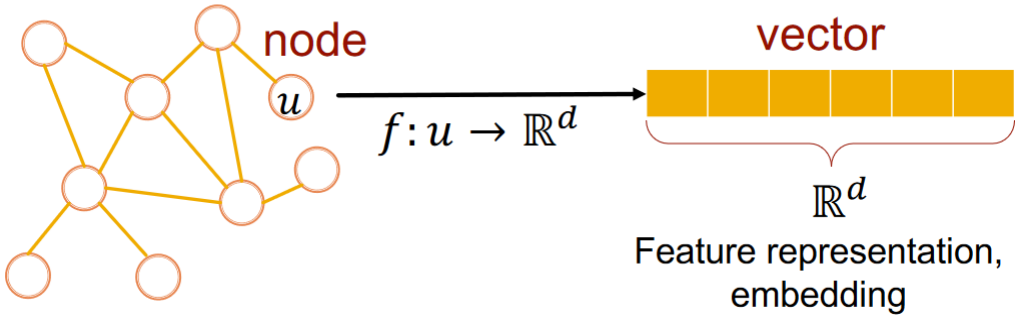
part2 Node Embeddings

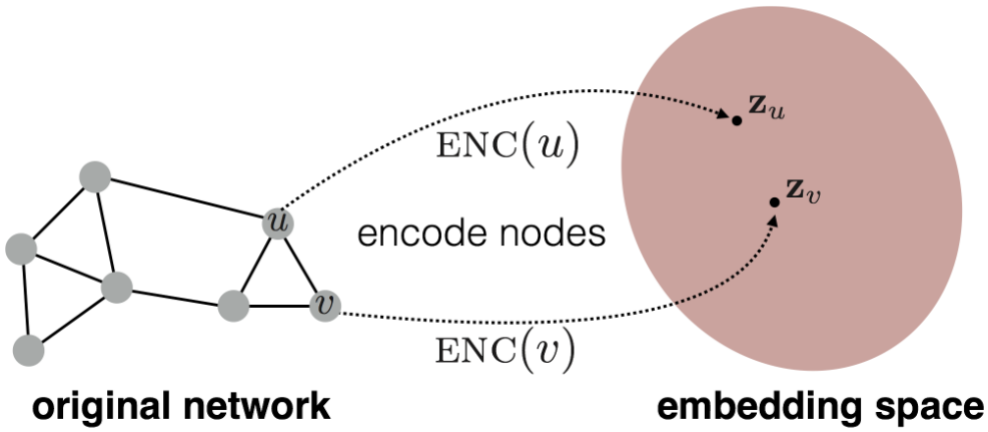
上一节中主要介绍了如何通过传统的ML方法定义节点特征，核心思想是通过构造特征来概括节点的空间特征结构。这种方法对于不同的graph需要手动进行特征构造和提取，那么我们是够可以通过更加有效地方式来构造task-independent的特征呢？这就引出了Node Embedidng 的思想，类比NLP的word Embedding，我们介绍如下的Node Embedding思路。（可以类比CBOW 和 Skip-gram的思路）

Why Embedding?



Embedding的思想就是将Graph中的Node映射为低维的特征向量表示，这种向量表示是dense的而且可以比较好的保持原先Graph中Nodes的相似关系。通过这种特征构造方式得到Node的特征向量后，我们就可以将他们应用在如Node Classification，Link Prediction之类的下游预测任务中。

为了简单起见，文中的Graph Embedding只使用了Node的结构特征，即no node features or extra information is used。



我们的目标是在Embedding space 中尽可能保留Graph space中Nodes之间的相似度：

$$similarity(u, v) \approx z_v^T z_u$$

我们将z_v的生成过程定义为:

$$ENC(v) = z_v = Z \cdot v$$ 其中Z为node embedding矩阵，每一列代表一个node的低维向量表示。v是指示向量，只有指示的列元素为1其余为0。

我们现在已知可以使用embedding向量的内积来近似表达映射后的node similarity。那么如何定义Graph 中的similarity呢?

- Key choice of methods is **how they define node similarity.**
- Should two nodes have a similar embedding if they…
  - are linked?
  - share neighbors?
  - have similar "structural roles"?
- We will now learn node similarity definition that uses **random walks**, and how to optimize embeddings for such a similarity measure.

