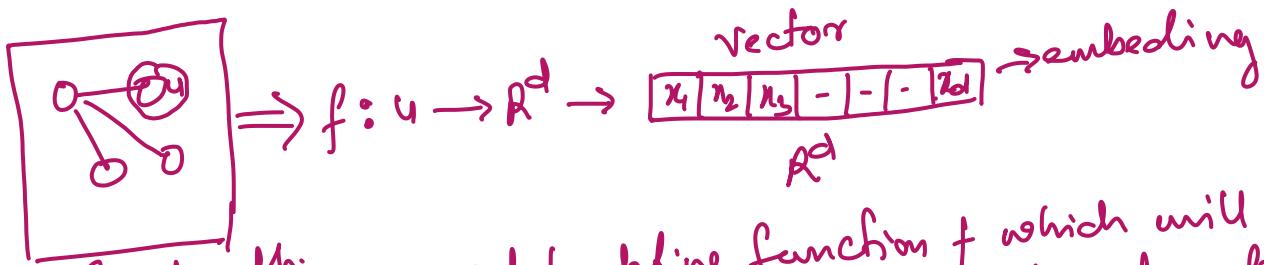


Node Embedding

Till now we have seen hand-crafted method to generate feature set for nodes, links, Graph where main idea was that based on structural property two features should be closer if structural property is same. But we have seen constructing hand-crafted feature set is computationally expensive and time consuming. To overcome this we want mechanism which will automatically extract the features which is relevant for the graph. Hand Crafted feature extraction is called Feature engineering. When we extract features automatically then that is called Graph representation learning.



Input-graph: we need to define function f which will read input-node u_i and transform it in d -dimensional real number vector space. This vector can be used for various downstream tasks like

- ① Node classification
- ② Link prediction
- ③ Graph classification
- ④ Anomalous node detection
- ⑤ Clustering etc.

Similarity in embedding also indicate similarity of node in network too. Process of embedding is also called encoding network information.

Node Embedding framework:

3 component:

- ① Encoder
- ② Node similarity function
- ③ Decoder

① Encoder: Process of generating R^d vector space embedding.

② In original network we need to define node similarity function which measures the similarity in original network.

(3) Decoder: Maps the embedding to similarity score.

(4) Optimize the parameter of encoder so that-

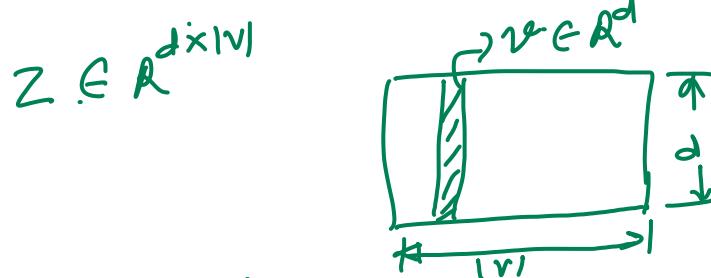
similarity(u, v) $\approx z_v^T z_u \rightarrow$ similarity in embedding space.

$ENC(u) \rightarrow z_u$ \downarrow
similarity in original
networks

We will study two kind of node embedding:

- (1) Random walk
- (2) Node2Vec

when size of embedding vector is d and suppose we have $|V|$ number of nodes then all will be stacked in single matrix Z



Hence each column represent one nodes embedding vector.

Since embedding is lookups in Matrix Z so this is called shallow embedding. When size of graph is large (billions of node) then size of Z becomes issue. Each element of Z is parameter that needs to learn.

Here objective function is to maximize $z_v^T z_u$ as per similarity function for actual network. So decoder is designed based on similarity function.

Similarity function can be defined based on many

criterion e.g

Nodes are linked

share neighbours

have similar structural role etc.

Here node embedding is task independent function and we do not use node label, node feature etc. Our aim is to create node embedding which captures some aspect of networks structure by encoder. This embedding can be used for any task.

(Deepwalk) Random Walk Approaches for Node Embeddings

- ① Random walk strategy: We need to fix how randomly we are going to walk over graph. One simple way can be pick randomly some node and choose randomly some next node and keep on walking till predefined fixed length not finished.



- ② We need to randomly initialize z_u vector of length d with real value which will be optimized based on the idea that similar node embedding will have closer value.

