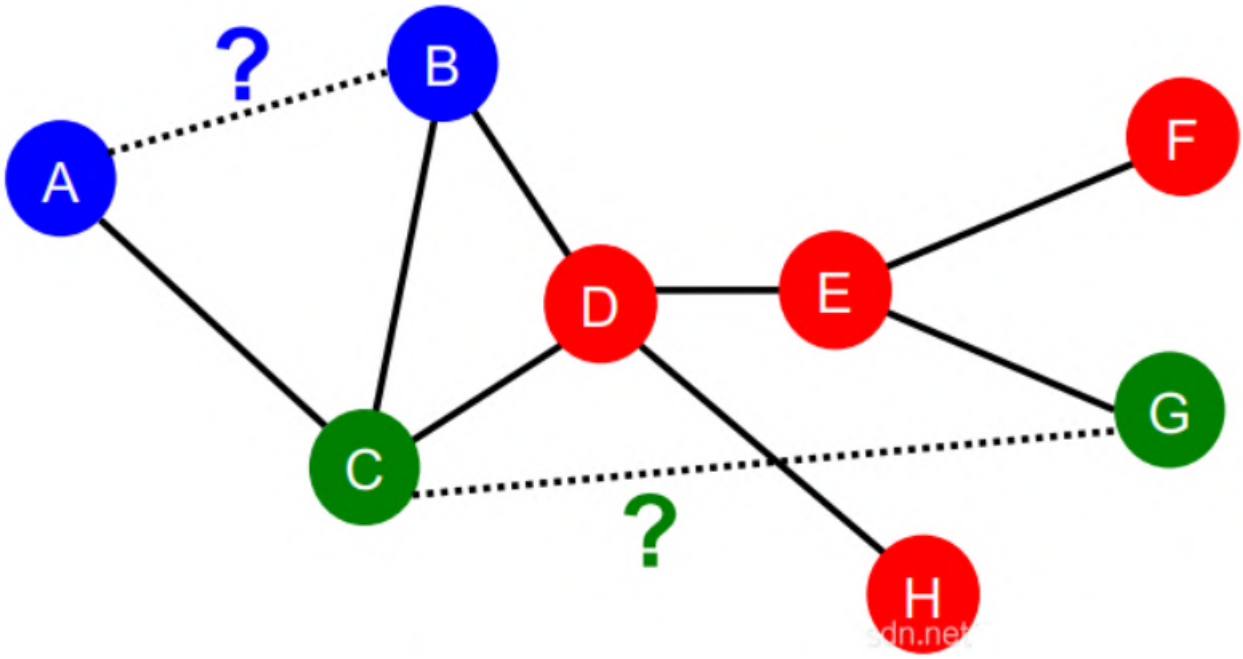
- **Structure-based features**: Capture topological properties of local neighborhood around a node.
  - **Node degree:**
    - Counts the number of neighboring nodes
  - **Clustering coefficient:**
    - Measures how connected neighboring nodes are
  - **Graphlet degree vector:**
    - Counts the occurrences of different graphlets
- **Useful for predicting a particular role a node plays in a graph:**
  - **Example:** Predicting protein functionality in a protein-protein interaction network.

结语：传统节点特征只能识别出结构上的相似，不能识别出图上空间、距离上的相似。



2.3 传统基于特征的方法：连接

预测任务是基于已知的边，预测新链接的出现。测试模型时，将每一对无链接的点对进行排序，取存在链接概率最高的K个点对，作为预测结果。

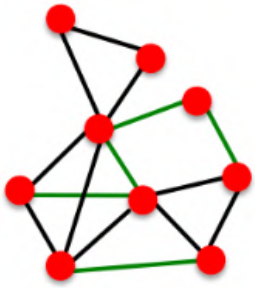


有时你也可以直接将两个点的特征合并concatenate起来作为点对的特征，来训练模型。但这样做的缺点就在于失去了点之间关系的信息。

链接预测任务的两种类型：随机缺失边、随时间演化边。

Two formulations of the link prediction task:

- **1) Links missing at random:**
  - Remove a random set of links and then aim to predict them
- **2) Links over time:**
  - Given $G[t_0, t_0']$ a graph on edges up to time t_0', output a ranked list L of links (not in $G[t_0, t_0']$) that are predicted to appear in $G[t_1, t_1']$
  - **Evaluation:**
    - $n = |E_{new}|$: # new edges that appear during the test period $[t_1, t_1']$
    - Take top n elements of L and count correct edges



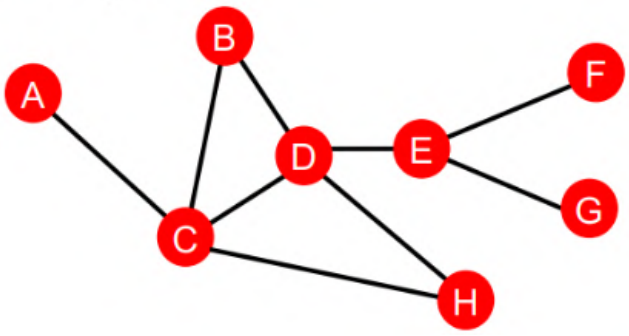
$G[t_0, t_0']$
$G[t_1, t_1']$

基于相似性进行链接预测：计算两点间的相似性得分（如用共同邻居衡量相似性），然后将点对进行排序，得分最高的n组点对就是预测结果，与真实值作比较。

- **Methodology:**
  - For each pair of nodes *(x,y)* compute score *c(x,y)*
    - For example, *c(x,y)* could be the # of common neighbors of *x* and *y*
  - Sort pairs *(x,y)* by the decreasing score *c(x,y)*
  - **Predict top *n* pairs as new links**
  - **See which of these links actually appear in $G[t_1, t_1']$**

基于距离的特征：两点间最短路径的长度。

- Example:



$$S_{BH} = S_{BE} = S_{AB} = 2$$
$$S_{BG} = S_{BF} = 3$$

这种方式的问题在于没有考虑两个点邻居的重合度the degree of neighborhood overlap，如B-H有2个共同邻居，B-E和A-B都只有1个共同邻居。

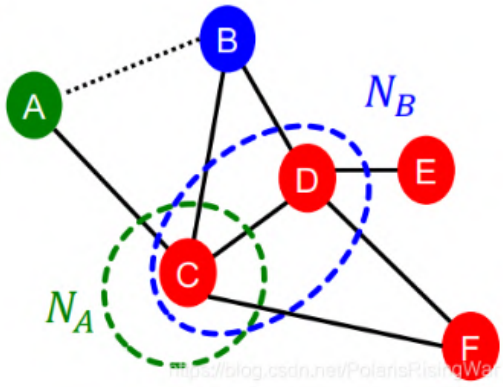
局部邻域重合（local neighborhood overlap）：捕获节点的共同邻居数。

Captures # neighboring nodes shared between two nodes v_1 and v_2:

- **Common neighbors:** $|N(v_1) \cap N(v_2)|$
  - Example: $|N(A) \cap N(B)| = |\{C\}| = 1$
- **Jaccard's coefficient:** $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$
  - Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{C,D\}|} = \frac{1}{2}$
- **Adamic-Adar index:**

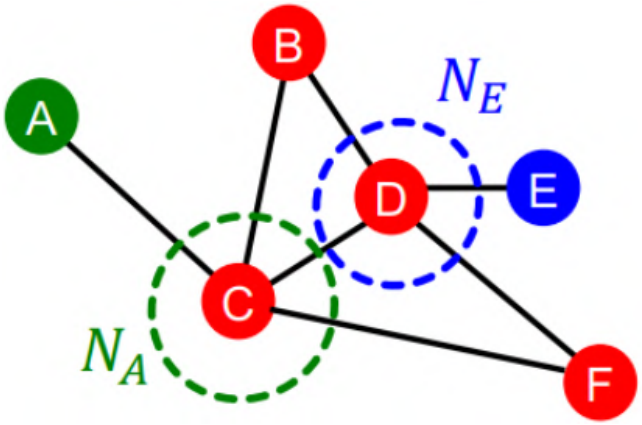
$$\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$$

  - Example: $\frac{1}{\log(k_C)} = \frac{1}{\log 4}$



common neighbors的问题在于度数高的点对就会有更高的结果，Jaccard's coefficient是其归一化后的结果。Adamic-Adar index在实践中表现得好。在社交网络上表现好的原因：有一堆度数低的共同好友比有一堆名人共同好友的得分更高。

全局邻域重合（global neighborhood overlap）：局部邻域重合的限制在于，如果两个点没有共同邻居，值就为0。



$$N_A \cap N_E = \phi$$
$$|N_A \cap N_E| = 0$$

但是这两个点未来仍有可能被连接起来。所以我们使用考虑全图的global neighborhood overlap来解决这一问题。

Katz指标（Katz index）：计算点对之间所有长度路径的条数。

- Katz index between v_1 and v_2 is calculated as

Sum over *all path lengths*

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \beta^l A^l_{v_1 v_2}$$

#paths of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

- Katz index matrix is computed in closed-form:

$$\boldsymbol{S} = \sum_{i=1}^{\infty} \beta^i \boldsymbol{A}^i = \underbrace{(\boldsymbol{I} - \beta \boldsymbol{A})^{-1}}_{= \sum_{i=0}^{\infty} \beta^i \boldsymbol{A}^i \text{ by geometric series of matrices}} - \boldsymbol{I},$$

（discount factor β会给比较长的距离以比较小的权重）

证明：

β 是权重衰减因子，为了保证数列的收敛性，其取值需小于邻接矩阵A最大特征值的倒数。

该方法权重衰减因子的最优值只能通过大量的实验验证获得，因此具有一定的局限性。

解析解推导过程：
矩阵形式的表达式为$\mathrm{S}=\beta \mathrm{A}+\beta^{2} \mathrm{A}^{2}+\beta^{3} \mathrm{A}^{3} \ldots$

$$
\begin{aligned}
&(\mathrm{I}-\beta \mathrm{A})(\mathrm{I}+\mathrm{S}) \\
=&(\mathrm{I}-\beta \mathrm{A})(\mathrm{I}+\beta \mathrm{A}+\beta^{2} \mathrm{A}^{2}+\beta^{3} \mathrm{A}^{3} \ldots) \\
=&(\mathrm{I}+\beta \mathrm{A}+\beta^{2} \mathrm{A}^{2}+\beta^{3} \mathrm{A}^{3} \ldots)-(\beta \mathrm{A}+\beta^{2} \mathrm{A}^{2}+\beta^{3} \mathrm{A}^{3} \ldots) \\
=&\mathrm{I}
\end{aligned}
$$

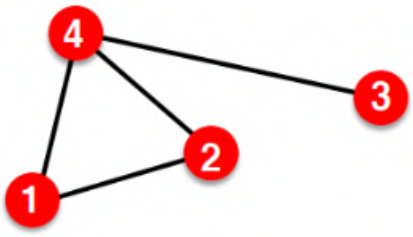
所以$\mathrm{I}+\mathrm{S}=(\mathrm{I}-\beta \mathrm{A})^{-1}$
故$\mathrm{S}=(\mathrm{I}-\beta \mathrm{A})^{-1}-\mathrm{I}$

计算方式：邻接矩阵求幂，邻接矩阵的 k 次幂结果，每个元素就是对应点对之间长度为 k 的路径的条数。

证明：

- **Computing #paths between two nodes**
  - **Recall:** $A_{uv} = 1$ if $u \in N(v)$
  - Let $P_{uv}^{(K)}$ = #paths of length K between u and v
  - We will show $P^{(K)} = A^k$
  - $P_{uv}^{(1)}$ = #paths of length 1 (direct neighborhood) between u and $v = A_{uv}$

$P_{12}^{(1)} = A_{12}$



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

- **How to compute $P_{uv}^{(2)}$?**
  - **Step 1:** Compute **#paths** of length 1 **between each of u's neighbor and v**
  - **Step 2: Sum up** these #paths across u's neighbors
  - $\boldsymbol{P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^{(1)} = \sum_i A_{ui} * A_{iv} = A_{uv}^2}$

Node 1's neighbors　　#paths of length 1 between Node 1's neighbors and Node 2　　$P_{12}^{(2)} = A_{12}^2$

Power of adjacency

$$A^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix}$$

- How to compute #paths between two nodes?
- Use **adjacency matrix powers**!
  - A_{uv} specifies #paths of length 1 (direct neighborhood) between u and v.
  - A_{uv}^2 specifies #paths of **length 2** (neighbor of neighbor) between u and v.
  - And, A_{uv}^l specifies #paths of **length l**.

结语:

1. **基于距离的特征**：使用两节点之间的最短路径长度，但是无法捕捉邻域重叠。
2. **局部邻域重合**：
   - 捕捉两节点共有的邻居节点的数量。
   - 当两节点没有共有节点时为0.
3. **全局邻域重合**：
   - 使用全局图结构计算两节点之间的值。
   - Katz 指标计算两节点之间所有长度的路径数量。

2.4 传统基于特征的方法：图

图级别特征构建目标：找到能够描述全图结构的特征。

背景：核方法（Kernel Methods）

- **Kernel methods** are widely-used for traditional ML for graph-level prediction.
- **Idea: Design kernels instead of feature vectors.**
- **A quick introduction to Kernels:**
  - Kernel $K(G, G') \in \mathbb{R}$ measures similarity b/w data
  - Kernel matrix $\boldsymbol{K} = \left(K(G, G')\right)_{G,G'}$ must always be positive semidefinite (i.e., has positive eigenvals)
  - There exists a feature representation $\phi(\cdot)$ such that $K(G, G') = \phi(\mathrm{G})^{\mathrm{T}}\phi(G')$
  - Once the kernel is defined, off-the-shelf ML model, such as kernel SVM, can be used to make predictions.

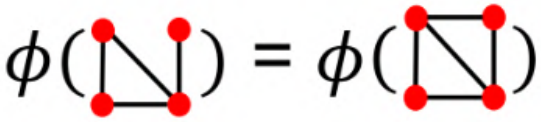
总结来说：两个图的核 $K(G, G')$ 用标量衡量相似度，存在特征表示 $\Phi(\cdot)$ 使得 $K(G, G') = \Phi(G)^T\Phi(G')$， 定义好核后就可以直接应用核SVM之类的传统机器学习模型。

概述：

- **Graph Kernels:** Measure similarity between two graphs:
  - Graphlet Kernel [1]
  - Weisfeiler-Lehman Kernel [2]
  - Other kernels are also proposed in the literature (beyond the scope of this lecture)
    - Random-walk kernel
    - Shortest-path graph kernel
    - And many more…

关键思想：

- **Goal**: Design graph feature vector $\phi(G)$
- **Key idea**: Bag-of-Words (BoW) for a graph
  - **Recall:** BoW simply uses the word counts as features for documents (no ordering considered).
  - Naïve extension to a graph: Regard nodes as words.
  - Since both graphs have 4 red nodes, we get the same feature vector for two different graphs…

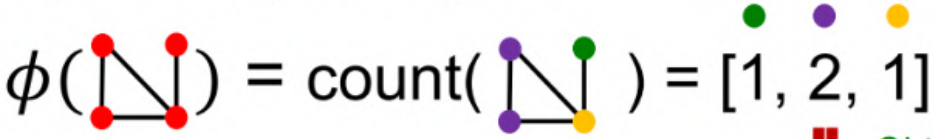


bag-of-words相当于是把文档表示成一个向量，每个元素代表对应word出现的次数。此处讲述的特征抽取方法也将是bag-of-something的形式，将图表示成一个向量，每个元素代表对应something出现的次数（这个something可以是node, degree, graphlet, color）

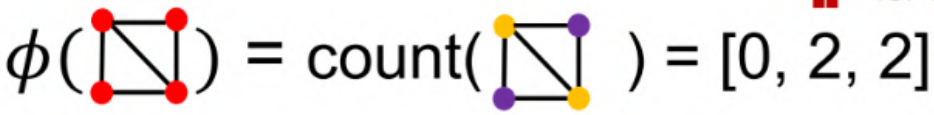
光用node不够的话，可以设置一个degree kernel，用bag-of-degrees来描述图特征。

What if we use Bag of **node degrees**?

Deg1: ● Deg2: ● Deg3: ●



Obtains different features for different graphs!



- Both Graphlet Kernel and Weisfeiler-Lehman (WL) Kernel use **Bag-of-*** representation of graph, where * is more sophisticated than node degrees!

graphlet 特征：计算图中不同 graphlet 的数量。

注意这里对graphlet的定义跟上文节点层面特征抽取里的graphlet不一样。

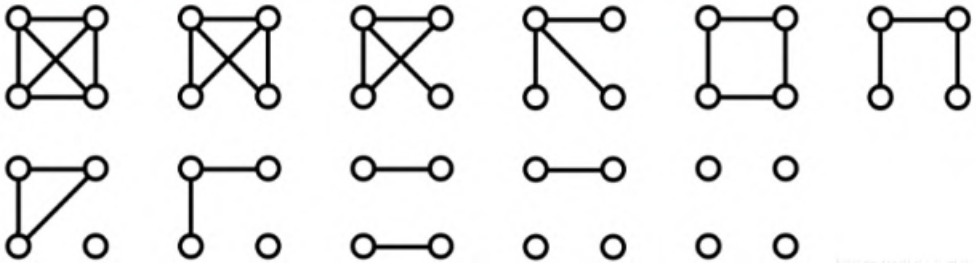
区别在于：这里 graphlets 中的节点不需要相连，可以有相互隔离的节点。

对每一种节点数，可选的graphlet：

Let $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$ be a list of graphlets of size k.

- For $k = 3$, there are 4 graphlets.

- For $k = 4$, there are 11 graphlets.

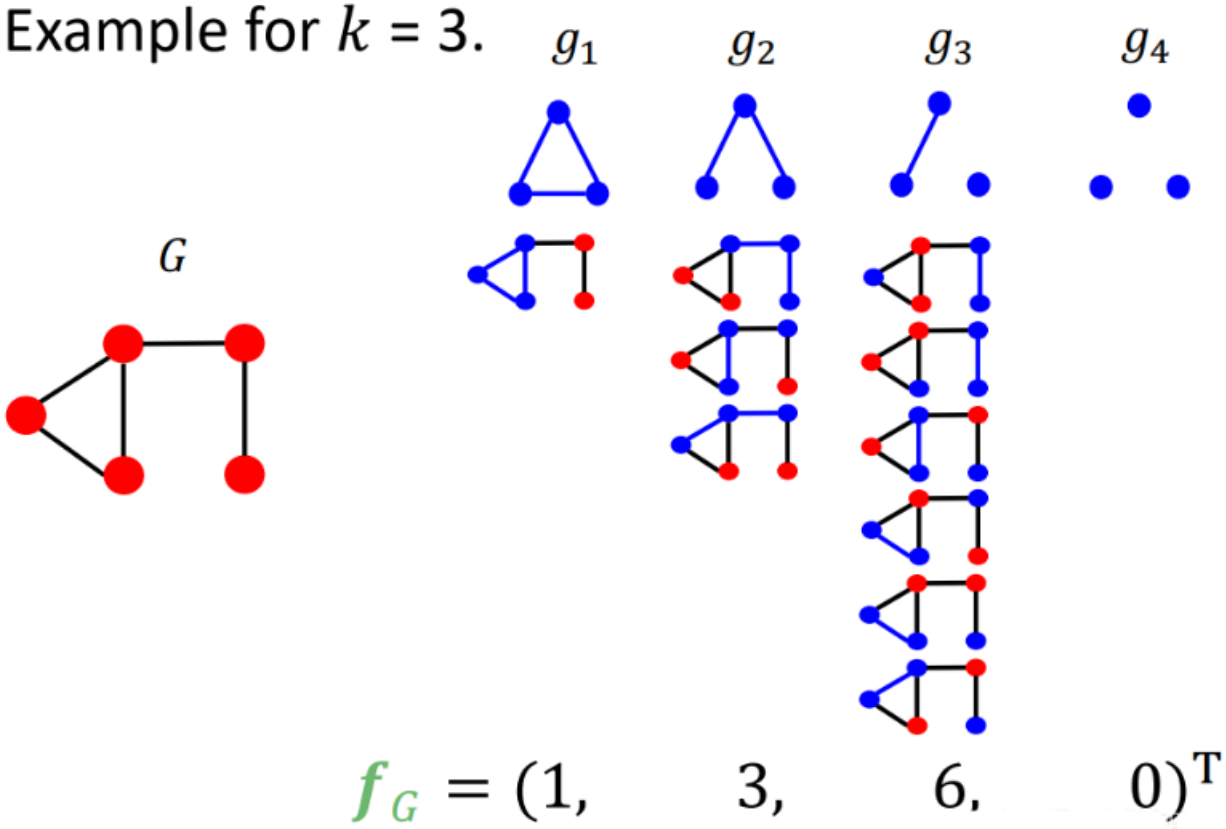


graphlet count vector：每个元素是图中对应graphlet的数量。

- Given graph G, and a graphlet list $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$, define the graphlet count vector $f_G \in \mathbb{R}^{n_k}$ as

$$(f_G)_i = \#(g_i \subseteq G) \text{ for } i = 1, 2, \ldots, n_k.$$

- Example for $k = 3$.



graphlet kernel就是直接点积两个图的graphlet count vector得到相似性。对于图尺寸相差较大的情况需进行归一化。

- Given two graphs, G and G', graphlet kernel is computed as

$$K(G, G') = \boldsymbol{f}_G{}^{\mathrm{T}} \boldsymbol{f}_{G'}$$

- **Problem:** if G and G' have different sizes, that will greatly skew the value.
- **Solution:** normalize each feature vector

$$\boldsymbol{h}_G = \frac{\boldsymbol{f}_G}{\mathrm{Sum}(\boldsymbol{f}_G)} \qquad K(G, G') = \boldsymbol{h}_G{}^{\mathrm{T}} \boldsymbol{h}_{G'}$$

graphlet kernel的限制：计算昂贵。

Limitations: Counting graphlets is expensive!

- Counting size-k graphlets for a graph with size n by enumeration takes n^k.
- This is unavoidable in the worst-case since subgraph isomorphism test (judging whether a graph is a subgraph of another graph) is **NP-hard**.
- If a graph's node degree is bounded by d, an $O(nd^{k-1})$ algorithm exists to count all the graphlets of size k.

WL 核（Weisfeiler-Lehman Kernel）：相比graphlet kernel代价较小，效率更高。用节点邻居结构迭代地来扩充节点信息。

- **Goal**: design an efficient graph feature descriptor $\phi(G)$
- **Idea**: use neighborhood structure to iteratively enrich node vocabulary.
  - Generalized version of **Bag of node degrees** since node degrees are one-hop neighborhood information.

实现算法：Weisfeiler-Lehman graph isomorphism test=color refinement

- **Given:** A graph G with a set of nodes V.
  - Assign an initial color $c^{(0)}(v)$ to each node v.
  - Iteratively refine node colors by

$$c^{(k+1)}(v) = \text{HASH}\left(\left\{c^{(k)}(v), \{c^{(k)}(u)\}_{u \in N(v)}\right\}\right),$$

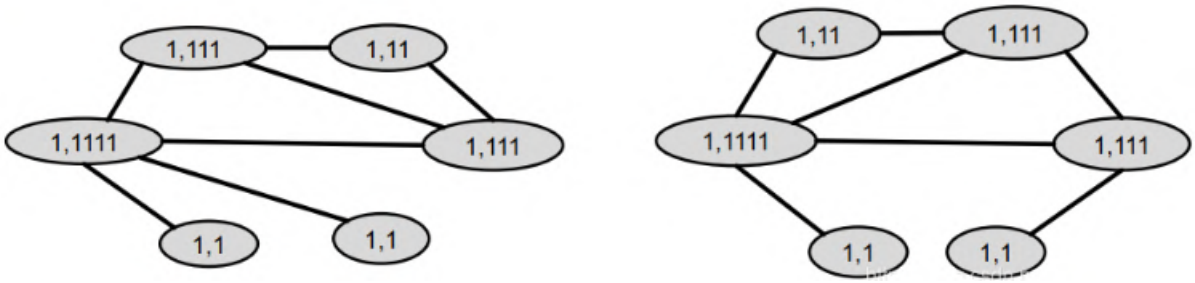
    where HASH maps different inputs to different colors.
  - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood

color refinement示例:

Example of color refinement given two graphs

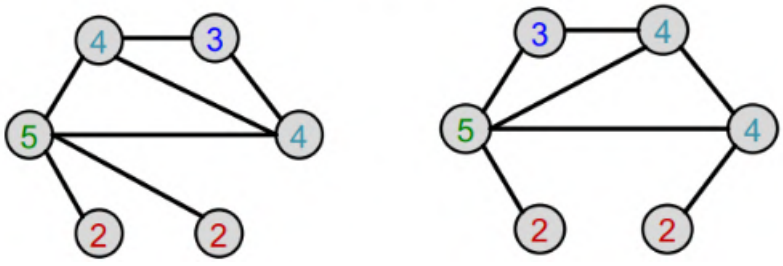
- Assign initial colors

- Aggregate neighboring colors



对聚集后颜色取哈希值:

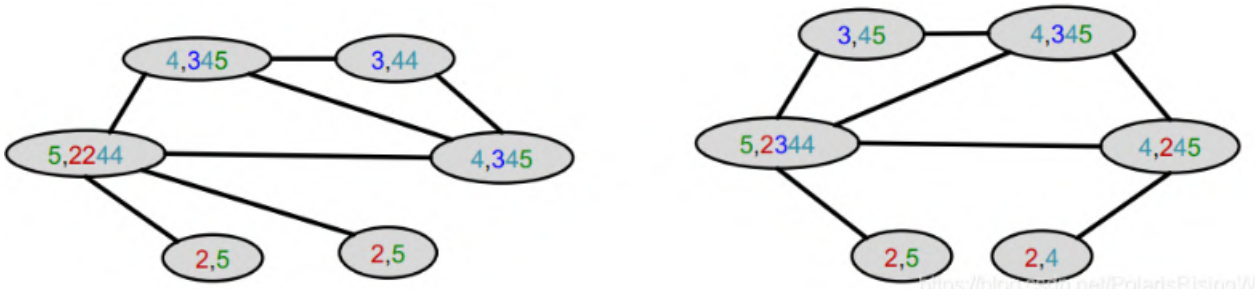
- Hash aggregated colors



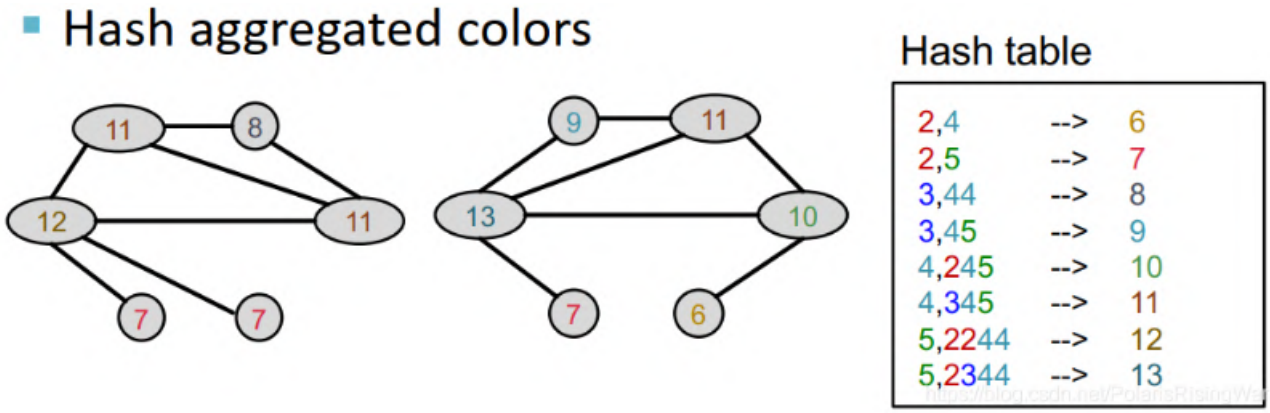
Hash table

1,1	-->	2
1,11	-->	3
1,111	-->	4
1,1111	-->	5

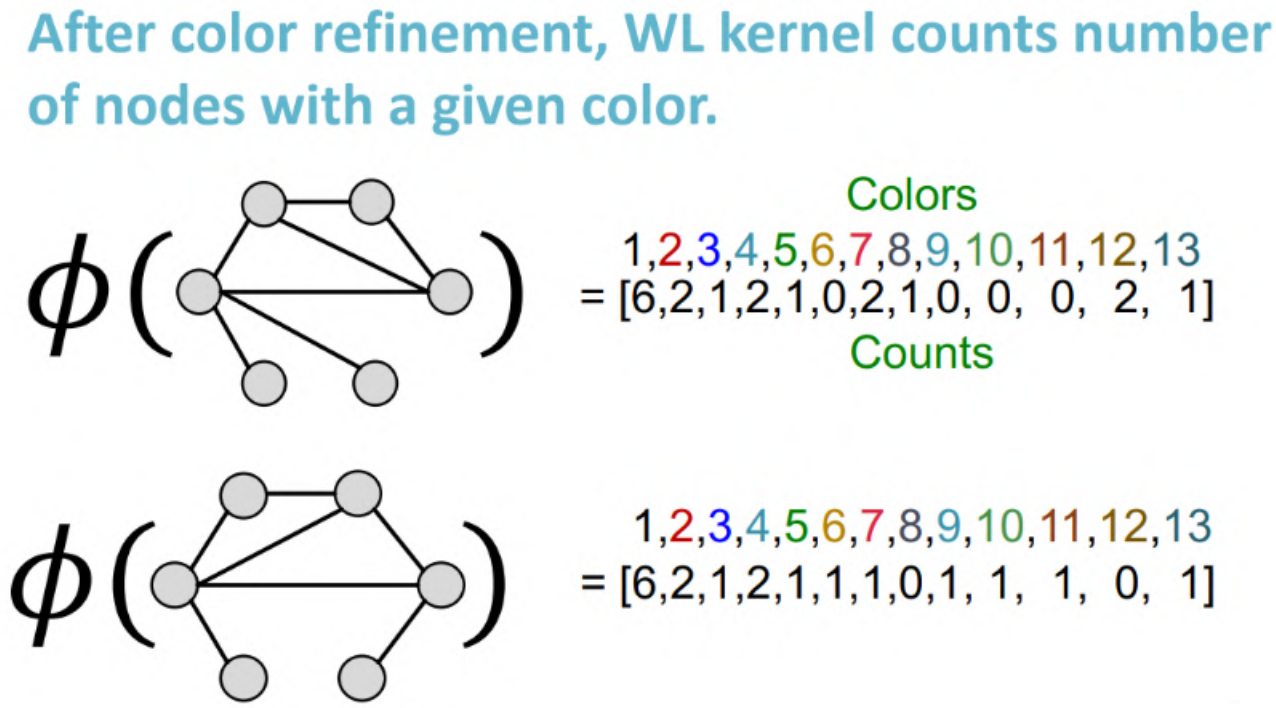
把邻居颜色聚集起来:



对聚集后颜色取哈希值:

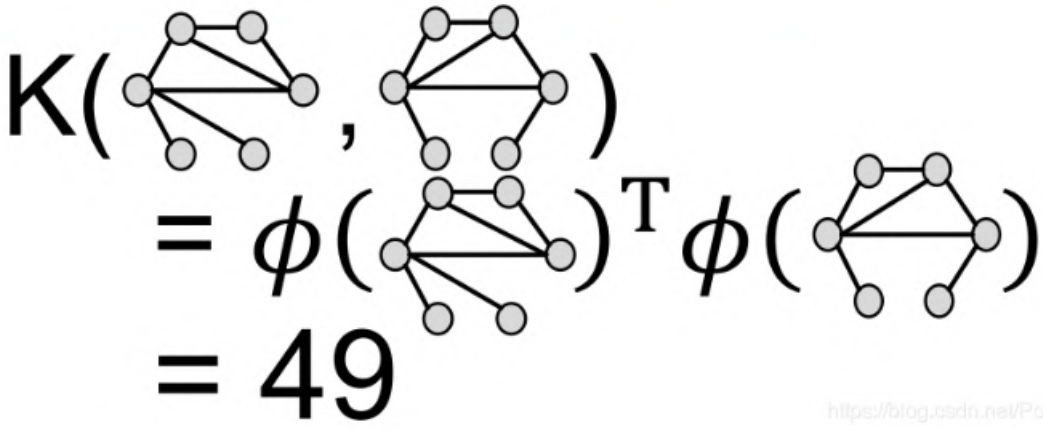


进行 K 次迭代后，用整个迭代过程中颜色出现的次数作为Weisfeiler-Lehman graph feature:



用上图的向量点积计算相似性，得到WL kernel:

The WL kernel value is computed by the inner product of the color count vectors:



WL kernel的优势在于计算成本低:

- WL kernel is **computationally efficient**
  - The time complexity for color refinement at each step is linear in #(edges), since it involves aggregating neighboring colors.
- When computing a kernel value, only colors appeared in the two graphs need to be tracked.
  - Thus, #(colors) is at most the total number of nodes.
- Counting colors takes linear-time w.r.t. #(nodes).
- In total, time complexity is **linear in #(edges)**.

结语:

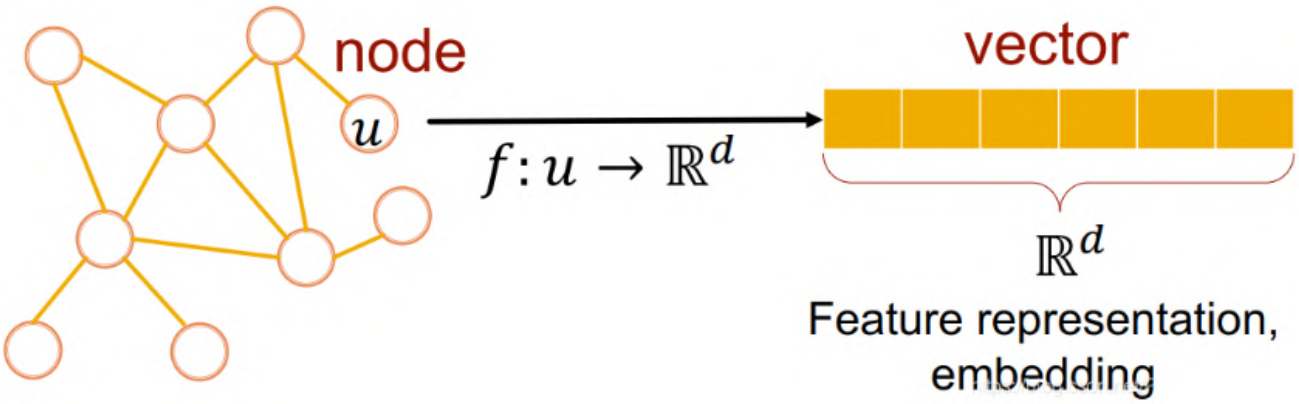
- **Graphlet Kernel**
  - Graph is represented as **Bag-of-graphlets**
  - **Computationally expensive**
- **Weisfeiler-Lehman Kernel**
  - Apply K-step color refinement algorithm to enrich node colors
    - Different colors capture different K-hop neighborhood structures
  - Graph is represented as **Bag-of-colors**
  - **Computationally efficient**
  - Closely related to Graph Neural Networks (as we will see!)

3 节点嵌入

3.1 章节前言

图表示学习（graph representation learning）：学习到图数据用于机器学习的、与下游任务无关的特征，我们希望这个向量能够抓住数据的结构信息。

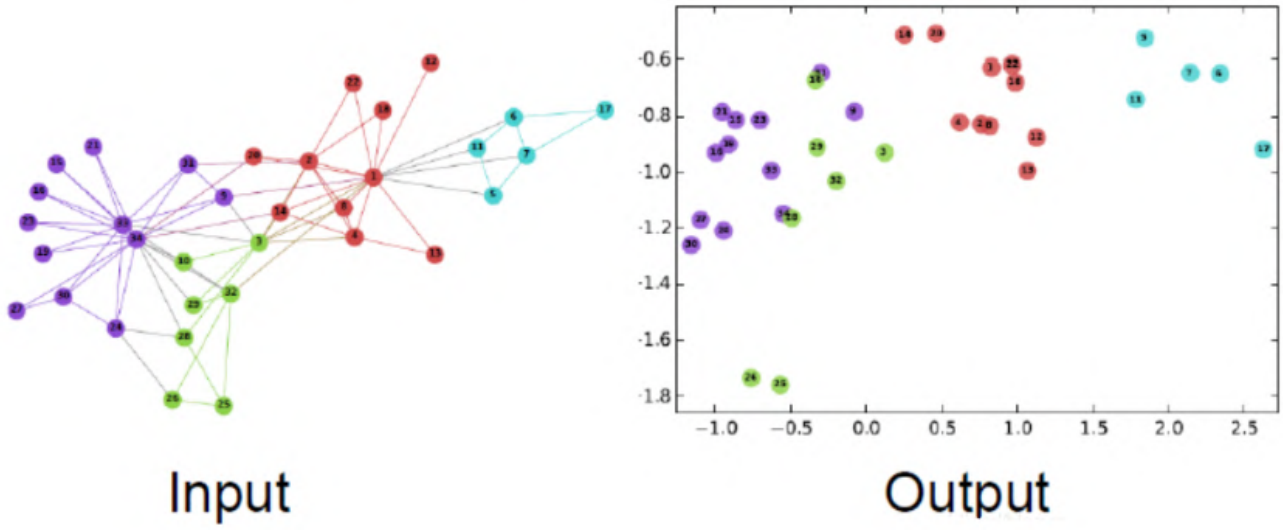
这个数据被称作**特征表示（feature representation）**或**嵌入（embedding）**。



为什么要嵌入：将节点映射到embedding space

- embedding的相似性可以反映原节点在网络中的相似性，比如定义有边连接的点对为相似的点，则这样的点的embedding应该离得更近。
- embedding编码网络信息。
- embedding可用于下游预测任务。

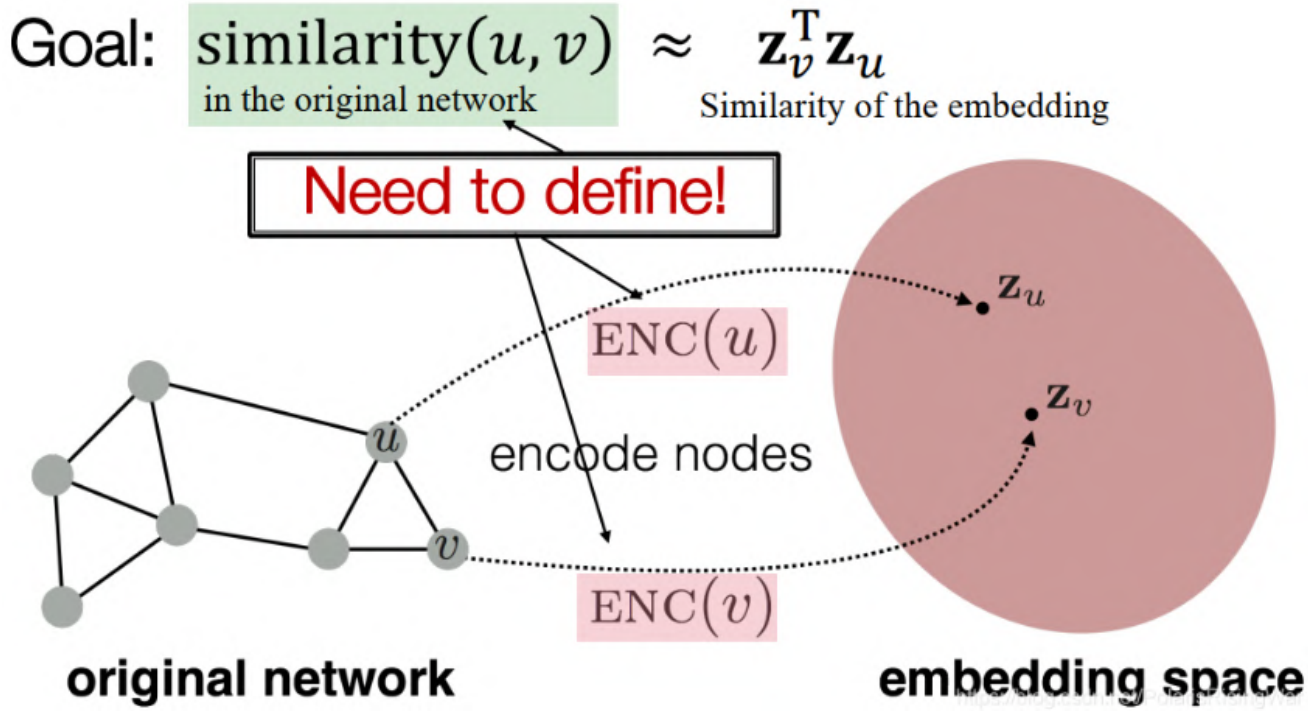
node embedding举例：二维节点嵌入可视化（将不同类的节点很好地分开了）



3.2 节点嵌入：编码与解码

图 G，节点集合 V，邻接矩阵 A（二元的）（简化起见：不考虑节点的特征或其他信息）

节点嵌入：目标是将节点编码为embedding space中的向量，使embedding的相似度（如点积）近似于图中节点的相似度（需要被定义）。



编码（Encoder）：将节点映射为embedding。

定义一个衡量节点相似度的函数（如衡量在原网络中的节点相似度）

解码（Decoder）：将embedding对映射为相似度得分。

Optimize the parameters of the encoder so that:

$\mathrm{DEC}(\mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u)$

$$\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$$

in the original network　　Similarity of the embedding

两个关键组件:

- **Encoder:** maps each node to a low-dimensional vector

$$\mathrm{ENC}(v) = \mathbf{z}_v$$

d-dimensional embedding

node in the input graph

- **Similarity function:** specifies how the relationships in vector space map to the relationships in the original network

$$\text{similarity}(u, v) \approx \mathbf{z}_v^{\mathrm{T}}\mathbf{z}_u$$

Decoder

Similarity of u and v in the original network

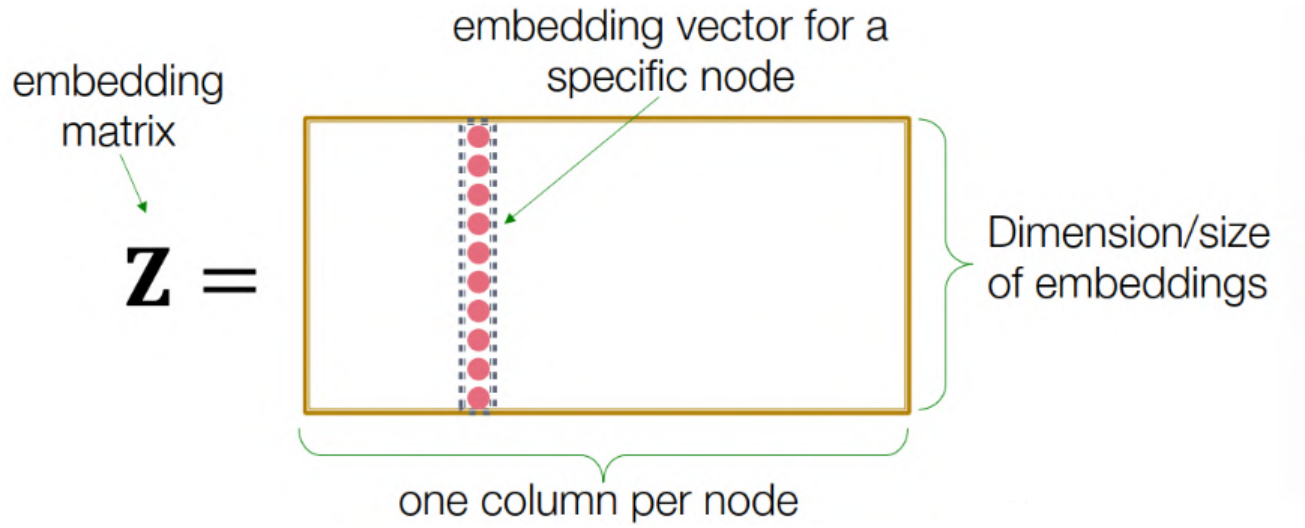
dot product between node embeddings

浅编码（shallow encoding）：最简单的编码方式，编码器只是一个**嵌入查找（embedding-lookup）**。

$$\mathrm{ENC}(v) = \mathbf{z}_v = \mathbf{Z} \cdot v$$

$\mathbf{Z} \in \mathbb{R}^{d \times |\mathcal{V}|}$ matrix, each column is a node embedding [what we learn / optimize]

$v \in \mathbb{I}^{|\mathcal{V}|}$ indicator vector, all zeroes except a one in column indicating node v



Z 每列是一个节点所对应的embedding向量。v 是一个其他元素都为0，对应节点位置的元素为1的向量。通过矩阵乘法的方式得到结果。

这种方式就是将每个节点直接映射为一个embedding向量，我们的学习任务就是直接优化这些embedding。

缺点：参数多，很难scale up到大型图上。

优点：如果获得了 Z，各节点就能很快得到embedding。

（有很多种方法：如DeepWalk，node2vec等）

节点相似的不同定义：

- 有边
- 共享邻居
- 有相似的结构特征
- **随机游走random walk**定义的节点相似度

3.3 节点嵌入：随机行走方法

统一符号表示notation：

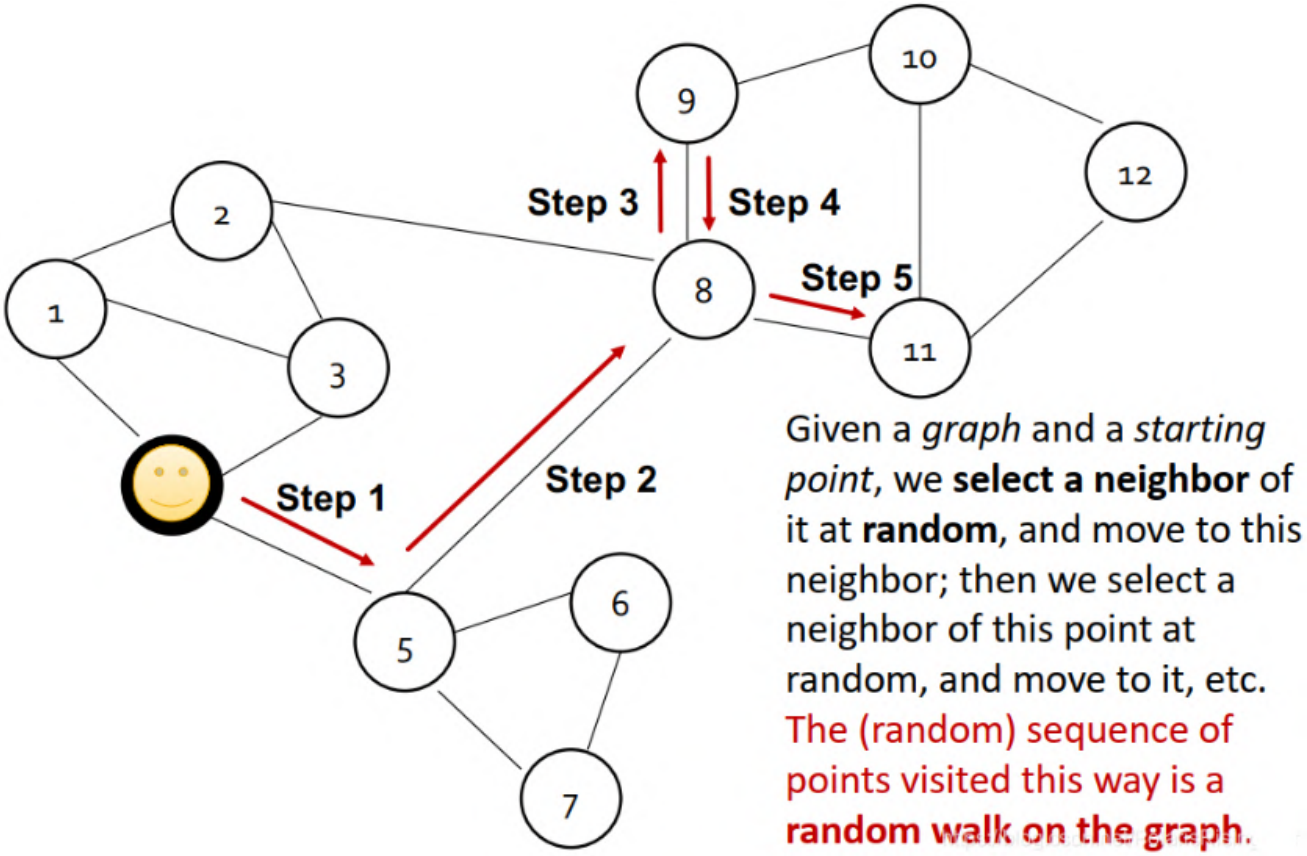
- **Vector** $\mathbf{z}_u$:
  - The embedding of node u (what we aim to find).
- **Probability** $P(v|\mathbf{z}_u)$: ⟸ Our model prediction based on $\mathbf{z}_u$
  - The **(predicted) probability** of visiting node v on random walks starting from node u.

