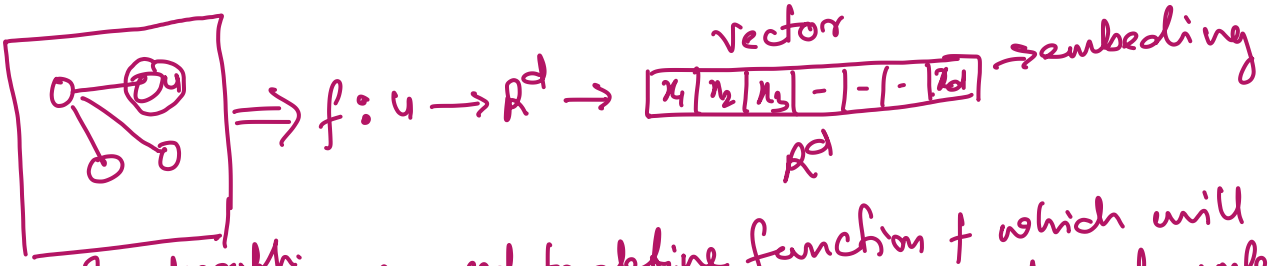
Node Embedding

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 feature set should be closer if structural property is same. But we have seen constructing hand-crafted feature set is computationaly expensive and time consuming.
To overcome this we want mechanism which will automatically extract the features which is relevent for the graph.
Hand Crafted feature extraction is called feature engineering.
When we extract features automaticaly then that is called Graph representation learning.



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

1. Node classification
2. Link prediction
3. Graph classification
4. Anomalus node detection
5. clustering etc.

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

Node Embeding frame work:
2 component:
1. Encoder
2. Node similarity function
3. Decoder

1. Encoder: Process of generating $\mathbb{R}^d$ vector space embeding.
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$(u,v) \approx z_v^T z_u$ → similarity in embedding space.

↓ similarity in original network

ENC$(u) \rightarrow z_u$

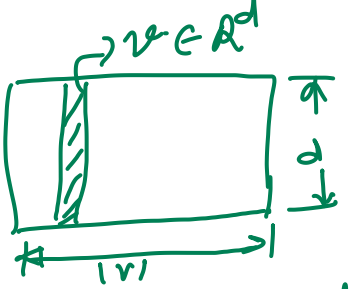
We will study two kind of node embedding:

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

$Z \in \mathbb{R}^{d \times |V|}$



Here each column represent one nodes embedding vector.

Since embedding is lookup 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 $z_v^T z_v$ 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

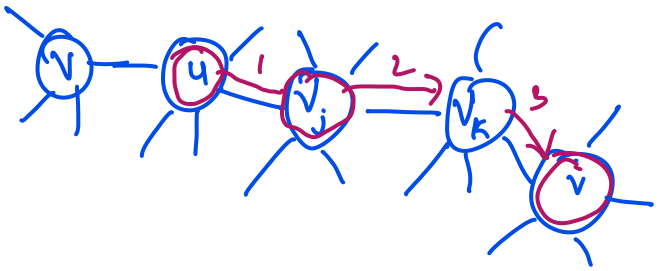
→ Share neighbours

→ have similar structural role etc.

Here node embedding is task independent function and we do not use node lable, node feature etc. Our aim is to create node embedding which capture some aspect of network 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 of simple way can be pick randomly some node and choose randomly some next node and keep on walking till predefined fixed length not finished.



Let 3-hop random walk from u to v with red path.

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 R 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

$$= \sum_{u \in V} \log P\left(N_R(u)/Z_u\right) \Rightarrow N_R(u) = \text{neigh. of } u \text{ by strategy } R$$

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

⑤ Using dot product - we can find similarity score

$Z_u^T Z_v$ $\quad$ θ: $Z_u \rightarrow Z_v$ $\quad$ $P_R(v/Z_u)$

We can use cos θ as encoding function.

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

⑦ Let $G(V,E)$ is graph where we need to define

$$f: u \rightarrow R^d, \quad f(u) = Z_u$$

⑧ Since nearby nodes should have similar embedding then we need to maximize log-likelihood function

$$\max_f \sum_{u \in V} \log P\left(N_R(u)/Z_u\right)$$ where $N_R(u)$ is neighbour nodes as per strategy R.