- ③ Probability of reaching node v from u using random walk strategy & will be multiplication of each step as all hop is independent - because its random walk.

$$p(v|z_u) = p(v_j|z_u) \times p(v_k|z_{v_j}) \times p(v|z_{v_k}) \rightarrow \text{similarity func.}$$

if we take log likelihood then we can generalize as

$$= \sum \log p(N_{\Delta}(u)|z_u)$$

- ④ In random walk if probability of some node occurrence is high means they are closer and consider as similar.

- ⑤ Using dot product - we can find similarity score
- $$z_u^T z_v \xrightarrow{\text{to}} z_v^T P(v|z_u)$$
- We can use cosθ as encoding function.

- ⑥ Our objective is to learn d -dimensional embedding vector that preserves the similarity.
Here if nodes are nearby then their embedding should be similar.

- ⑦ Let $G(V, E)$ is graph where we need to define
- $$f: u \rightarrow \mathbb{R}^d, f(u) = z_u$$

- ⑧ Since nearby nodes should have similar embedding then we need to maximize log-likelihood function
- $$\max \sum_{u \in V} \log p(N_{\Delta}(u)|z_u)$$
- where $N_{\Delta}(u)$ is neighbour nodes as per strategy Δ .

Here $\text{NA}(u)$ will be multiset - because of strategy A.

- ⑨ Now our problem converted into optimization problem which can be written as minimization problem.

$$f = \sum_{u \in V} \sum_{v \in \text{NA}(u)} -\log(P(v|z_u)) \rightarrow \text{for given fixed length.}$$

For all node $v \in G(V, E) \Rightarrow u \in V$

- ⑩ Similarity probability can be defined using softmax function.

$z_u^T z_v \Rightarrow$ Gives similarity score.

$$\text{Softmax function} \Rightarrow \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)} \Rightarrow \text{This must be same as } P_a(v|z_u)$$

$$\Rightarrow P_a(v|z_u) = \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)} \rightarrow \text{node embedding similarity function}$$

similarity function of actual network

- ⑪ Now optimization function be written as

$$f = \sum_{n \in V} \sum_{v \in \text{NA}(u)} -\log \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)} \Rightarrow \text{Minimization problem.}$$

- ⑫ Solving this naively will be very expensive because nesting of summation over all nodes. Complexity $O(|V|^2)$

- ⑬ Normalization in softmax $\sum_{n \in V} \exp(z_u^T z_n)$ is culprit. we need to approximate this. This can be done using negative sampling. We do not take all nodes but k -nodes for approximation and instead of exponential function we use sigmoid function (σ) for probability.

$$\log \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)} \approx \log \left[\frac{\sigma(z_u^T z_v)}{\sum_{n \in V} \sigma(z_u^T z_n)} \right]$$

$$= \log \sigma(z_u^T z_v) - \sum_{i=1}^k \log (\sigma(z_u^T z_{n_i}))$$

Here η_i is random distribution over nodes.

By using k -sample improves the efficiency of calculation.

④ Higher k gives more robust estimate

⑤ Higher k corresponds to higher bias on negative ~~est~~ events.

⑥ Sample K negative nodes each with probability proportional to its degree.

⑦ $K=5-20$ suggested

⑧ Usually negative sampling for any node alone but most efficient way is to take nodes which are not on the walk.

14 Once optimization function defined then using gradient descent- we can optimize L w.r.t z_u

$$\frac{\partial L}{\partial z_u}$$

$$z_{ut} \leftarrow z_{ut} - \eta \frac{\partial L}{\partial z_u} \quad \text{when } \eta \text{ is learning rate.}$$

15 By stochastic gradient-descent- method we can train z_u by optimizing L for given training examples.

We choose random walk because this is expressive due to giving higher probability to closer nodes. Other reason is we do not need to consider all pair node but only those node which is participating in random walk.

NODE2VEC

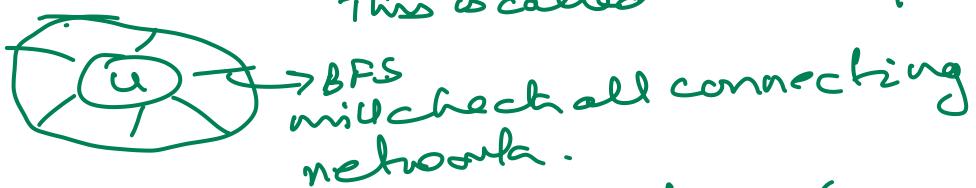
Now question came, can we tune the random walk more so that embedding is more expressive? How to make random walk (DeepWalk) more richer? currently using uniformly distributed random walk which may be too constraint- for downstream task.