在进行embedding时，我们并没有使用node features 和 node labels。所以node embedding可以被看做是无监督或者半监督的学习。我们的目标是估计node对应的低维向量表示来最大限度的保留network structure。

Random Walk

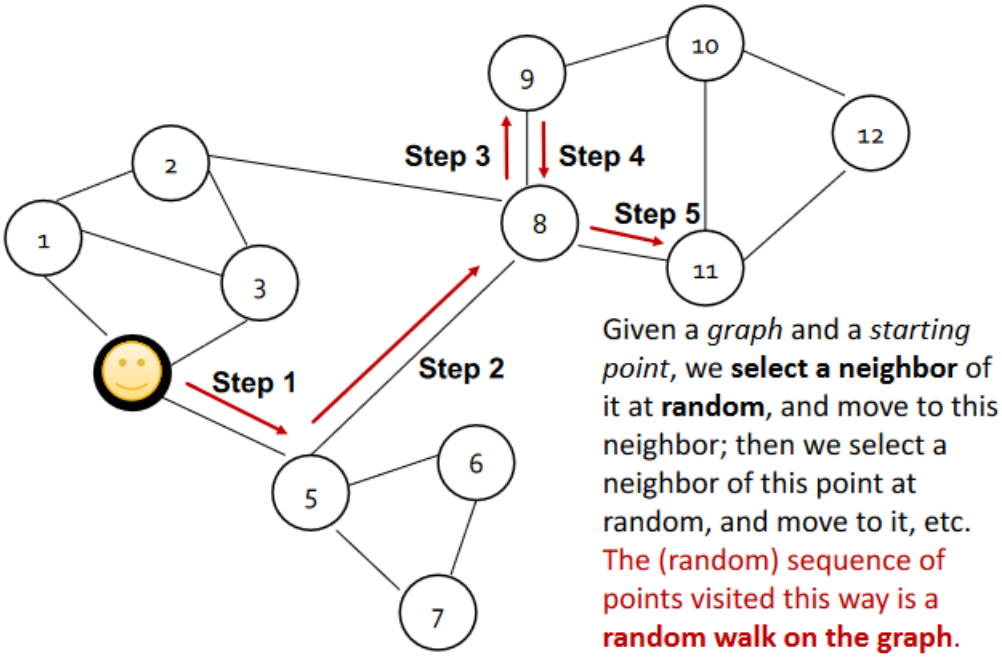
首先我们给出一些定义：

- **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 $S(x) = \frac{1}{1+e^{-x}}$.

类比word2vec算法，首先在语料库的句子中构建滑窗模型，窗口中的单词被分为中心词和背景词。模型的目标是最大化给定中心词时，背景词的后验概率（skip-gram）或者是最大化给定背景词时，中心词的后验概率（CBOW）。而node embedding中，将graph类比作为语料库，graph中并非天然存在类似于句子的序列结构。那么我们是否可以通过在graph中构造出序列特征，从而应用w2v中的方法呢?



如上图所示，我们首先选定graph中的一个node作为strat point，随机选择它的neighbor。在neighbor point的基础上，继续随机选择下一个neighbor point.....这种在graph上随机访问node并且产生一系列node sequence的方法称之为Random Walk。

将$z_u^T z_v$定义为uv共同出现在一个随机游走序列上的概率。那么Random Walk实际上计算的是在给定某种随机游走策略R时，从$node\ u$开始游走，访问到$node\ v$的概率，即$P_R(v|u)$。类比w2v，出现在同一个窗口内的中心词和背景词是相似的，那么对于Random Walk，出现在同一个游走序列内的node也应该是相似的。同时由于游走的随机性，游走可能发生在start point的neighbor，也可能游走到距start point较远的地方，所以这种采样的方式同时包含了node 的局部特征和高阶特征(local and higher-order neighborhood information)。同时Random Walk采样的好处还在于我们并不需要训练graph中的所有node pairs，只需要考虑在Random Walk序列中共现的所有pairs即可。

类比w2v，给出Random Walk的目标函数：

$max_f \sum_{u \in V} logP(N_R(u)|z_u)$　其中$N_R(u)$表示给出游走策略R时，u的随机游走序列。

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 $N_R(u)$

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

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

我们等价的将极大似然估计转化为以下的最小化问题：

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

- **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: $\sum_i \exp(x_i) \approx \max_i \exp(x_i)$