Non-linear functions used to produce predicted **probabilities**

- **Softmax** function
  - Turns vector of K real values (model predictions) into K probabilities that sum to 1: $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$.
- **Sigmoid** function:
  - S-shaped function that turns real values into the range of (0, 1). Written as $S(x) = \frac{1}{1+e^{-x}}$.

$P(v|\mathbf{z}_u)$ 是从 u 开始随机游走能得到 v 的概率，衡量 u 和 v 的相似性，用节点embedding向量相似性算概率。
用于计算预测概率的非线性函数：softmax会将一组数据归一化为和为1的形式，最大值的结果会几乎为1。sigmoid会将实数归一化到 (0,1) 上。

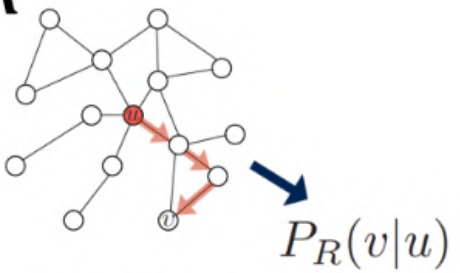
随机行走（random walk）：从某一节点开始，每一步按照概率选一个邻居，走过去，重复。停止时得到一系列节点。



随机行走嵌入（random walk embeddings）：

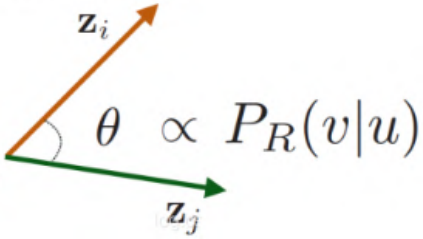
$\mathbf{z}_u^T \mathbf{z}_v \approx u$和$v$在一次随机游走中(以$u$为起点)同时出现的概率

1. **Estimate probability of visiting node v on a random walk starting from node u using some random walk strategy R**



2. **Optimize embeddings to encode these random walk statistics:**

Similarity in embedding space (Here: dot product=$\cos(\theta)$) encodes random walk "similarity"



为什么选择随机行走方法:

表达性：灵活的节点相似性随机定义，结合了局部和高阶邻域信息。

想法：如果从节点 $\boldsymbol{u}$ 开始随机行走访问 $\boldsymbol{v}$ 的概率很高，则 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 相似（高阶多跳信息）

效率：训练时不需要考虑所有节点对；只需要考虑随机行走中同时出现的节点对。

无监督特征学习：

直觉：查找节点嵌入d-维空间的向量，可以保持相似性。

想法：学习节点嵌入，使网络中邻接的节点相互靠近。

【问题】给定节点 u，如何定义邻接节点?

$N_R(u)$ 用于表示节点 u 通过一些随机行走策略 R 得到的邻居。

作为优化的特征学习：

目标是使每个节点 u，$N_R(u)$ 的节点与 $\mathbf{z}_u$ 靠近，也就是 $P(N_R(u)|\mathbf{z}_u)$ 值很大。

$$f: u \rightarrow R^d, f(u) = \mathbf{z}_u$$

极大似然目标函数：

$$\max_f \sum_{u \in V} logP(N_R(u)|\mathbf{z}_u)$$

对这个目标函数的理解是：对节点 u，我们希望其表示向量对其random walk neighborhood $N_R(u)$ 的节点是predictive的（可以预测到它们的出现）

随机行走的优化：

1. Run **short fixed-length random walks** starting from each node u in the graph using some random walk strategy R
2. For each node u collect $N_R(u)$, the multiset* of nodes visited on random walks starting from u
3. Optimize embeddings according to: Given node u, predict its neighbors $N_R(u)$

$$\max_f \sum_{u \in V} \log \mathrm{P}(N_R(u)|\, \mathbf{z}_u) \Rightarrow \text{Maximum likelihood objective}$$

*$N_R(u)$ can have repeat elements since nodes can be visited multiple times on random walks

把最大似然估计翻过来，拆开，就成了需被最小化的损失函数 L：

$$L = \sum_{u \in V} \sum_{v \in N_R(u)} -log(P(v|\mathbf{z}_u))$$

- **Intuition:** Optimize embeddings z_u to maximize the likelihood of random walk co-occurrences
- **Parameterize $P(v|\mathbf{z}_u)$ using softmax:**

$$P(v|\mathbf{z}_u) = \frac{\exp(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_n)}$$

Why softmax?
We want node v to be most similar to node u (out of all nodes n).
Intuition: $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

这个计算概率 $P(v|\mathbf{z}_u)$ 选用 softmax 的 intuition 就是前文所提及的，softmax 会使最大元素输出靠近1，也就是在节点相似性最大时趋近于1。

将 $P(v|\mathbf{z}_u)$ 代入 L 得到：

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}}\mathbf{z}_n)}\right)$$

sum over all nodes u

sum over nodes v seen on random walks starting from u

predicted probability of u and v co-occuring on random walk

优化 random walk embeddings 就是找到 $\mathbf{z}$ 使得 L 最小。

但是计算这个公式代价很大，因为需要内外加总2次所有节点，复杂度达 $O(|V|^2)$ 。

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

Nested sum over nodes gives $O(|V|^2)$ complexity!

我们发现问题就在于用于softmax归一化的这个分母：

$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log\left(\frac{\exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)}{\sum_{n \in V} \exp(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_n)}\right)$$

为了解决这个分母，我们使用negative sampling的方法：简单来说就是原本我们是用所有节点作为归一化的负样本，现在我们只抽出一部分节点作为负样本，通过公式近似减少计算。

- **Solution:** Negative sampling

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

Why is the approximation valid?
Technically, this is a different objective. But Negative Sampling is a form of Noise Contrastive Estimation (NCE) which approx. maximizes the log probability of softmax.

New formulation corresponds to using a logistic regression (sigmoid func.) to distinguish the target node v from nodes n_i sampled from background distribution P_v.

More at https://arxiv.org/pdf/1402.3722.pdf

$$\approx \log\left(\sigma(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_v)\right) - \sum_{i=1}^{k} \log\left(\sigma(\mathbf{z}_u^{\mathrm{T}} \mathbf{z}_{n_i})\right), n_i \sim P_V$$

sigmoid function (makes each term a "probability" between 0 and 1)

random distribution over nodes

Instead of normalizing w.r.t. all nodes, just normalize against k random "**negative samples**" n_i

这个随机分布不是uniform random，而是random in a biased way：概率与其度数成比例。

负样本个数 k 的考虑因素：

- 更高的 k 会使估计结果更鲁棒robust（我的理解是方差）

- 更高的k会使负样本上的偏差bias更高（其实我没搞懂这是为什么）
- 实践上的k常用值：5-20

优化目标函数（最小化损失函数）的方法：随机梯度下降 SGD

随机行走总结：

1. Run **short fixed-length** random walks starting from each node on the graph
2. For each node u collect $N_R(u)$, the multiset of nodes visited on random walks starting from u
3. Optimize embeddings using Stochastic Gradient Descent:

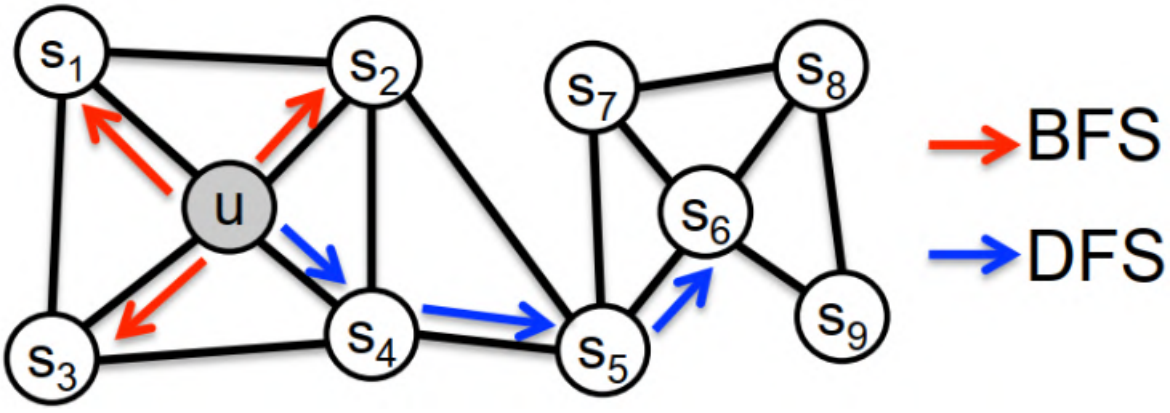
$$\mathcal{L} = \sum_{u \in V} \sum_{v \in N_R(u)} -\log(P(v|\mathbf{z}_u))$$

We can efficiently approximate this using negative sampling!

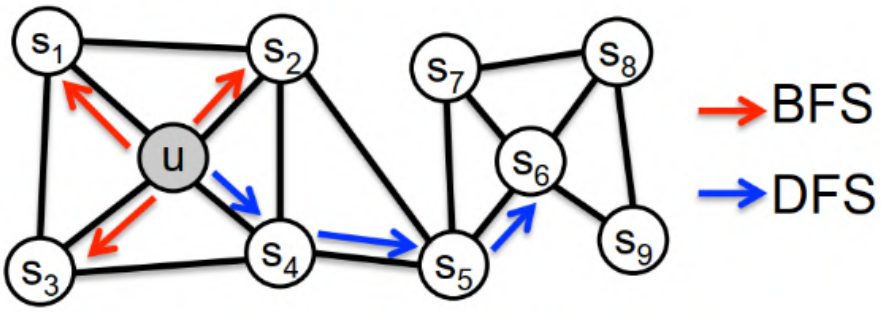
随机行走策略：

1. **DeepWalk**：仅使用固定长度，无偏地从起始节点开始进行随机行走。但是相似度概念受限。
2. **node2vec**：有弹性的网络邻居 $N_R(u)$ 定义使 u 的embedding更丰富，因此使用有偏的二阶随机游走策略以产生 $N_R(u)$ 。

   用有弹性、有偏的随机游走策略平衡local（BFS）和global（DFS）的节点网络结构信息。



Two classic strategies to define a neighborhood $N_R(u)$ of a given node u:



Walk of length 3 ($N_R(u)$ of size 3):

$N_{BFS}(u) = \{s_1, s_2, s_3\}$ Local microscopic view

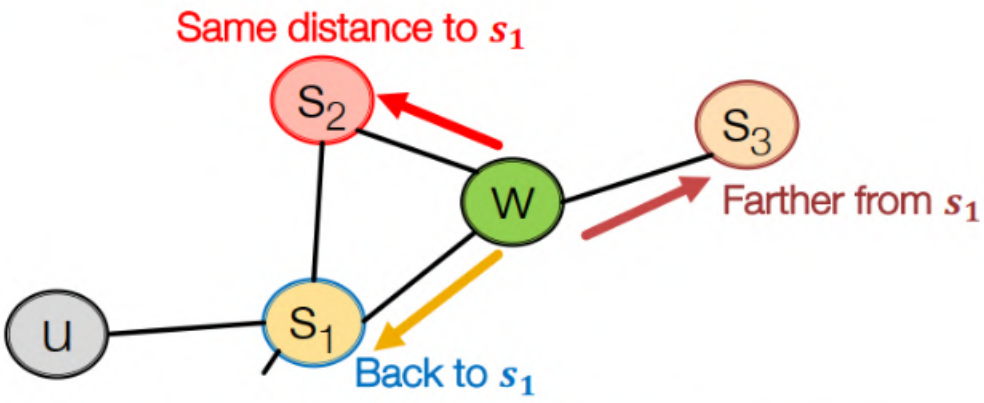
$N_{DFS}(u) = \{s_4, s_5, s_6\}$ Global macroscopic view

有偏定长随机游走的参数:

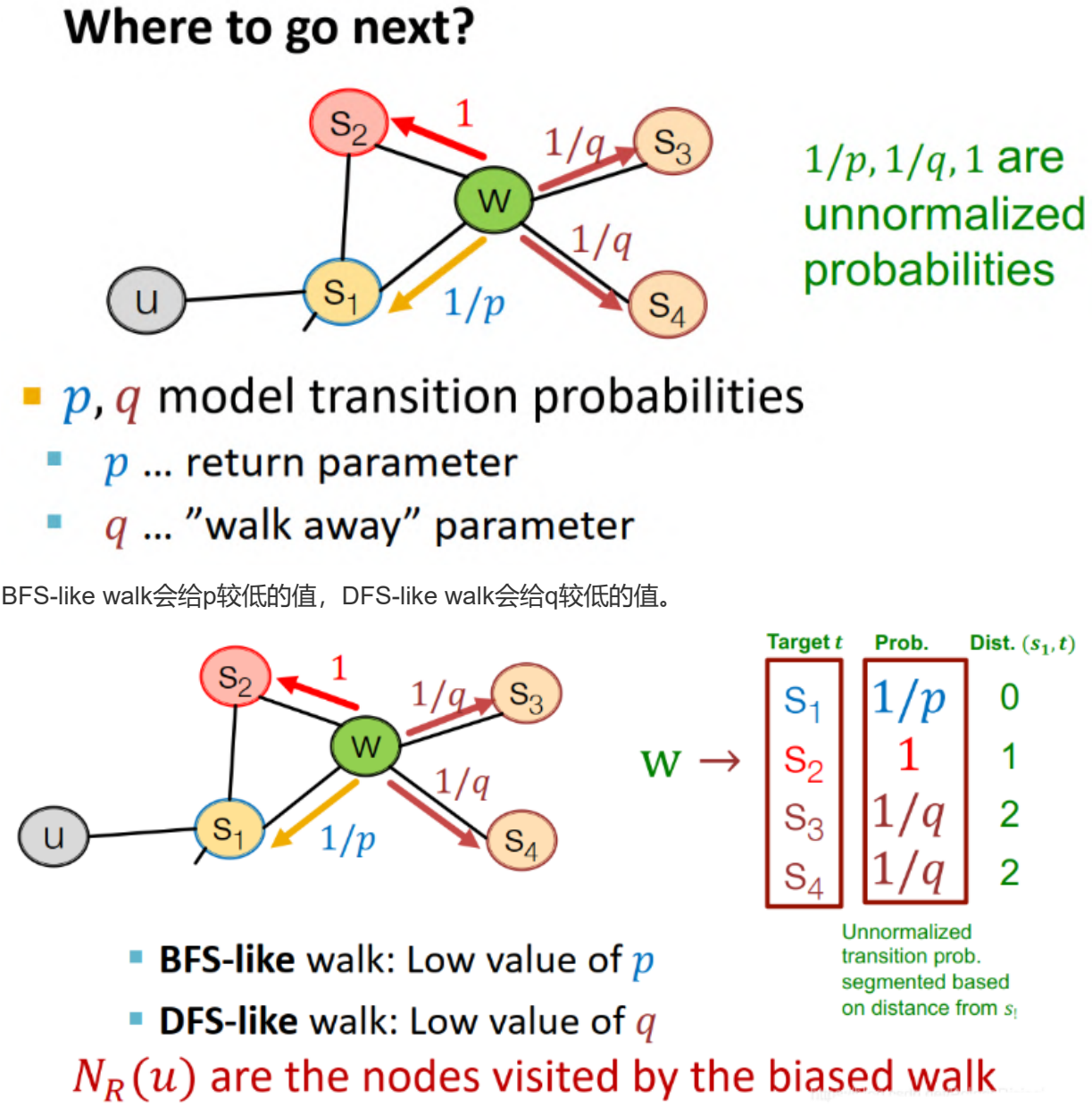
- return parameter p： 返回上一个节点的概率
- in-out parameter q： 向外走（DFS）VS 向内走（BFS），相比于DFS，选择BFS的概率。

有偏随机游走举例：上一步是 (s_1, w)

- **Insight:** Neighbors of w can only be:

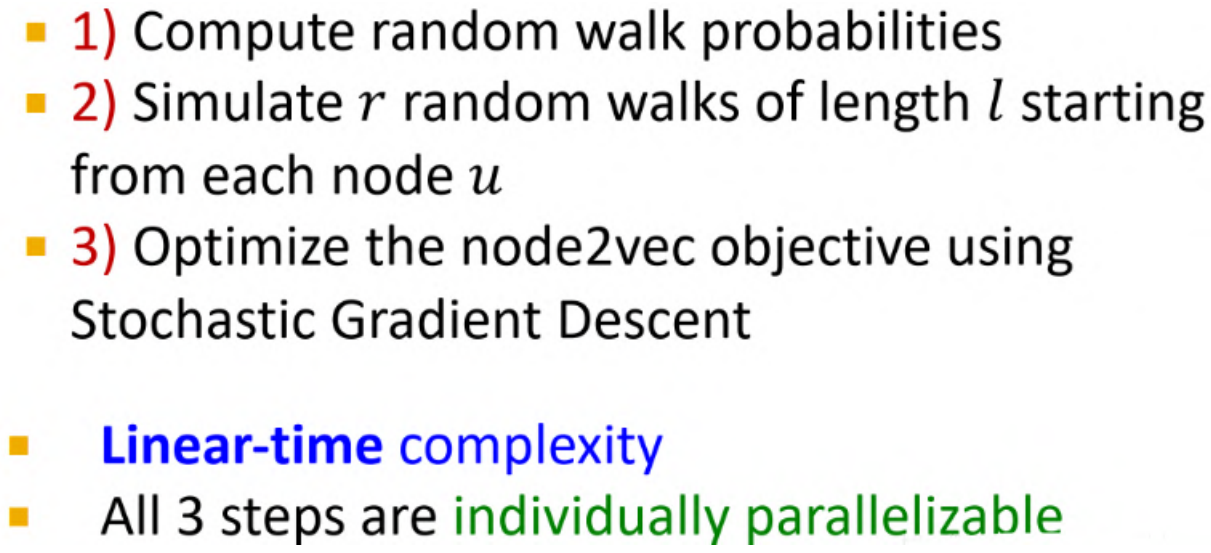


Idea: Remember where the walk came from



BFS-like walk会给p较低的值，DFS-like walk会给q较低的值。

node2vec算法:



线性时间复杂度是因为节点邻居数量是固定的。

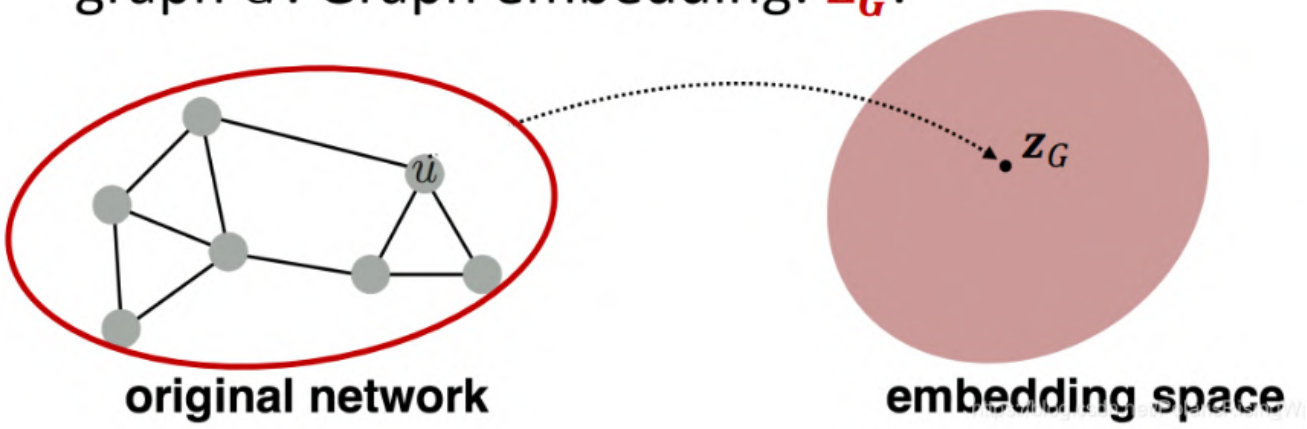
3. **其他随机游走方法**:

- **Different kinds of biased random walks:**
  - Based on node attributes (Dong et al., 2017).
  - Based on learned weights (Abu-El-Haija et al., 2017)
- **Alternative optimization schemes:**
  - Directly optimize based on 1-hop and 2-hop random walk probabilities (as in LINE from Tang et al. 2015).
- **Network preprocessing techniques:**
  - Run random walks on modified versions of the original network (e.g., Ribeiro et al. 2017's struct2vec, Chen et al. 2016's HARP).

3.4 嵌入整个图

任务目标：嵌入子图或整个图 G，得到表示向量 $\mathbf{z}_G$ 。

- **Goal:** Want to embed a subgraph or an entire graph G. Graph embedding: $\mathbf{z}_G$.



任务示例：

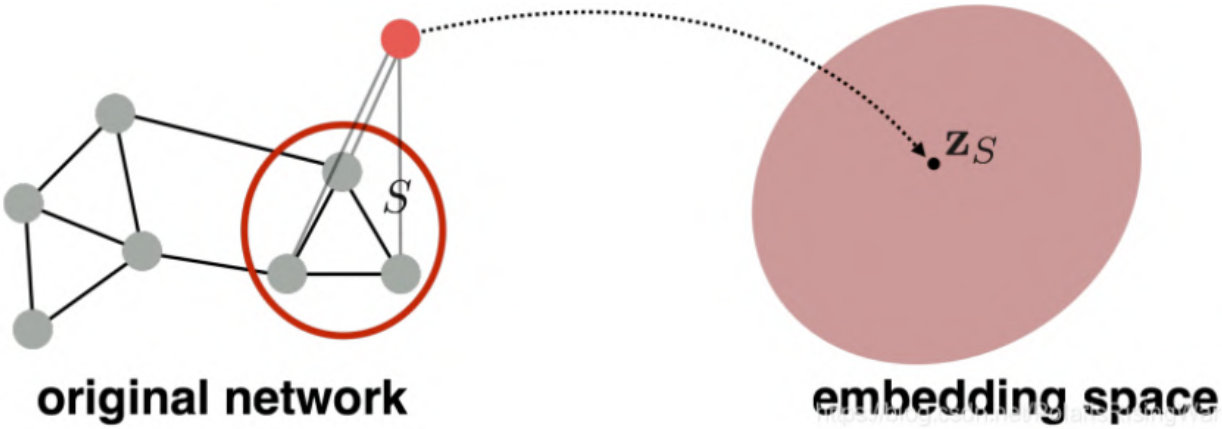
- 分类有毒/无毒分子
- 识别异常图

也可以视为对节点的一个子集的嵌入：

1. **方法1**：聚合（加总或求平均）节点的嵌入

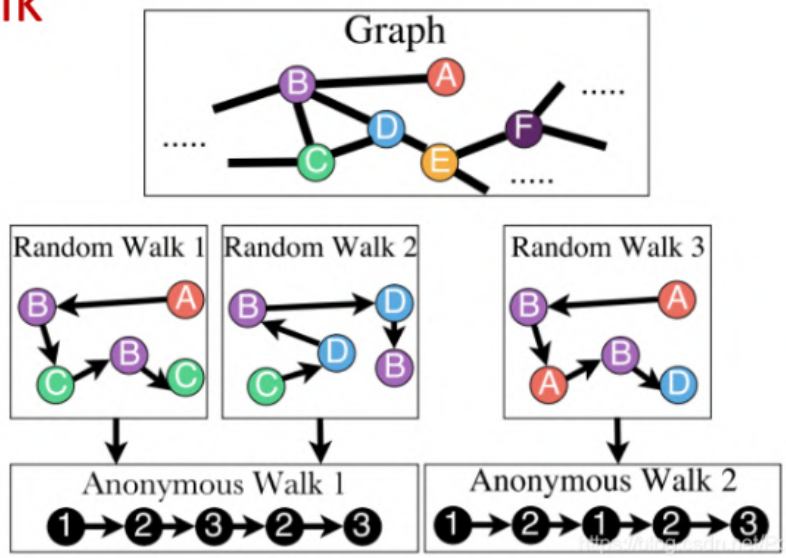
$$\mathbf{z}_G = \sum_{v \in G} \mathbf{z}_v$$

2. **方法2**：创造一个假节点（virtual node），用这个节点嵌入来作为图嵌入。这个virtual node和它想嵌入的节点子集（比如全图）相连。



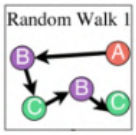
3. **方法3**： anonymous walk embeddings 以节点第一次出现的序号（是第几个出现的节点）作为索引。

States in **anonymous walks** correspond to the index of the **first time** we visited the node in a random walk



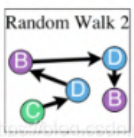
这种做法会使具体哪些节点被游走到这件事不可知（因此匿名）。

- Example RW1:

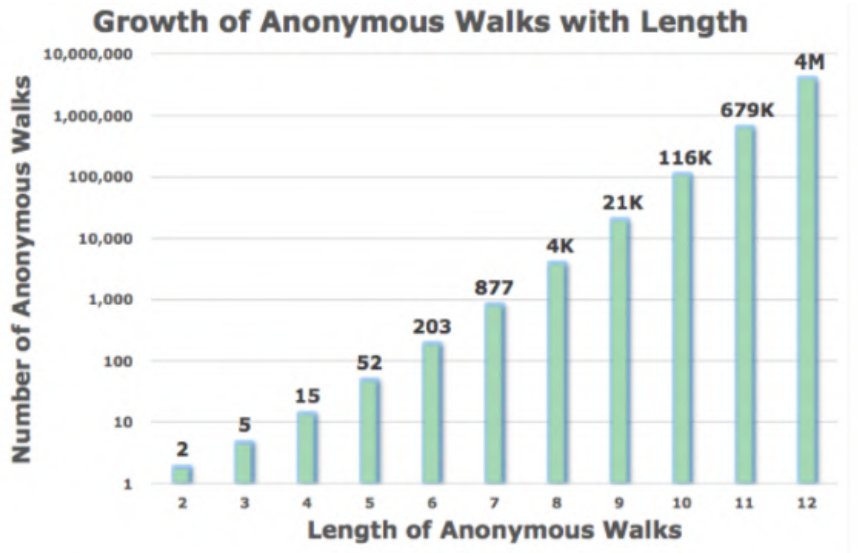


- Step **1**: node A ⟶ node 1
- Step **2**: node B ⟶ node 2 (different from node 1)
- Step **3**: node C ⟶ node 3 (different from node 1, 2)
- Step **4**: node B ⟶ node 2 (same as the node in step 2)
- Step **5**: node C ⟶ node 3 (same as the node in step 3)

- Note: RW2 gives the same anonymous walk



anonymous walks的个数随walk长度指数级增长：



Number of anonymous walks grows exponentially:

- There are 5 anon. walks w_i of length 3:
  w_1=111, w_2=112, w_3= 121, w_4= 122, w_5= 123

anonymous walks的应用：

- **模拟图上长为 l 的匿名随机游走**：将图表示为walks的概率分布向量（我感觉有点bag-of-anonymous walks、有点像GDV那些东西，总之都是向量每个元素代表其对应object的出现概率/次数）

  - Simulate anonymous walks w_i of l steps and record their counts
  - **Represent the graph as a probability distribution over these walks**
  - **For example:**
    - Set $l = 3$
    - Then we can represent the graph as a 5-dim vector
      - Since there are 5 anonymous walks w_i of length 3: 111, 112, 121, 122, 123
    - $\boldsymbol{Z_G}[i]$ = probability of anonymous walk w_i in G

  sampling anonymous walks：

- **Sampling anonymous walks:** Generate independently a set of m random walks
- Represent the graph as a probability distribution over these walks

- How many random walks m do we need?
  - We want the distribution to have error of more than ε with prob. less than δ:

$$m = \left\lceil \frac{2}{\varepsilon^2} (\log(2^\eta - 2) - \log(\delta)) \right\rceil$$

where: η is the total number of anon. walks of length l.

For example:
There are $\eta = 877$ anonymous walks of length $l = 7$. If we set $\varepsilon = 0.1$ and $\delta = 0.01$ then we need to generate m=122,500 random walks

- **行走嵌入（walk embeddings）**：$\mathbf{Z}_G$ 和匿名游走嵌入 $\mathbf{z}_i$ 。得到 $\mathbf{Z}_G$ 后可用于预测任务，可以视其内积为核，照第二章所介绍的核方法来进行机器学习；也可以照后续课程将介绍的神经网络方法来进行机器学习。

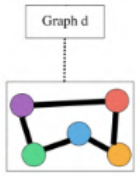
Rather than simply represent each walk by the fraction of times it occurs, we **learn embedding z_i of anonymous walk w_i**

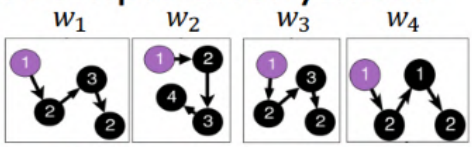
- **Learn a graph embedding Z_G together with all the anonymous walk embeddings z_i**
  $Z = \{z_i : i = 1 \dots \eta\}$, where η is the number of sampled anonymous walks.

How to embed walks?

- **Idea:** Embed walks s.t. the next walk can be predicted

- A vector parameter z_G for input graph
  - The embedding of entire graph to be learned
- Starting from **node 1**: Sample anonymous random walks, e.g.





- **Learn to predict walks that co-occur in Δ-size window** (e.g. predict w_2 given w_1, w_3 if $\Delta = 1$)
- Objective:

$$\max \sum_{t=\Delta}^{T-\Delta} \log P(w_t | w_{t-\Delta}, \dots, w_{t+\Delta}, z_G)$$

- Sum the objective over all nodes in the graph
- **Run T different random walks from u each of length l:**

$$N_R(u) = \{w_1^u, w_2^u \dots w_T^u\}$$

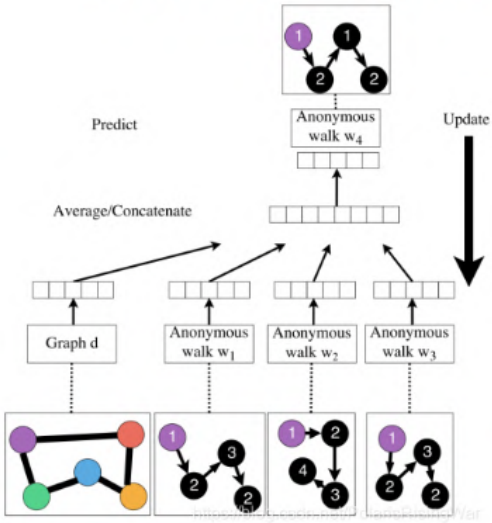
- **Learn to predict walks that co-occur in Δ-size window**
- Estimate embedding z_i of anonymous walk w_i
  Let η be number of all possible walk embeddings

Objective: $\max_{z,d} \frac{1}{T} \sum_{t=\Delta}^{T-\Delta} \log P(w_t | \{w_{t-\Delta}, \dots, w_{t+\Delta}, z_G\})$

- $P(w_t | \{w_{t-\Delta}, \dots, w_{t+\Delta}, z_G\}) = \frac{\exp(y(w_t))}{\sum_{i=1}^{\eta} \exp(y(w_i))}$ ← All possible walks (require negative sampling)
- $y(w_t) = b + U \cdot \left(cat(\frac{1}{2\Delta} \sum_{i=-\Delta}^{\Delta} z_i, z_G)\right)$
  - $cat(\frac{1}{2\Delta} \sum_{i=-\Delta}^{\Delta} z_i, z_G)$ means an average of anonymous walk embeddings in window, concatenated with the graph embedding z_G
  - $b \in \mathbb{R}, U \in \mathbb{R}^D$ are learnable parameters. This represents a linear layer.

- We obtain the graph embedding z_G (learnable parameter) after optimization
- Use z_G to make predictions (e.g. graph classification)
  - **Option1**: Inner product Kernel $z_{G_1}^T z_{G_2}$ (Lecture 2)
  - **Option2**: Use a neural network that takes z_G as input to classify

Overall Architecture



4 链接分析：网页排名（图转换为矩阵）

4.1 图转换为矩阵

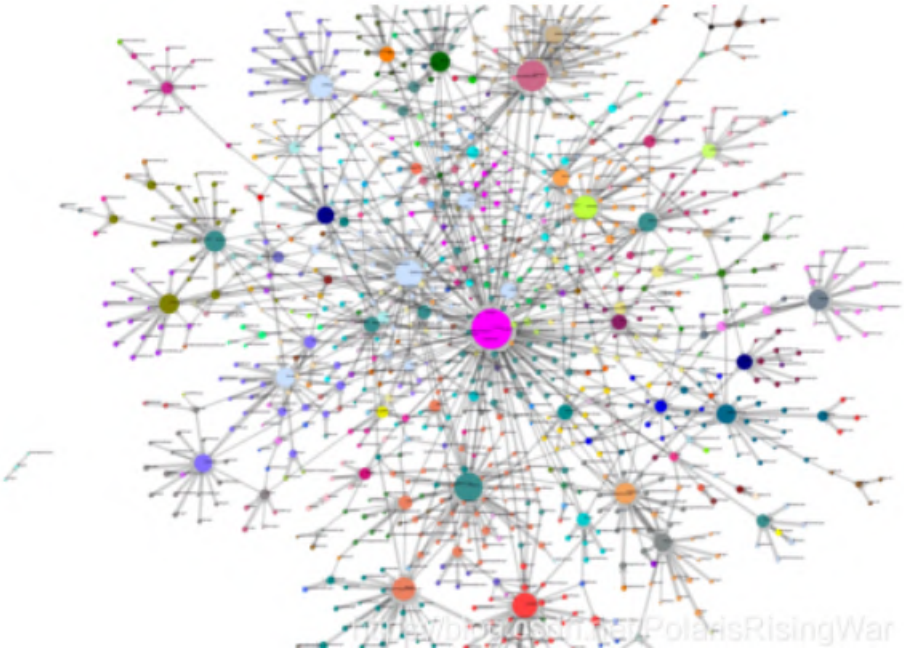
本节课研究矩阵角度的图分析和学习。这里的矩阵就是指邻接矩阵。

将图视为矩阵形式，可以通过**随机游走**的方式定义节点重要性（即PageRank），通过**矩阵分解**matrix factorization (MF)来获取**节点嵌入**，将其他节点嵌入（如node2vec）也视作MF。

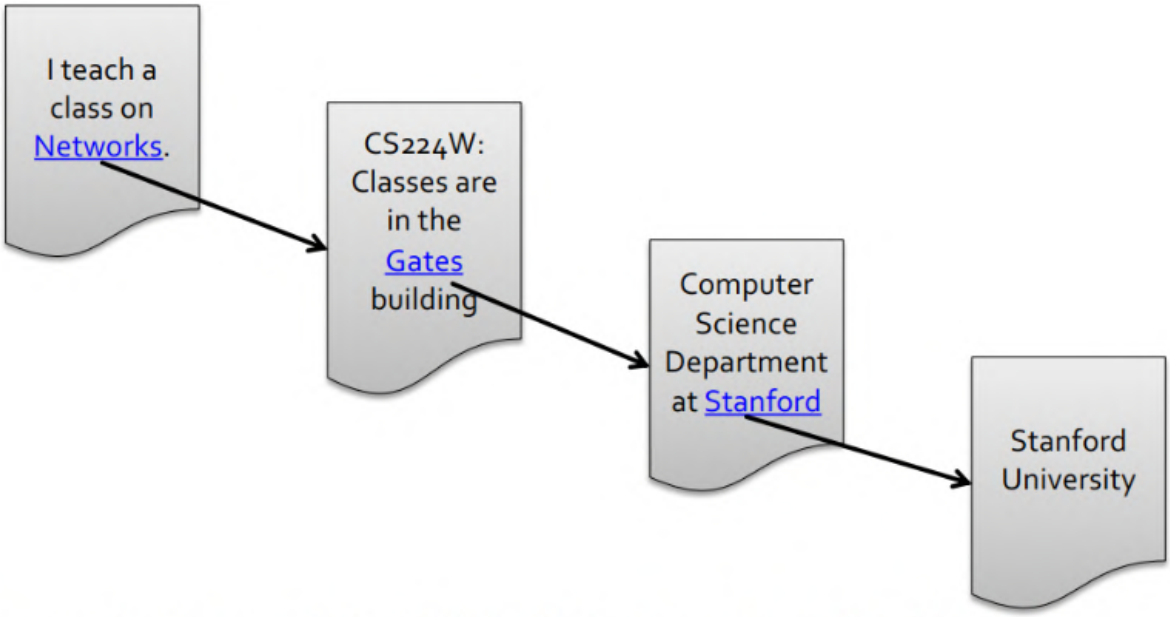
4.2 网页排名/谷歌算法

PageRank是谷歌搜索用的算法，用于对网页的重要性进行排序。在搜索引擎应用中，可以对网页重要性进行排序，从而辅助搜索引擎结果的网页排名。

在现实世界中，将整个互联网视作图：将网页视作节点，将网页间的超链接视作边



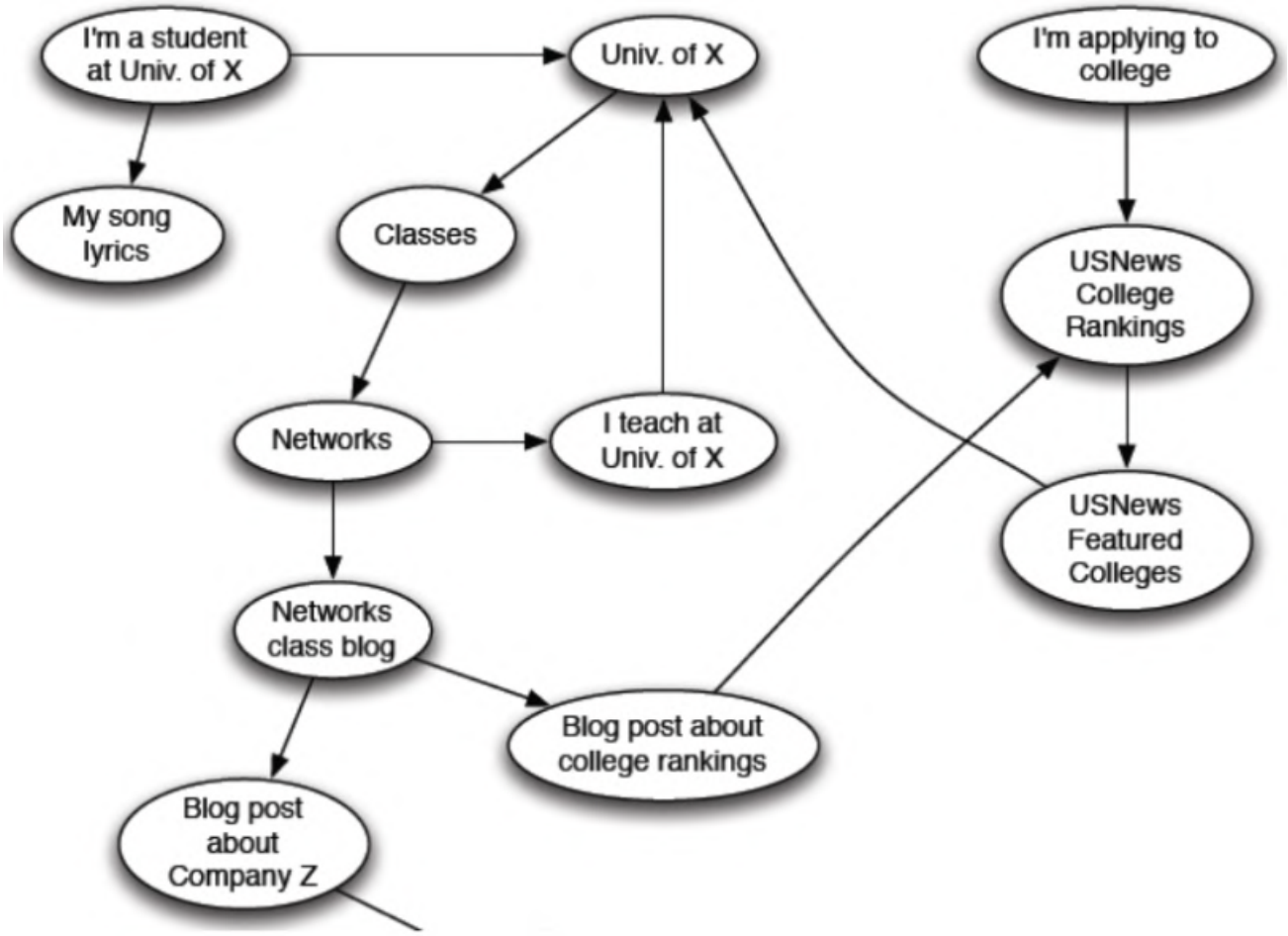
一个网页之间互相链接的情况的示例：



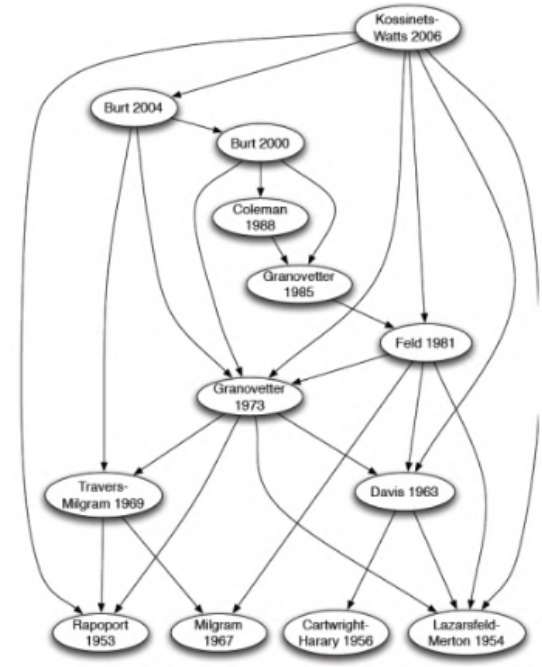
- In early days of the Web links were **navigational**
- Today many links are **transactional** (used not to navigate from page to page, but to post, comment, like, buy, …)