Our goal is to embed node such that - similar network neighbourhood should have embedding also closer. So maximizing likelihood optimization which is independent- to downstream task. Only thing will change $N_d(u)$: how we select neighbourhood.

This time instead of uniformly distributed random walk, we will choose 2nd order biased random walk to choose $N_d(u)$.

2nd order random walk:

- (*) It remembers from where it came (Previous state)
- (*) Neighbours are defined based on local & global views of network.
- (*) BFS reflects local view of network.
This is called microscopic view



- (*) DFS reflects global view as if one try to go away from starting point. This gives macroscopic view

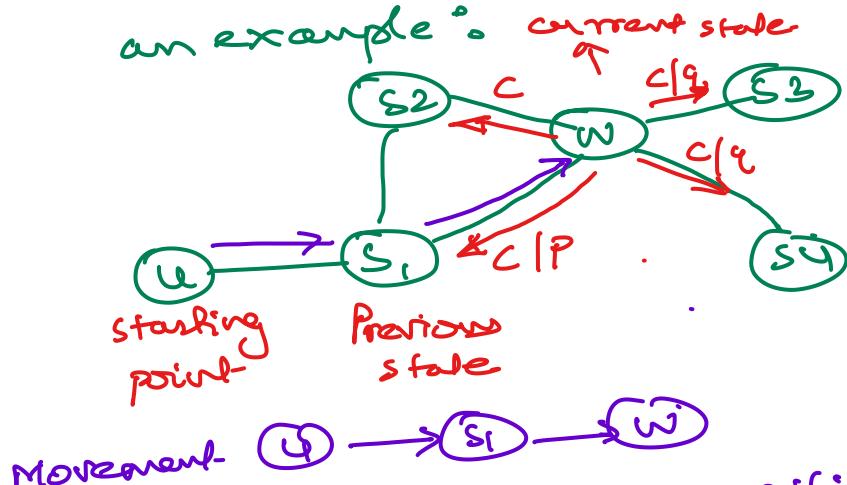


- (*) We fix the random walk length l from any random node u .
- (*) Based on fixed length we will generate $N_d(u)$.
- (*) Neighbours are generated based on two params
p: return back to previous state.
q: in-out parameters & this is trade-off between

BFS and DFS - Intuitively question of BFS vs DFS

- (*) From any intermediate step there will be 3 possible next move
- ① Return back to previous state
 - ② Move to next step as per BFS move which will keep the distance same with respect to previous state.
 - ③ Move away from previous state (DFS)

- (*) We create biasness in above move by introducing P, Q in transition probability. Let's understand transition probability using an example :-



- From (i) there is 4 possibility
- ① go to s_2 . Weightage is some constant c
 - ② return back to previous state s_1 with transition weightage c/P
 - ③ go away s_3 or s_4 with transition weightage c/q

These transition weightage is unnormalize distribution. We can normalize this

$$s = c + \frac{c}{P} + \frac{c}{Q} \Rightarrow$$

Position $\rightarrow s_1$ Probability $\frac{c/P}{s}$
 $\rightarrow s_2$ Probability $\frac{c}{s}$
 $\rightarrow s_3$ Probability $\frac{c/Q}{s}$
 $\rightarrow s_4$ Probability $\frac{c/Q}{s}$

This probability distribution creates biasness in next move. We flip a biased coin in order to choose next move -

- ★ Based on this random walk we generate $NA(u)$
- ★ For each node u we make ' τ ' random walks of length ℓ to generate $NA(u)$.
- ★ For this $NA(u)$ we will optimise node2vec objective function using stochastic gradient-method.
- ★ Since generation of neighbours for each node is independent, so we can parallelize the encoding function to generate embedding.
- ★ Time complexity of this algorithm is linear.
- ★ There are many other ways in which we can generate embedding by
 - ① Different kind of Random walks
 - ② Alternative optimisation techniques
 - ③ Network prepossessing techniques.
- ★ The expressiveness of embedding is highly dependent on selection of various methods during generation like node2vec perform better on node classification while other method better on link prediction.
- ★ Definition of node similarity must be chosen based on application requirement.
- ★ Scalability is issue as it is good upto medium size network. Not for very large network.

Graph embedding

Till now we have seen embedding of nodes in a graph which can be used for downstream task, but what is effective way to embed entire graph? The embedding requirement remain same as similar graph should have embedding also closer. The graph embedding can be used for various downstream task like clustering, classification, anomaly etc. Graph embedding could be of entire graph or subgraph also. Let us see various idea on this.