对于集合V中的所有node u进行Random Walk采样，同时计算所有采样序列中的node pairs。我们用softmax定义给出node v的后验概率。得到最终的目标函数：

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

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

Optimizing random walk embeddings =

Finding embeddings $\mathbf{z}_u$ that minimize $\mathcal{L}$

和w2v类似，上式中分母项对于V进行遍历求和计算代价过高。我们使用负采样的方式进行优化。因此node u，v共同出现在同一条random walk序列中的概率可以通过如下的两个独立事件来近似：u和v共现在同一条random walk序列上，且有k个噪声节点i和node u没有共现在同一条random walk序列上。我们只需要采样K个负样本来近似损失函数就可以避免在softmax时，对于分母项进行V的遍历操作。

- **Solution:** Negative sampling

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

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

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

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

我们看到近似后的损失函数放弃了softmax映射概率的方式，而是使用了sogmoid函数将uv的内积结果映射为概率。同时对负样本进行以node degree为权重的加权随机抽样。

- Two considerations for k (# negative samples):
  1. Higher k gives more robust estimates
  2. Higher k corresponds to higher bias on negative events

  In practice k =5-20

在Random Walk的论文中，作者直接在Random Walk采样后对采样序列进行了Skip gram算法。那么Random Walk的采样策略R是否可以进一步优化呢?

Random Walks: Summary

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:

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

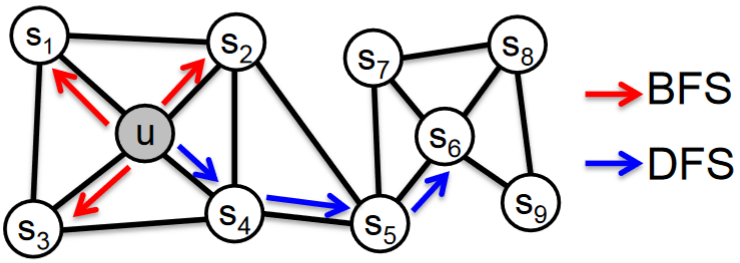
We can efficiently approximate this using negative sampling!

Node2vec

Node2vec在Random Walk的基础上对于游走策略R进行改进，使用了有偏的二阶随机游走策略来对于node u的neighborhood进行采样，即

- Develop biased 2nd order random walk R to generate network neighborhood $N_R(u)$ of node u

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

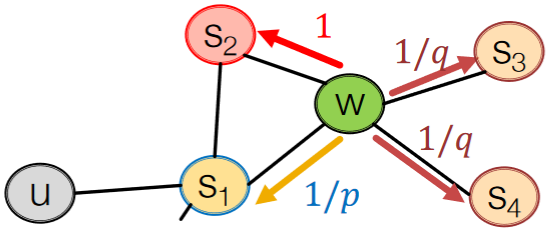
$N_{DFS}(u) = \{s_4, s_5, s_6\}$ Global macroscopic view

Node2vec的游走策略可以分为两种，第一种为BFS采样，即广度优先搜索。第二种为DFS采样，即深度有限搜索。我们从上图可以看出BFS倾向于在node u的近邻附近游走，而DFS倾向远离node u。因此BFS捕捉了节点的微观结构特征，而DFS捕捉了节点的宏观结构特征。

- Two parameters:
  - **Return parameter p:**
    - Return back to the previous node
  - **In-out parameter q:**
    - Moving outwards (DFS) vs. inwards (BFS)
    - Intuitively, q is the "ratio" of BFS vs. DFS

对于上述的两种游走策略，使用$Return\ parameter\ p$和$In-out\ parameter\ q$来控制。

- **Walker came over edge (s_1, w) and is at w. Where to go next?**



$1/p, 1/q, 1$ are unnormalized probabilities

对于上述两个参数可以结合上图理解。当从S1开始的随机游走到达W时，继续远离S1前往S3/S4的概率为1/q。返回S1的概率为1/p。前往S2,即保持和S1的距离为1，的概率为1.（上述的概率都未经归一化）。

node2vec algorithm

- 1) Compute random walk probabilities
- 2) Simulate r random walks of length l starting from each node u
- 3) Optimize the node2vec objective using Stochastic Gradient Descent