老一点的网页超链接都是navigational纯导航到其他页面的，当代的很多链接则是transactional用于执行发布、评论、点赞、购买等功能事务的。本课程中主要仅考虑那种网页之间互相链接的情况。

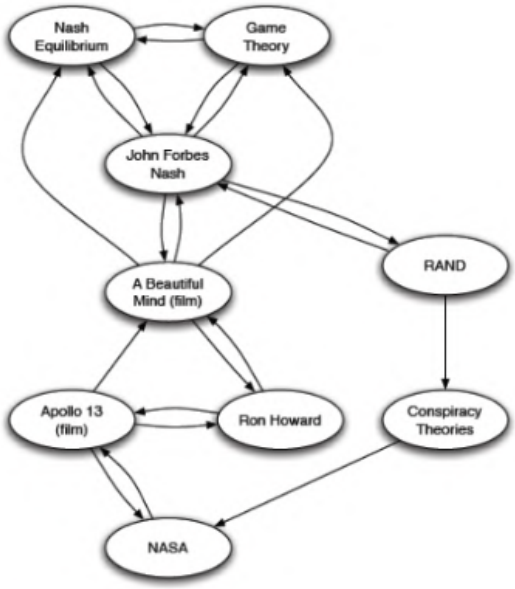
将网页看作有向图，以链接指向作为边的方向（这个网页/节点能直接跳转到的网页就作为其下一个节点successor）：



其他可表现为有向图形式的信息网络示例：论文引用，百科全书中词条间的互相引用：



Citations



References in an Encyclopedia

在图中，我们想要定义节点的**重要性**importance，通过网络图链接结构来为网页按重要性分级rank。以下将介绍3种用以计算图中节点重要性的方法：

- **PageRank**
- **Personalized PageRank (PPR)**
- **Random Walk with Restarts (RWR)**

衡量节点重要性：认为一个节点的链接越多，那么这个节点越重要。

有向图有in-coming links和out-going links两种情况。可以想象，in-links比较不容易造假，比较靠谱，所以用in-links来衡量一个节点的重要性。可以认为一个网页链接到下一网页，相当于对该网页重要性投了票（vote）。所以我们认为一个节点的in-links越多，那么这个节点越重要。同时，我们认为来自更重要节点的in-links，在比较重要性时的权重vote更大。

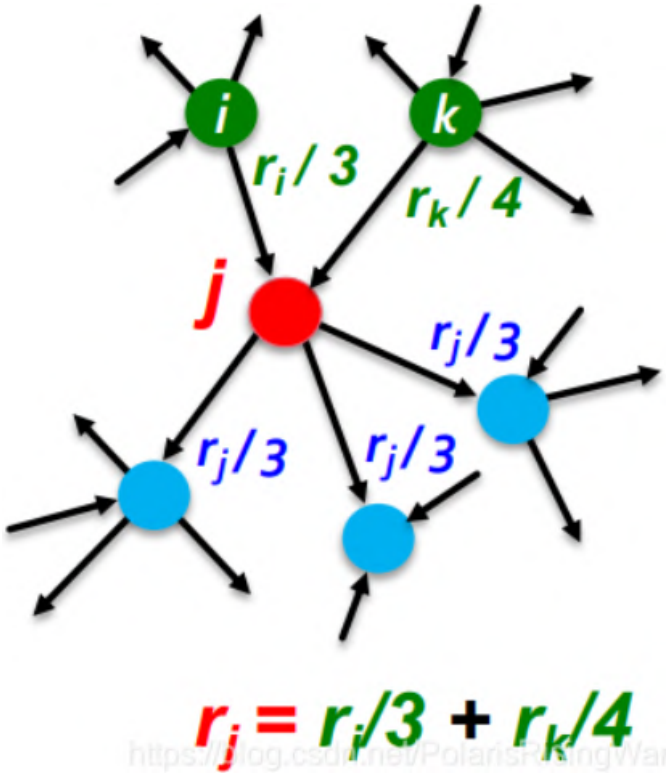
这就成了一个递归recursive的问题——要计算一个节点的重要性就要先计算其predecessors的重要性，计算这些predecessors的重要性又要先计算它们predecessors的重要性……

4.2.1 网页排名：流模型

链接权重与其source page的重要性成正比例。

如果网页 i 的重要性是 r_i ，有 d_i 个out-links，那么每个边的权重就是 r_i/d_i 。

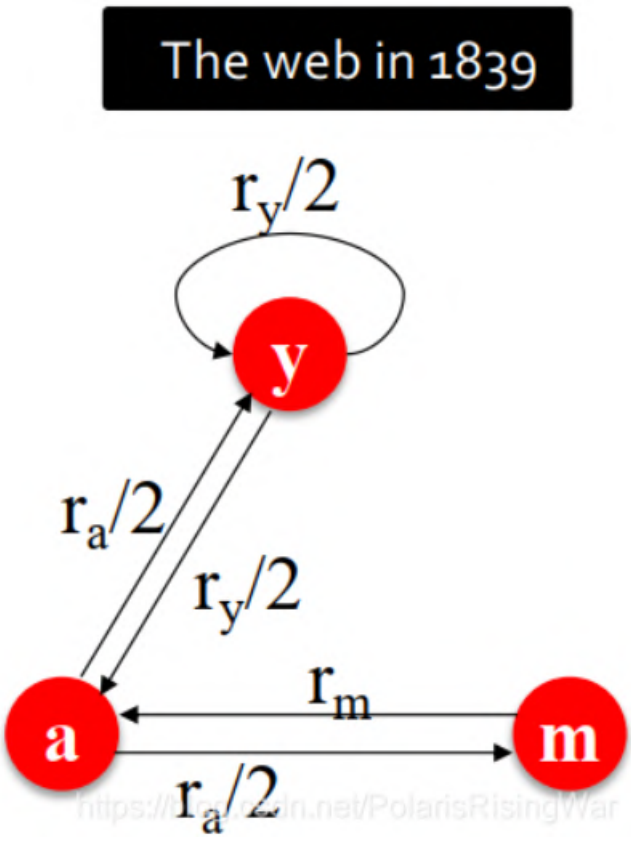
网页 j 的重要性 r_j 是其 in-links 上权重的总和。



从而得到对节点 j 的级别 r_j 的定义：

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}, \text{其中} d_i \text{是节点} i \text{的出度}$$

以这个1839年的网络为例：



我们就可以得到这样的"flow" equations:

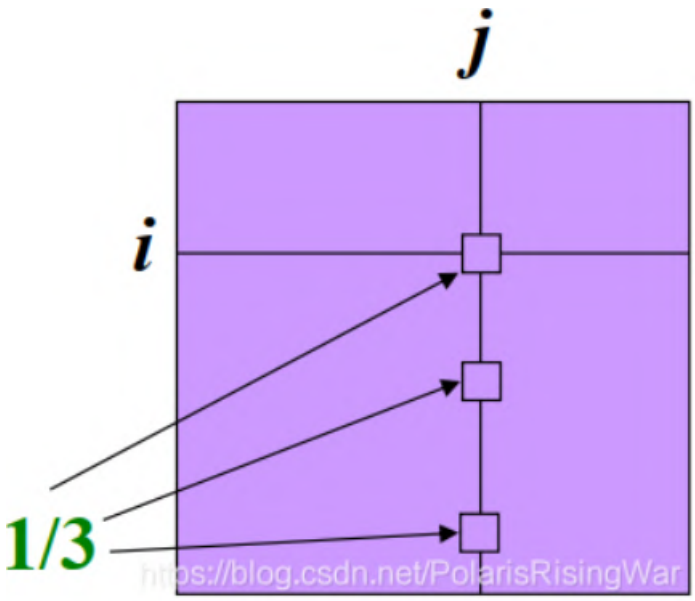
$$\begin{aligned}\mathrm{r_y} &= \mathrm{r_y}/2 + \mathrm{r_a}/2 \\ \mathrm{r_a} &= \mathrm{r_y}/2 + \mathrm{r_m} \\ \mathrm{r_m} &= \mathrm{r_a}/2\end{aligned}$$

在直觉上我们好像可以用高斯消元法Gaussian elimination来解这个线性方程组，但这种方法不scalable。所以我们寻找更scalable的矩阵形式解法。

4.2.2 网页排名：矩阵表达

建立**随机邻接矩阵（stochastic adjacency matrix）** M，网页 j 有 d_j 条 out-links，如果 $j \to i$，$M_{ij} = \frac{1}{d_j}$。

M 是列随机矩阵column stochastic matrix（列和为1），M 的第 j 列可以视作 j 在邻居节点上的概率分布。

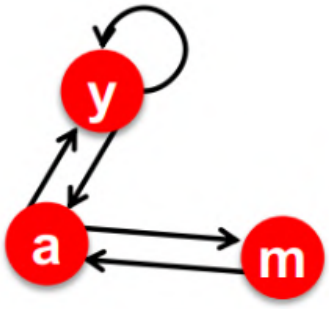


rank vector r：每个网页 i 的重要性得分 r_i，$\sum_i r_i = 1$，所以 r 也可被视为是网络中所有节点的概率分布。

flow equations可以被写成：$\mathbf{r} = M \cdot \mathbf{r}$。

回忆一下原公式 $r_j = \sum_{i \to j} \frac{r_i}{d_i}$，$M$ 的第 j 行被指向的节点对应的列 i 的元素就是 $\frac{1}{d_i}$，该列对应的是 r_i，加总起来就得到上个公式。

flow equation和矩阵对比的举例：



	r_y	r_a	r_m
r_y	½	½	0
r_a	½	0	1
r_m	0	½	0

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix}$$

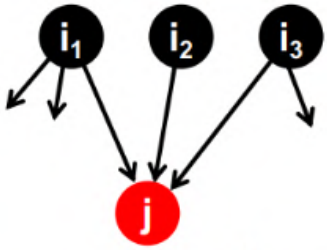
r M r

4.2.3 连接到随机行走

假想一个web surfer的随机游走过程，在 t 时间在网页 i 上，在 $t+1$ 时间从 i 的out-links中随机选一条游走。如此重复过程。

向量 $\mathbf{p}(t)$ 的第 i 个坐标是 t 时间web surfer在网页 i 的概率，因此向量 $\mathbf{p}(t)$ 是网页间的概率分布向量。

- **Imagine a random web surfer:**
  - At any time t, surfer is on some page i
  - At time $t+1$, the surfer follows an out-link from i uniformly at random
  - Ends up on some page j linked from i
  - Process repeats indefinitely
- **Let:**
  - $p(t)$ ... vector whose i^{th} coordinate is the prob. that the surfer is at page i at time t
  - So, $p(t)$ is a probability distribution over pages



$$r_j = \sum_{i \to j} \frac{r_i}{\mathrm{d}_{\mathrm{out}}(\mathrm{i})}$$

平稳分布（stationary distribution）：

$$\mathbf{p}(t+1) = M \cdot \mathbf{p}(t)$$

M 是web surfer的转移概率，这个公式的逻辑感觉和 $\mathbf{r} = M \cdot \mathbf{r}$ 其实类似。

如果达到这种条件，即下一时刻的概率分布向量与上一时刻的相同：

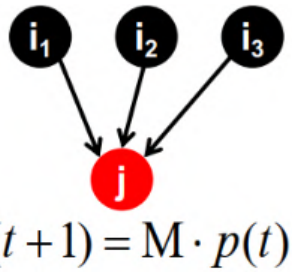
$$\mathbf{p}(t+1) = M \cdot \mathbf{p}(t) = \mathbf{p}(t)$$

则 $\mathbf{p}(t)$ 是随机游走的stationary distribution。

$\mathbf{r} = M \cdot \mathbf{r}$，所以 $\mathbf{r}$ 是随机游走的stationary distribution。

- **Where is the surfer at time *t+1*?**
  - Follow a link uniformly at random

    $\boldsymbol{p(t+1) = M \cdot p(t)}$



- Suppose the random walk reaches a state

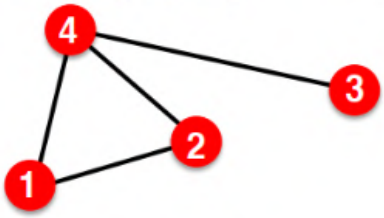
  $\boldsymbol{p(t+1) = M \cdot p(t) = p(t)}$

  then $\boldsymbol{p(t)}$ is **stationary distribution** of a random walk
- **Our original rank vector $\boldsymbol{r}$** satisfies $\boldsymbol{r = M \cdot r}$
  - **So, $\boldsymbol{r}$ is a stationary distribution for the random walk**

4.2.4 特征向量表达

无向图的邻接矩阵的特征向量是节点特征eigenvector centrality，而PageRank定义在有向图的随机邻接矩阵上。

- Recall from lecture 2 (eigenvector centrality), let $\boldsymbol{A \in \{0,1\}^{n \times n}}$ be an adj. matrix of undir. graph:



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

- Eigenvector of adjacency matrix: vectors satisfying $\boldsymbol{\lambda c = Ac}$
- $\boldsymbol{c}$: eigenvector; $\boldsymbol{\lambda}$: eigenvalue

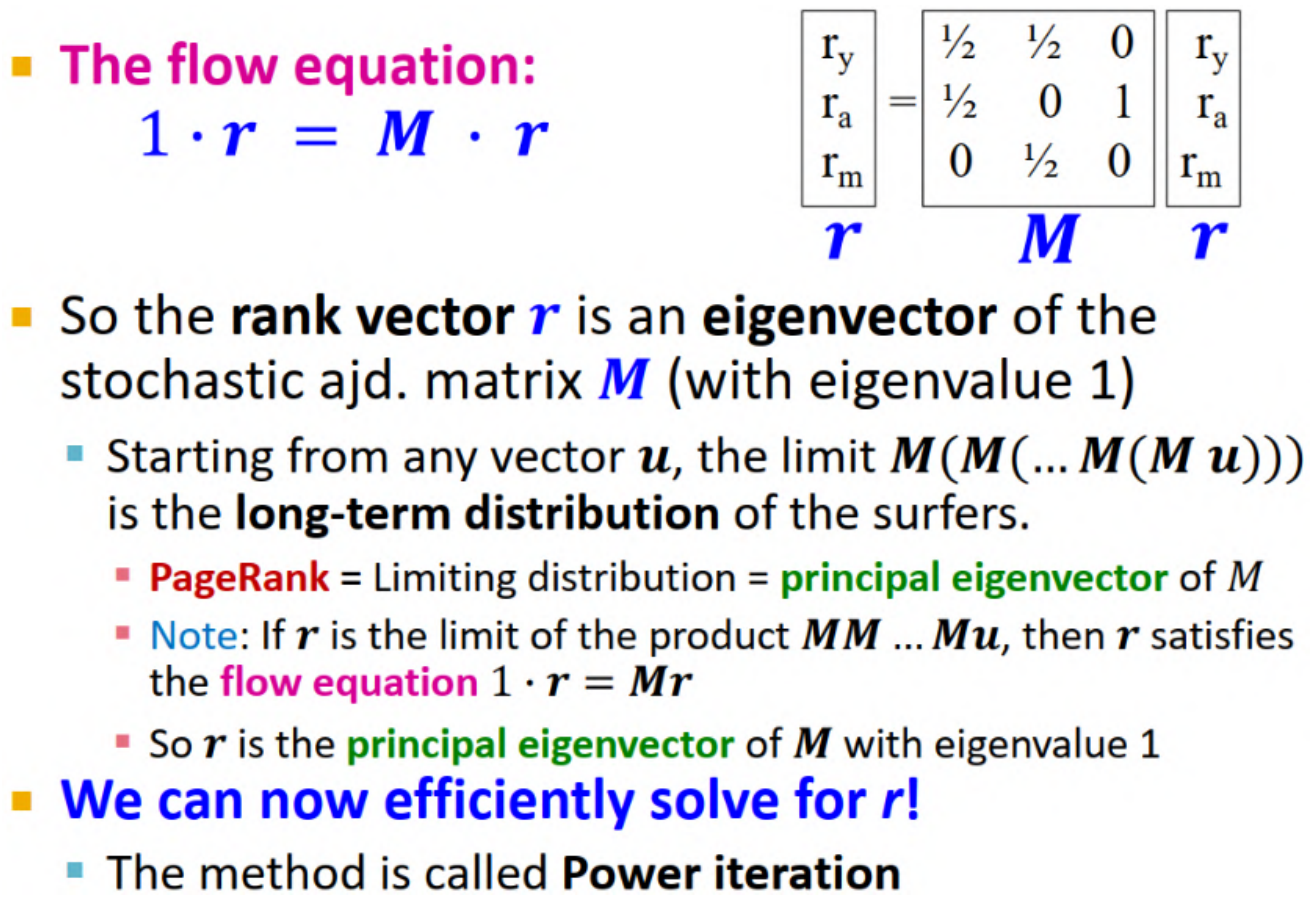
- Note:
  - This is the definition of eigenvector centrality (for undirected graphs).
  - PageRank is defined for directed graphs

$$1 \cdot \mathbf{r} = M \cdot \mathbf{r}$$

rank vector $\mathbf{r}$ 是随机邻接矩阵 M 的特征向量，对应特征值为1。

从任一向量 $\mathbf{u}$ 开始，极限 $M(M(\dots M(M\mathbf{u})))$ 是web surfer的长期分布，也就是 $\mathbf{r}$（意思是无论怎么开局，最后结果都一样，这个感觉可以比较直觉地证明），PageRank=极限分布=M的principal eigenvector。

根据这个思路，我们就能找到PageRank的求解方式：power iteration



极限分布（limiting distribution）：相当于是random surfer一直随机游走，直至收敛，到达稳定状态。这个 M 的叠乘可以让人联想到Katz index叠乘邻接矩阵 A。相比高斯消元法，power iteration是一种scalable的求解PageRank方法。

4.2.5 网页排名总结

1. 通过网络链接结构衡量图中节点的重要性。
2. 用随机邻接矩阵M建立web surfer随机游走模型。
3. PageRank解方程：$\mathbf{r} = M \cdot \mathbf{r}$，$\mathbf{r}$ 可被视作 M 的principle eigenvector，也可被视作图中随机游走的stationary distribution。

4.3 网页排名：如何求解?

对每个节点赋初始PageRank。重复计算每个节点的PageRank $r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$（$d_i$ 是节点 i 的出度）直至收敛 $\sum_i |r_i^{(t+1)} - r_i^t| < \epsilon$。

Given a graph with *n* nodes, we use an iterative procedure:

- Assign each node an initial page rank
- Repeat until convergence ($\sum_i \left| r_i^{t+1} - r_i^t \right| < \epsilon$)
  - Calculate the page rank of each node

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

d_i .... out-degree of node i

幂迭代法（power iteration method）：初始化 $r^0 = [1/N, \ldots, 1/N]^T$，迭代 $\mathbf{r}^{(t+1)} = M \cdot \mathbf{r}^t$ 直到 $|\mathbf{t}^{(t+1)} - \mathbf{t}^{(t)}| < \epsilon$，约50次迭代即可逼近极限。

- **Given a web graph with *N* nodes, where the nodes are pages and edges are hyperlinks**
- **Power iteration:** a simple iterative scheme
  - Initialize: $\boldsymbol{r}^0 = [1/N, \ldots, 1/N]^T$
  - Iterate: $\boldsymbol{r}^{(t+1)} = \boldsymbol{M} \cdot \boldsymbol{r}^t$
  - Stop when $|\boldsymbol{r}^{(t+1)} - \boldsymbol{r}^t|_1 < \varepsilon$

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

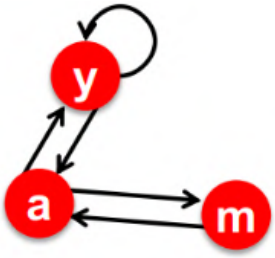
d_i .... out-degree of node i

$|x|_1 = \sum_1^N |x_1|$ is the L_1 norm
Can use any other vector norm, e.g., Euclidean

About 50 iterations is sufficient to estimate the limiting solution.

power iteration示例:

- **Power Iteration:**
  - Set $r_j \leftarrow 1/N$
  - **1:** $r'_j \leftarrow \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** If $|r - r'| > \varepsilon$:
    - $r \leftarrow r'$
  - **3:** go to **1**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + rm$$
$$r_m = r_a/2$$

- **Example:**

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} \begin{bmatrix} 1/3 \\ 3/6 \\ 1/6 \end{bmatrix} \begin{bmatrix} 5/12 \\ 1/3 \\ 3/12 \end{bmatrix} \begin{bmatrix} 9/24 \\ 11/24 \\ 1/6 \end{bmatrix} \dots \begin{bmatrix} 6/15 \\ 6/15 \\ 3/15 \end{bmatrix}$$

Iteration 0, 1, 2, …

PageRank的问题及其解决方案:

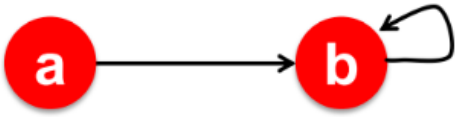
$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i} \quad \text{or equivalently} \quad r = Mr$$

- **Does this converge?**
- **Does it converge to what we want?**
- **Are results reasonable?**

Two problems:

- **(1)** Some pages are **dead ends** (have no out-links)
  - Such pages cause importance to "leak out"
- **(2) Spider traps** (all out-links are within the group)
  - Eventually spider traps absorb all importance

- **spider trap**：所有出边都在一个节点组内，会吸收所有重要性。
  - **The "Spider trap" problem:**



$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

  - **Example:**

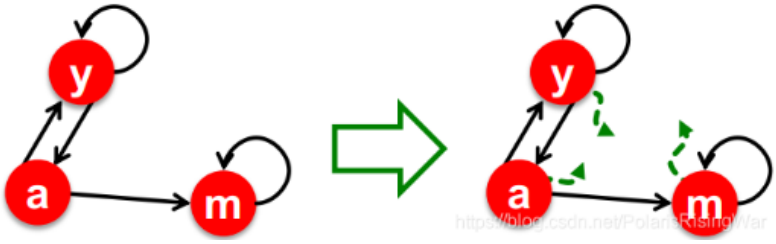
Iteration:	0,	1,	2,	3...
r_a	1	0	0	0
r_b	0	1	1	1

spider traps解决方案：random jumps or teleports（teleport（通常见于科幻作品）（被）远距离传送，大概我就翻译成直接跳了）

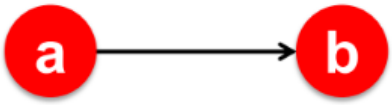
random surfer每一步以概率 β 随机选择一条链接（用 M ），以概率 $1-\beta$ 随机跳到一个网页上（ β 一般在0.8-0.9之间）。

这样surfer就会在几步后跳出spider trap。

- **Solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. **1-β,** jump to a random page
  - Common values for β are in the range 0.8 to 0.9
- **Surfer will teleport out of spider trap within a few time steps**



- **dead end**：没有出边，造成重要性泄露。
  - **The "Dead end" problem:**



$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{d_i}$$

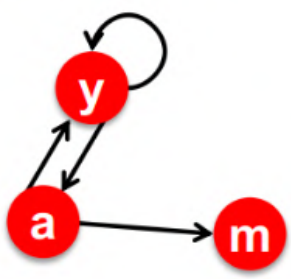
  - **Example:**

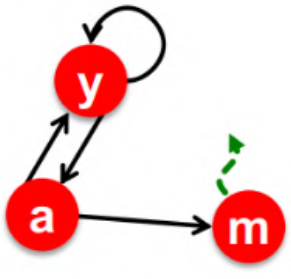
Iteration:	0,	1,	2,	3...
r_a =	1	0	0	0
r_b =	0	1	0	0

dead ends解决方案：random jumps or teleports。

从dead-ends出发的web surfer随机跳到任意网页（相当于从dead end出发，向所有节点连了一条边）

- **Teleports:** Follow random teleport links with total probability **1.0** from dead-ends
  - Adjust matrix accordingly





	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

	y	a	m
y	½	½	⅓
a	½	0	⅓
m	0	½	⅓

spider-traps在数学上不是个问题，但是无法得到我们想要的PageRank结果；因此要在有限步内跳出spider traps。

dead-ends在数学上就是问题（其随机邻接矩阵列和不为0，初始假设直接不成立），因此要直接调整随机邻接矩阵，让web surfer无路可走时可以直接teleport。

Why are dead-ends and spider traps a problem and why do teleports solve the problem?

- **Spider-traps** are not a problem, but with traps PageRank scores are **not** what we want
  - **Solution:** Never get stuck in a spider trap by teleporting out of it in a finite number of steps
- **Dead-ends** are a problem
  - The matrix is not column stochastic so our initial assumptions are not met
  - **Solution:** Make matrix column stochastic by always teleporting when there is nowhere else to go

整体解决方案：random teleport

random surfer每一步以概率 β 随机选择一条链接（M），以概率 $1-\beta$ 随机跳到一个网页上。

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{N}, d_i是节点i的出度$$

M 需要没有dead ends，可以通过直接去除所有dead ends或显式将dead ends跳到随机节点的概率设置到总和为1。

- **Google's solution that does it all:**
  At each step, random surfer has two options:
  - With probability β, follow a link at random
  - With probability $1-\beta$, jump to some random page
- **PageRank equation** [Brin-Page, 98]

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

d_i … out-degree of node i

This formulation assumes that M has no dead ends. We can either preprocess matrix M to remove all dead ends or explicitly follow random teleport links with probability 1.0 from dead-ends.

The Google Matrix G:

$$G = \beta M + (1-\beta)[\frac{1}{N}]_{N \times N}$$

其中 $[\frac{1}{N}]_{N \times N}$ 是每个元素都是 $\frac{1}{N}$ 的 $N \times N$ 的矩阵。

现在 $\mathbf{r} = G \cdot \mathbf{r}$ 又是一个迭代问题，power iteration方法仍然可用，β 在实践中一般用0.8或0.9（平均5步跳出一次）。

- **PageRank equation** [Brin-Page, '98]

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

- **The Google Matrix *G*:**

$[1/N]_{NxN}$...N by N matrix where all entries are 1/N

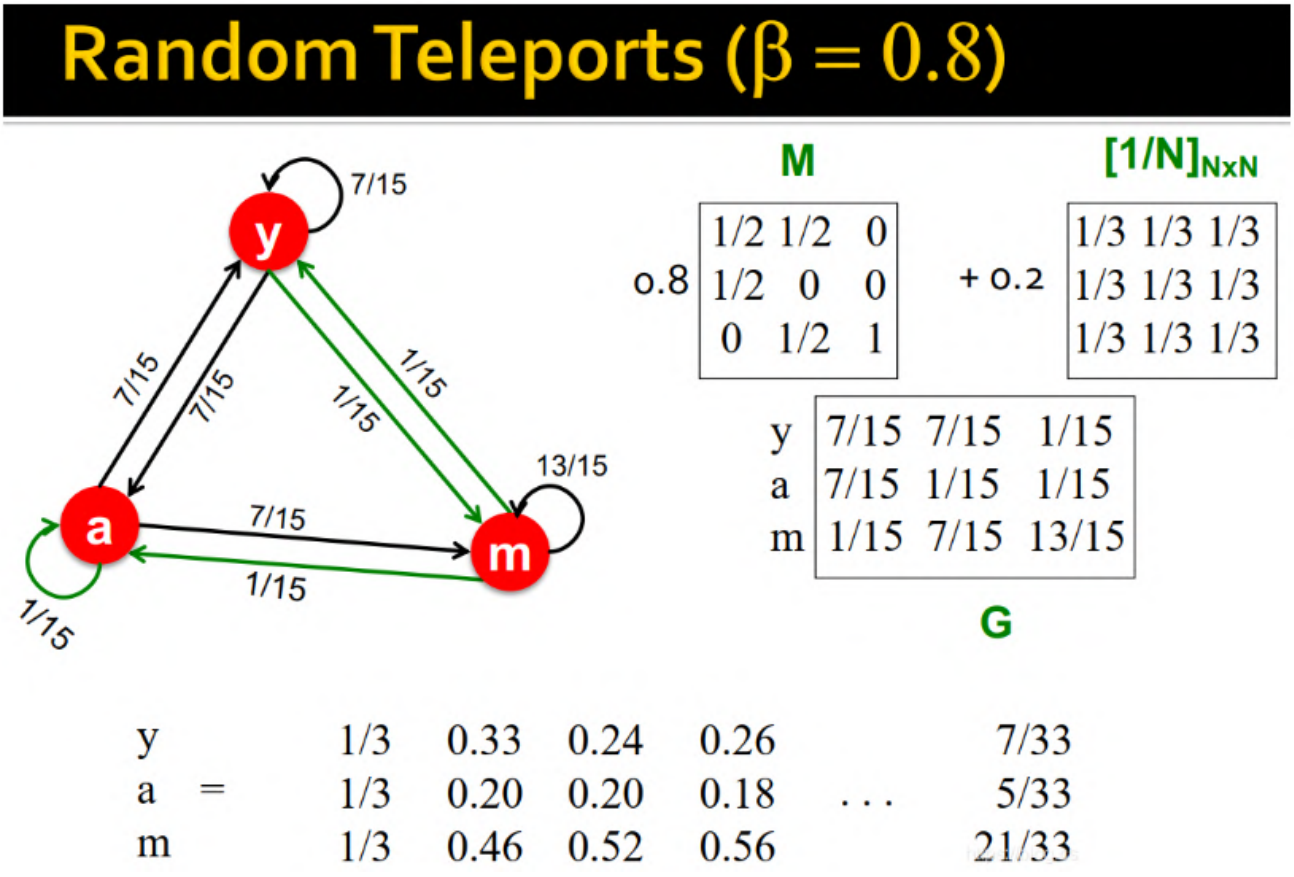
$$G = \beta M + (1-\beta)\left[\frac{1}{N}\right]_{N\times N}$$

- **We have a recursive problem: $r = G \cdot r$**

  And the Power method still works!

- **What is β?**
  - In practice $\beta = 0.8, 0.9$ (make 5 steps on avg., jump)

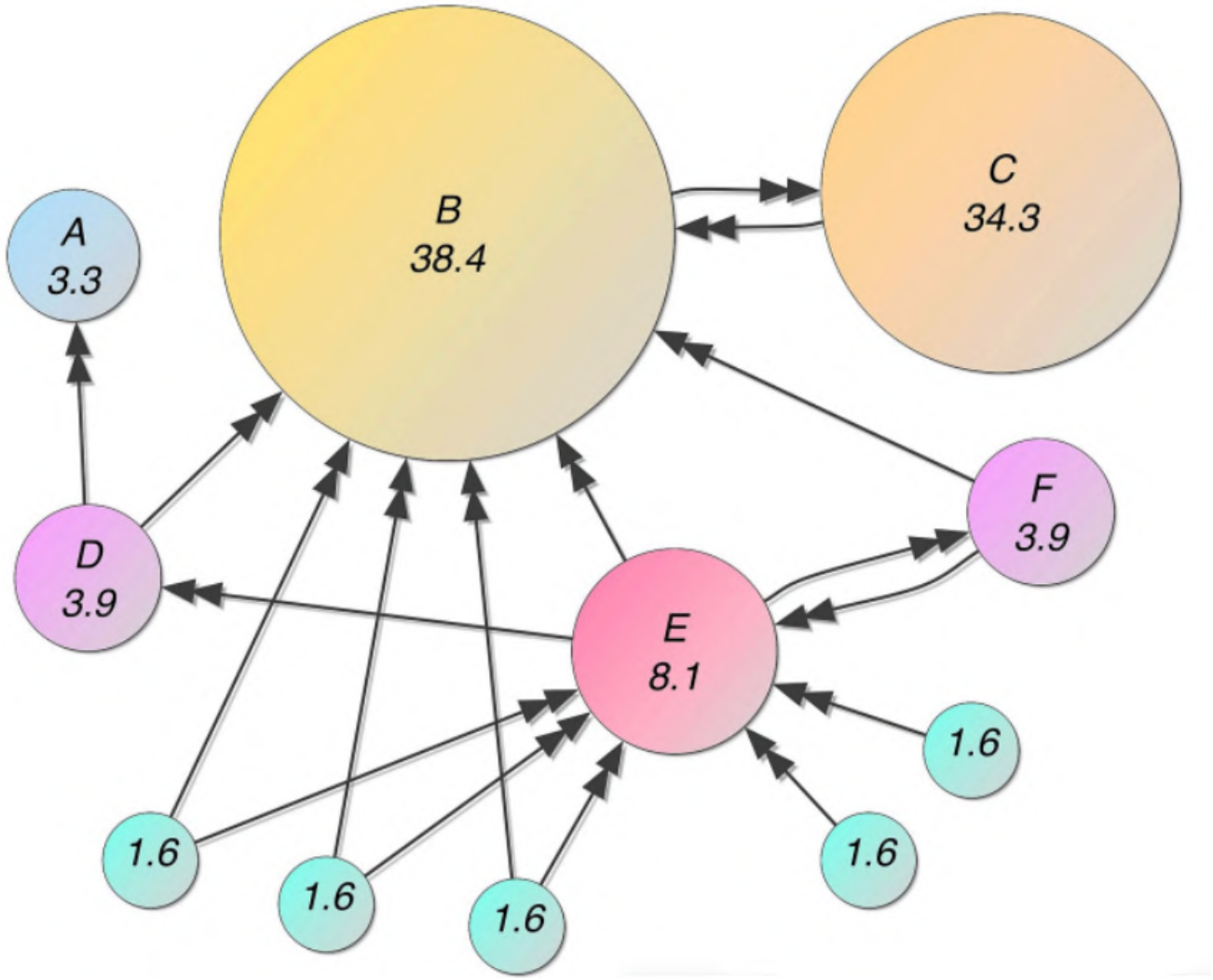
random teleports举例:



y		1/3	0.33	0.24	0.26	7/33
a	=	1/3	0.20	0.20	0.18	... 5/33
m		1/3	0.46	0.52	0.56	21/33

M 是个spider trap，所以加上random teleport links，G 也是一个转移概率矩阵。

PageRank结果示例:



PageRank求解部分总结:

用power iteration方法求解 $\mathbf{r} = G \cdot \mathbf{r}$（$G$ 是随机邻接矩阵）。

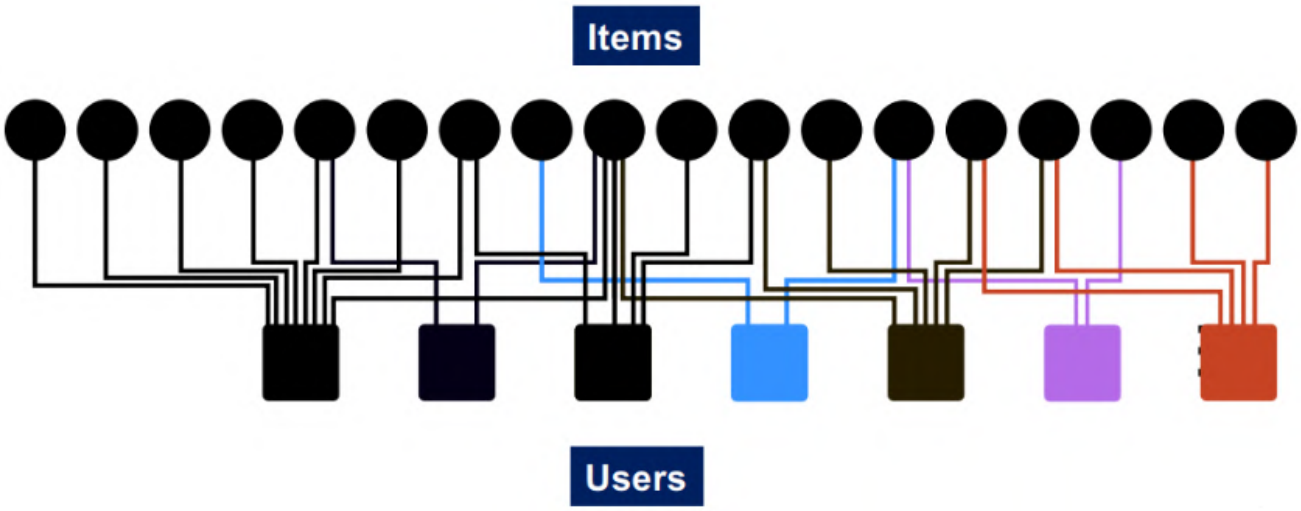
用random uniform teleporation解决dead-ends和spider-traps问题。

- PageRank solves for $\boldsymbol{r} = \boldsymbol{Gr}$ and can be efficiently computed by power iteration of the stochastic adjacency matrix ($\boldsymbol{G}$)
- Adding random uniform teleportation solves issues of dead-ends and spider-traps

4.4 重启动随机行走&个性化网页排名

举例：推荐问题（一个由user和item两种节点组成的bipartite graph）

- **Given:**
  A bipartite graph representing user and item interactions (e.g. purchase)

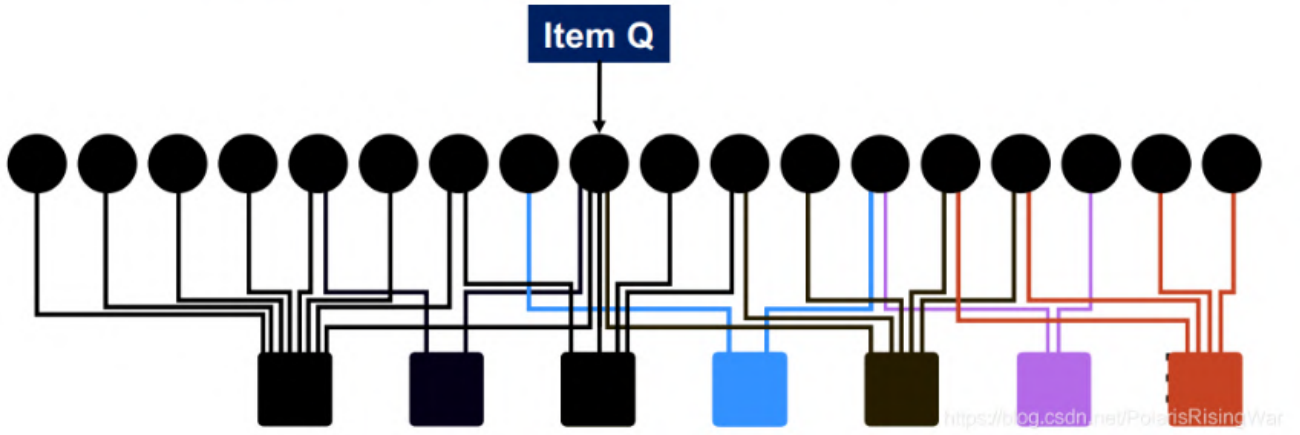


Bipartite User-Item Graph:

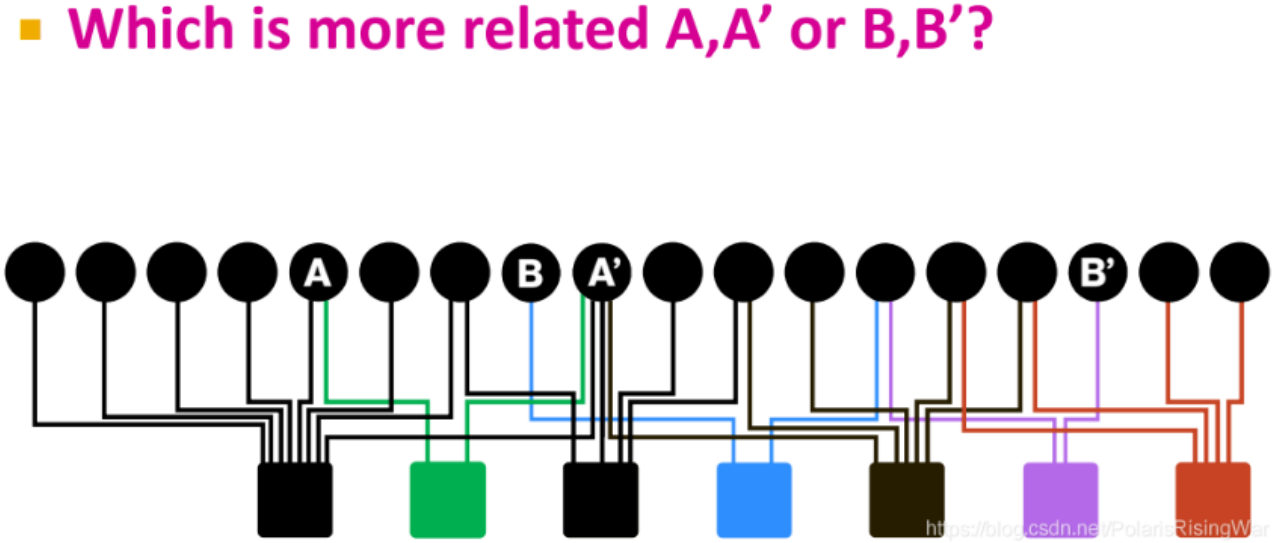
求解目标：图节点间相似性（针对与item Q交互的user，应该给他推荐什么别的item？）

可以直觉地想到，如果item Q和P都与相似user进行过交互，我们就应该推荐Q。

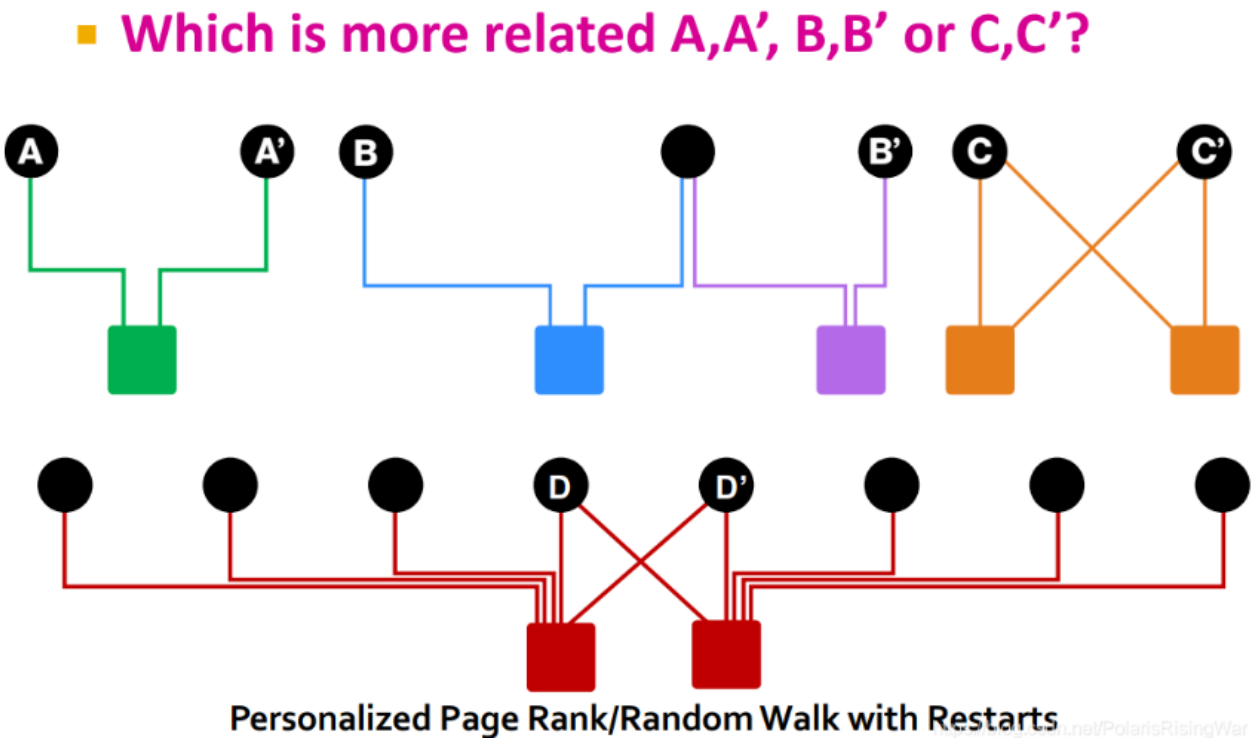
- **Goal:** Proximity on graphs
  - **What items should we recommend to a user who interacts with item Q?**
  - **Intuition**: if items Q and P are interacted by similar users, recommend P when user interacts with Q



但是我们如何量化这个相似性呢?