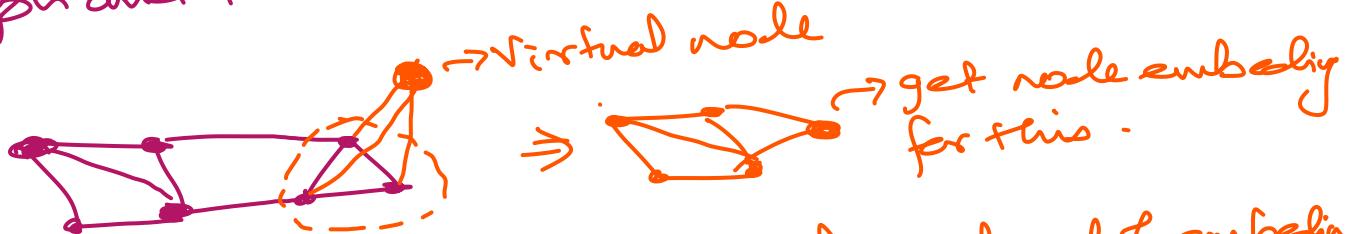
- ① Simple idea: Generate embedding for all nodes and then sum(average) all embedding to create graph embedding Z_G

$$Z_G = \sum_{v \in G} Z_v$$

In case subgraph then take subgraph node for this.

This was used for molecule classification by Duvenaud (2016) which was effective.

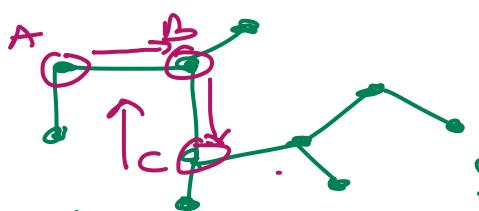
- ② Idea 2: Create virtual node for subgraph of original graph and then run standard node embedding



This is general technique for subgraph embedding.

- ③ Idea 3: Anonymous Walk Embedding

For this we need to understand what is anonymous walk. Let's understand this using example:
Random walk(uniformly distributed): A walk that starts from any random node and traversing randomly to their neighbours.



lets represent - this random walk as list of node visited
so we say ordered list of nodes visited
 $W = \{A, B, C, B\}$
Index $\rightarrow 1 \quad 2 \quad 3 \quad 4$

Positional function (pos) can be defined as list of positions of nodes in a given random walk.

e.g. $pos(W, A) = \{1\} \Rightarrow$ Index value list

$$pos(W, B) = \{2, 4\} \Rightarrow pos(W, C) = \{3\}$$

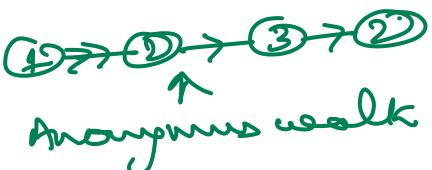
We will define other function $f(v_i) = \min [pos(W, v_i)]$
means for given index list we take minimum value.

e.g. $f(B) = \min \{2, 4\} = 2 \quad f(C) = \min \{3\} = 3$
 $f(A) = \min \{1\} = 1$

Now replace each node in Random walk list - with value $f(v_i)$. This list is called anonymous walk.

$$W = \{A, B, C, B\}$$

$$AW1 = \{1, 2, 3, 2\}$$



Even though there node sequence is different - but - there anonymous walk sequence will be same because there positional sequence min value will be same.

e.g. let $AW2 = \{A, B, C, B, C\}$

$$\& W2 = \{C, D, B, D, B\}$$

(lets create- $pos(RW1, A) = \{1\}$)

$$pos(RW1, B) = \{2, 4\}$$

$$pos(RW1, C) = \{3, 5\}$$

$$f(A) = \min [pos(RW1, A)] = 1$$

$$f(B) = \min [pos(RW1, B)] = 2$$

$$f(C) = \min [pos(RW1, C)] = 3$$

$$AW1 = (1, 2, 3, 2, 3)$$

We can see $AW1 = AW2$, so both is same.

$$\left| \begin{array}{l} pos(AW2, C) = \{1\} \\ pos(AW2, D) = \{2, 4\} \\ pos(AW2, B) = \{3, 5\} \\ f(C) = \min [pos(AW2, C)] = 1 \\ f(D) = \min [pos(AW2, D)] = 2 \\ f(B) = \min [pos(AW2, B)] = 3 \\ AW2 = (1, 2, 3, 2, 3) \end{array} \right.$$