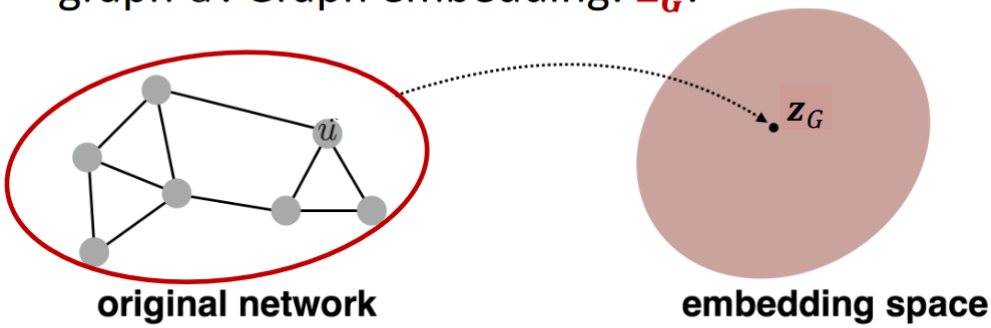
- **Linear-time** complexity
- All 3 steps are individually parallelizable

- **Core idea:** Embed nodes so that distances in embedding space reflect node similarities in the original network.
- **Different notions of node similarity:**
  - Naïve: similar if 2 nodes are connected
  - Neighborhood overlap (covered in Lecture 2)
  - Random walk approaches **(covered today)**

Embedding Entire Graph

在w2v中，我们在对word进行embedding后，通常会对句子也进行相应的embedding。对于graph来说，我们也可以使用相同的思路。

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



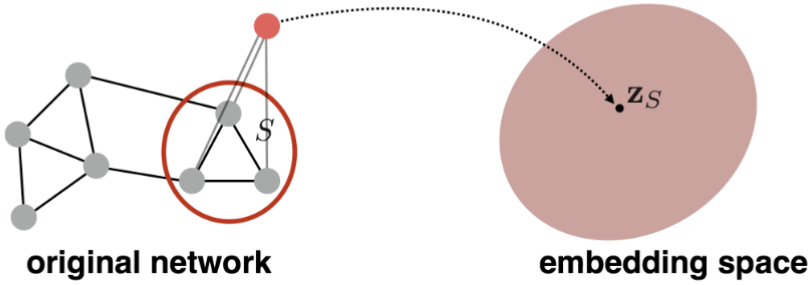
- **Tasks:**
  - Classifying toxic vs. non-toxic molecules
  - Identifying anomalous graphs

现在的目标是对于整张graph做出相应的embedding。

第一种方法是叠加所有的node embedding，即$z_G = \sum_{v \in G} z_v$

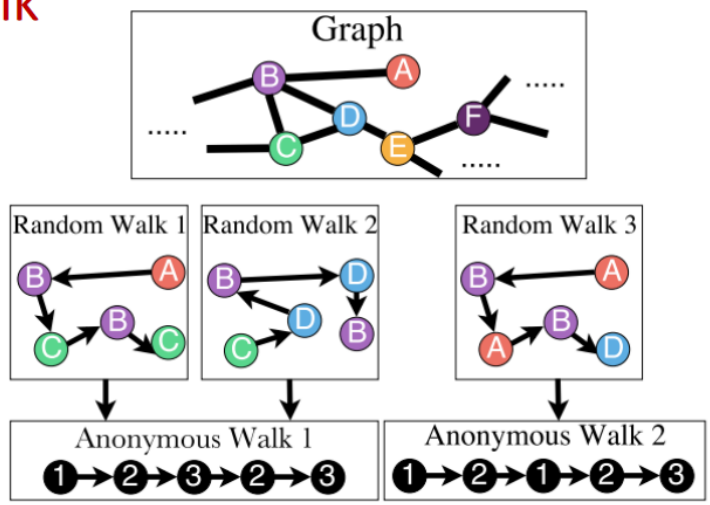
第二种方法是创造一个假节点（virtual node），用这个节点嵌入来作为图嵌入
这个virtual node和它想嵌入的节点子集（比如全图）相连接。