衡量节点相似性的问题：



A A'比B B'近可以因为它们之间距离较短；但是A A'和C C'距离相等，C C'却有着共同用户，这又如何比较呢？如果引入shared neighbors作为标准，D D'和C C'有相同的share neighbors，但是D D'的共同用户之间的相似性却很低，这又如何衡量呢？

图上的相似性：Random Walks with Restarts

- PageRank用重要性来给节点评级，随机跳到网络中的任何一个节点。
- Personalized PageRank衡量节点与teleport nodes $\mathbf{S}$ 中的节点的相似性。
- 用于衡量其他item与item Q的相似性：Random Walks with Restarts只能跳到起始节点：$\mathbf{S}=\{\mathbf{Q}\}$ 。

- **PageRank:**
  - Ranks nodes by “importance”
  - Teleports with uniform probability to any node in the network
- **Personalized PageRank:**
  - Ranks proximity of nodes to the teleport nodes S
- **Proximity on graphs:**
  - Q: What is most related item to **Item Q**?
  - **Random Walks with Restarts**
    - Teleport back to the starting node: $S = \{Q\}$

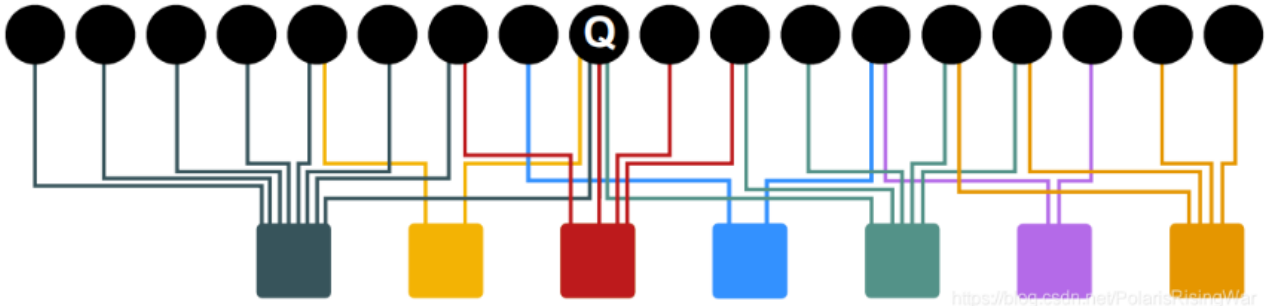
Random Walks：每个节点都有重要性，在其边上均匀分布，传播到邻居节点。对 query_nodes 模拟随机游走：

1. 随机游走到一个邻居，记录走到这个节点的次数（visit count）
2. 以 alpha 概率从 query_nodes 中某点重启
3. 结束随机游走后，visit count最高的节点就与 query_nodes 具体最高的相似性（直觉上这就是 query_nodes 最容易走到、最近的点了）

- **Idea**
  - Every node has some importance
  - Importance gets evenly split among all edges and pushed to the neighbors:
- **Given a set of QUERY_NODES, we simulate a random walk:**
  - Make a step to a random neighbor and record the visit (visit count)
  - With probability ALPHA, restart the walk at one of the QUERY_NODES
  - The nodes with the highest visit count have highest proximity to the QUERY_NODES

以 Q 作为示例：

- Given a set of **QUERY NODES Q**, simulate a random walk:



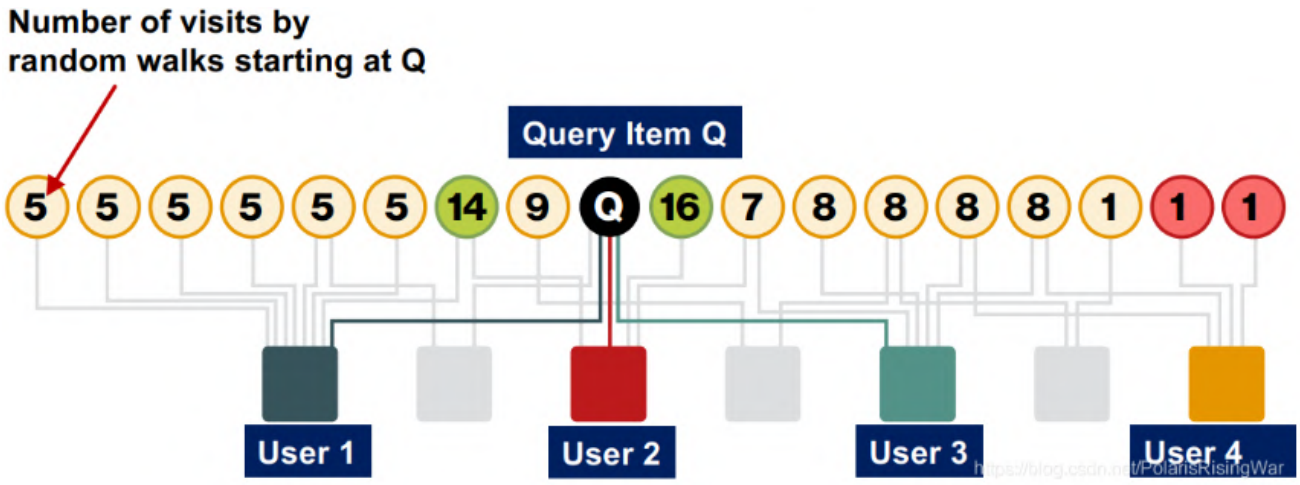
算法伪代码（从item随机游走到另一个item，记录visit_count；以一定概率返回query_nodes）：

- Proximity to query node(s) ***Q***:

```
ALPHA = 0.5
QUERY_NODES = { Q }

item = QUERY_NODES.sample_by_weight( )
for i in range( N_STEPS ):
    user = item.get_random_neighbor( )
    item = user.get_random_neighbor( )
    item.visit_count += 1
    if random( ) < ALPHA:
        item = QUERY_NODES.sample.by_weight ( )
```

结果示例：



在示例中是模拟的random walk，但其实也可以用power iteration的方式来做。

RWR的优点在于，这种相似性度量方式考虑到了网络的如下复杂情况：

1. multiple connections
2. multiple paths
3. direct and indirect connections
4. degree of the node

对不同PageRank变体的总结：

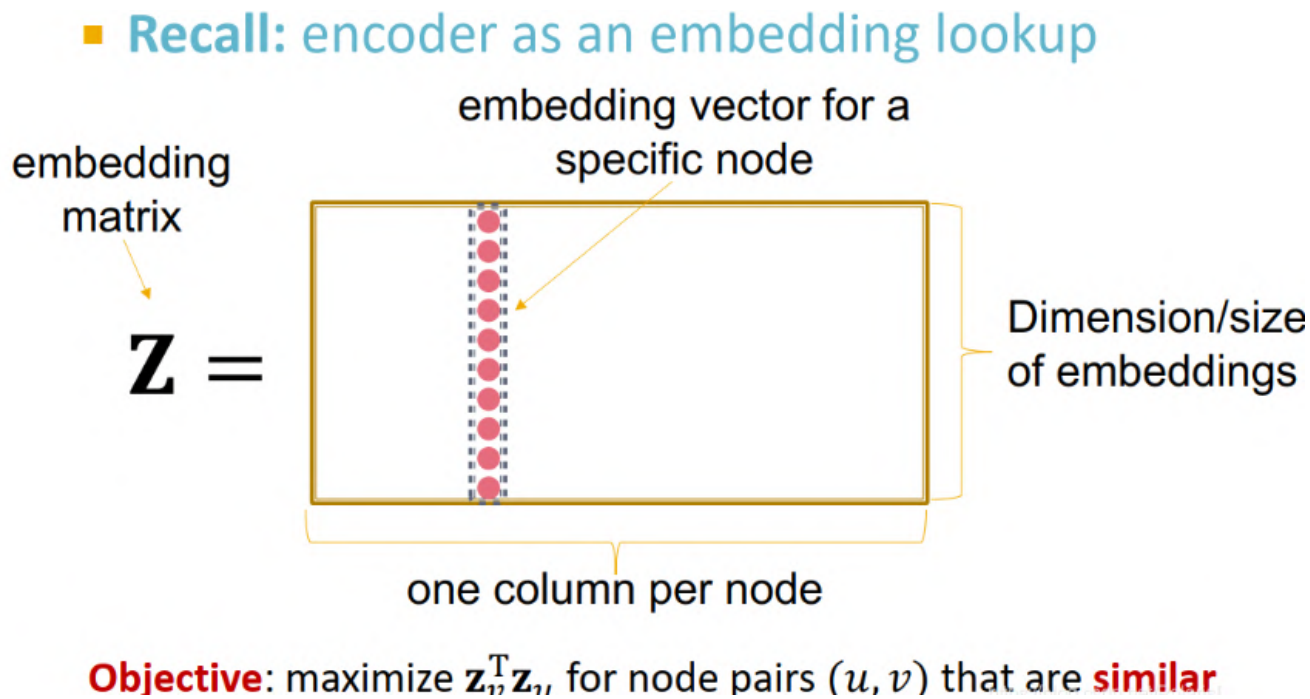
- **PageRank:**
  - Teleports to any node
  - Nodes can have the same probability of the surfer landing:
    $$\boldsymbol{S} = [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1]$$
- **Topic-Specific PageRank** aka **Personalized PageRank:**
  - Teleports to a specific set of nodes
  - Nodes can have different probabilities of the surfer landing there:
    $$\boldsymbol{S} = [0.1, 0, 0, 0.2, 0, 0, 0.5, 0, 0, 0.2]$$
- **Random Walk with Restarts:**
  - Topic-Specific PageRank where teleport is always to the same node:
    $$\boldsymbol{S} = [0, 0, 0, 0, \mathbf{1}, 0, 0, 0, 0, 0, 0]$$

总结：

- A graph is naturally represented as a matrix
- We defined a random walk process over the graph
  - Random surfer moving across the links and with random teleportation
  - Stochastic adjacency matrix M
- PageRank = Limiting distribution of the surfer location represented node importance
  - Corresponds to the leading eigenvector of transformed adjacency matrix M.

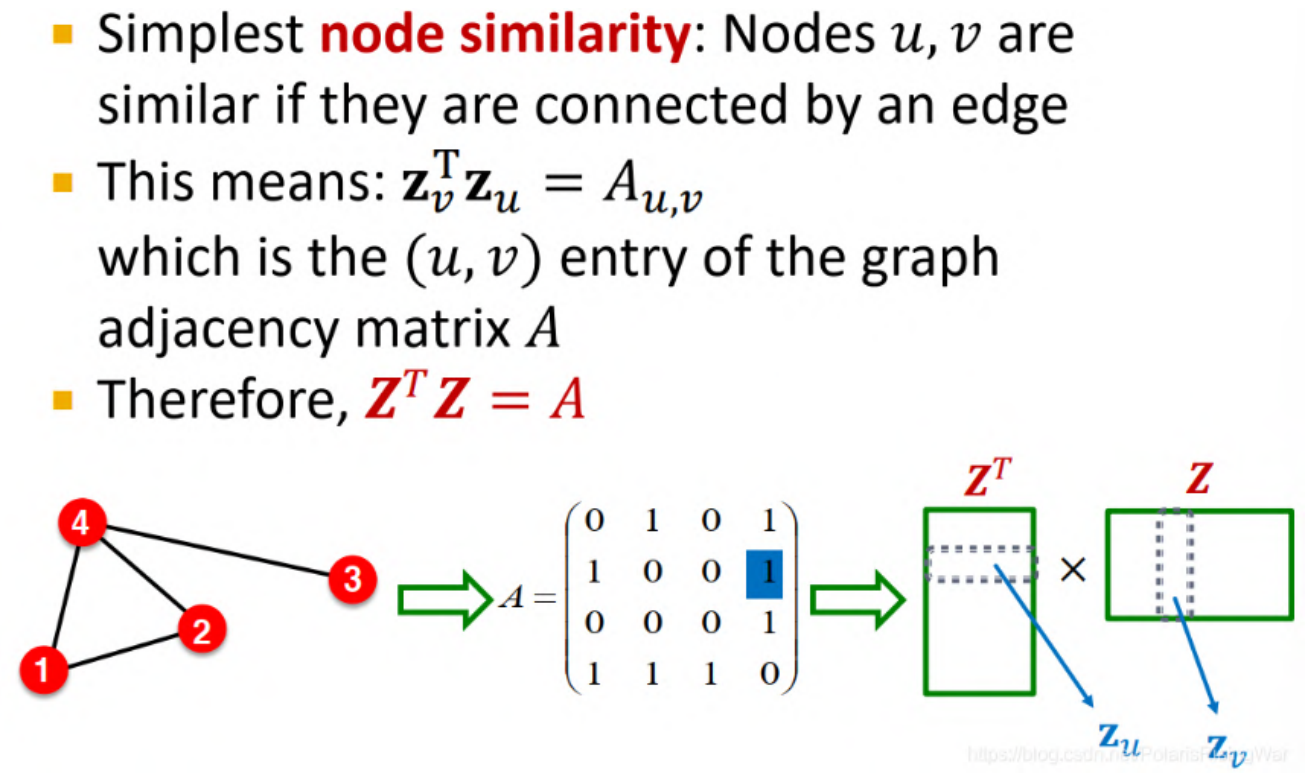
4.5 矩阵表示与节点嵌入

回忆上一章讲到的embedding lookup的encoder：



将节点嵌入视作矩阵分解：

假设有边的节点是相似节点，则 $\mathbf{z}_v^T\mathbf{z}_u = A_{u,v}$（A是邻接矩阵）(如有边连接，则节点嵌入相似性直接取到1)，则 $\mathbf{Z}^T\mathbf{Z} = A$ 。



矩阵分解问题：节点表示向量维度远低于节点数。如上一序号，将节点嵌入视作矩阵分解问题，严格的矩阵分解 $A = \mathbf{Z}^T\mathbf{Z}$ 很难实现（因为没有那么多维度来表示），因此通过最小化 A 和 $\mathbf{Z}^T\mathbf{Z}$ 间的L2距离（元素差的平方和）来近似目标。

在Lecture 3中我们使用softmax来代替L2距离完成目标。

所以用边连接来定义节点相似性的inner product decoder等同于A的矩阵分解问题。

- The embedding dimension d (number of rows in $\boldsymbol{Z}$) is much smaller than number of nodes n.
- Exact factorization $A = Z^T Z$ is generally not possible
- However, we can learn $\boldsymbol{Z}$ approximately
- **Objective**: $\min_{Z} \| A - \boldsymbol{Z}^T \boldsymbol{Z} \|_2$
  - We optimize $\boldsymbol{Z}$ such that it minimizes the L2 norm (Frobenius norm) of $A - \boldsymbol{Z}^T \boldsymbol{Z}$
  - Note in lecture 3 we used softmax instead of L2. But the goal to approximate A with $\boldsymbol{Z}^T \boldsymbol{Z}$ is the same.
- Conclusion: **inner product decoder with node similarity defined by edge connectivity is equivalent to matrix factorization of A**

矩阵分解问题：基于random walk定义的相似性

DeepWalk和node2vec有基于random walk定义的更复杂的节点相似性，但还是可以视作矩阵分解问题，不过矩阵变得更复杂了。（相当于是把上面的 A 给换了）

DeepWalk：

Volume of graph

$$vol(G) = \sum_i \sum_j A_{i,j}$$

Diagonal matrix D

$$D_{u,u} = \deg(u)$$

$$\log\left(vol(G)\left(\frac{1}{T}\sum_{r=1}^{T}(D^{-1}A)^r\right)D^{-1}\right) - \log b$$

context window size
See Lec 3 slide 30:
$T = |N_R(u)|$

Power of normalized adjacency matrix

Number of negative samples

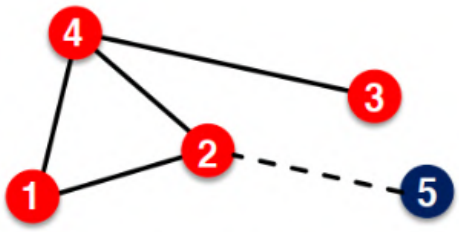
albeit尽管

$\mathrm{vol}(G)$ 是2倍的边数

通过矩阵分解和随机游走进行节点嵌入的限制：

- 无法获取不在训练集中的节点嵌入，每次新增节点都要对全部数据集重新计算嵌入。

- Cannot obtain embeddings for nodes not in the training set



Training set

A newly added node 5 at test time (e.g. new user in a social network)

Cannot compute its embedding with DeepWalk / node2vec. Need to recompute all node embeddings.

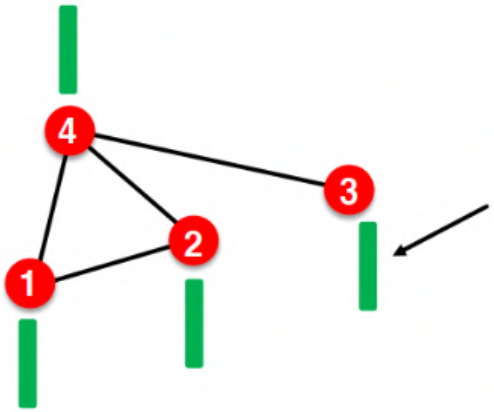
- 无法捕获结构相似性：比如图中节点1和节点11在结构上很相似，但是节点嵌入会差别很大（随机游走走不过去）

- Cannot capture **structural similarity**:

- Node 1 and 11 are **structurally similar** – part of one triangle, degree 2
- However, they have very **different** embeddings
  - It's unlikely that a random walk will reach node 11 from node 1
- **DeepWalk and node2vec do not capture structural similarity**

- 无法使用节点、边和图上的特征信息。

- Cannot utilize node, edge and graph features



Feature vector
(e.g. protein properties in a protein-protein interaction graph)

DeepWalk / node2vec embeddings do not incorporate such node features

4.6 总结

- **PageRank**
  - Measures importance of nodes in graph
  - Can be efficiently computed by **power iteration of adjacency matrix**
- **Personalized PageRank (PPR)**
  - Measures importance of nodes with respect to a particular node or set of nodes
  - Can be efficiently computed by **random walk**
- **Node embeddings** based on random walks can be expressed as **matrix factorization**
- **Viewing graphs as matrices plays a key role in all above algorithms!**

5. Colab：节点嵌入

本colab以无向图 Karate Club Network1 （有34个节点，78条边）为例，探索该数据集的相关统计量，并将从NetworkX下载的数据集转换为PyTorch的Tensor格式，用边连接作为节点相似性度量指标实现shallow encoder（以 nn.Embedding 为embedding-lookup）的节点嵌入代码。

节点嵌入训练概览：

- 用图中原本的边作为正值，从不存在的边中抽样作为负值，将对应边/节点对的点积结果用sigmoid归一化后视作输出值，将1视为正值的标签，0视为负值的标签。用BCELoss计算损失函数。

- 将nn.Embedding作为参数，用PyTorch在神经网络中以随机梯度下降的方式进行训练。
- 最后通过PCA将nn.Embedding.weight（即embedding-lookup的值）降维到二维上，通过可视化的方式直观检验训练效果。

5.0 Python 包导入

```
import networkx as nx

import torch
import torch.nn as nn
from torch.optim import SGD

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

import random
```

6. 消息传递与节点分类

本章主要内容:
我们的任务是：已知图中一部分节点的标签，用图中节点之间的关系来将标签分配到所有节点上。属于半监督学习任务。

本节课我们学习message passing方法来完成这一任务。对某一节点的标签进行预测，需要其本身特征、邻居的标签和特征。

message passing的假设是图中相似的节点之间会存在链接，也就是相邻节点有标签相同的倾向。这种现象可以用homophily（**相似节点倾向于聚集**）、influence（**关系会影响节点行为**）、confounding（**环境影响行为和关系**）来解释。

collective classification给所有节点同时预测标签的概率分布，基于马尔科夫假设（某一点标签仅取决于其邻居的标签）。

local classifier（用节点特征预测标签）→ relational classifier（用邻居标签 和/或 特征，预测节点标签）→ collective inference（持续迭代）

本节课讲如下三种collective classification的实现技术：

- relational classification：用邻居标签概率的加权平均值来代表节点标签概率，循环迭代求解。
- iterative classification：在训练集上训练用 (节点特征) 和 (节点特征，邻居标签summary z) 两种自变量预测标签的分类器 ϕ_1 和 ϕ_2，在测试集上用 ϕ_1 赋予初始标签，循环迭代求解 $z \rightleftharpoons$ 用 ϕ_2 重新预测标签
- belief propagation：在边上传递节点对邻居的标签概率的置信度（belief）的message/estimate，迭代计算边上的message，最终得到节点的belief。有环时可能出现问题。

6.1 消息传递与节点分类

本章重要问题：给定网络中部分节点的标签，如何用它们来分配整个网络中节点的标签？（举例：已知网络中有一些诈骗网页，有一些可信网页，如何找到其他诈骗和可信的网页节点？）

训练数据中一部分有标签，剩下的没标签，这种就是半监督学习。

对这个问题的一种解决方式：node embeddings。

- **Main question today:** Given a network with labels on some nodes, how do we assign labels to all other nodes in the network?
- **Example:** In a network, some nodes are fraudsters and some other nodes are fully trusted. **How do you find the other fraudsters and trustworthy nodes?**
- We already discussed node embeddings as a method to solve this in Lecture 3

本章我们介绍一个解决上述问题的方法：**message passing**

使用message passing基于“网络中存在关系correlations”这一直觉，亦即相似节点间存在链接。

message passing是通过链接传递节点的信息，感觉上会比较类似于 PageRank 将节点的vote通过链接传递到下一节点，但是在这里我们更新的不再是重要性的分数，而是对节点标签预测的概率。

核心概念 **collective classification**：同时将标签分配到网络中的所有节点上。

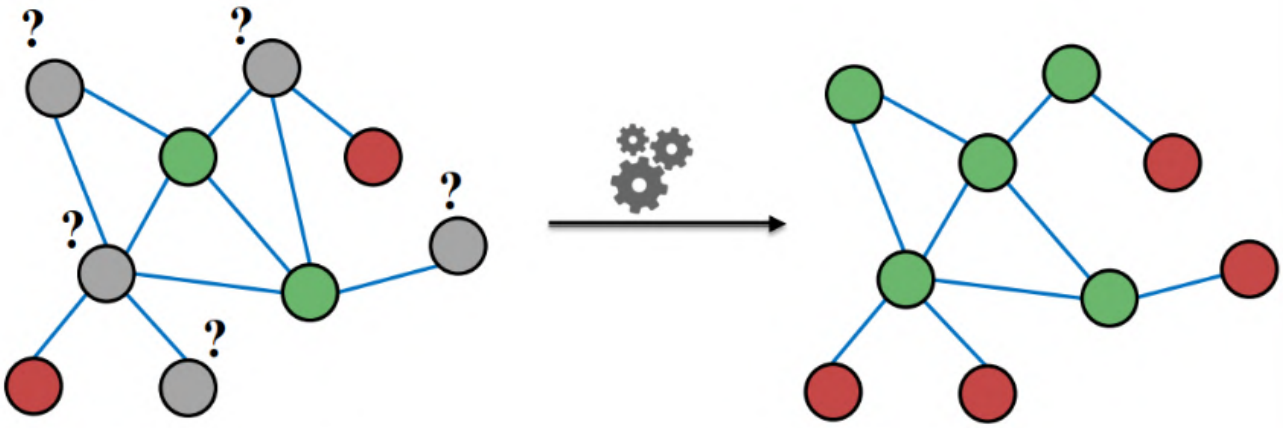
本章将讲述三种实现技术：

- **关系分类（relational classification）**
- **迭代分类（iterative classification）**
- **信念传播（belief propagation）**

- **Main question today:** Given a network with labels on some nodes, how do we assign labels to all other nodes in the network?
- **Today we will discuss an alternative framework: message passing**
- **Intuition: Correlations** exist in networks.
  - In other words: Similar nodes are connected
  - **Key concept** is **collective classification**: Idea of assigning labels to all nodes in a network together
- **We will look at three techniques today:**
  - **Relational classification**
  - **Iterative classification**
  - **Belief propagation**

举例：节点分类

半监督学习问题：给出一部分节点的标签（如图中给出了一部分红色节点、一部分绿色节点的标签），预测其他节点的标签



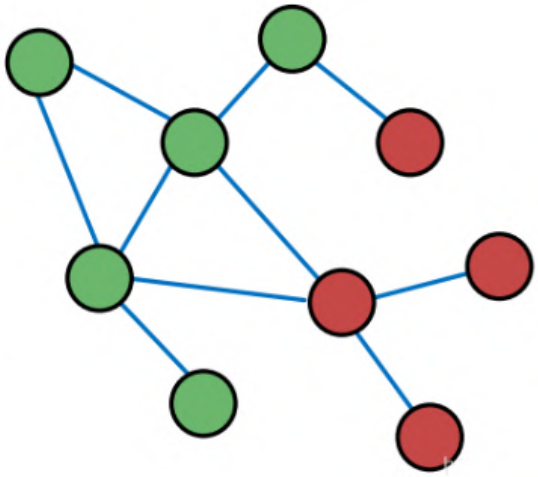
- Given labels of some nodes
- Let's predict labels of unlabeled nodes
- This is called semi-supervised node classification

网络中存在关系correlations：

相似的行为在网络中会互相关联。

correlation：相近的节点会具有相同标签

- Individual behaviors are **correlated** in the network
- **Correlation**: nearby nodes have the same color (belonging to the same class)

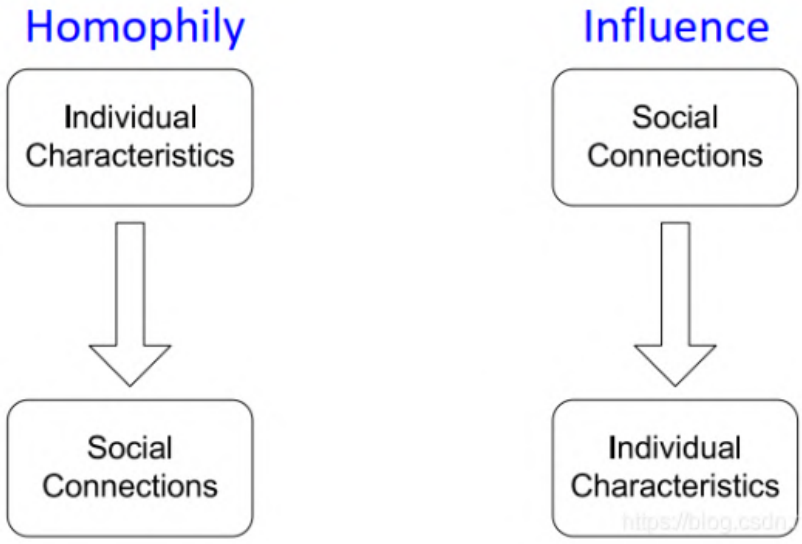


导致correlation的两种解释：

homophily（同质性，趋同性，同类相吸原则）：个体特征影响社交连接

influence：社交连接影响个体特征

- **Main types of dependencies that lead to correlation:**

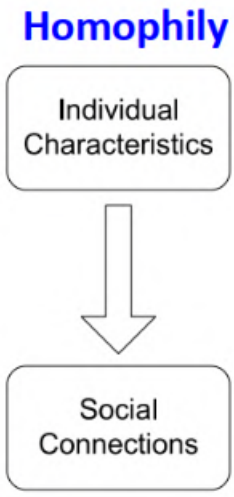


homophily：相似节点会倾向于交流、关联（物以类聚，人以群分）

在网络研究中得到了大量观察

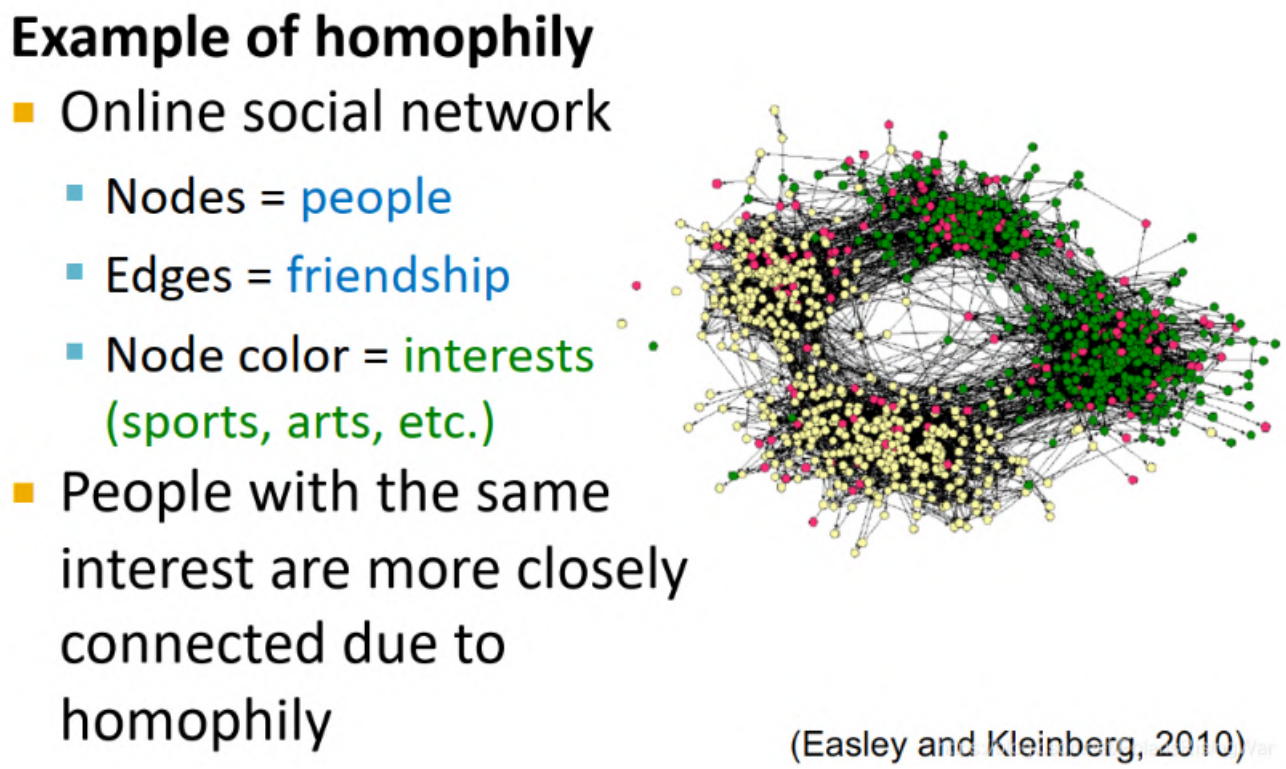
举例：同领域的研究者更容易建立联系，因为他们参加相同的会议、学术演讲……等活动

- **Homophily**: The tendency of individuals to associate and bond with similar others
  - *"Birds of a feather flock together"*
  - It has been observed in a vast array of network studies, based on a variety of attributes (e.g., age, gender, organizational role, etc.)
  - **Example**: Researchers who focus on the same research area are more likely to establish a connection (meeting at conferences, interacting in academic talks, etc.)



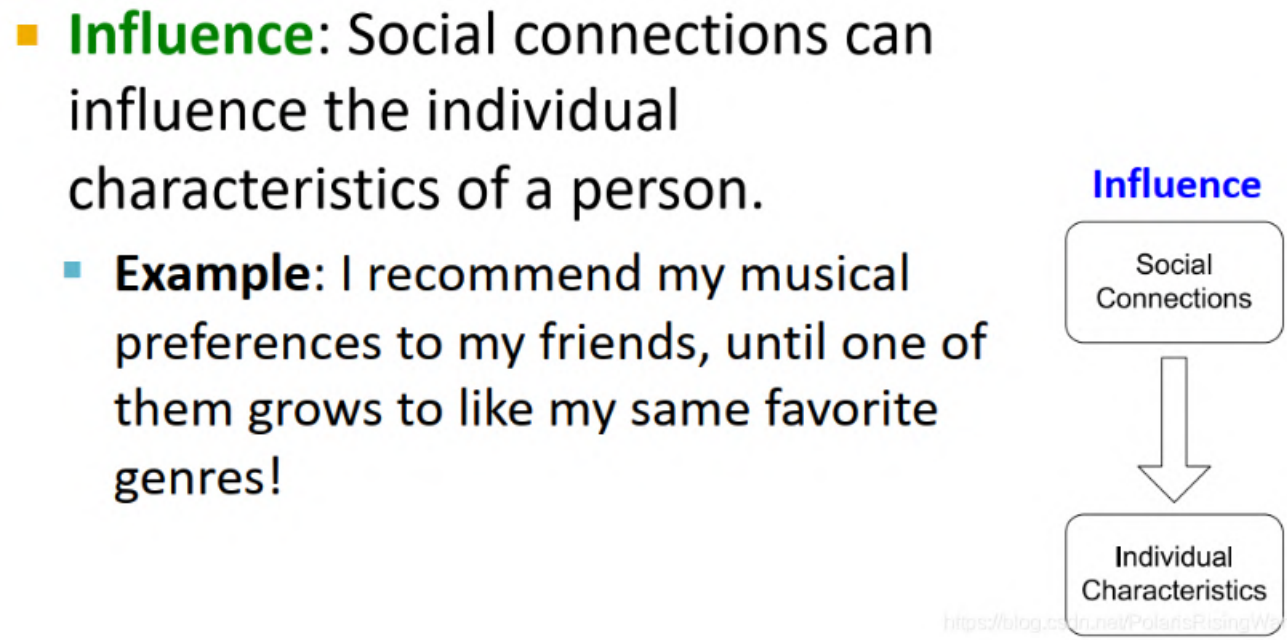
homophily举例：一个在线社交网络，以人为节点，友谊为边，通过人们的兴趣将节点分为多类（用颜色区分）。

从图中可以看出，各种颜色都分别聚在一起，亦即有相同兴趣（同色）的人们更有聚集在一起的倾向。



influence：社交链接会影响个人行为。

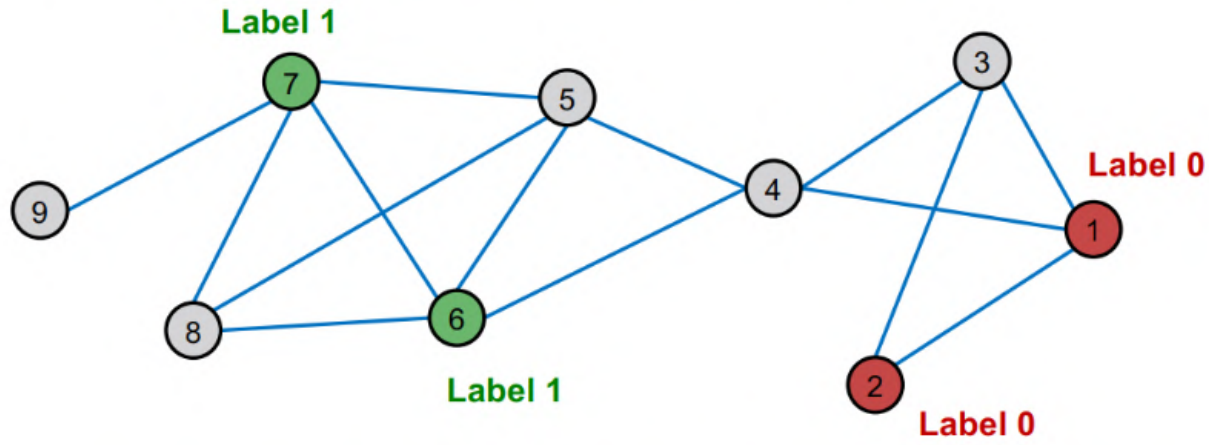
举例：用户将喜欢的音乐推荐给朋友。



既然知道了网络中关系的影响机制，我们就希望能够通过网络中的链接关系来辅助预测节点标签。

如图举例，我们希望根据已知的绿色（label 1）和红色（label 0）节点来预测灰色（标签未知）节点的标签。将各节点从 1-9 标注上node-id。

- How do we leverage this correlation observed in networks to help predict node labels?



How do we predict the labels for the nodes in grey?