Here $N_R(u)$ will be multiset- because of strategy R.

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

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

For all node 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.

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

$$\Rightarrow P_R(v|z_u) = \frac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)}$$

$P_R(v|z_u)$: similarity function of actual network

$\dfrac{\exp(z_u^T z_v)}{\sum_{n \in V} \exp(z_u^T z_n)}$: node embedding similarity function

(11) Now optimization function be written as

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

(12) Solving this naively will be very expensive because of nesting of summation over all nodes. Complexity $O(|V|^2)$ ($u \in V$, $n \in V$ (nested summation two times))

(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.

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

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

Here n_i is random distribution over nodes.

By using K-sample improves the efficiency of calculation.

- Higher K gives more robust estimate
- Higher K corresponds to higher bias on negative events.
- Sample K negative nodes each with probability proportional to its degree.
- K = 5-20 suggested
- Usualy negative sampling for any node done bcz most efficient way is to take nodes which ~~on the~~ not on the walk.

(14) Once optimation function defined then using gradiant descent we can optimize L wrt z_4

$$\frac{\partial(L)}{\partial(z_4)}$$

$$z_4 \leftarrow z_4 - \eta \frac{\partial L}{\partial z_4}$$ when η is learning rate.

(15) By stochastic gradient-descent method we can train z_4 by optizing L for given training examples.

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

NODE2VEC

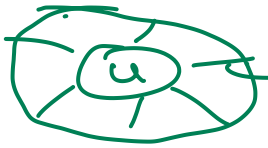
Now question came, can we tune the random walk more so that embedding is more expressive? How to make random walk (DeepWalk) more richer? currently using uniformly distributed random walk which may be too constraint for downstream task.

Our goal is to embed node such that similar network neighbourhood should have embedding also closer. So maximizing likelihood optimization which is independent of downstream task. Only thing we'll change $N_R(u)$: How we select neighbourhood.

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

2nd order random walk:

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



BFS will check all connecting network.

- DFS reflect global view as if we'll 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 $N_R(u)$.
- Neighbours are generated based on two parameters

p: Return back to previous state.

q: In-out parameter. This is trade-off between

BFS and DFS. Intuitively q is ratio of BFS Vs DFS

(*) From any intermediate step there will be 3 possible next move

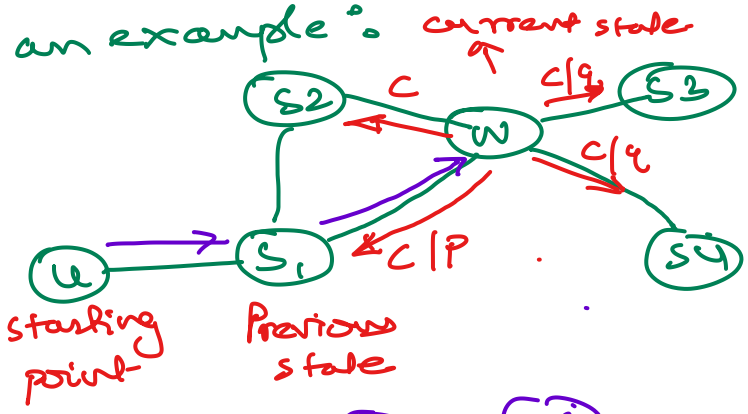
(1) Return back to previous state

(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.



Movement- u → S1 → W

From W there is 3 possibility

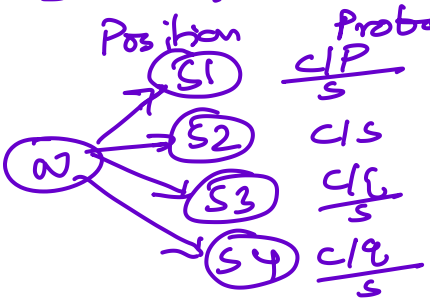
(1) go to S2. Weightage is same constant c

(2) Return back to previous state S1 with transition weightage c/p

(3) go away S3 or S4 with transition weightage c/q

These transition weightage is unnormalize distribution. We can normalize this

$S = C + \frac{c}{p} + \frac{c}{q} \Rightarrow$



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 $N_R(u)$

(*) For each node u we make 'r' random walk of length l to generate $N_R(u)$.