Idea 2: Introduce a **"virtual node"** to represent the (sub)graph and run a standard graph embedding technique

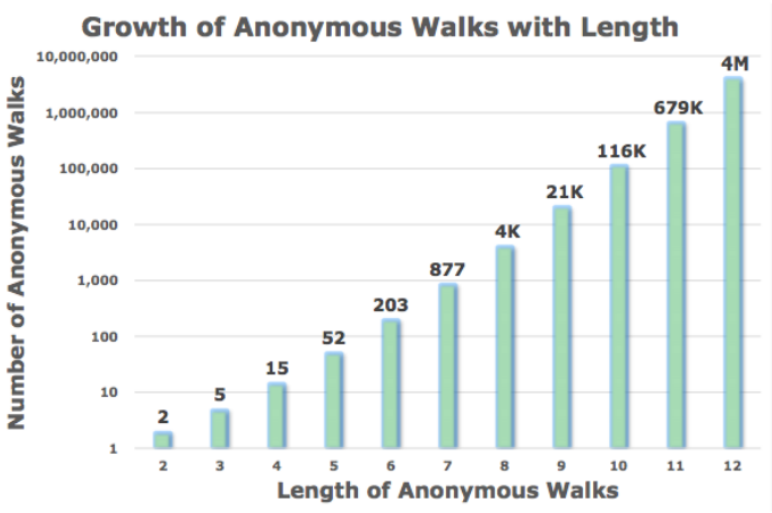


第三种方法被称为Anonymous Random Embedding（匿名随机游走）

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



在一张graph中进行Random Walk，得到了随机游走序列后，我们对序列中第一次出现的node进行编号。以上图为例，左边的随机游走序列中A首先出现，编号为1。BC依次出现，分别编号为2，3。当B再次出现时，仍然编号为2。中间的随机游走序列中，C首先出现，编号为1，DB依次出现，编号为2，3。D再次出现时，仍然编号为2。我们可以发现，虽然左边和中间的序列中出现的node不同，但是我们只在意不同的节点出现的先后顺序，至于节点本身是谁，我们并不在意。因此他们的Anonymous Walk序列是一样的，都为12323。因此称之为匿名（不区分node本身是谁）的随机游走。而对于右侧的序列，编号为12123。



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

随着Anonymous Walk序列长度的增加，Anonymous Walk序列的种类呈指数增长。

- 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

我们对于一张graph多次进行Anonymous Walk同时记录下他们的index。我们可以通过不同Anonymous Walk种类的概率分布来对于graph进行表征。（类似于graphlet的思想）例如我们给定序列长度为3，那么就会有五种random walk序列可能，Embedding为五维随机向量。定义

$$Z_G[i] = \text{随机游走}wi\text{在}graph\text{中出现的概率}$$

我们需要进行的Anonymous Walk次数可以通过如下方法计算：

- 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

其中η为给定序列长度l时，walk的种类数目。上式代表我们希望分布误差大于ϵ的概率小于σ时，需要的random walk序列数目m。

另一种思路是Learning Walk Embeddings

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

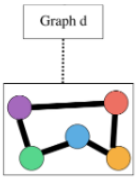
- **Learn a graph embedding Z_G together with all the anonymous walk embeddings z_i**
  $Z = \{z_i : i = 1 \ldots \eta\}$, where η is the number of sampled anonymous walks.

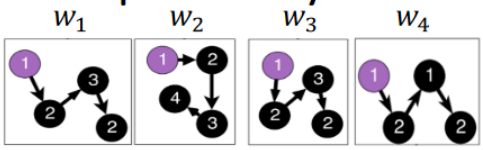
How to embed walks?