解决分类问题的逻辑：我们已知相似节点会在网络中更加靠近，或者直接相连。

因此根据关联推定guilt-by-association：如果我与具有标签 X 的节点相连，那么我也很可能具有标签 X（基于马尔科夫假设）

举例：互联网中的恶意/善意网页：恶意网页往往会互相关联，以增加曝光，使其看起来更可靠，并在搜索引擎中提高排名。

- **Similar nodes are typically close together or directly connected in the network:**
  - **Guilt-by-association**: If I am connected to a node with label X, then I am likely to have label X as well.
  - **Example: Malicious/benign web page:** Malicious web pages link to one another to increase visibility, look credible, and rank higher in search engines

预测节点 v 的标签需要：

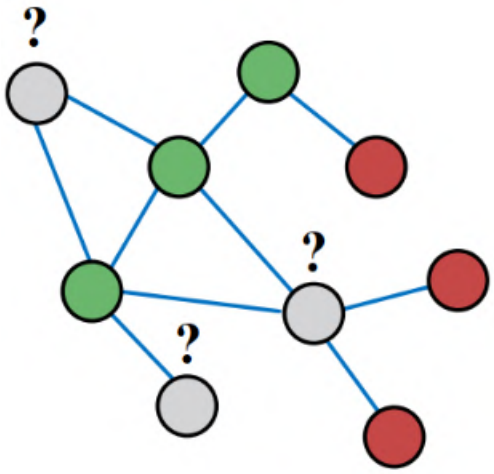
- v 的特征

- v 邻居的标签
- v 邻居的特征

- **Classification label** of a node v in network may depend on:
  - Features of v
  - Labels of the nodes in v's neighborhood
  - Features of the nodes in v's neighborhood

半监督学习:

任务：假设网络中存在homophily，根据一部分已知标签（红/绿）的节点预测剩余节点的标签



Given:
- Graph
- Few labeled nodes

Find: class (red/green) of remaining nodes

Main assumption: There is homophily in the network

示例任务：A 是 $n \times n$ 的邻接矩阵，$Y = \{0,1\}^n$ 是标签向量，目标是预测未知标签节点属于哪一类。

Example task:

- Let $\boldsymbol{A}$ be a $n \times n$ adjacency matrix over n nodes
- Let $\mathrm{Y} = \{0, 1\}^n$ be a vector of **labels**:
  - $\mathrm{Y}_v = 1$ belongs to Class 1
  - $\mathrm{Y}_v = 0$ belongs to Class 0
  - There are unlabeled node needs to be classified
- **Goal:** Predict which unlabeled nodes are likely Class 1, and which are likely **Class 0**

解决方法：**collective classification**

collective classification的应用：

1. Document classification
2. 词性标注Part of speech tagging
3. Link prediction
4. 光学字符识别Optical character recognition
5. Image/3D data segmentation
6. 实体解析Entity resolution in sensor networks
7. 垃圾邮件Spam and fraud detection

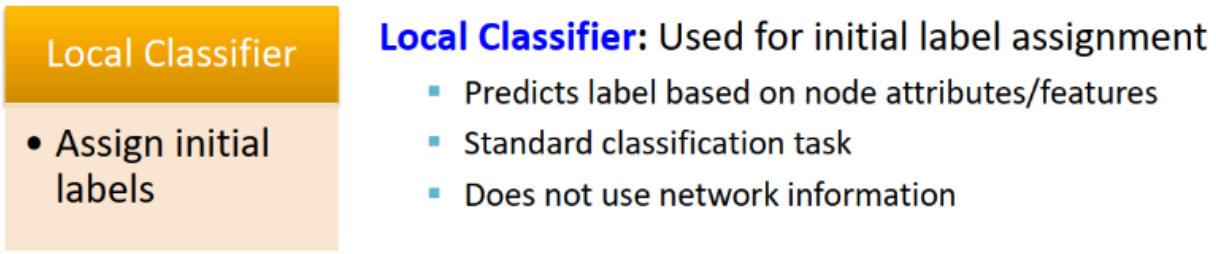
collective classification概述：使用网络中的关系同时对相连节点进行分类，使用概率框架（propabilistic framework），基于马尔科夫假设，节点 v 的标签 Y_v 取决于其邻居 N_v 的标签 $P(Y_v) = P(Y_v|N_v)$ 。

- **Intuition:** Simultaneous classification of interlinked nodes using correlations
- Probabilistic framework
- Markov Assumption: *the label Y_v of one node v depends on the labels of its neighbors N_v*

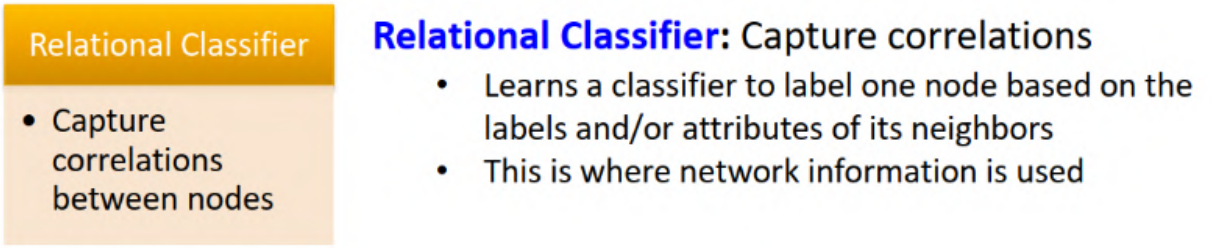
$$P(Y_v) = P(Y_v|N_v)$$

集体分类法（collective classification）分成三步：

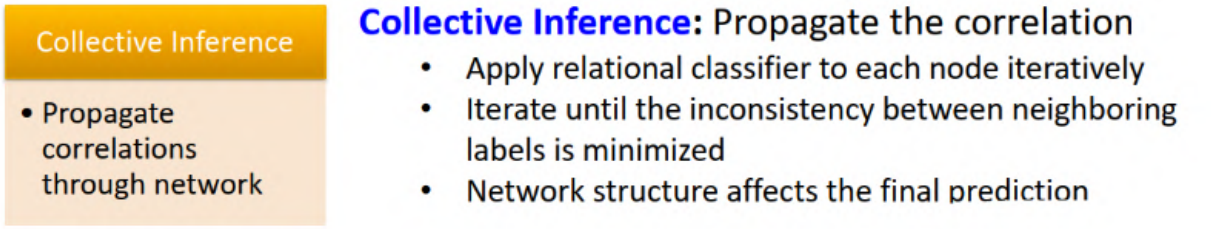
1. **局部分类器（local classifier）**：分配节点的初始标签（基于节点特征建立标准分类，不使用网络结构信息）



2. **关系分类器（relational classifier）**：捕获关系（基于邻居节点的标签 和/或 特征，建立预测节点标签的分类器模型）（应用了网络结构信息）



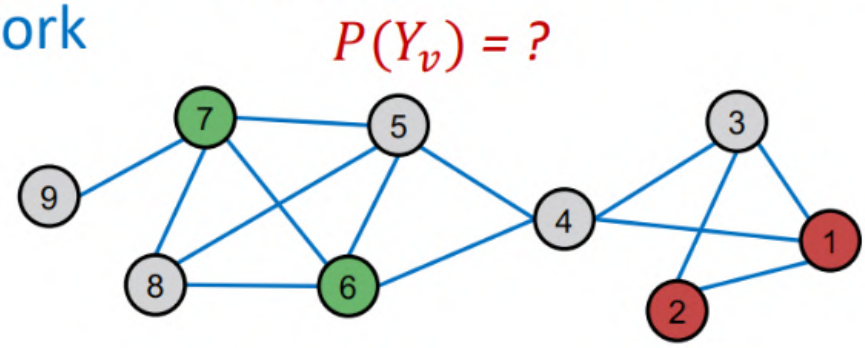
3. **集体推论（collective inference）**：传播关系（在每个节点上迭代relational classifier，直至邻居间标签的不一致最小化。网络结构影响最终预测结果）



问题设置：预测无标签（灰色）节点 v 的标签 Y_v。所有节点 v 具有特征向量 f_v。部分节点的标签已给出（绿色是1，红色是0）

任务：求解 $P(Y_v)$

- How to predict the labels Y_v for the unlabeled nodes v (in grey color)?
- Each node v has a feature vector f_v
- Labels for some nodes are given (1 for green, 0 for red)
- **Task:** Find $P(Y_v)$ given all features and the network



对本章后文内容的概述:

- We focus on semi-supervised node classification
- Intuition is based on **homophily**: Similar nodes are typically close together or directly connected
- **Three techniques we will introduce:**
  - **Relational classification**
  - **Iterative classification**
  - **Belief propagation**

6.2 关系分类/概率关系分类器

基础思想：节点 v 的类概率 Y_v 是其邻居类概率的加权平均值。

- 对有标签节点 v，固定 Y_v 为真实标签（ground-truth label）Y_v^* 。
- 对无标签节点 v，初始化 Y_v 为 0.5。

以随机顺序（不一定要是随机顺序，但实证上发现最好是。这个顺序会影响结果，尤其对小图而言）更新所有无标签节点，直至收敛或到达最大迭代次数。

- **Basic idea:** Class probability Y_v of node v is a weighted average of class probabilities of its neighbors
- For **labeled nodes** v, initialize label Y_v with ground-truth label Y_v^*
- For **unlabeled nodes**, initialize $Y_v = 0.5$
- **Update** all nodes in a random order until convergence or until maximum number of iterations is reached

更新公式:

$$P(Y_v = c) = \frac{1}{\sum_{(v,u)\in E} A_{v,u}} \sum_{(v,u)\in E} A_{v,u} P(Y_u = c)$$

邻接矩阵 $A_{v,u}$ 可以带权；分母是节点 v 的度数或入度；$P(Y_u = c)$ 是节点 v 标签为 c 的概率。

问题:

- 不一定能收敛
- 模型无法使用节点特征信息

- **Update** for each node v and label c (e.g. 0 or 1)

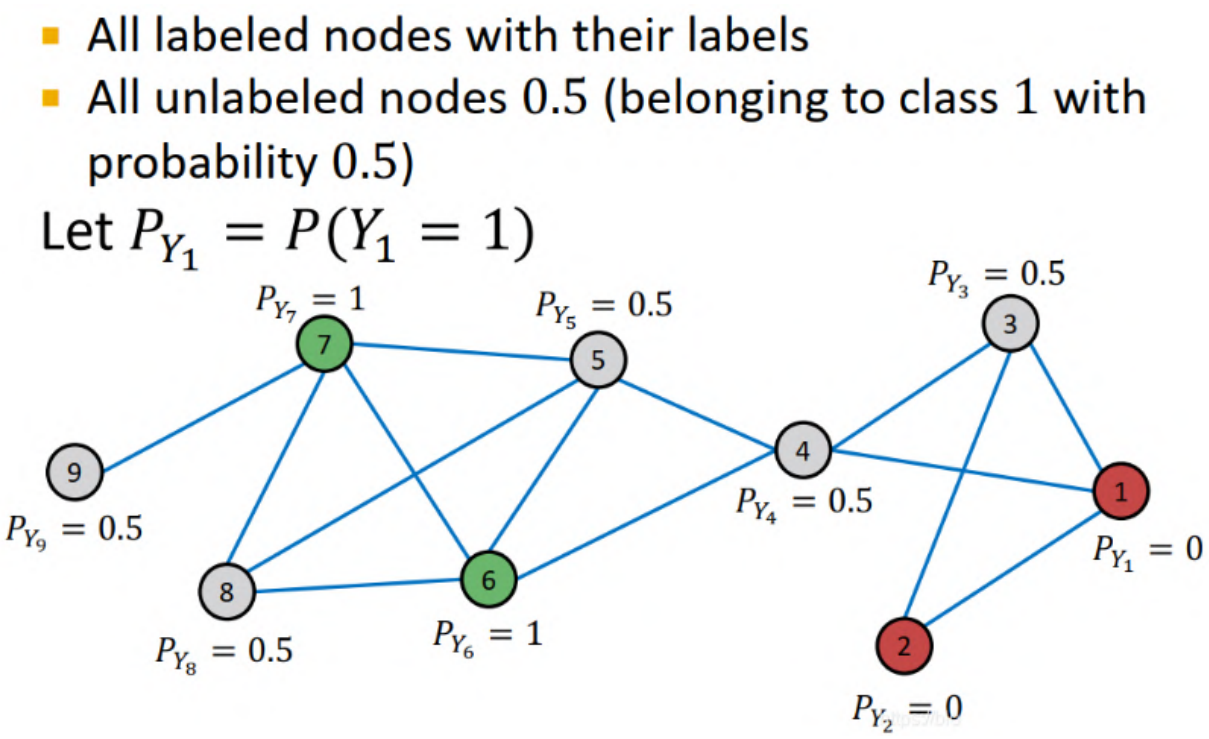
$$P(Y_v = c) = \frac{1}{\sum_{(v,u)\in E} A_{v,u}} \sum_{(v,u)\in E} A_{v,u}\, P(Y_u = c)$$

  - If edges have strength/weight information, $A_{v,u}$ can be the edge weight between v and u
  - $P(Y_v = c)$ is the probability of node v having label c

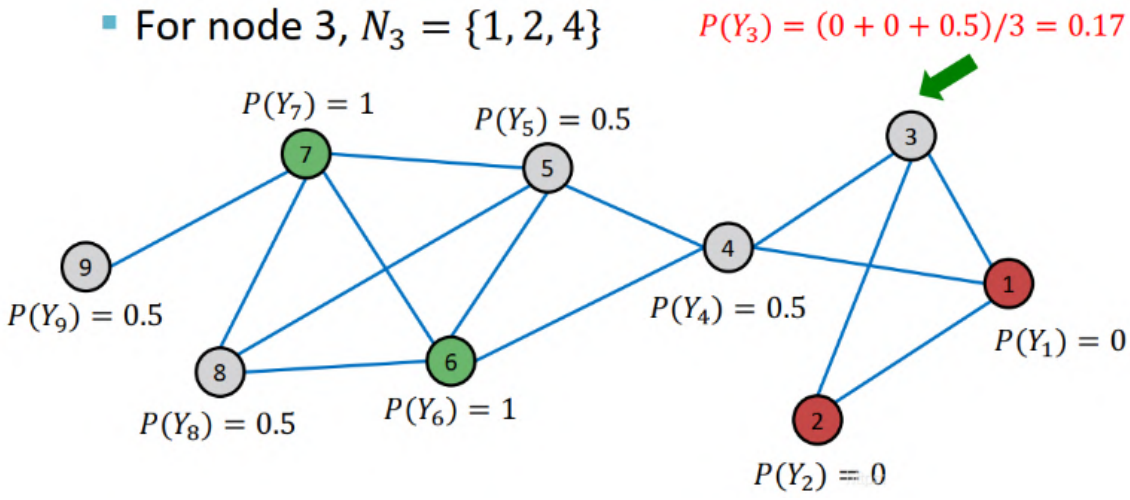
- Challenges:
  - Convergence is not guaranteed
  - Model cannot use node feature information

举例：

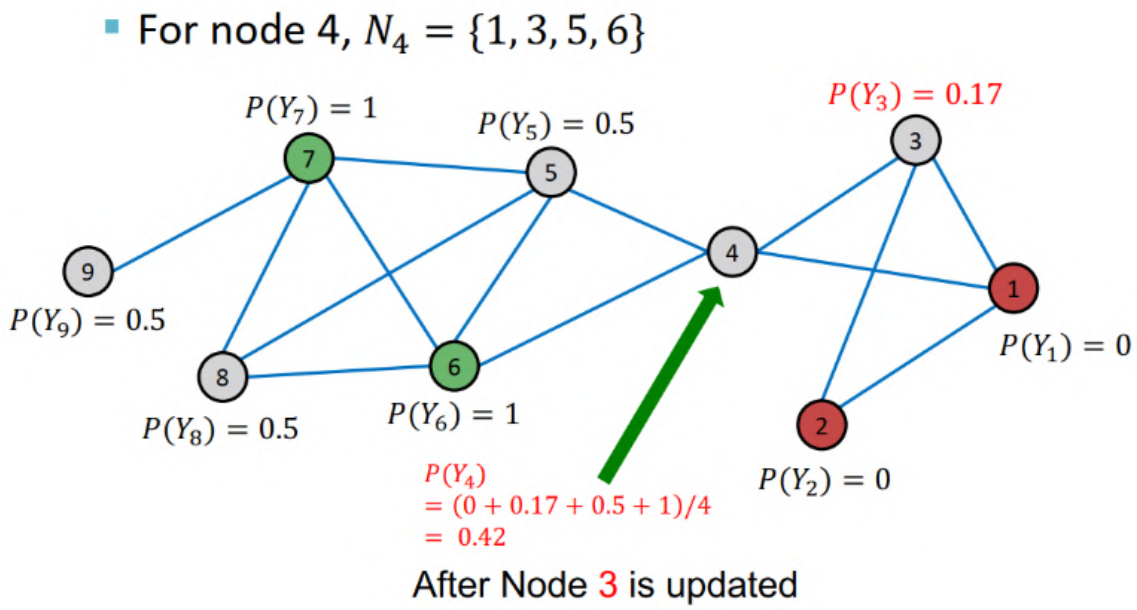
1. 初始化：迭代顺序就是node-id的顺序。有标签节点赋原标签，无标签节点赋0，$P_{Y_v} = P(Y_v = 1)$。



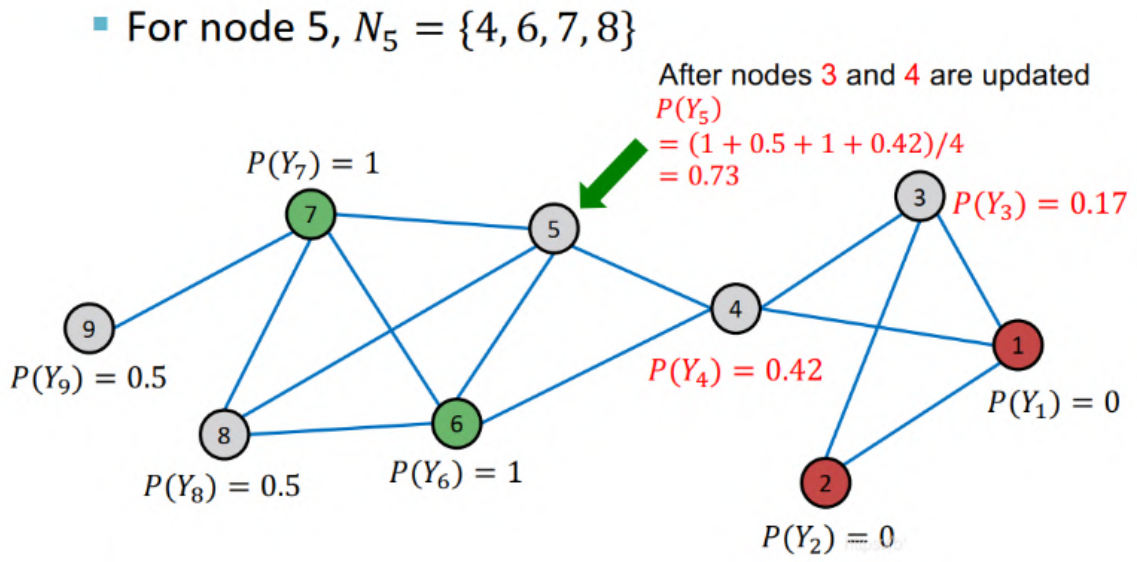
2. **第一轮迭代**：
   - 更新节点3：



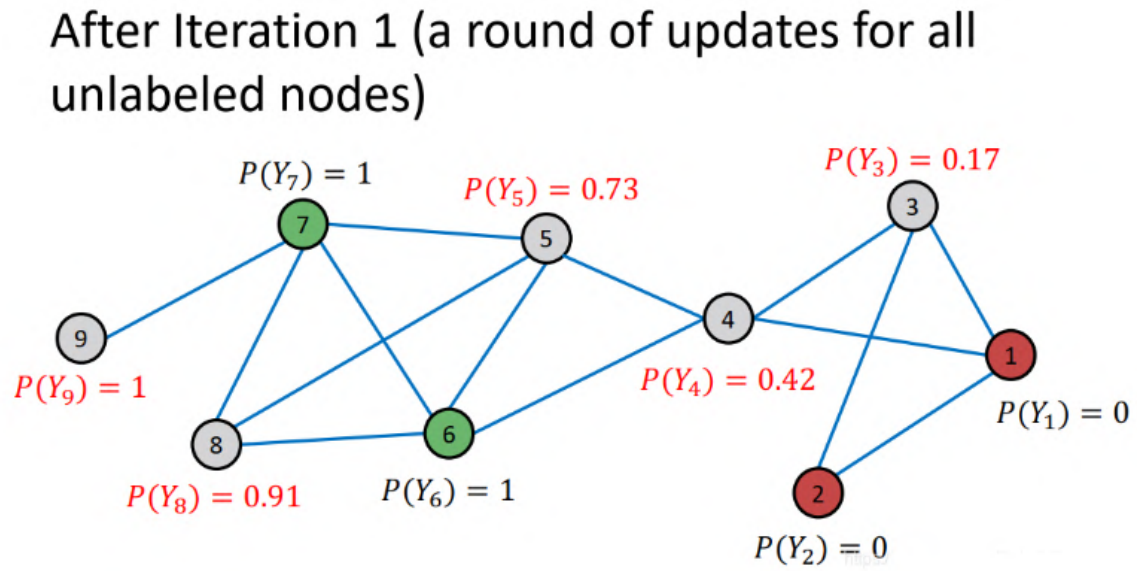
   - 更新节点4：



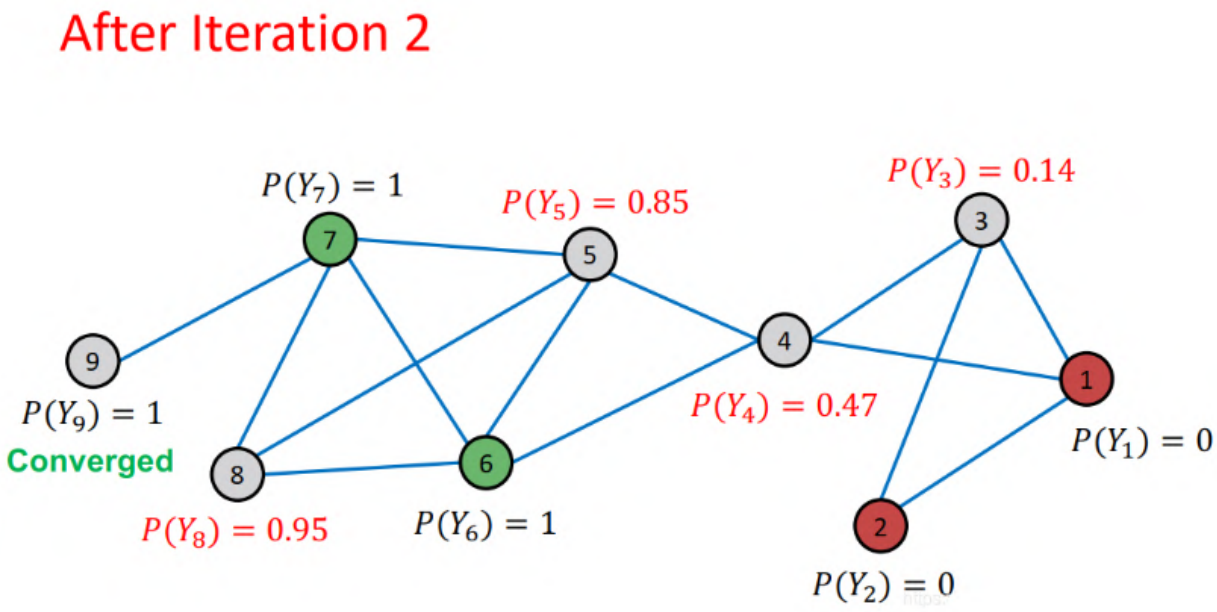
- 更新节点5：



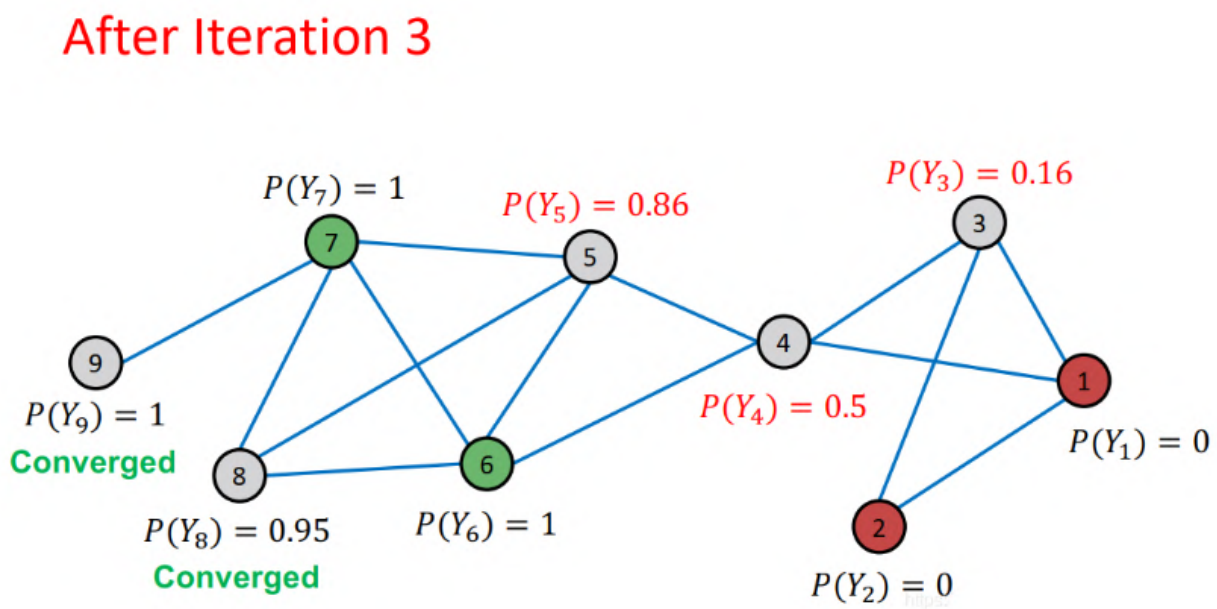
- 更新完所有无标签节点：



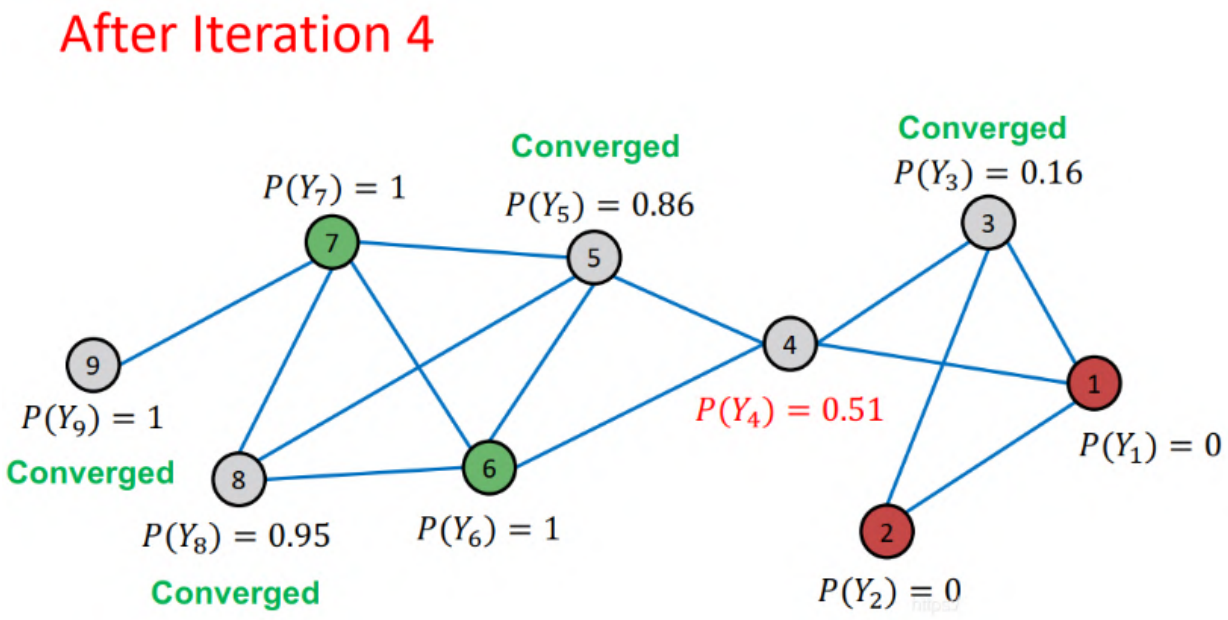
3. 第二轮迭代：结束后发现节点9收敛



4. 第三轮迭代：结束后发现节点8收敛

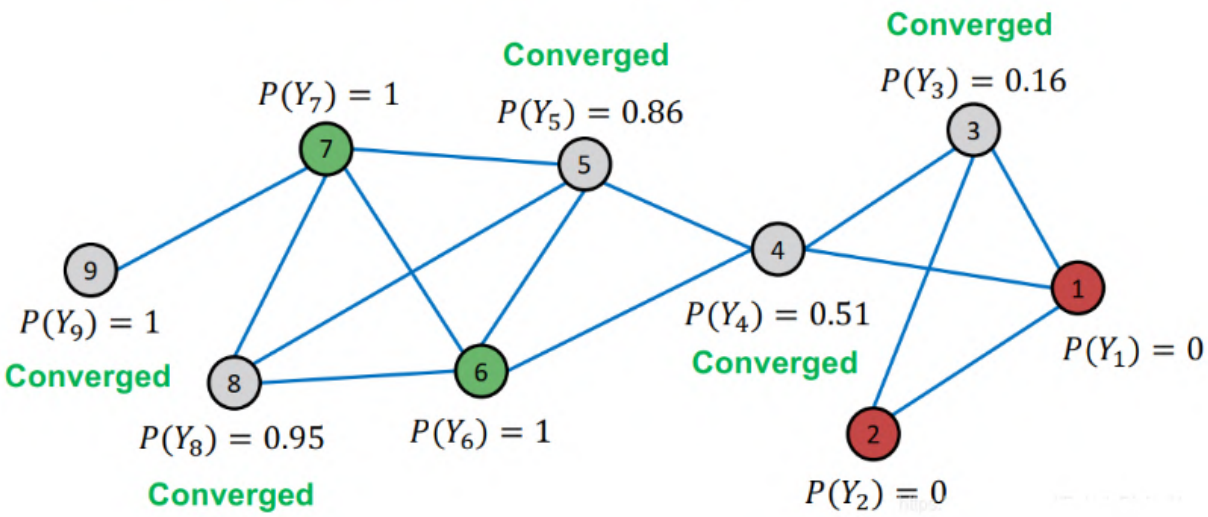


5. 第四轮迭代：



6. 收敛后：预测概率>0.5的节点为1，<0.5的为0

- All scores stabilize after 4 iterations. We therefore predict:
  - **Nodes 4, 5, 8, 9 belong to class 1** ($P_{Y_v} > 0.5$)
  - **Nodes 3 belong to class 0** ($P_{Y_v} < 0.5$)



6.3 迭代分类

关系分类器（relational classifiers）没有使用节点特征信息，所以我们使用新方法迭代分类（iterative classification）。

迭代分类（iterative classification）主思路：基于节点特征及邻居节点标签对节点进行分类

- **Relational classifiers do not use node attributes. How can one leverage them?**

- **Main idea of iterative classification:** Classify node v based on its **attributes f_v** as well as **labels z_v** of neighbor set N_v

迭代分类的方法：训练两个分类器

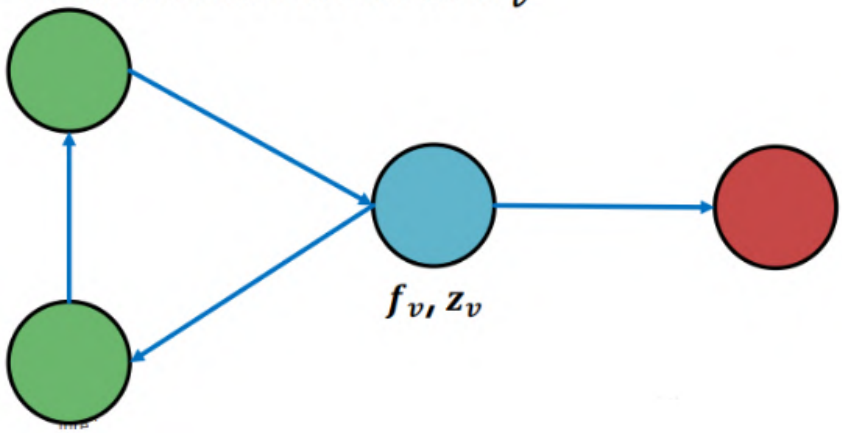
- $\phi_1(f_v)$ 基于节点特征向量 f_v 预测节点标签。
- $\phi_2(f_v, z_v)$ 基于节点特征向量 f_v 和邻居节点标签summary z_v 预测节点标签。

- **Input: Graph**
  - f_v : feature vector for node v
  - Some nodes v are labeled with Y_v
- **Task:** Predict label of unlabeled nodes
- **Approach: Train two classifiers:**
- $\phi_1(f_v)$ = Predict node label based on node feature vector f_v
- $\phi_2(f_v, z_v)$ = Predict label based on node feature vector f_v and summary z_v of labels of v's neighbors.

计算summary z_v：z_v 是个向量，可以是邻居标签的直方图（各标签数目或比例），邻居标签中出现次数最多的标签，邻居标签的类数。

How do we compute the summary z_v of labels of v's neighbors N_v?

- Ideas: $\boldsymbol{z_v}$ **= vector**
  - Histogram of the number (or fraction) of each label in N_v
  - Most common label in N_v
  - Number of different labels in N_v



迭代分类器iterative classifier的结构：

1. **第一步：基于节点特征建立分类器。**在训练集上训练如下分类器（可以用线性分类器、神经网络等算法）：
    - $\phi_1(f_v)$ 基于 f_v 预测 Y_v 。
    - $\phi_2(f_v, z_v)$ 基于 f_v 和 z_v 预测 Y_v 。
2. **第二步：迭代至收敛。**在测试集上，用 ϕ_1 预测标签，根据 ϕ_1 计算出的标签计算 z_v 并用 ϕ_2 预测标签。对每个节点重复迭代计算 z_v ，用 ϕ_2 预测标签这个过程，直至收敛或到达最大迭代次数（10, 50, 100……这样，不能太多）。注意：模型不一定能收敛。

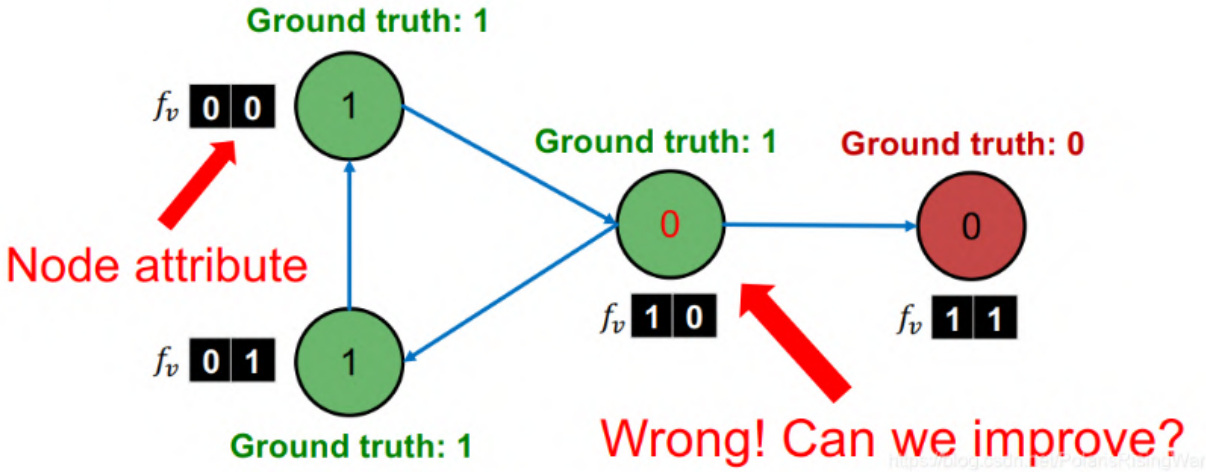
- **Phase 1: Classify based on node attributes alone**
    - On a **training set**, train classifier (e.g., linear classifier, neural networks, …):
    - $\phi_1(f_v)$ to predict Y_v based on f_v
    - $\phi_2(f_v, z_v)$ to predict Y_v based on f_v and summary z_v of labels of v's neighbors
- **Phase 2: Iterate till convergence**
    - On **test set**, set labels Y_v based on the classifier ϕ_1, compute z_v and predict the labels with ϕ_2
    - **Repeat** for each node v
        - Update z_v based on Y_u for all $u \in N_v$
        - Update Y_v based on the new z_v (ϕ_2)
    - Iterate until class labels stabilize or max number of iterations is reached
    - Note: Convergence is not guaranteed

计算举例：网页分类问题。

节点是网页，链接是超链接，链接有向。节点特征简化设置为2维二元向量。预测网页主题。

- **Input:** Graph of web pages
- **Node:** Web page
- **Edge:** Hyper-link between web pages
  - **Directed edge:** a page points to another page
- **Node features:** Webpage description
  - For simplicity, we only consider 2 binary features
- **Task:** Predict the topic of the webpage

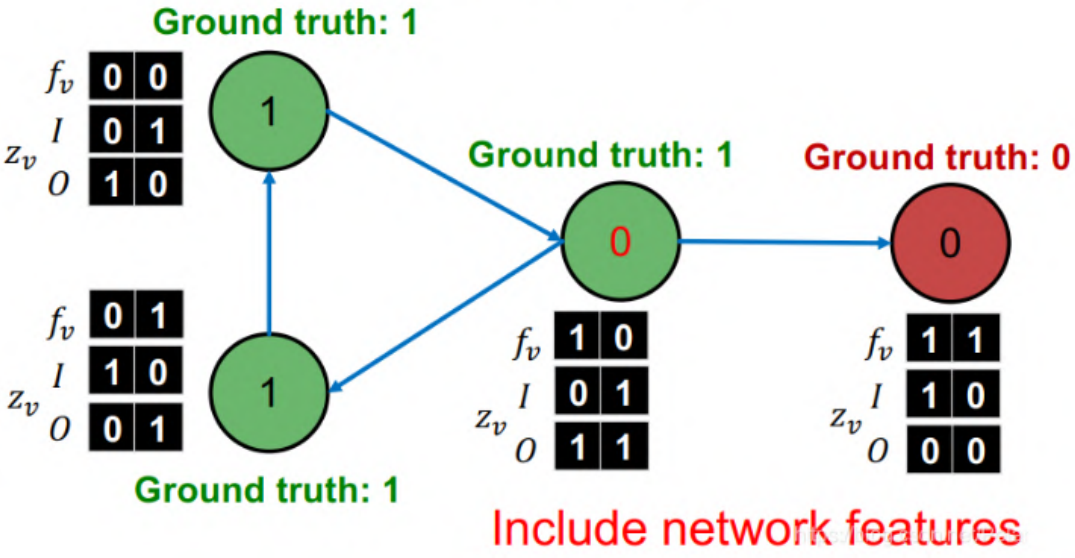
1. **基于节点特征训练分类器** ϕ_1：
   - Baseline: train a classifier (e.g., linear classifier) to classify pages based on binary node attributes.



可以假设分类器以特征第一个元素作为分类标准，于是对中间节点分类错误。

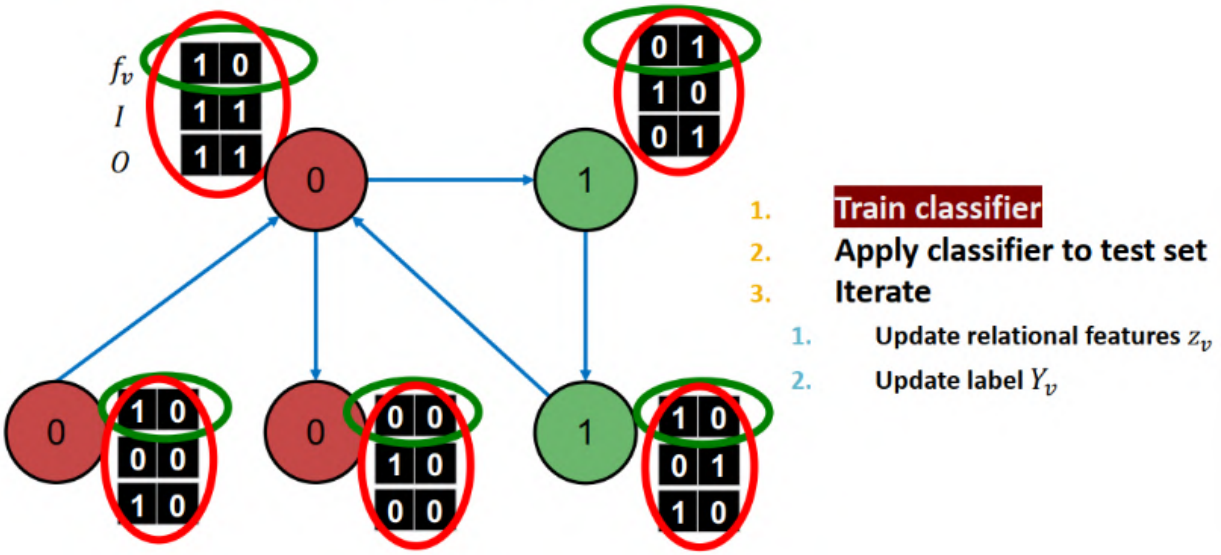
2. **根据** ϕ_1 **得到的结果计算** z_v：此处设置 z_v 为四维向量，四个元素分别为指向该节点的节点中标签为0和1的数目、该节点指向的节点中标签为0和1的数目。在这一步应用了网络结构信息。

- Each node maintains vectors z_v of neighborhood labels:
  I = **Incoming** neighbor label information vector
  O = **Outgoing** neighbor label information vector
- $I_0 = 1$ if at least one of the incoming pages is labelled 0.
  Similar definitions for I_1, O_0, and O_1



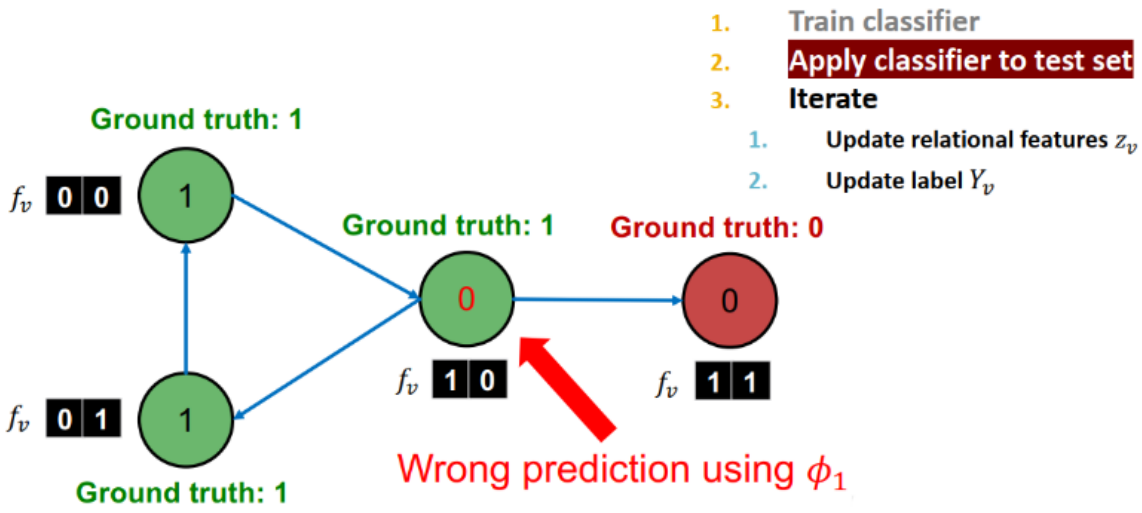
过程举例:

1. 第一步：在训练集上训练 ϕ_1 和 ϕ_2 。
   - On a different **training set**, train two classifiers:
     - Node attribute vector only (green circles): ϕ_1
     - Node attribute and link vectors (red circles): ϕ_2

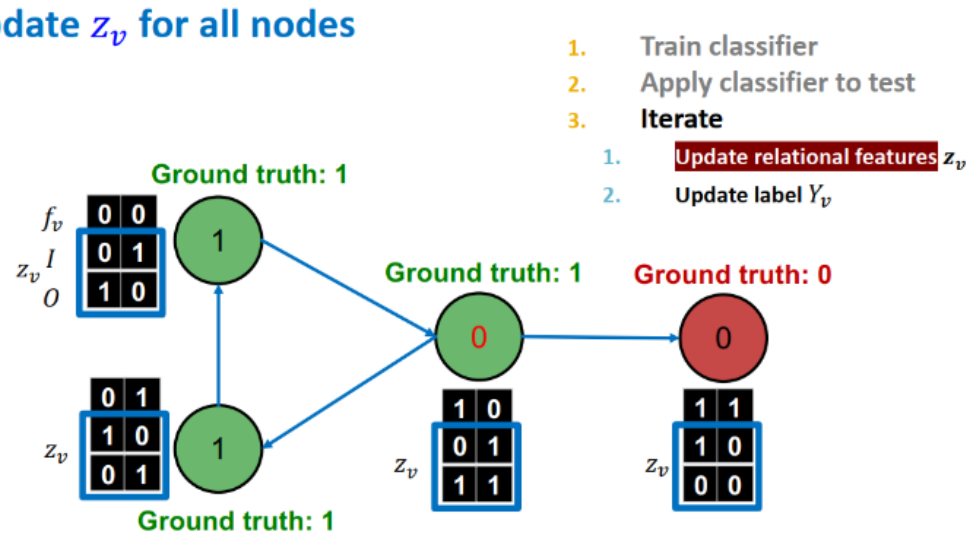


2. 第二步：在测试集上预测标签。
   - 用 ϕ_1 预测 Y_v :

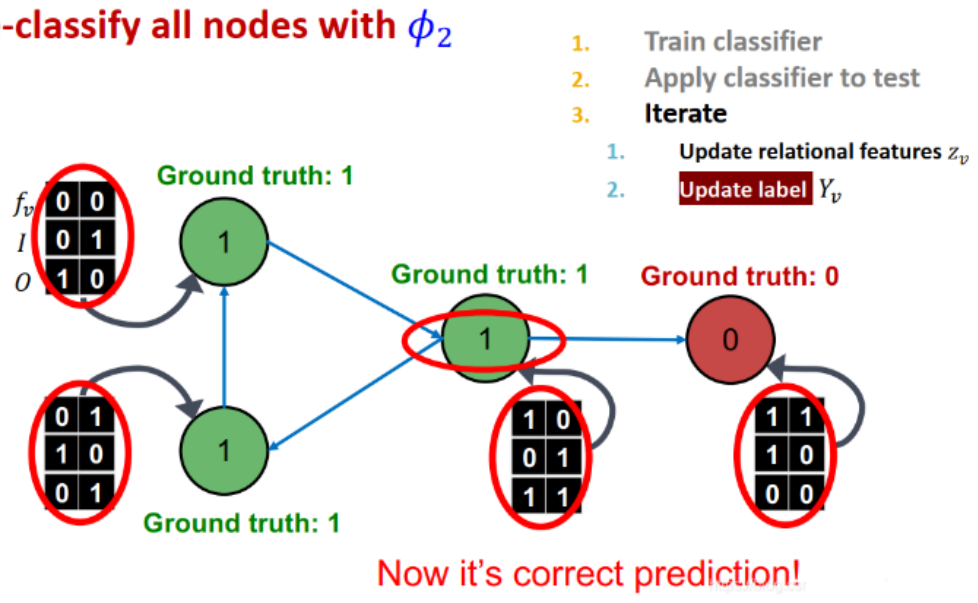
- On the **test set**:
  - Use trained node feature vector classifier ϕ_1 to set Y_v



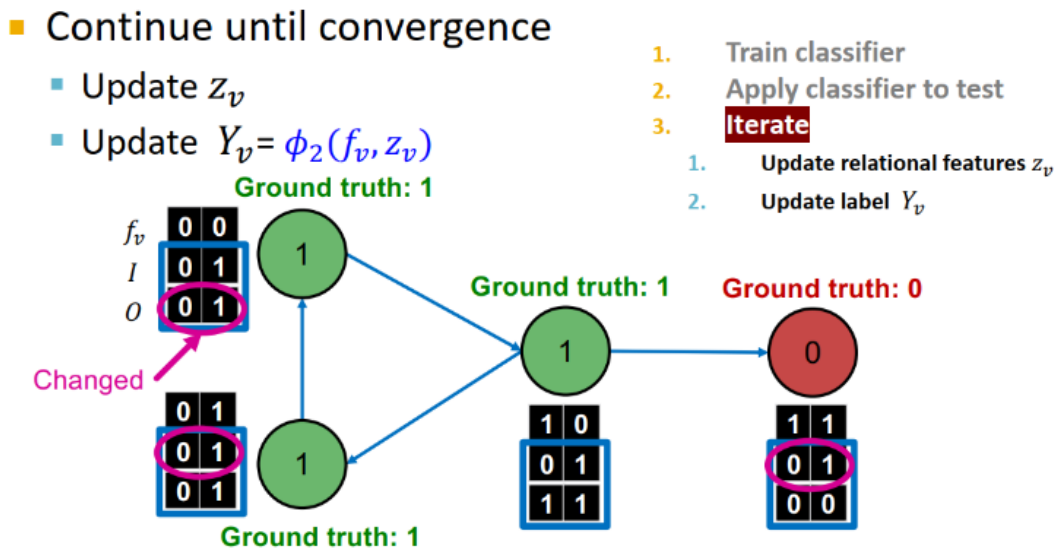
- 循环迭代：
  - 用 Y_v 计算 z_v
    - **Update z_v for all nodes**



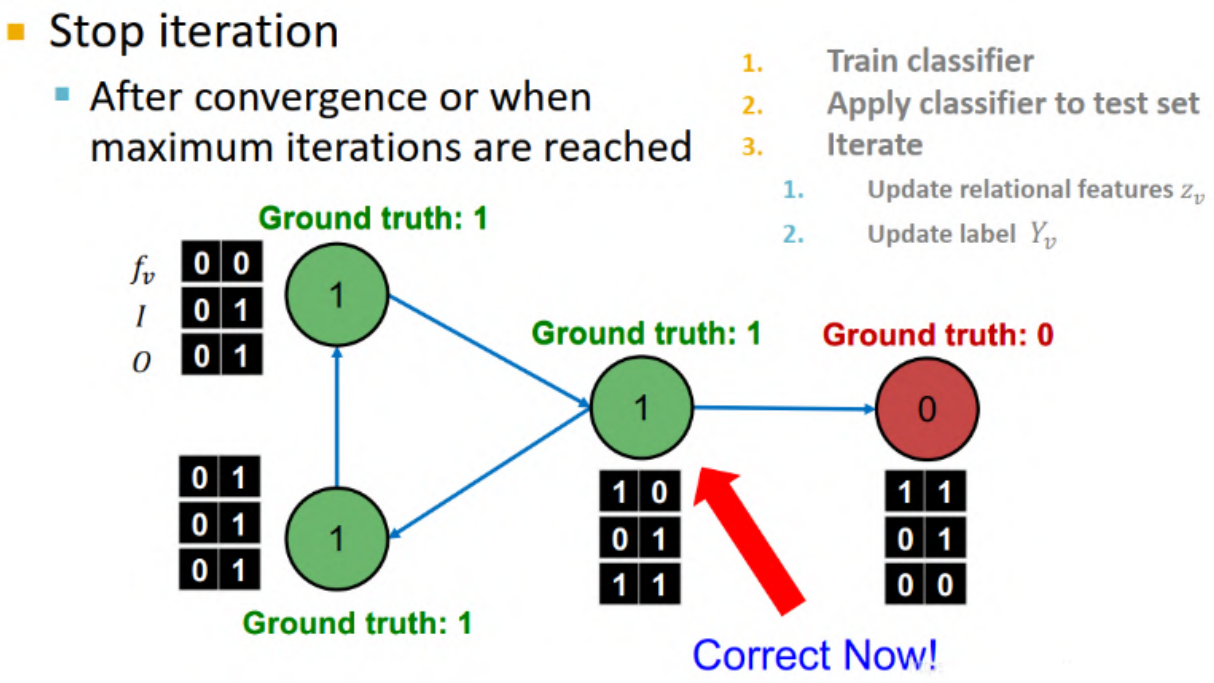
  - 用 ϕ_2 预测标签
    - **Re-classify all nodes with ϕ_2**



  - 迭代直至收敛



3. 结束迭代（收敛或达到最大迭代次数），得到最终预测结果。



对relational classification和iterative classification的总结：

- **We talked about 2 approaches to collective classification**
- **Relational classification**
  - Iteratively update probabilities of node belonging to a label class based on its neighbors
- **Iterative classification**
  - Improve over collective classification to handle attribute/feature information
  - Classify node i based on its **features** as well as **labels** of neighbors

6.4 循环信念传播

名字叫loopy是因为loopy BP方法会应用在有很多环的情况下。

信念传播（belief propagation）是一种动态规划方法，用于回答图中的概率问题（比如节点属于某类的概率）。

邻居节点之间迭代传递信息pass message（如传递相信对方属于某一类的概率），直至达成共识（大家都这么相信），计算最终置信度（也有翻译为信念的）belief。

问题中节点的状态并不取决于节点本身的belief，而取决于邻居节点的belief。

- Belief Propagation is a dynamic programming approach to **answering probability queries in a graph** (e.g. probability of node v belonging to class 1)
- Iterative process in which neighbor nodes "talk" to each other, **passing messages**

- **When consensus is reached, calculate final belief**

消息传递（message passing）：

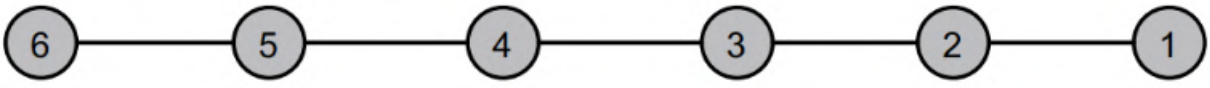
例子介绍：

任务：计算图中的节点数（注意，如果图中有环会出现问题，后文会讲有环的情况。在这里不考虑）

限制：每个节点只能与邻居交互（传递信息）

举例：path graph（节点排成一条线）

- **Task**: Count the number of nodes in a graph*
- **Condition:** Each node can only interact (pass message) with its neighbors
- **Example**: path graph

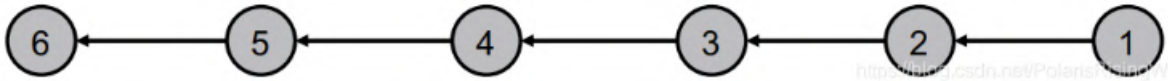


* Potential issues when the graph contains cycles.
We'll get back to it later!

算法：

1. 定义节点顺序（生成一条路径）
2. 基于节点顺序生成边方向，从而决定message passing的顺序
3. 按节点顺序，计算其对下一节点的信息（至今数到的节点数），将该信息传递到下一节点

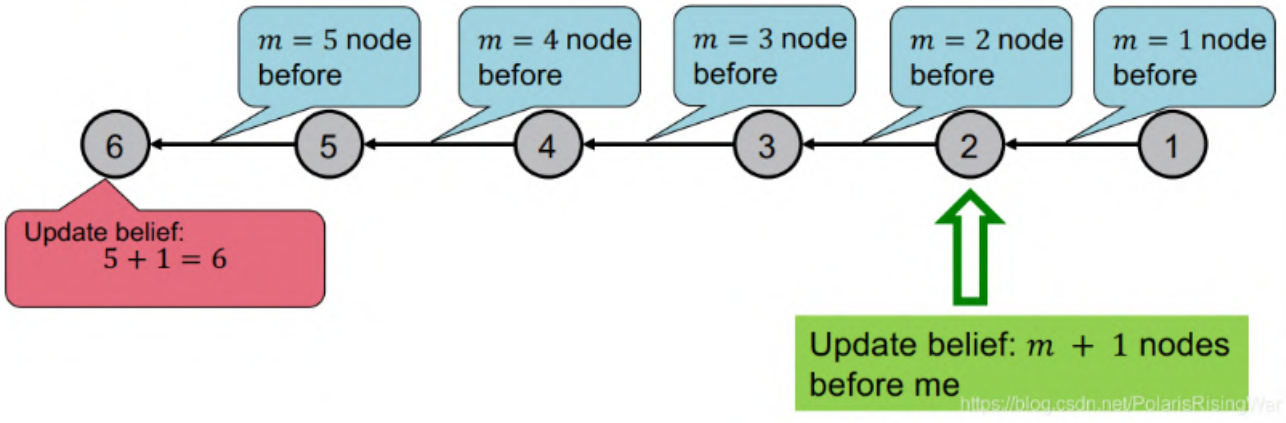
- **Algorithm:**
  - Define an ordering of nodes (that results in a path)
  - Edge directions are according to order of nodes
    - Edge direction defines the order of message passing
  - For node i from 1 to 6
    - Compute the message from node i to $i+1$ (number of nodes counted so far)
    - Pass the message from node i to $i+1$



每个节点**接收邻居信息，更新信息，传递信息**

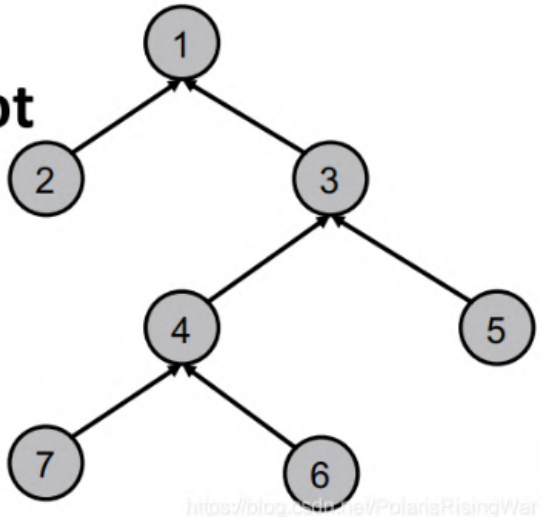
Solution: Each node listens to the message from its neighbor, updates it, and passes it forward

m: the message

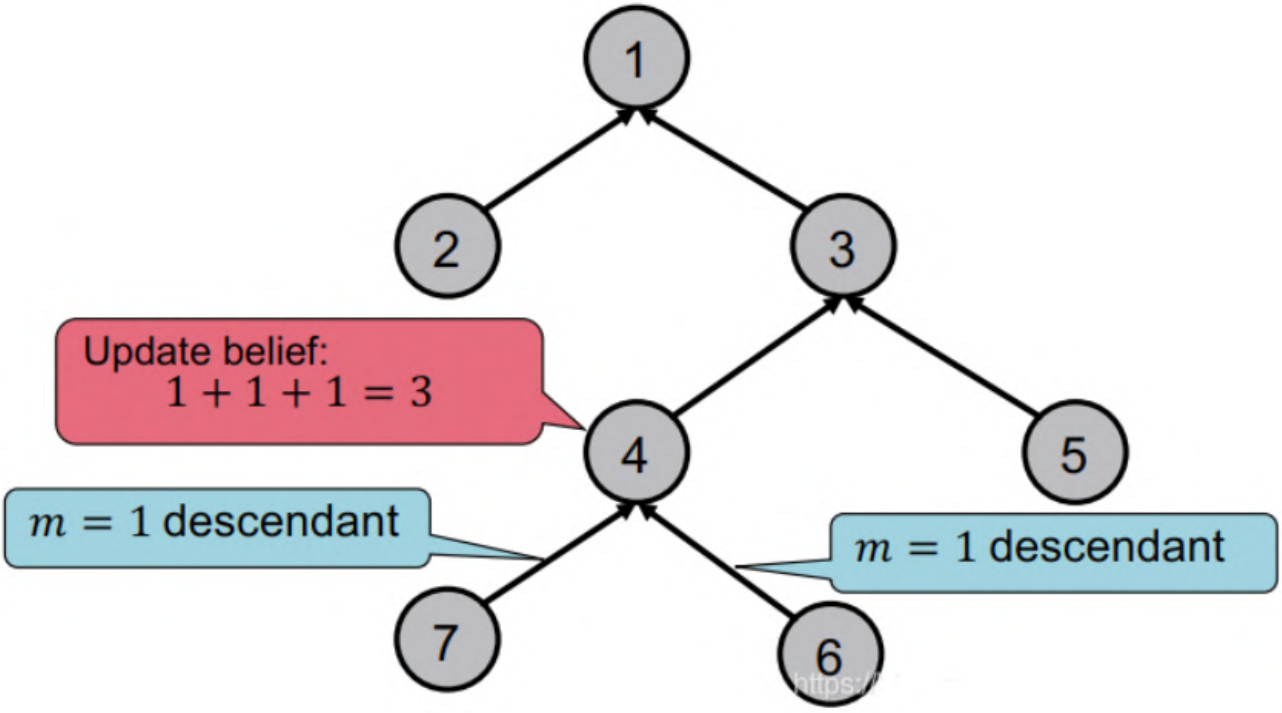


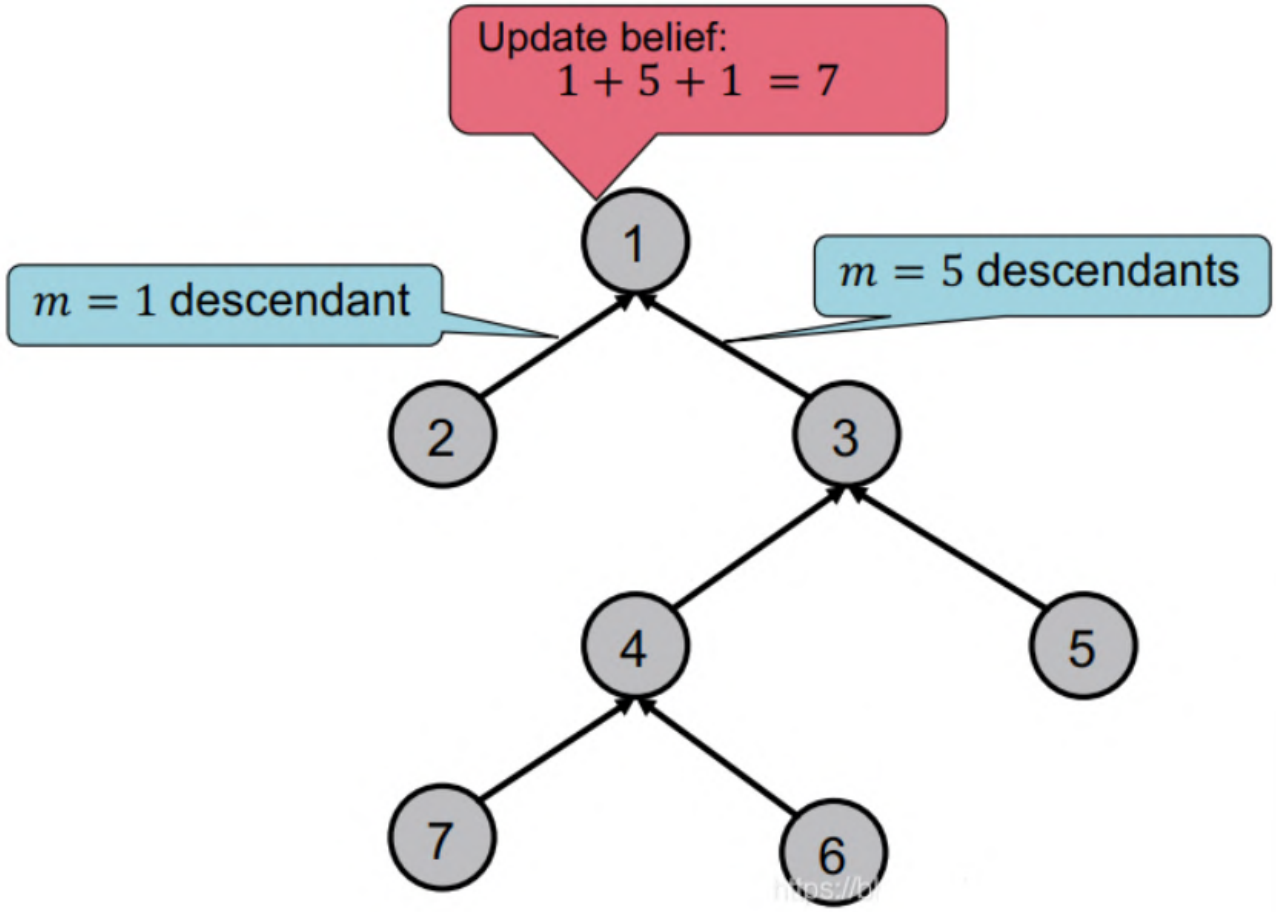
将path graph的情况泛化到树上：从叶子节点到根节点传递信息

- We can perform message passing not only on a path graph, but also on a tree-structured graph
- Define order of message passing from **leaves** to **root**

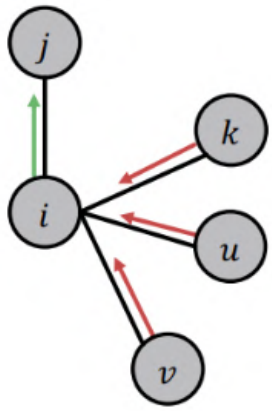


在树结构上更新置信度：





Loopy BP Algorithm：从 i 传递给 j 的信息，取决于 i 从邻居处接收的信息。每个邻居给 i 对其状态的置信度的信息。



What message will i send to j?

- It depends on what i hears from its neighbors
- Each neighbor passes a message to i its beliefs of the state of i

I (node i) believe that you (node j) belong to class Y_j with probability …

一些记号说明：

- label-label potential matrix ψ： 节点及其邻居间的dependency。$\psi(Y_i, Y_j)$ 表示，节点 i 是节点 j 的邻居，已知 i 属于类 Y_i ， $\psi(Y_i, Y_j)$ 与 j 属于类 Y_j 的概率成比例。
- prior belief ϕ： $\phi(Y_i)$ 与节点 i 属于类 Y_i 的概率成比例。
- $m_{i \to j}(Y_j)$： i 对 j 属于类 Y_j 的message/estimate。
- L： 所有类/标签的集合。

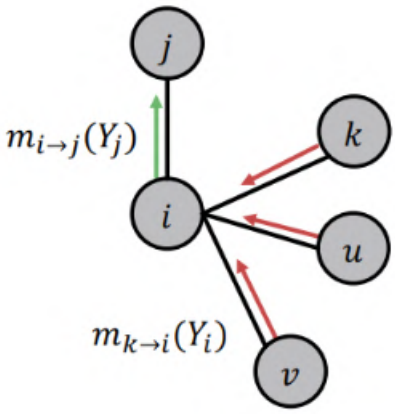
- ***Label-label potential matrix*** ψ : Dependency between a node and its neighbor. $\psi(Y_i, Y_j)$ is proportional to the probability of a node j being in class Y_j given that it has neighbor i in class Y_i.
- ***Prior belief*** ϕ : $\phi(Y_i)$ is proportional to the probability of node i being in class Y_i.
- $m_{i \to j}(Y_j)$ is i's message / estimate of j being in class Y_j.
- $\mathcal{L}$ is the set of all classes/labels

Loopy BP Algorithm:

1. 将所有信息初始化为1。
2. 对每个节点重复：

$$m_{i \to j}(Y_j) = \sum_{Y_i \in L} \psi(Y_i, Y_j) \phi_i(Y_i) \prod_{k \in N_i / j} m_{i \to j}(Y_i), \forall Y_j \in L$$

（反斜杠指除了 j ）



1. Initialize all messages to 1
2. Repeat for each node:

All messages sent by neighbors from previous round

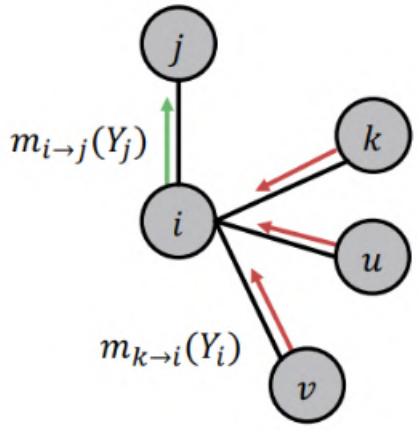
Label-label potential

$$m_{i\to j}(Y_j) = \sum_{Y_i\in\mathcal{L}} \psi(Y_i, Y_j)\phi_i(Y_i) \prod_{k\in N_i\setminus \mathrm{j}} m_{k\to i}(Y_i), \forall Y_j \in \mathcal{L}$$

Sum over all states　　Prior

3. 收敛后，计算 $b_i(Y_i)$ = 节点i属于类Y_i的置信度

$$b_i(Y_i) = \phi_i(Y_i) \prod_{j\in N_i} m_{k\to i}(Y_i)$$



After convergence:

$b_i(Y_i)$ = node i's belief of being in class Y_i

Prior　　All messages from neighbors

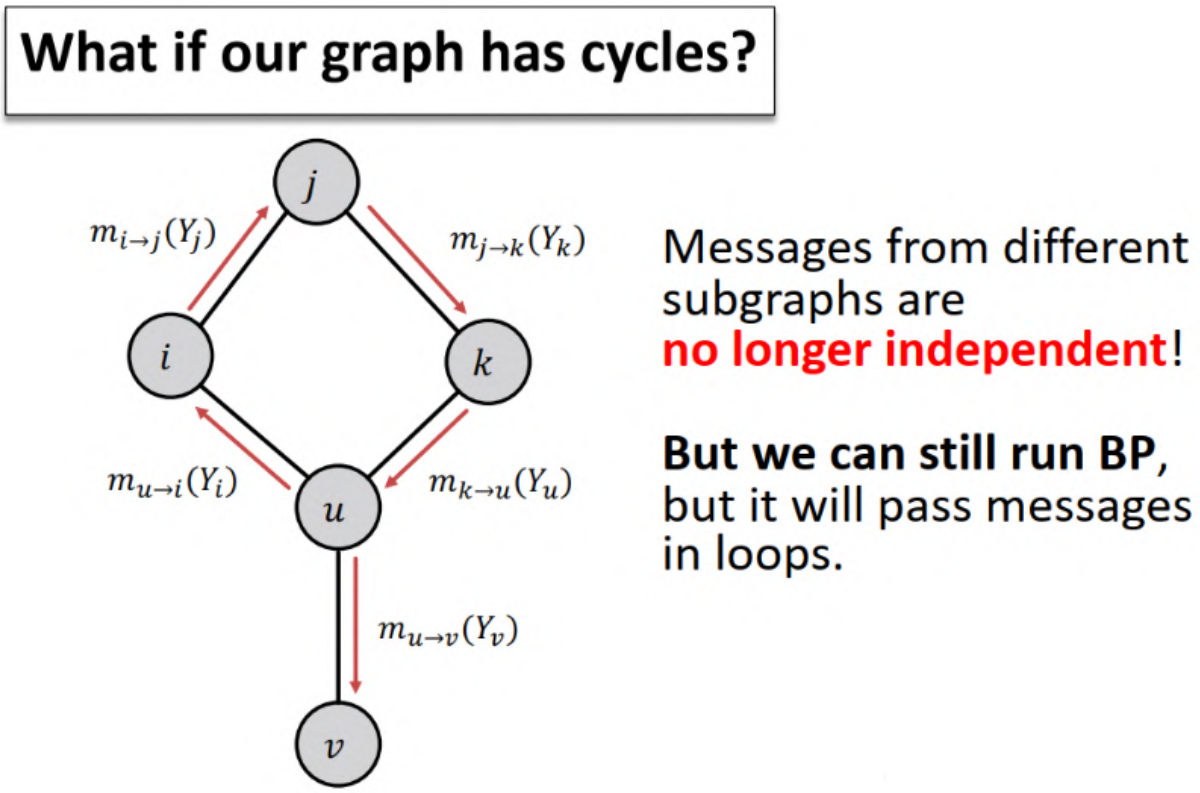
$$b_i(Y_i) = \phi_i(Y_i) \Pi_{j\in N_i} m_{j\to i}(Y_i), \forall\ Y_i \in \mathcal{L}$$

示例:

1. 现在我们考虑图中有环的情况，节点没有顺序了。我们采用上述算法，从随机节点开始，沿边更新邻居节点。

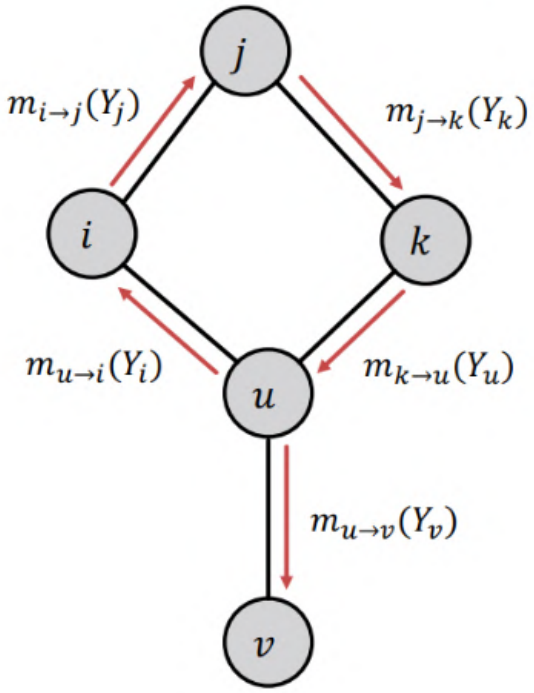
- Now we consider a graph with cycles
- There is no longer an ordering of nodes
- We apply the same algorithm as in previous slides:
  - Start from arbitrary nodes
  - Follow the edges to update the neighboring nodes

2. 由于图中有环，来自各子图的信息就不独立了。信息会在圈子里加强（就像 PageRank 里的 spider trap）

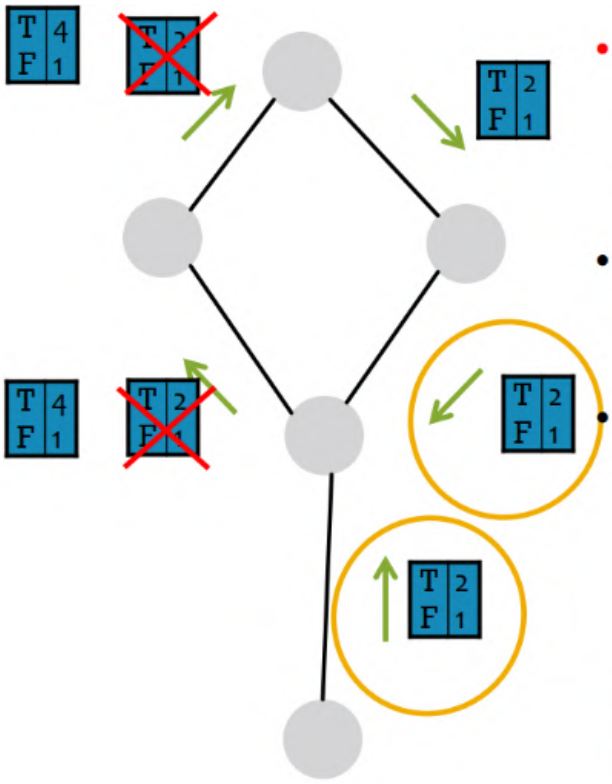


可能出现的问题：置信度不收敛（如图，信息在环里被加强了）

但是由于现实世界的真实复杂图会更像树，就算有环也会有弱连接，所以还是能用Loopy BP



- **Beliefs may not converge**
  - Message $m_{u\to i}(Y_i)$ is based on initial belief of i, not a **separate evidence** for i
  - The initial belief of i (which could be incorrect) is reinforced by the cycle $i \to j \to k \to u \to i$
- However, in practice, Loopy BP is still a good heuristic for complex graphs which contain many branches.



- Messages loop around and around: 2, 4, 8, 16, 32, ... More and more convinced that these variables are T!
- BP incorrectly treats this message as **separate evidence** that the variable is T (true).
- Multiplies these two messages as if they were independent.
  - But they don't actually come from *independent* parts of the graph.
  - One influenced the other (via a cycle).

This is an extreme example. Often in practice, the cyclic influences are weak. (As cycles are long or include at least one weak correlation.)

置信度传播方法的特点:

- **优点**:
  1. 易于编程及同步运算
  2. 可泛化到任何形式potentials（如高阶）的图模型上
- **挑战**：不一定能收敛（参考：尤其在闭环多的情况下）
- potential functions (parameters) (label-label potential matrix) 需要训练来估计

- **Advantages:**
  - Easy to program & parallelize
  - General: can apply to any graph model with any form of potentials
    - Potential can be higher order: e.g. $\psi(Y_i, Y_j, Y_k, Y_v \ldots)$
- **Challenges:**
  - Convergence is not guaranteed (when to stop), especially if many closed loops
- **Potential functions** (parameters)
  - Require training to estimate

6.5 总结

学习了如何利用图中的关系来对节点做预测。

主要技术：

1. relational classification
2. iterative classification
3. loopy belief propagation

- We learned how to leverage correlation in graphs to make prediction on nodes
- Key techniques:
  - Relational classification
  - Iterative classification
  - Loopy belief propagation

7. 图神经网络1：GNN模型

本章主要内容：

介绍深度学习基础。

介绍GNN思想：聚合邻居信息。

每一层都产生一种节点嵌入。将上一层的邻居信息聚合起来，连接本节点上一层信息，产生新的节点嵌入。

第一层节点嵌入就是节点特征。

GCN：用平均值作为聚合函数。

GraphSAGE：用各种聚合函数。

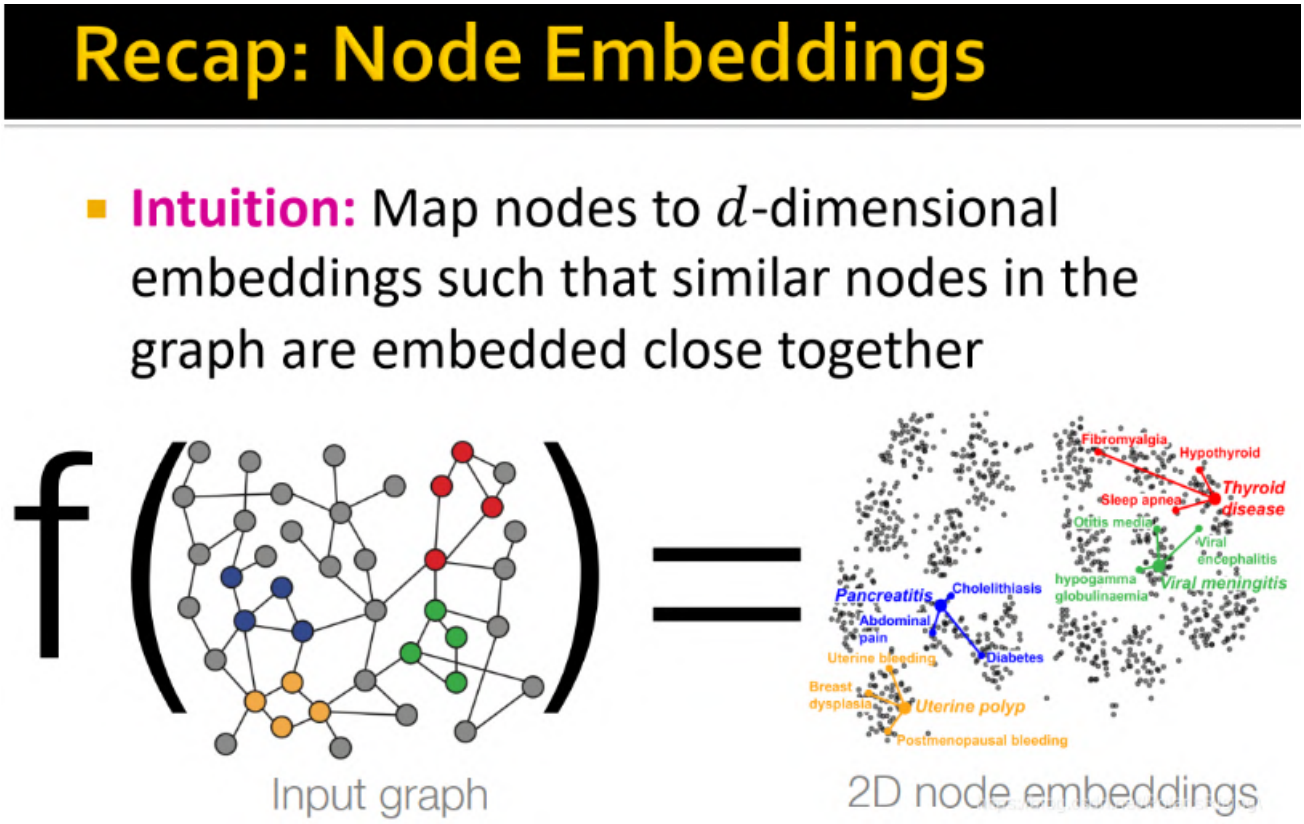
7.1 图神经网络1：GNN模型

回忆一下节点嵌入任务。其目的在于将节点映射到 d 维向量，使得在图中相似的节点在向量域中也相似。

我们已经学习了 “Shallow” Encoding 的方法来进行映射过程，也就是使用一个大矩阵直接储存每个节点的表示向量，通过矩阵与向量乘法来实现嵌入过程。

这种方法的缺陷在于：

- 需要 $O(|V|)$ 复杂度（矩阵的元素数，即表示向量维度 d ×节点数 $|V|$）的参数，太多了节点间参数不共享，每个节点的表示向量都是完全独特的。
- 直推式（transductive）：无法获取在训练时没出现过的节点的表示向量。
- 无法应用节点特征信息。



Recap: Shallow Encoders

- **Limitations** of shallow embedding methods:
  - $O(|V|)$ **parameters are needed:**
    - No sharing of parameters between nodes
    - Every node has its own unique embedding
  - **Inherently "transductive":**
    - Cannot generate embeddings for nodes that are not seen during training
  - **Do not incorporate node features:**
    - Many graphs have features that we can and should leverage

本节课将介绍deep graph encoders，也就是用图神经网络GNN来进行节点嵌入。

映射函数，即之前讲过的node embedding中的encoder：$ENC(v) =$ 基于图结构的多层非线性转换。

（对节点相似性的定义仍然可以使用之前Lecture 3中的DeepWalk、node2vec等方法）

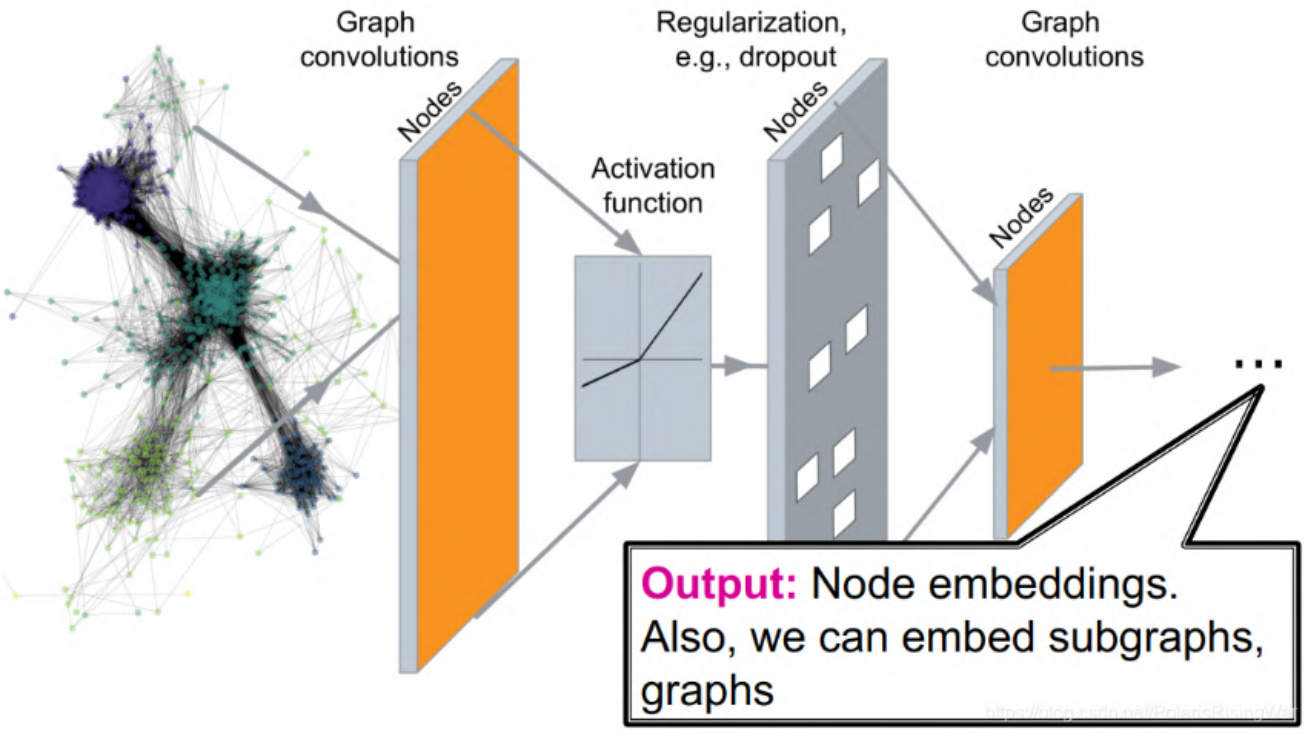
Today: Deep Graph Encoders

- **Today:** We will now discuss deep methods based on **graph neural networks (GNNs):**

$$\mathrm{ENC}(v) =$$ **multiple layers of non-linear transformations based on graph structure**

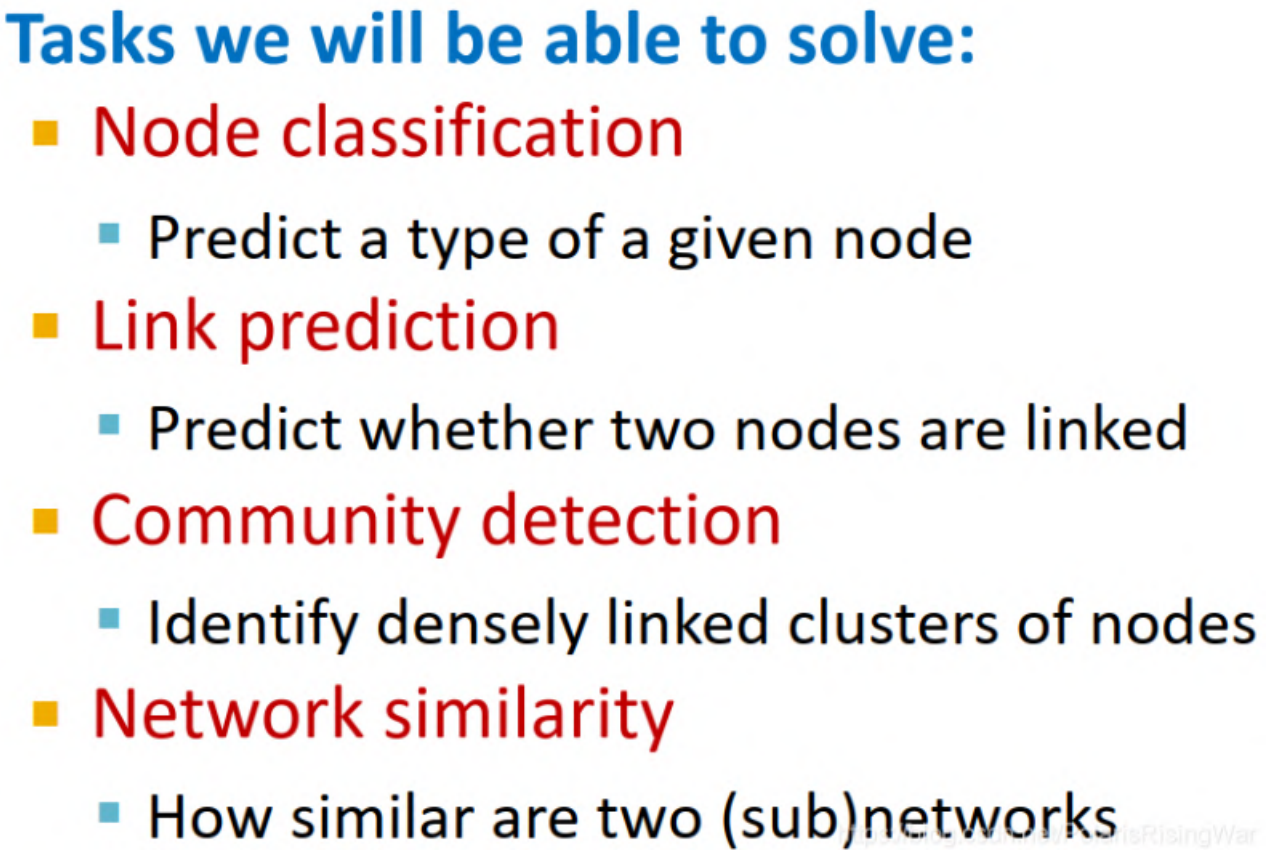
- **Note:** All these deep encoders can be **combined with node similarity functions** defined in the lecture 3

一个GNN网络的结构如图:



通过网络可以解决的任务有:

- 节点分类：预测节点的标签
- 链接预测：预测两点是否相连
- 社区发现：识别密集链接的节点簇
- 网络相似性：度量图/子图间的相似性



传统机器学习难以应用在图结构上：图上的节点不全是独立同分布的,因此传统的机器学习无法直接运用到图上。

7.2 深度学习基础

机器学习：一个优化任务

有监督学习：输入自变量 x，预测标签 y 。

将该任务视作一个优化问题：

$$\min_{\Theta} L(y, f(x))$$

Θ 是参数集合，优化对象，可以是一至多个标量、向量或矩阵。如在shallow encoder中 $\Theta = \{Z\}$ （就是embedding lookup，那个大矩阵）。

L 是目标函数/损失函数。

举例：$L2$ loss（回归任务），$L(y, f(x)) = ||y - f(x)||_2$ 。

其他常见损失函数：L1 loss、huber loss、max margin（hinge loss）、交叉熵（后文将详细介绍）……等。

- **Supervised learning:** we are given input x, and the goal is to predict label y
- **Input x can be:**
  - Vectors of real numbers
  - Sequences (natural language)
  - Matrices (images)
  - Graphs (potentially with node and edge features)
- **We formulate the task as an optimization problem**
- **Formulate the task as an optimization problem:**

$$\min_{\Theta} \mathcal{L}(y, f(x))$$

Objective function

- Θ: a set of **parameters** we optimize
  - Could contain one or more scalars, vectors, matrices ...
  - E.g. $\Theta = \{Z\}$ in the shallow encoder (the embedding lookup)
- $\mathcal{L}$: **loss function**. Example: L2 loss

$$\mathcal{L}(y, f(x)) = \|y - f(x)\|_2$$

  - Other common loss functions:
    - L1 loss, huber loss, max margin (hinge loss), cross entropy ...
    - See https://pytorch.org/docs/stable/nn.html#loss-functions

损失函数举例：常用于分类任务的**交叉熵（cross entropy）**

标签 y 是一个独热编码（所属类别索引的元素为1，其他元素为0）的分类向量，如 $y = [0, 0, 1, 0, 0]$ 。

输出结果 $f(x)$ 是经过softmax的概率分布向量，即 $f(x) = Softmax(g(x))$ ，如 $f(x) = [0.1, 0.3, 0.4, 0.1, 0.1]$ 。