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

我们之前的想法用给定长度l是，walk不同种类的经验分布来对于graph进行表征。现在我们对于random walk本身进行表征，即learn embedding z_i of walk w_i。 我们在学习graph embeddingZ_G的同时，学习不同Anonymous Walks 自身的embeddingz_i $i = 1 \ldots \eta$ η为进行的random walk次数。

- 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

定义graph embedding 为 Z_G。从node 1 开始，进行多次Anonymous Walk，结果为$w_1, \dots, w_i$。那么我们得到了元素为一条Anonymous Walk w_i的序列。对这条序列构造滑窗特征并且使用背景序列来预测中心序列。

- **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:

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

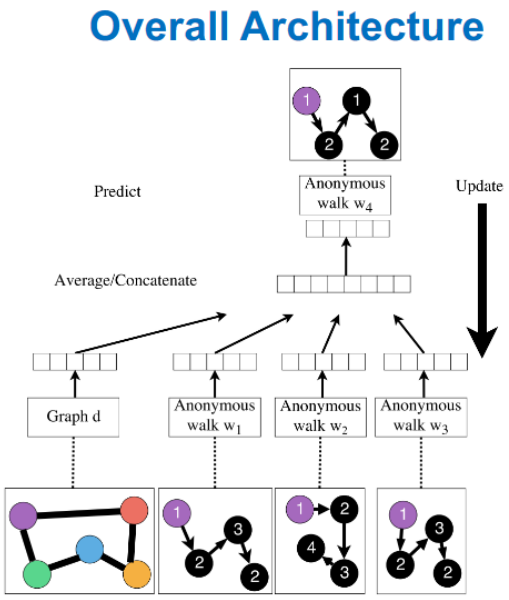
- $P(w_t | \{w_{t-\Delta}, \dots, w_{t+\Delta}, z_G\}) = \frac{\exp(y(w_t))}{\sum_{i=1}^{\eta} \exp(y(w_i))}$ ← All possible walks (require negative sampling)
- $y(w_t) = b + U \cdot \left(cat(\frac{1}{2\Delta} \sum_{i=-\Delta}^{\Delta} z_i, z_G)\right)$
  - $cat(\frac{1}{2\Delta} \sum_{i=-\Delta}^{\Delta} z_i, z_G)$ 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.

Anonymous Walk Embeddings, ICML 2018 https://arxiv.org/pdf/1805.11921.pdf

我们要预测是的整个游走序列w_i的embedding z_i。损失函数与skip gram的损失函数类似，但是注意我们的损失函数中包含了graph embedding Z_G，那么我们将graph embedding也作为参数进行优化，在学习walk embedding的同时也学习了graph embedding。上图中损失函数的后验概率，我们同样使用softmax来表示。注意分母项我们需要对所有可能的Anonymous Walks种类进行求和，根据"随着Anonymous Walk序列长度的增加，Anonymous Walk序列的种类呈指数增长。"这一条件，我们在序列长度比较大时仍然需要对softmax函数进行负采样的近似。（实际上这种方法与node embedding类似，只是将node替换为Anonymous walk序列，同时将graph embedding的参数加入损失函数中一起进行计算得到）

softmax映射的score我们用$y(w_t)$表示。根据上图的表达式可以看出，$y(w_t)$实际上是将滑窗内背景序列的embedding求平均后，平均向量与graph embedding Z_G拼接，在经过线性变化U得到的。其中偏置项b和线性变换U都是可学习的参数。

- 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



Summary

最后我们给出Embedidng的使用方法

How to Use Embeddings

- **How to use embeddings z_i of nodes:**
  - **Clustering/community detection:** Cluster points z_i
  - **Node classification:** Predict label of node i based on z_i
  - **Link prediction:** Predict edge (i, j) based on (z_i, z_j)
    - Where we can: concatenate, avg, product, or take a difference between the embeddings:
      - Concatenate: $f(z_i, z_j) = g([z_i, z_j])$
      - Hadamard: $f(z_i, z_j) = g(z_i * z_j)$ (per coordinate product)
      - Sum/Avg: $f(z_i, z_j) = g(z_i + z_j)$
      - Distance: $f(z_i, z_j) = g(||z_i - z_j||_2)$
  - **Graph classification**: graph embedding z_G via aggregating node embeddings or anonymous random walks. Predict label based on graph embedding z_G

We discussed **graph representation learning**, a way to learn **node and graph embeddings** for downstream tasks, **without feature engineering**.

- **Encoder-decoder framework:**
  - **Encoder: embedding lookup**
  - **Decoder: predict score based on embedding to match node similarity**
- **Node similarity measure: (biased) random walk**
  - **Examples: DeepWalk, Node2Vec**
- **Extension to Graph embedding: Node embedding aggregation and Anonymous Walk Embeddings**