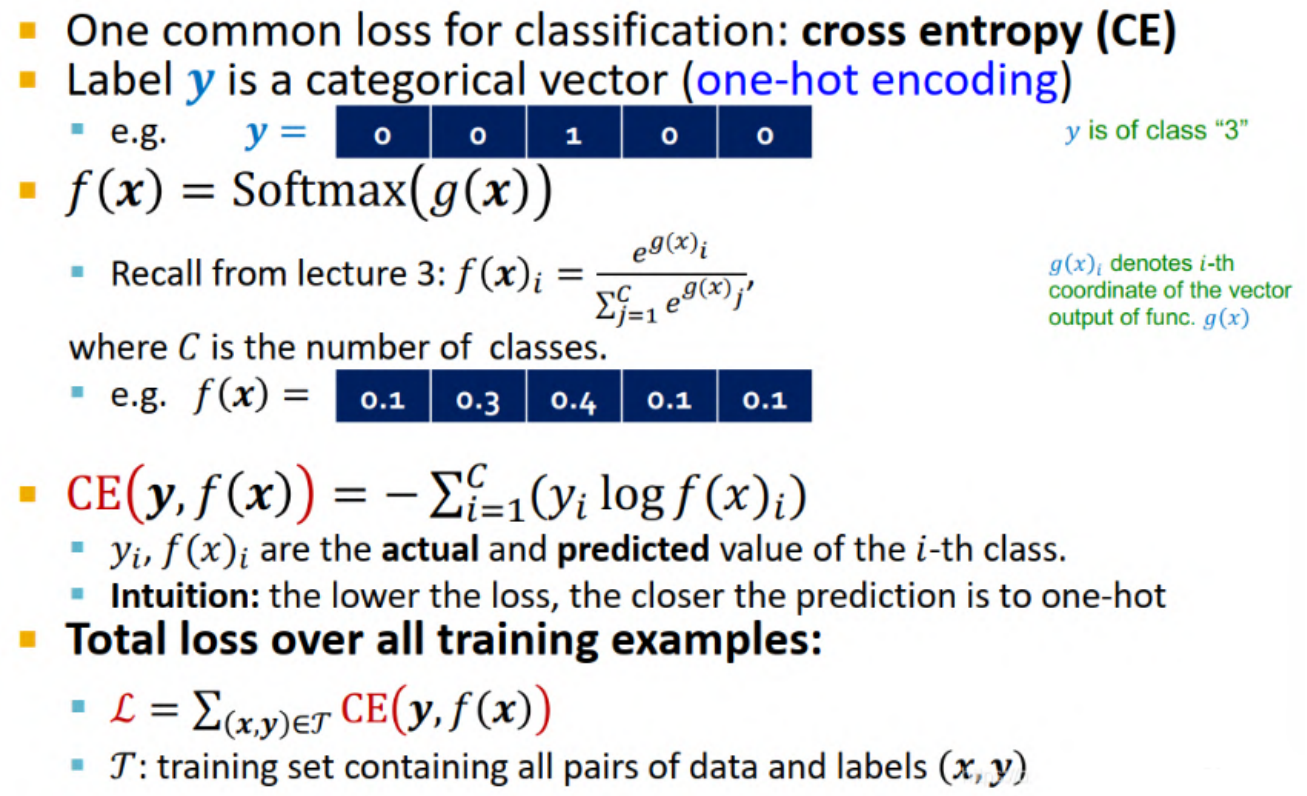
$$CE(y, f(x)) = -\sum_{i=1}^{C}(y_i log f(x)_i)$$

其中 C 是类别总数，下标 i 代表向量中第 i 个元素。CE 越低越好，越低说明预测值跟真实值越近。

在所有训练集数据上的总交叉熵：

$$L = \sum_{(x,y)\in T} CE(y, f(x))$$

其中 T 是所有训练集数据。



梯度向量 $\nabla_\Theta L$：函数增长最快的方向和增长率，每个元素是对应参数在损失函数上的偏微分。

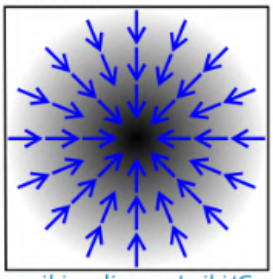
方向导数：函数在某个给定方向上的变化率。

梯度是函数增长率最快的方向的方向导数。

- **How to optimize the objective function?**
- **Gradient vector:** Direction and rate of fastest increase

Partial derivative

$$\nabla_\Theta \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \Theta_1}, \frac{\partial \mathcal{L}}{\partial \Theta_2}, \dots\right)$$



https://en.wikipedia.org/wiki/Gradient

  - $\Theta_1, \Theta_2 \dots$: components of Θ
- Recall **directional derivative** of a multi-variable function (e.g. $\mathcal{L}$) along a given vector represents the instantaneous rate of change of the function along the vector.
- Gradient is the directional derivative in the **direction of largest increase**

梯度下降:

迭代：将参数向梯度负方向更新：$\Theta \leftarrow \Theta - \eta \nabla_\Theta L$

学习率learning rate η 是一个需要设置的超参数，控制梯度下降每一步的步长，可以在训练过程中改变（有时想要学习率先快后慢：LR scheduling）

理想的停止条件是梯度为0，在实践中一般则是用“验证集上的表现不再提升”作为停止条件。（据我的经验一般是设置最大迭代次数，如果在验证集上表现不再增加就提前停止迭代（early stopping））

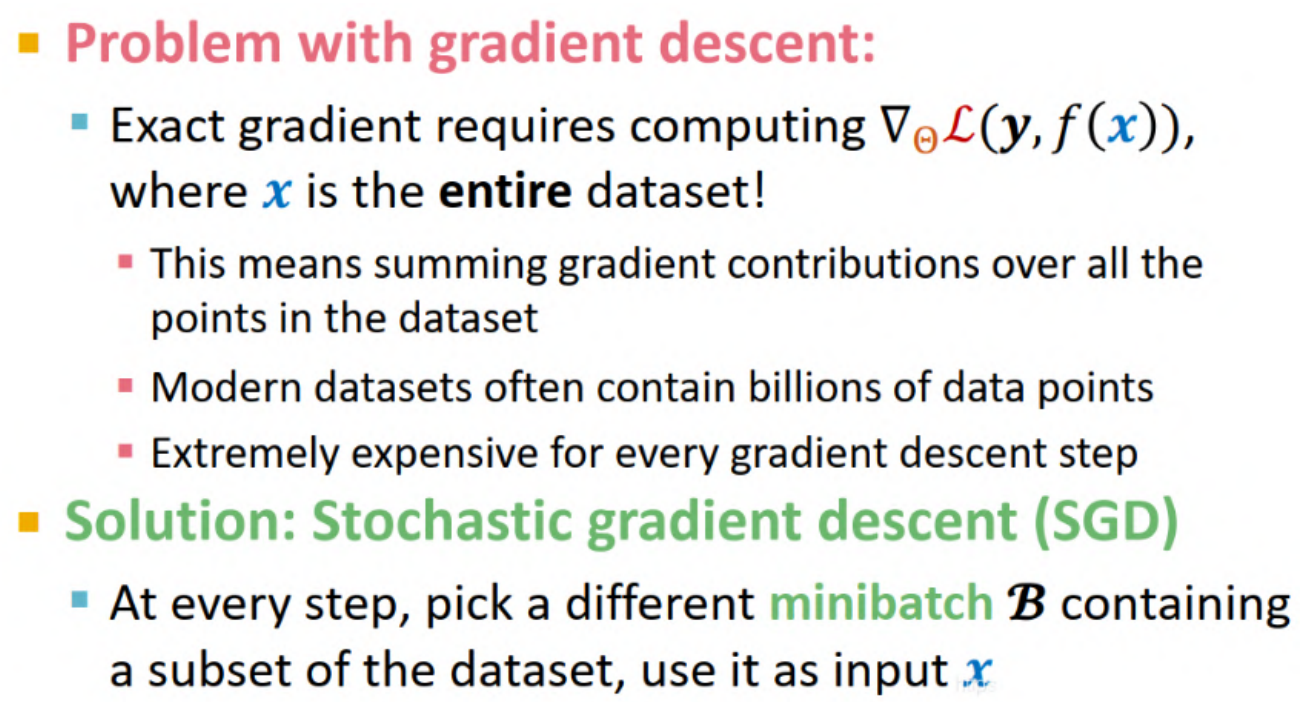
- **Iterative algorithm:** repeatedly update weights in the (opposite) direction of gradients until convergence

$$\Theta \leftarrow \Theta - \eta \nabla_\Theta \mathcal{L}$$

- **Training:** Optimize Θ iteratively
  - **Iteration**: 1 step of gradient descent
- **Learning rate (LR) η:**
  - Hyperparameter that controls the size of gradient step
  - Can vary over the course of training (LR scheduling)
- Ideal termination condition: **0** gradient
  - In practice, we stop training if it no longer improves performance on **validation set** (part of dataset we hold out from training)

随机梯度下降stochastic gradient descent (SGD)：

每一次梯度下降都需要计算所有数据集上的梯度，耗时太久，因此我们使用SGD的方法，将数据分成多个minibatch，每次用一个minibatch来计算梯度。



minibatch SGD： SGD是梯度的无偏估计，但不保证收敛，所以一般需要调整学习率。

对SGD的改进优化器：Adam，Adagrad，Adadelta，RMSprop……等。

一些概念：

- batch size：每个minibatch中的数据点数
- iteration：在一个minibatch上做一次训练
- epoch：在整个数据集上做一次训练（在一个epoch中iteration的数量是 $\frac{dataset\_size}{batch\_size}$ ）

- **Concepts:**
  - **Batch size**: the number of data points in a minibatch
    - E.g. number of nodes for node classification task
  - **Iteration**: 1 step of SGD on a minibatch
  - **Epoch**: one full pass over the dataset (# iterations is equal to ratio of dataset size and batch size)

- SGD is unbiased estimator of full gradient:
  - But there is no guarantee on the rate of convergence
  - In practice often requires tuning of learning rate
- Common optimizer that improves over SGD:
  - Adam, Adagrad, Adadelta, RMSprop ...

神经网络目标函数:

$$\min_{\Theta} L(y, f(x))$$

深度学习中的 f 可能非常复杂，为了简化，先假设一个线性函数：$f(x) = W \cdot x$ ， $\Theta = \{W\}$ 。

- 如果 f 返回一个标量，则 W 是一个可学习的向量 $\nabla_W f = (\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial w_3}, \dots)$ 。
- 如果 f 返回一个向量，则 W 是一个权重矩阵 $\nabla_W f = f$的雅可比矩阵 。

- **Objective:** $\min_{\Theta} \mathcal{L}(y, f(x))$
- In deep learning, the function f can be very complex
- To start simple, consider linear function

$$f(x) = W \cdot x, \qquad \Theta = \{W\}$$

- If f returns a scalar, then W is a learnable **vector**

$$\nabla_W f = (\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial w_3} \dots)$$

- If f returns a vector, then W is the **weight matrix**

$$\nabla_W f = W^T$$

Jacobian matrix of f

反向传播（Back Propagation）：

对更复杂的函数，如 $f(x) = W_2(W_1 x)$，$\Theta = \{W_1, W_2\}$。

将该函数视为：$h(x) = W_1 x$，$f(h) = W_2 h$。

应用链式法则计算梯度：$\nabla_x f = \frac{\partial f}{\partial h} \cdot \frac{\partial h}{\partial x} = \frac{\partial f}{\partial (W_1 x)} \cdot \frac{\partial (W_1 x)}{\partial x}$。

反向传播就是应用链式法则反向计算梯度，最终得到 L 关于参数的梯度。

- How about a more complex function:

$$f(x) = W_2(W_1 x), \qquad \Theta = \{W_1, W_2\}$$

- Recall **chain rule**:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

In other words:
$f(x) = W_2(W_1 x)$
$h(x) = W_1 x$
$g(z) = W_2 z$

- E.g. $\nabla_x f = \frac{\partial f}{\partial (W_1 x)} \cdot \frac{\partial (W_1 x)}{\partial x}$

- **Back-propagation**: Use of **chain rule** to propagate gradients of intermediate steps, and finally obtain gradient of $\mathcal{L}$ w.r.t. Θ

神经网络举例：

简单两层线性网络 $f(x) = g(h(x)) = W_2(W_1 x)$

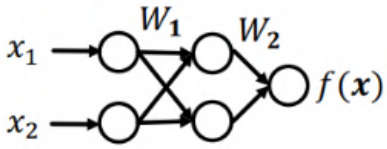
在一个minibatch上的 $L2$ loss：$L_{(x,y)\in \mathrm{B}} = ||y - f(x)||_2$

隐藏层：x 的中间表示向量。这里我们用 $h(x) = W_1 x$ 来表示隐藏层，$f(x) = W_2 h(x)$。

前向传播：从输入计算输出，用输出计算loss。

反向传播：计算梯度 $\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial W_2}$，$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial W_2} \cdot \frac{\partial W_2}{\partial W_1}$

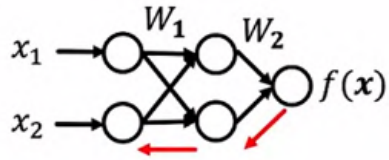
- **Example:** Simple 2-layer linear network
- $f(\boldsymbol{x}) = g\big(h(x)\big) = W_2(W_1\boldsymbol{x})$



- $\mathcal{L} = \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{B}} \left|\left|\big(\boldsymbol{y}, -f(\boldsymbol{x})\big)\right|\right|_2$ sums the L2 loss in a minibatch $\mathcal{B}$
- **Hidden layer:** intermediate representation for input $\boldsymbol{x}$
  - Here we use $h(x) = W_1\boldsymbol{x}$ to denote the hidden layer
  - $f(\boldsymbol{x}) = W_2 h(x)$

- **Forward propagation:**
  Compute loss starting from input
  - $\boldsymbol{x} \longrightarrow h \longrightarrow g \longrightarrow \mathcal{L}$
    Multiply W_1 Multiply W_2 Loss

Remember:
$f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x})$
$h(x) = W_1\boldsymbol{x}$
$g(z) = W_2 z$



- **Back-propagation to compute gradient of**

$$\Theta = \{W_1, W_2\}$$

Start from loss, compute the gradient

$$\frac{\partial\mathcal{L}}{\partial W_2} = \frac{\partial\mathcal{L}}{\partial f}\cdot\frac{\partial f}{\partial W_2}, \qquad \frac{\partial\mathcal{L}}{\partial W_1} = \frac{\partial\mathcal{L}}{\partial f}\cdot\frac{\partial f}{\partial W_2}\cdot\frac{\partial W_2}{\partial W_1}$$

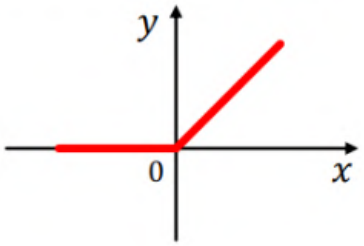
Compute backwards Compute backwards

非线性:

- ReLU: $ReLU(x) = max(x, 0)$
- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$

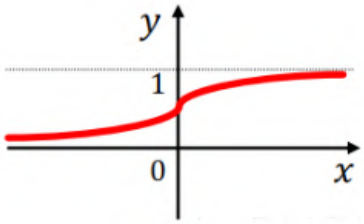
- Note that in $f(\boldsymbol{x}) = W_2(W_1\boldsymbol{x})$, W_2W_1 is another matrix (vector, if we do binary classification)
- Hence $f(\boldsymbol{x})$ is still linear w.r.t. $\boldsymbol{x}$ no matter how many weight matrices we compose
- **Introduce non-linearity:**
  - **Rectified linear unit (ReLU)**

    $ReLU(x) = \max(x, 0)$



  - **Sigmoid**

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



多层感知器Multi-layer Perceptron (MLP):

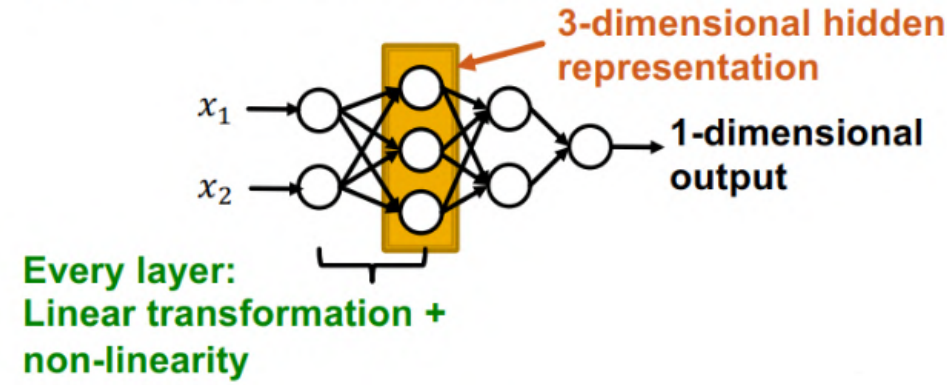
MLP每一层都是线性转换和非线性结合

$$x^{(l+1)} = \sigma(W_l x^{(l)} + b^l)$$

- **Each layer of MLP combines linear transformation and non-linearity:**

$$\boldsymbol{x}^{(l+1)} = \sigma(W_l\boldsymbol{x}^{(l)} + b^l)$$

  - where W_l is weight matrix that transforms hidden representation at layer l to layer $l+1$
  - b^l is bias at layer l, and is added to the linear transformation of $\boldsymbol{x}$
  - σ is non-linearity function (e.g., sigmod)
- Suppose $\boldsymbol{x}$ is 2-dimensional, with entries x_1 and x_2



总结:

- **Objective function:**

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x}))$$

- f can be a simple linear layer, an MLP, or other neural networks (e.g., a GNN later)
- Sample a minibatch of input $\boldsymbol{x}$
- **Forward propagation:** compute $\mathcal{L}$ given $\boldsymbol{x}$
- **Back-propagation:** obtain gradient $\nabla_{\Theta}\mathcal{L}$ using a chain rule
- Use **stochastic gradient descent (SGD)** to optimize for Θ over many iterations

7.3 图上的深度学习

本节内容:

1. local network neighborhoods
    - 聚合策略
    - 计算图
2. 叠层
    - 模型、参数、训练
    - 如何学习?
    - 无监督和有监督学习举例

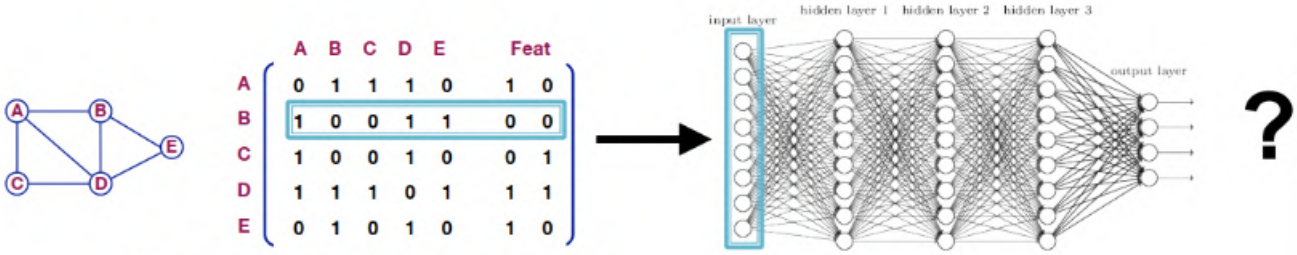
Setup：图 G，节点集 V，邻接矩阵 A（二元，无向无权图。这些内容都可以泛化到其他情况下），节点特征矩阵 $X \in \mathbb{R}^{m \times |V|}$，一个节点 v，v 的邻居集合 $N(v)$。如果数据集中没有节点特征，可以用指示向量indicator vectors（节点的独热编码），或者所有元素为常数1的向量。有时也会用节点度数来作为特征。

- **Assume we have a graph G:**
  - V is the **vertex set**
  - $\boldsymbol{A}$ is the **adjacency matrix** (assume binary)
  - $\boldsymbol{X} \in \mathbb{R}^{m \times |V|}$ is a matrix of **node features**
  - v: a node in V; $N(v)$: the set of neighbors of v.
  - **Node features:**
    - Social networks: User profile, User image
    - Biological networks: Gene expression profiles, gene functional information
    - When there is no node feature in the graph dataset:
      - Indicator vectors (one-hot encoding of a node)
      - Vector of constant 1: [1, 1, ..., 1]

我们可能很直接地想到，将邻接矩阵和特征合并在一起应用在深度神经网络上（如图，直接一个节点的邻接矩阵+特征合起来作为一个观测）。这种方法的问题在于：

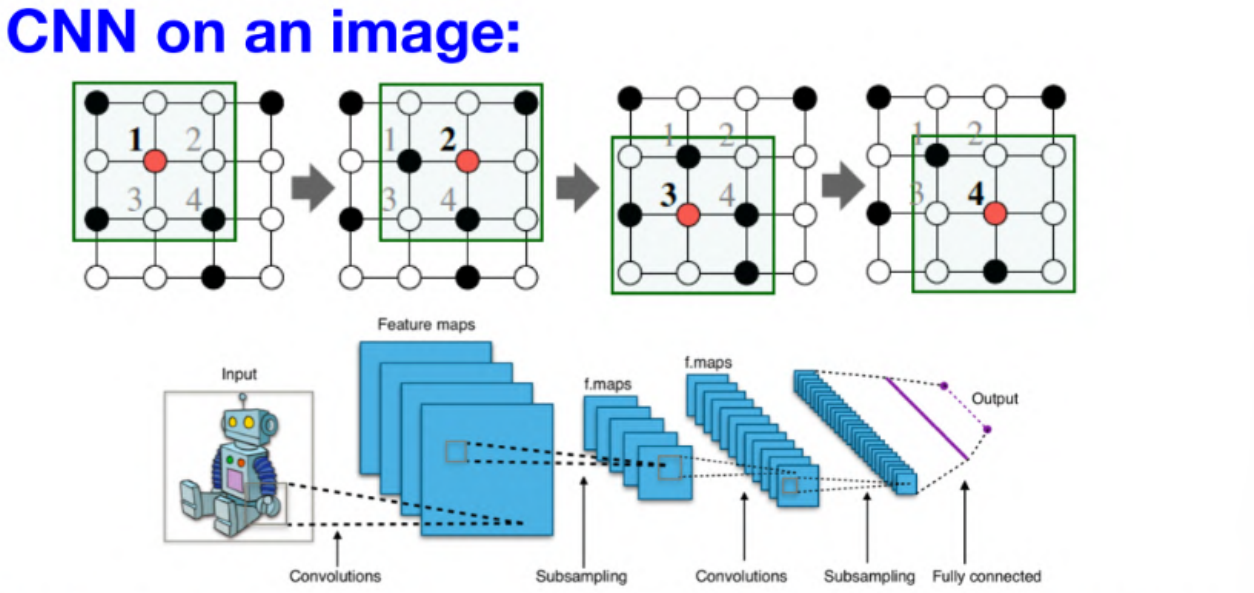
- 需要 $O(|V|)$ 的参数。
- 不适用于不同大小的图。
- 对节点顺序敏感（我们需要一个即使改变了节点顺序，结果也不会变的模型）。

- Join adjacency matrix and features
- Feed them into a deep neural net:



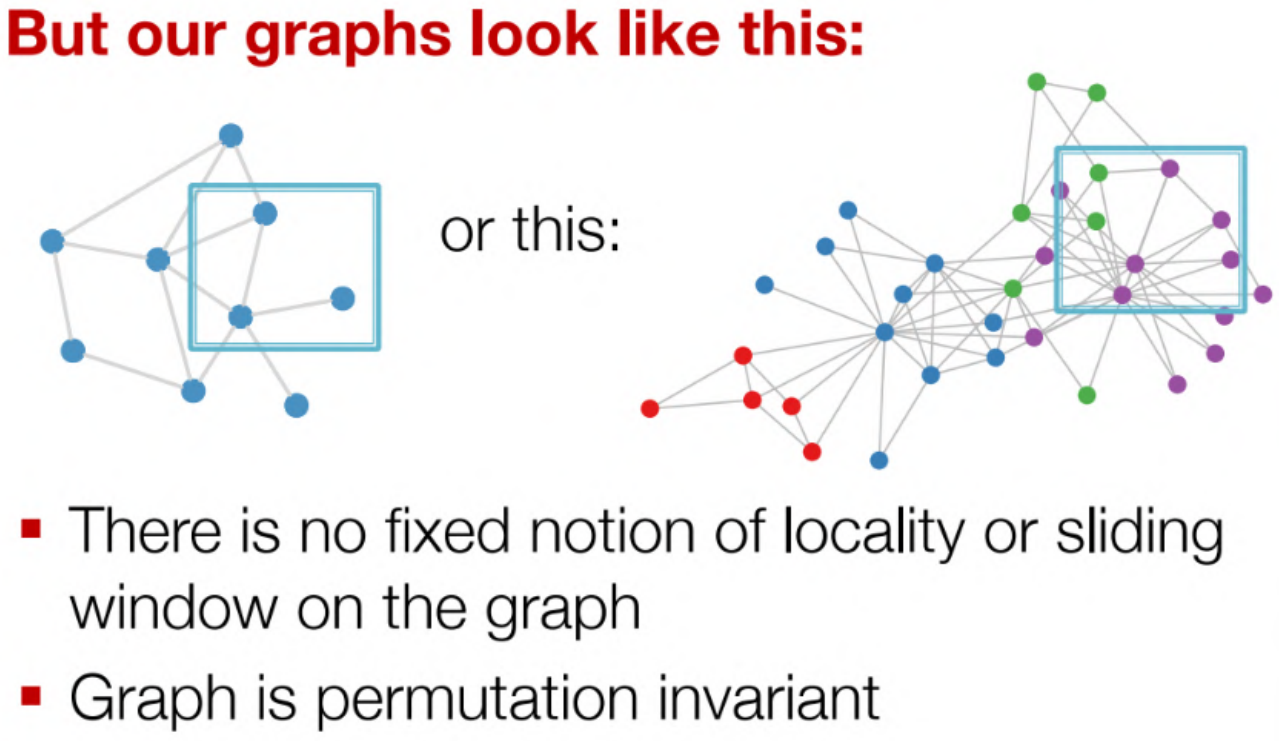
- Issues with this idea:
  - $O(|V|)$ parameters
  - Not applicable to graphs of different sizes
  - Sensitive to node ordering

将网格上的卷积神经网络泛化到图上，并应用到节点特征数据：



Goal is to generalize convolutions beyond simple lattices
Leverage node features/attributes (e.g., text, images)

图上无法定义固定的locality或滑动窗口，而且图是permutation invariant的（节点顺序不固定）：

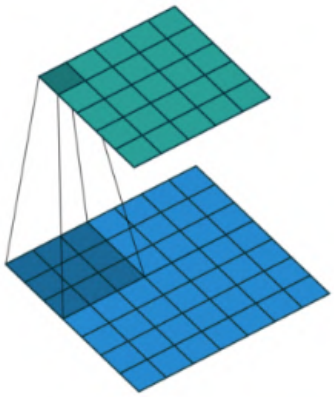


- There is no fixed notion of locality or sliding window on the graph
- Graph is permutation invariant

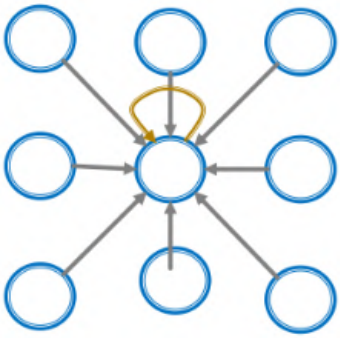
从image到graph：聚合邻居信息

过程：转换邻居信息 $W_i h_i$，将其加总 $\sum_i W_i h_i$。

Single Convolutional neural network (CNN) layer with 3x3 filter:



Image



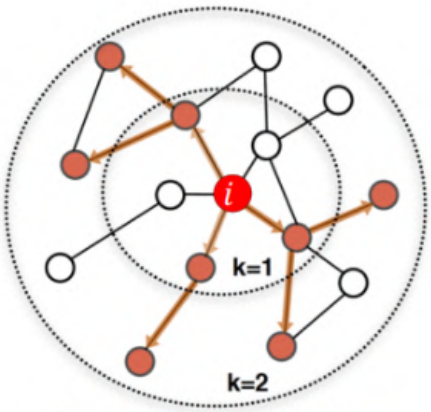
Graph

Idea: transform information at the neighbors and combine it:

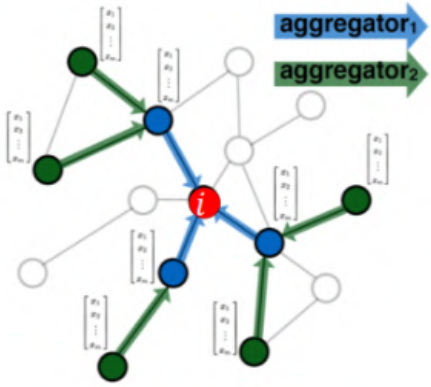
- Transform "messages" h_i from neighbors: $W_i\ h_i$
- Add them up: $\sum_i W_i\ h_i$

图卷积神经网络（Graph Convolutional Networks）：通过节点邻居定义其计算图，传播并转换信息，计算出节点表示（可以说是用邻居信息来表示一个节点）

Idea: Node's neighborhood defines a computation graph



Determine node computation graph

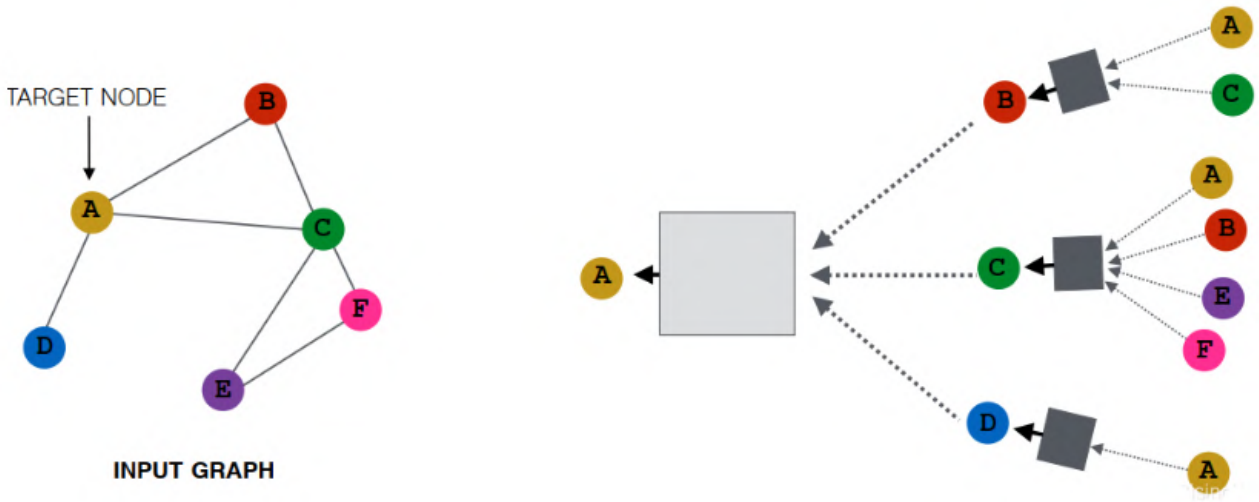


Propagate and transform information

Learn how to propagate information across the graph to compute node features

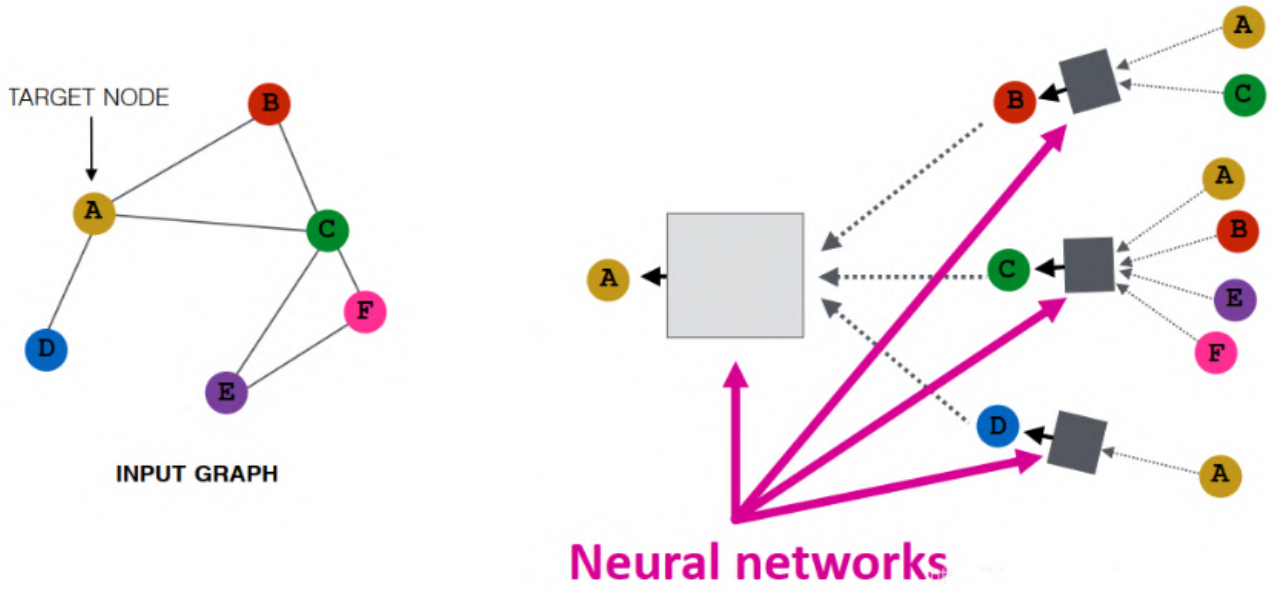
核心思想：通过聚合邻居来生成节点嵌入

- **Key idea:** Generate node embeddings based on **local network neighborhoods**

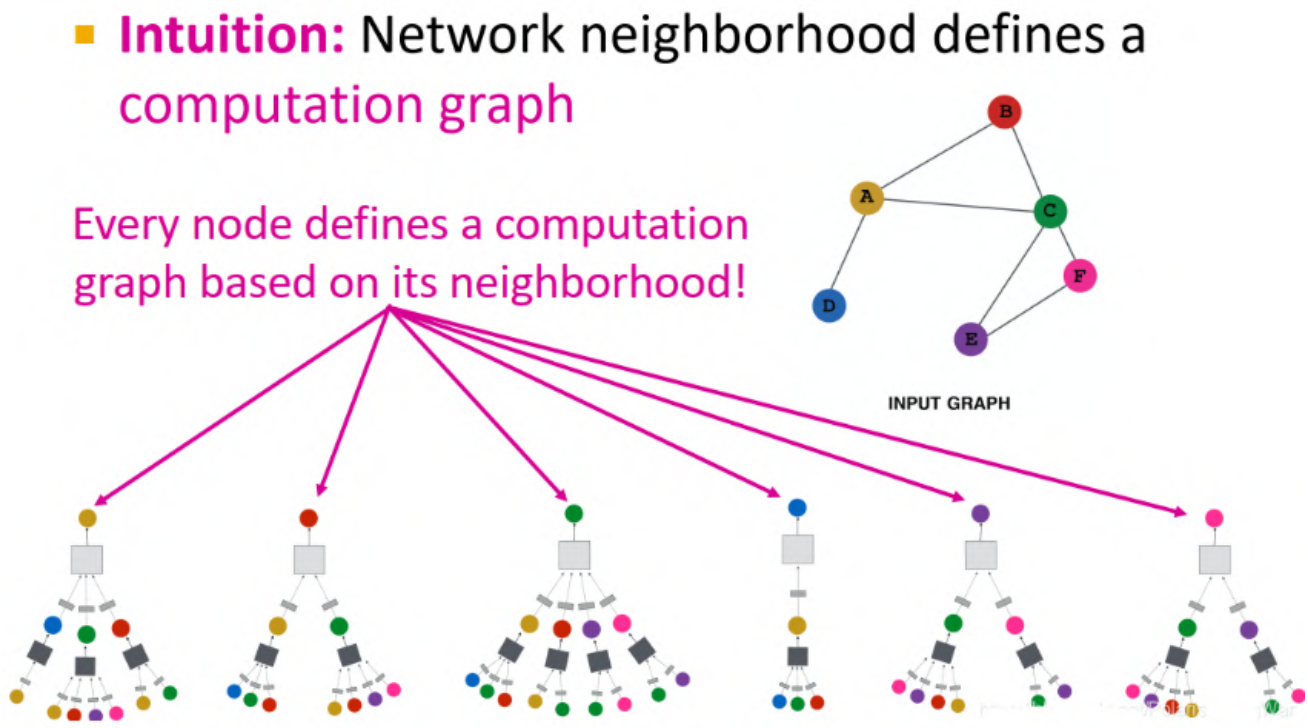


直觉：通过神经网络聚合邻居信息

- **Intuition:** Nodes aggregate information from their neighbors using neural networks



直觉：通过节点邻居定义计算图（它的邻居是子节点，子节点的邻居又是子节点们的子节点……）

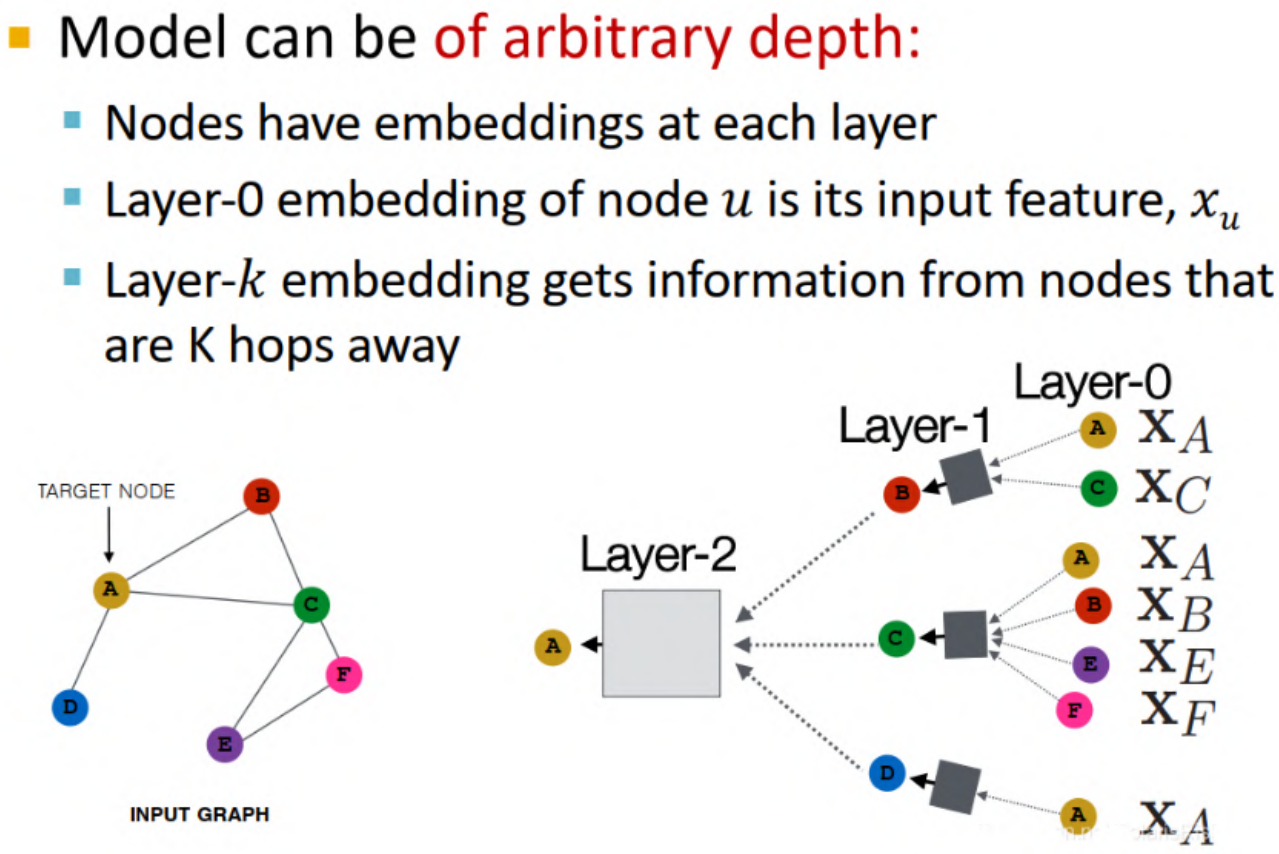


深度模型就是有很多层。

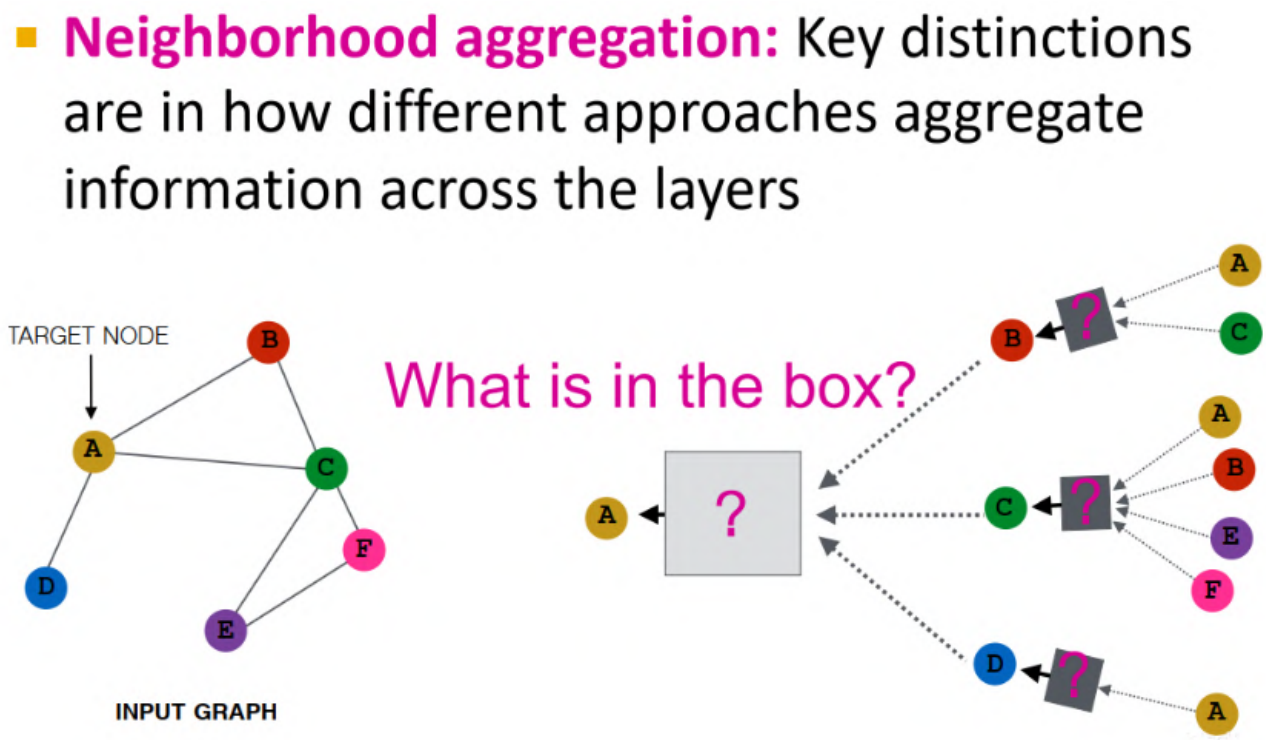
节点在每一层都有不同的表示向量，每一层节点嵌入是邻居上一层节点嵌入再加上它自己（相当于添加了自环）的聚合。

第0层是节点特征，第k层是节点通过聚合k hop邻居所形成的表示向量。

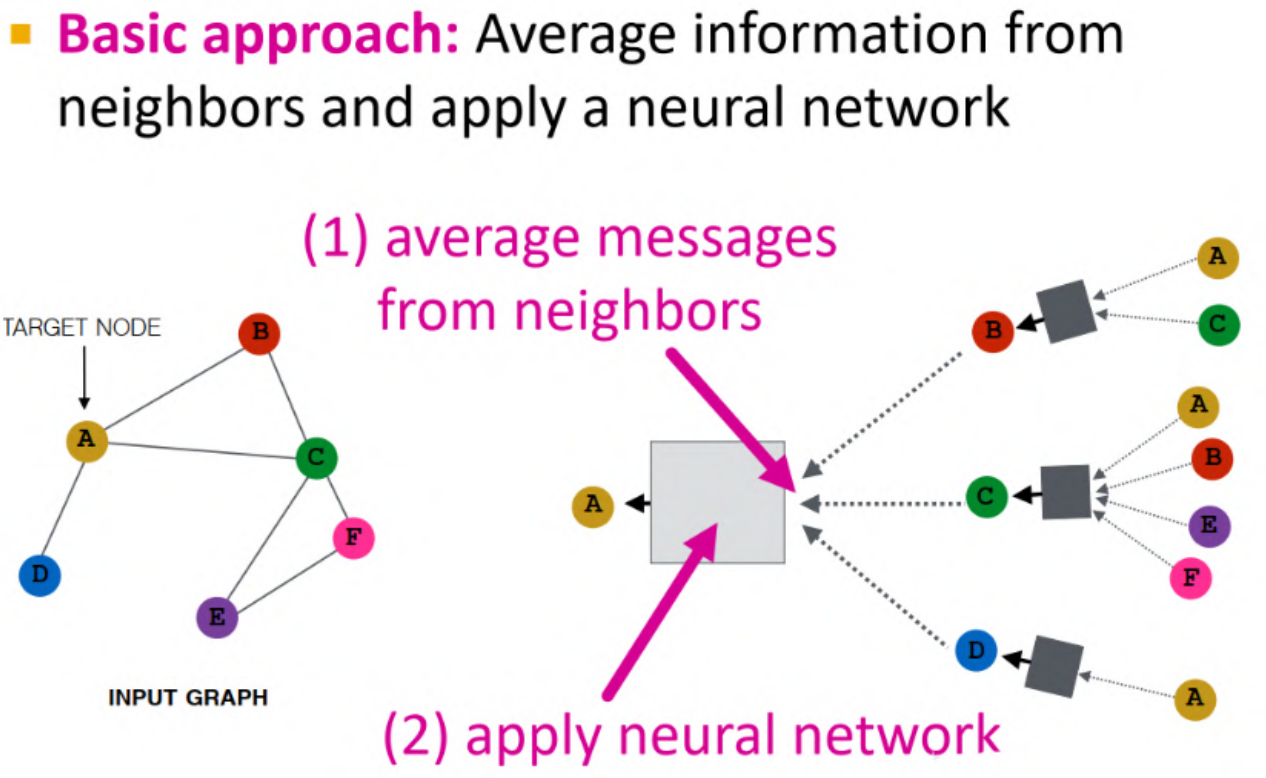
在这里就没有收敛的概念了，直接选择跑有限步（k）层。



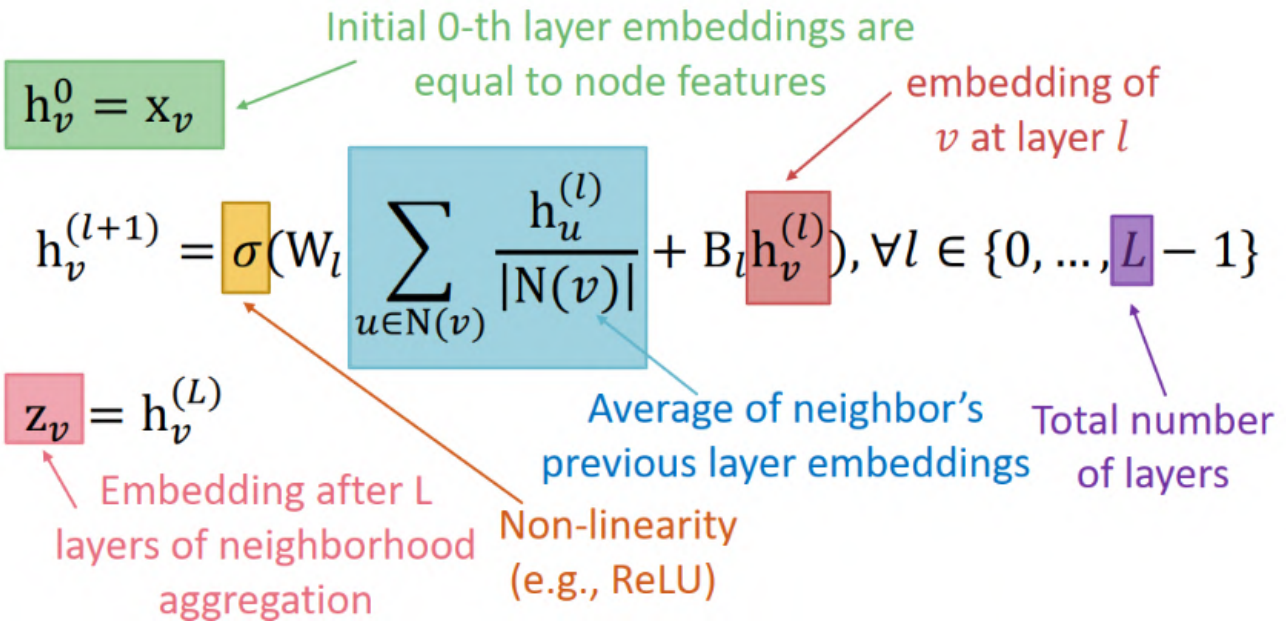
邻居信息聚合（neighborhood aggregation）：不同聚合方法的区别就在于如何跨层聚合邻居节点信息。neighborhood aggregation方法必须要order invariant或者说permutation invariant。



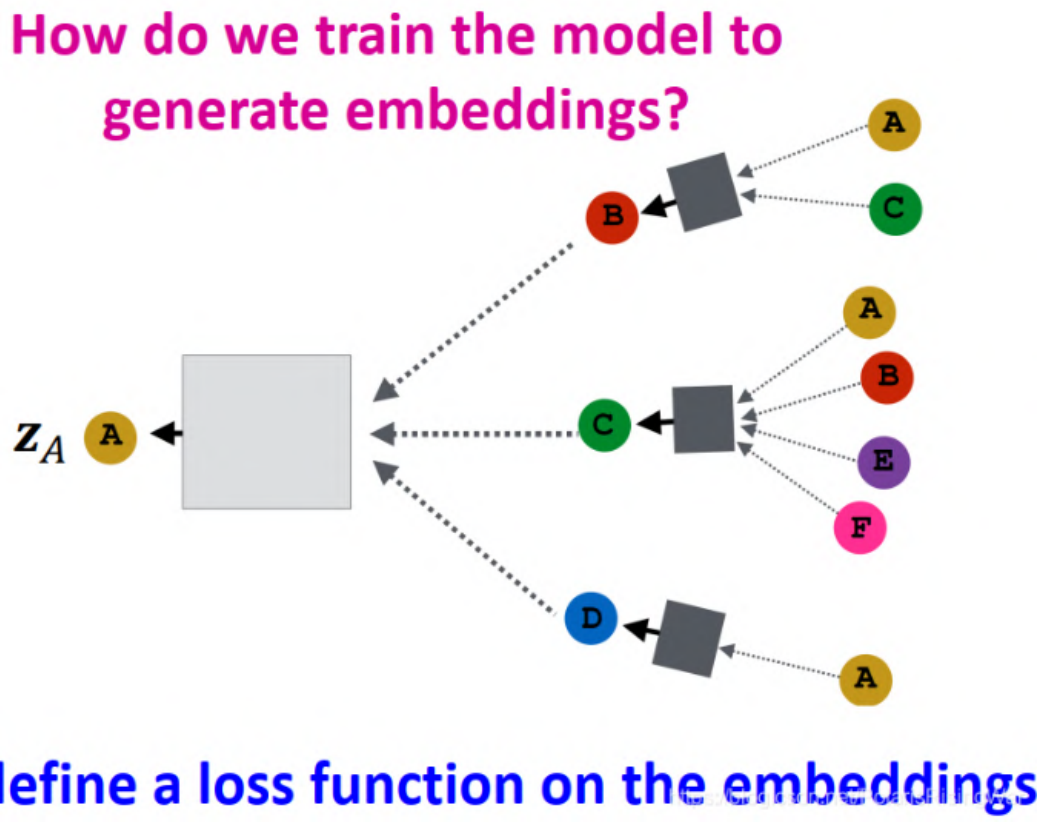
基础方法：从邻居获取信息求平均，再应用神经网络



这种deep encoder的数学公式：



如何训练模型：需要定义节点嵌入上的损失函数



h_v^l 是 l 层 v 的隐藏表示向量。

模型上可以学习的参数有 W_l （neighborhood aggregation的权重）和 B_l （转换节点自身隐藏向量的权重），注意，每层参数在不同节点之间是共享的。

可以通过将输出的节点表示向量输入损失函数中，运行SGD来训练参数。

Trainable weight matrices
(i.e., what we learn)

$$h_v^{(0)} = x_v$$

$$h_v^{(l+1)} = \sigma\left(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}\right), \forall l \in \{0, \ldots, L-1\}$$

$$z_v = h_v^{(L)}$$

Final node embedding

We can feed these **embeddings into any loss function** and run SGD to **train the weight parameters**

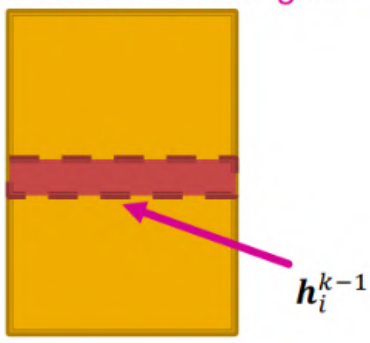
h_v^l: the hidden representation of node v at layer l

- W_k: weight matrix for neighborhood aggregation
- B_k: weight matrix for transforming hidden vector of self

矩阵形式：很多种聚合方式都可以表示为（稀疏）矩阵操作的形式，如这个基础方法可以表示成图中这种形式

- **Many aggregations can be performed efficiently by (sparse) matrix operations**
- Let $H^{(l)} = [h_1^{(l)} \ldots h_{|V|}^{(l)}]^T$
- Then: $\sum_{u \in N_v} h_u^{(l)} = A_{v,:} H^{(l)}$
- Let D be diagonal matrix where $D_{v,v} = \text{Deg}(v) = |N(v)|$
  - The inverse of D: D^{-1} is also diagonal: $D_{v,v}^{-1} = 1/|N(v)|$
- **Therefore,**

Matrix of hidden embeddings H^{k-1}



$$\sum_{u \in N(v)} \frac{h_u^{(l-1)}}{|N(v)|} \Rightarrow H^{(l+1)} = D^{-1} A H^{(l)}$$

补充：向量点积/矩阵乘法就是逐元素相乘然后累加，对邻接矩阵来说相当于对存在边的元素累加

对整个公式的矩阵化也可以实现：

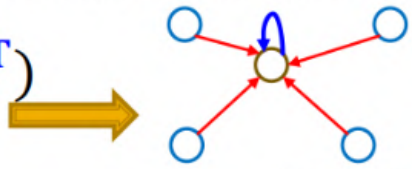
这样就可以应用有效的稀疏矩阵操作。

同时，也要注意，当aggregation函数过度复杂时，GNN可能无法被表示成矩阵形式。

- Re-writing update function in matrix form:

$$H^{(l+1)} = \sigma(\tilde{A}H^{(l)}W_l^{\mathrm{T}} + H^{(l)}B_l^{\mathrm{T}})$$

where $\tilde{A} = D^{-1}A$



$$H^{(l)} = [h_1^{(l)} \dots h_{|V|}^{(l)}]^T$$

  - Red: neighborhood aggregation
  - Blue: self transformation

- In practice, this implies that efficient sparse matrix multiplication can be used ($\tilde{A}$ is sparse)

- **Note**: not all GNNs can be expressed in matrix form, when aggregation function is complex

如何训练GNN：节点嵌入 $\mathbf{z}_v$ 。

- **监督学习**：优化目标 $\min_\Theta L(\mathbf{y}, f(\mathbf{z}_v))$

  如回归问题可以用L2 loss，分类问题可以用交叉熵。

  比如二分类交叉熵：$L = -\sum_{v \in V}(y_v log(\sigma(z_v^T\theta)) + (1 - y_v)log(1 - \sigma(z_v^T\theta)))$，其中 z_v 是encoder输出的节点嵌入向量，θ 是classification weight。这个式子中，前后两个加数只有一个会被用到（y要么是1要么是0）

- **无监督学习**：用图结构作为学习目标

  比如节点相似性（随机游走、矩阵分解、图中节点相似性……等）

  损失函数为 $L = \sum_{z_u, z_v} CE(y_{u,v}, DEC(z_u, z_v))$，如果 u 和 v 相似，则 $y_{u,v} = 1$，CE 是交叉熵，DEC 是节点嵌入的 decoder 。

- Node embedding $\boldsymbol{z}_v$ is a function of input graph
- **Supervised setting**: we want to minimize the loss $\mathcal{L}$ (see also slide 15):

$$\min_{\Theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{z}_v))$$

  - $\boldsymbol{y}$: node label
  - $\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical
- **Unsupervised setting:**
  - No node label available
  - **Use the graph structure as the supervision!**

Unsupervised Training

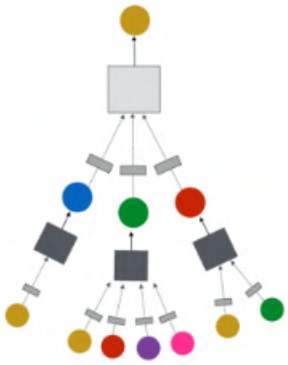
- **"Similar" nodes have similar embeddings**

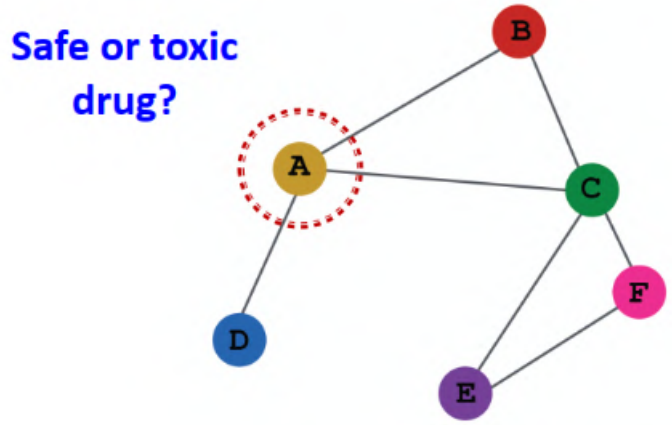
$$\mathcal{L} = \sum_{z_u, z_v} \mathrm{CE}(y_{u,v}, \mathrm{DEC}(z_u, z_v))$$

  - Where $y_{u,v} = 1$ when node u and v are **similar**
  - CE is the cross entropy (slide 16)
  - DEC is the decoder such as inner product (lecture 4)
- **Node similarity** can be anything from lecture 3, e.g., a loss based on:
  - **Random walks** (node2vec, DeepWalk, struc2vec)
  - **Matrix factorization**
  - **Node proximity in the graph**

Supervised Training

Directly train the model for a supervised task (e.g., node classification)





E.g., a drug-drug interaction network

Supervised Training

Directly train the model for a supervised task (e.g., **node classification**)

- Use cross entropy loss (slide 16)

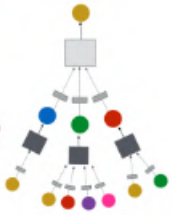
$$\mathcal{L} = \sum_{v \in V} y_v \log(\sigma(z_v^T \theta)) + (1 - y_v) \log(1 - \sigma(z_v^T \theta))$$

Encoder output: node embedding

Classification weights

Node class label

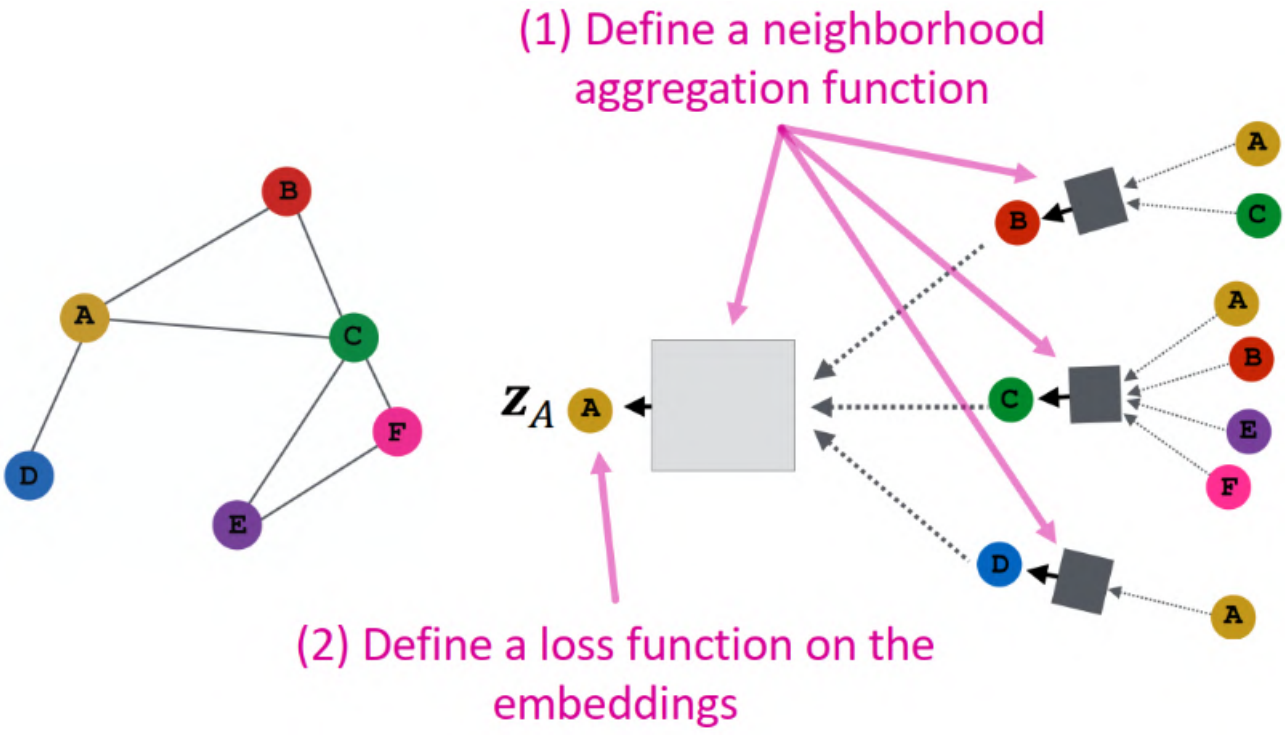
Safe or toxic drug?

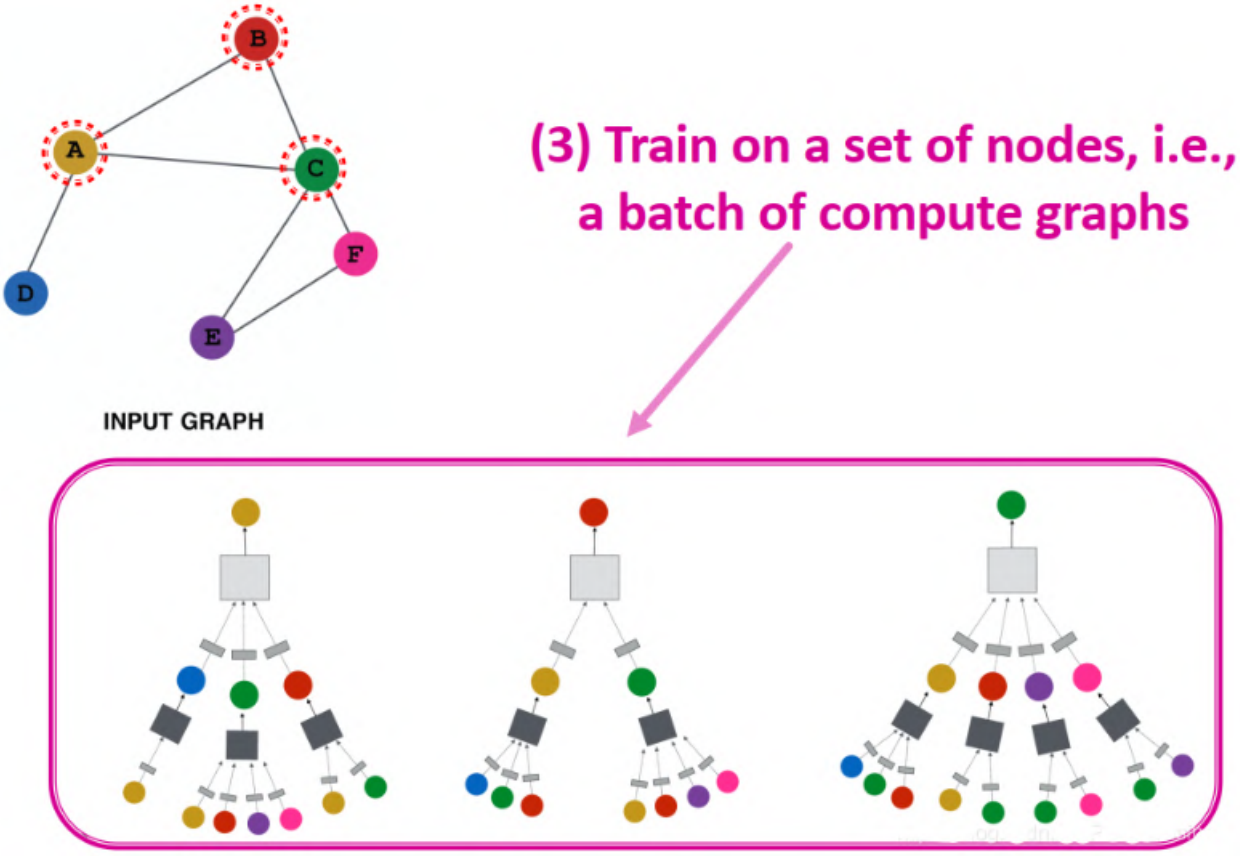


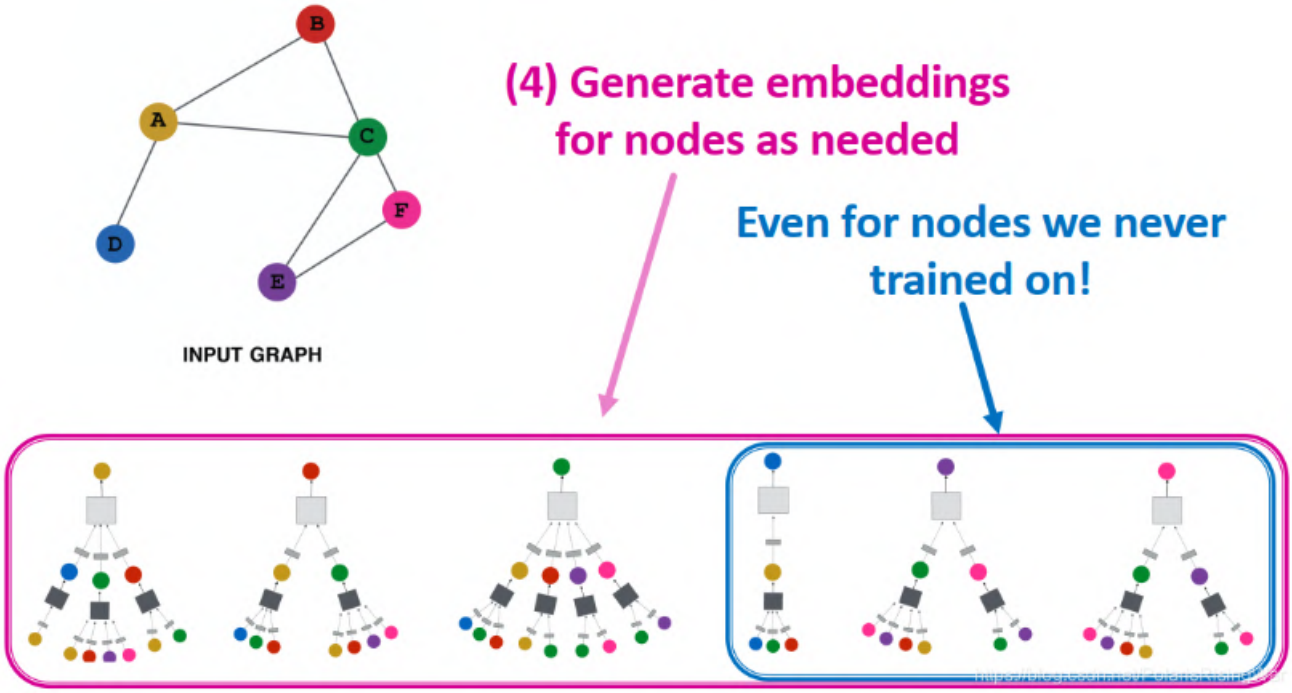
模型设计：overview

1. 定义邻居聚合函数
2. 定义节点嵌入上的损失函数

3. 在节点集合（如计算图的batch）上做训练
4. 训练后的模型可以应用在训练过与没有训练过的节点上

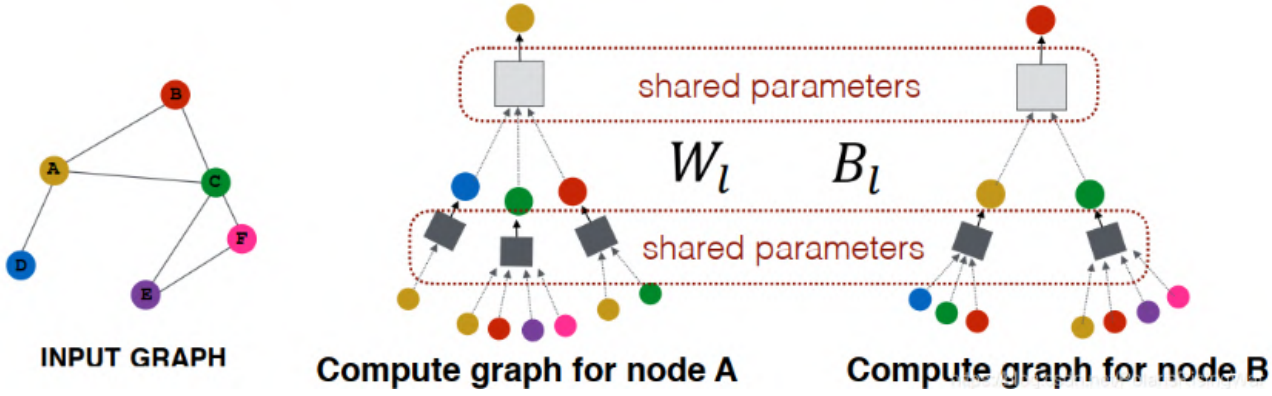




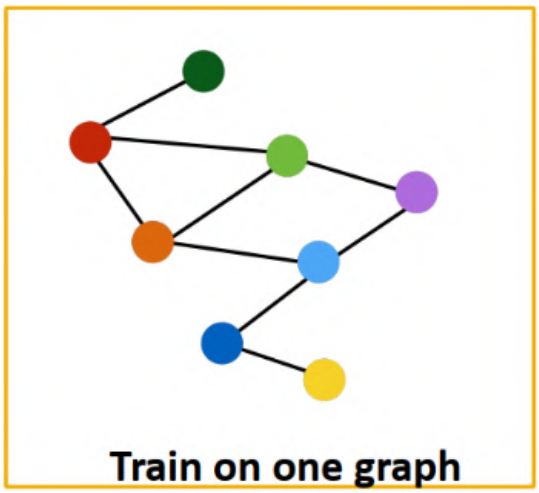


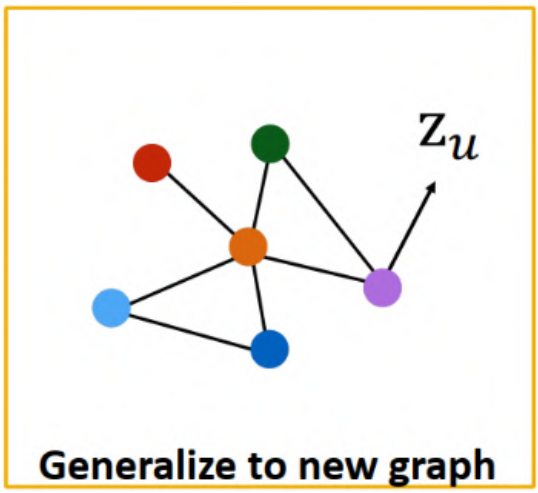
归约能力（inductive capability）：因为聚合邻居的参数在所有节点之间共享，所以训练好的模型可以应用在没见过的节点/图上。比如动态图就有新增节点的情况。模型参数数量是亚线性sublinear于 $|V|$ 的（仅取决于嵌入维度和特征维度）（矩阵尺寸就是下一层嵌入维度×上一层嵌入维度，第0层嵌入维度就是特征维度）。

- **The same aggregation parameters are shared for all nodes:**
  - The number of model parameters is sublinear in $|V|$ and we can **generalize to unseen nodes**!



Inductive Capability: New Graphs

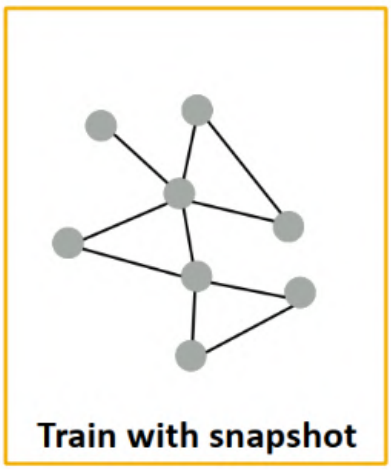


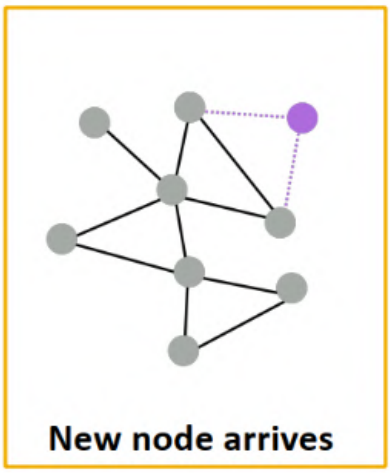


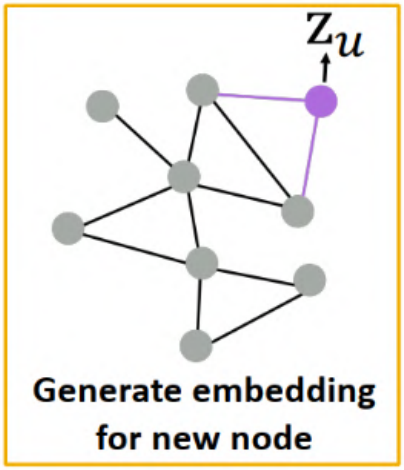
Inductive node embedding → Generalize to entirely unseen graphs

E.g., train on protein interaction graph from model organism A and generate embeddings on newly collected data about organism B

Inductive Capability: New Nodes

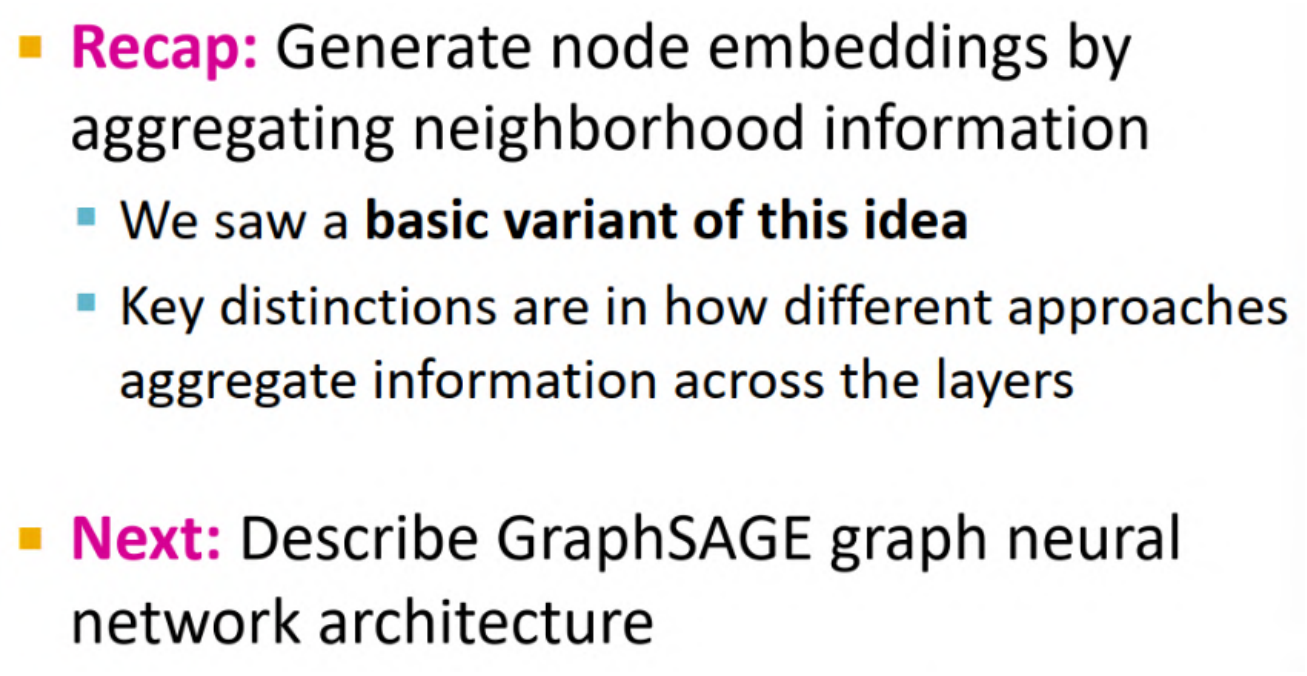






- Many application settings constantly encounter previously unseen nodes:
  - E.g., Reddit, YouTube, Google Scholar
- Need to generate new embeddings "on the fly"

结语：通过聚合邻居信息产生节点嵌入，本节阐述了这一总思想下的一个基本变体。具体GNN方法的区别在于信息如何跨层聚合。



7.4 图卷积神经网络与GraphSAGE

GraphSAGE：这个聚合函数可以是任何将一组向量（节点邻居的信息）映射到一个向量上的可微函数

$$h_v^{(l+1)} = \sigma([W_l \cdot AGG(\{h_u^{(l)}, \forall u \in N(v)\}), B_l h_v^{(l)}])$$

aggregation变体：

- Mean：

$$AGG = \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|}$$

- Pool：

$$AGG = \gamma(\{MLP(h_u^{(l)}, \forall u \in N(u)\})$$

  对邻居信息向量做转换，再应用对称向量函数。

- LSTM：

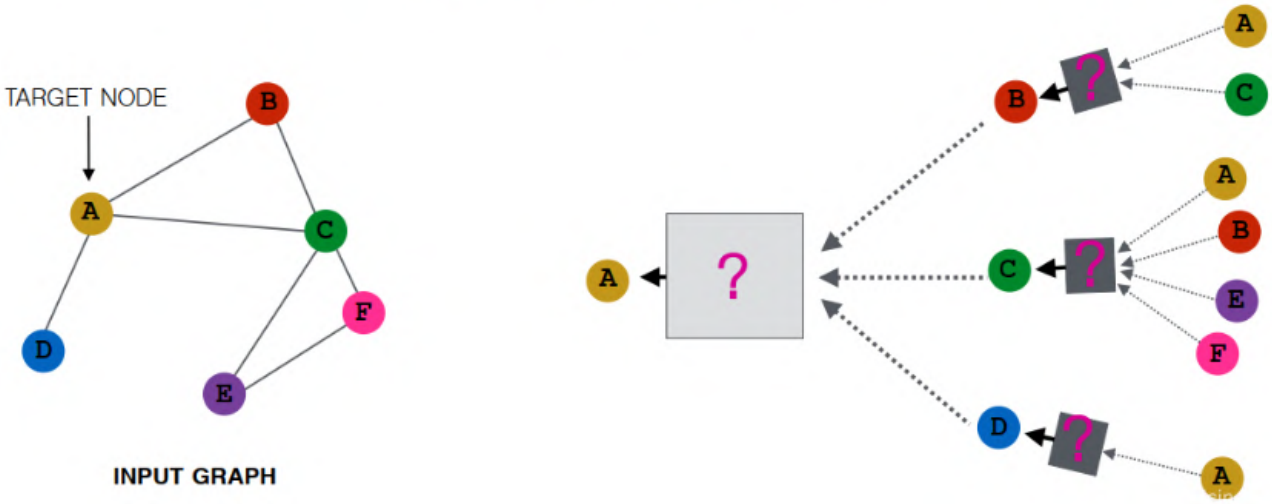
$$AGG = LSTM([h_u^{(l)}, \forall u \in \pi(N(v))])$$

  在reshuffle的邻居上应用LSTM。

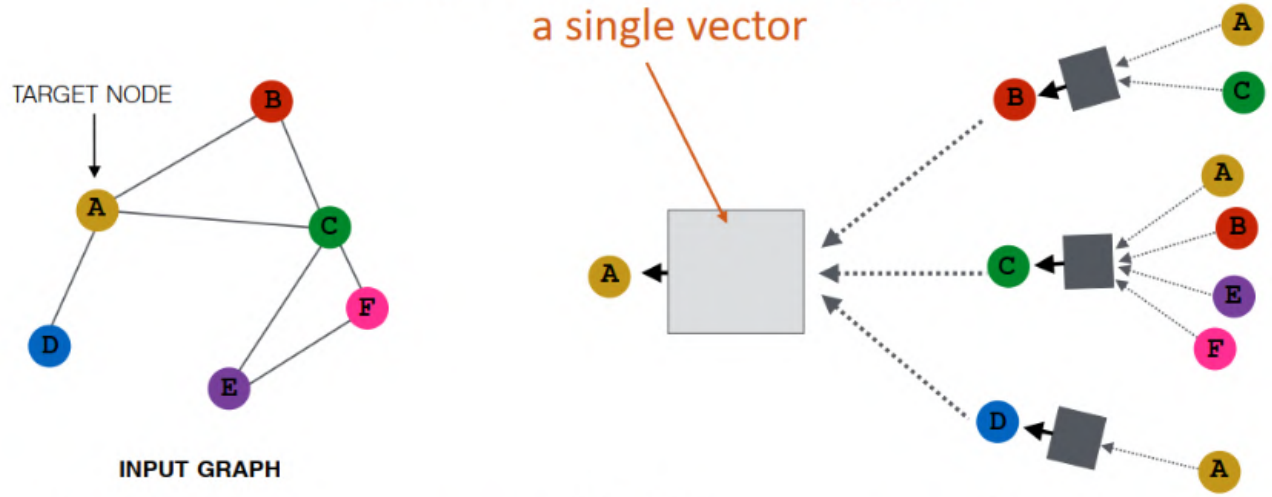
在每一层的节点嵌入上都可以做 $L2$ 归一化 $h_v^k \leftarrow \frac{h_v^k}{||h_v^k||_2}$ ，有时可以提升模型效果。

So far we have aggregated the neighbor messages by taking their (weighted) average

Can we do better?



Any differentiable function that maps set of vectors in $N(u)$ to a single vector



$$\mathrm{h}_v^{(l+1)} = \sigma([\mathrm{W}_l \cdot \mathrm{AGG}\left(\left\{\mathrm{h}_u^{(l)}, \forall u \in N(v)\right\}\right), \mathrm{B}_l \mathrm{h}_v^{(l)}])$$

How does this message passing architecture differ?

Optional: Apply L2 normalization to $h_v^{(l+1)}$ embedding at every layer

- **ℓ_2 Normalization:**
  - $h_v^k \leftarrow \frac{h_v^k}{\|h_v^k\|_2}\ \forall v \in V$ where $\|u\|_2 = \sqrt{\sum_i u_i^2}$ (ℓ_2-norm)
  - Without ℓ_2 normalization, the embedding vectors have different scales (ℓ_2-norm) for vectors
  - In some cases (not always), normalization of embedding results in performance improvement
  - After ℓ_2 normalization, all vectors will have the same ℓ_2-norm

Neighborhood Aggregation

- **Simple neighborhood aggregation:**

$$h_v^{(l+1)} = \sigma\left(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}\right)$$

- **GraphSAGE:**

Concatenate neighbor embedding and self embedding

$$h_v^{(l+1)} = \sigma\left([W_l \cdot \mathrm{AGG}\left(\left\{h_u^{(l)}, \forall u \in N(v)\right\}\right), B_l h_v^{(l)}]\right)$$

Flexible aggregation function instead of mean

Neighbor Aggregation: Variants

- **Mean:** Take a weighted average of neighbors

$$\text{AGG} = \sum_{u \in N(v)} \frac{\mathrm{h}_u^{(l)}}{|N(v)|}$$

- **Pool:** Transform neighbor vectors and apply symmetric vector function

Element-wise mean/max

$$\text{AGG} = \gamma(\{\text{MLP}(\mathrm{h}_u^{(l)}), \forall u \in N(v)\})$$

- **LSTM:** Apply LSTM to reshuffled of neighbors

$$\text{AGG} = \text{LSTM}([\mathrm{h}_u^{(l)}, \forall u \in \pi(N(v))])$$

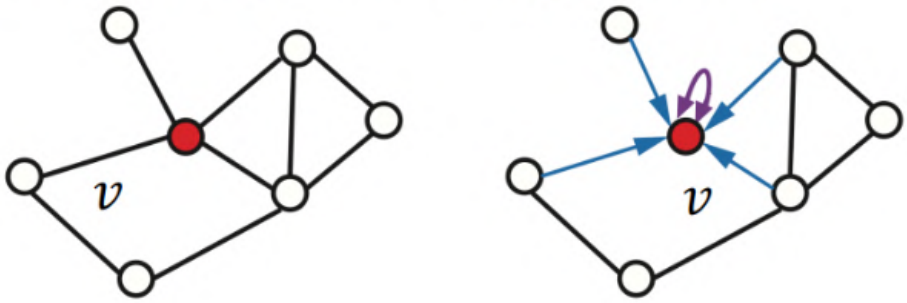
GCN 与 GraphSAGE：核心思想都是基于local neighborhoods产生节点嵌入，用神经网络聚合邻居信息。

- GCN：邻居信息求平均，叠网络层。
- GraphSAGE：泛化neighborhood aggregation所采用的函数。

Recap: GCN, GraphSAGE

Key idea: Generate node embeddings based on local neighborhoods

- Nodes aggregate "messages" from their neighbors using neural networks
- **Graph convolutional networks:**
  - **Basic variant:** Average neighborhood information and stack neural networks
- **GraphSAGE:**
  - Generalized neighborhood aggregation



7.5 总结

本节课中介绍了：

1. 神经网络基础：损失函数loss，优化optimization，梯度gradient，随机梯度下降SGD，非线性non-linearity，多层感知器MLP
2. 图深度学习思想：
   - 多层嵌入转换
   - 每一层都用上一层的嵌入作为输入
   - 聚合邻居和本身节点
3. GCN Graph Convolutional Network：用求平均的方式做聚合，可以用矩阵形式来表示。
4. GraphSAGE：更有弹性的聚合函数。

- **In this lecture, we introduced**
  - Basics of neural networks
    - Loss, Optimization, Gradient, SGD, non-linearity, MLP
  - Idea for Deep Learning for Graphs
    - Multiple layers of embedding transformation
    - At every layer, use the embedding at previous layer as the input
    - Aggregation of neighbors and self embeddings
  - Graph Convolutional Network
    - Mean aggregation; can be expressed in matrix form
  - GraphSAGE: more flexible aggregation