(*) For this $N_R(u)$ we will optimize 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) Diffrent kind of Random walks
   (2) Alternative optimization techniques
   (3) Network preprocessing 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 choosen 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 close. 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.

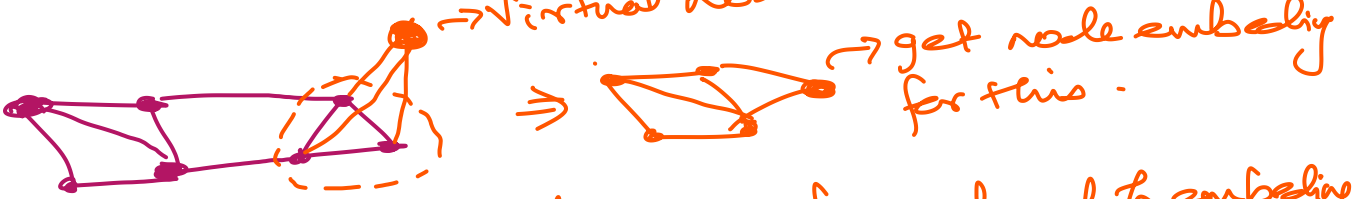
① Simple idea: Generate embedding for all nodes and then sum(average) all embedding to create graph embedding Z_G

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

In case sub-graph 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 from original graph and then run standard node embedding

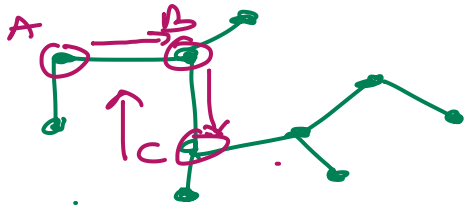


This is general technique for subgraph embedding.

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.



Let's represent this random walk as list of node visited so we say ordered list of nodes visited

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

Index → 1 2 3 4

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

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

$POS(W, B) = \{2, 4\}$ ⇏ $POS(W, C) = \{3\}$

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

e.g $f(B) = \min\{2, 5\} = 2$ $f(C) = \min\{3\} = 3$

$f(A) = \min\{1\} = 1$

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

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

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

Anonymous walk

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

e.g: let $AW1 = \{A, B, C, B, C\}$

$AW2 = \{C, D, B, D, B\}$

Let's check - $POS(AW1, A) = \{1\}$

$POS(AW1, B) = \{2, 4\}$

$POS(AW1, C) = \{3, 5\}$

$f(A) = \min[POS(AW1, A)] = 1$

$f(B) = \min[POS(AW1, B)] = 2$

$f(C) = \min[POS(AW1, C)] = 3$

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

$POS(AW2, C) = \{1\}$

$POS(AW2, D) = \{2, 4\}$

$POS(AW2, B) = \{3, 5\}$

$f(C) = \min[POS(AW2, C)] = 1$

$f(D) = \min[POS(AW2, D)] = 2$

$f(B) = \min[POS(AW2, B)] = 3$

$AW2 = (1, 2, 3, 2, 3)$

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

* based on anonymous walk length no. of possible move is fixed

e.g

Anonymous walk length	#possible move
2	2
3	5
4	15
5	52
⋮	⋮
10	115K
12	4M

* For given anonymous walk length 'l' we will simulate the count of AW of particular type covered in 'y' trails

e.g let $l=3$ then possible AW will be

$w_1=111$, $w_2=112$, $w_3=121$, $w_4=122$, $w_5=123$

let 3 trials

	w_1	w_2	w_3	w_4	w_5
trial-1	0	0	1	0	0
trial-2	1	0	0	0	0
trial-3	1	0	0	0	0
	2	0	1	0	0

Then we can represent graph embedding as probability distribution of these trials.

$Z_G = \{P(w_1), P(w_2), P(w_3), P(w_4), P(w_5)\}$

$Z_G = \{\frac{2}{3}, 0, \frac{1}{3}, 0, 0\}$

$Z_G[i]$ is probability of AW w_i in G.

* Now question comes how many walks we need to consider for given length of AW. For large graph complete counting of all possible AW is infeasible. we need to select sampling approach will be approximation of true distribution.

Let ε is error level with probability δ then sample m can be defined as

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

where η is possible AW move for given length l.

Here $\varepsilon > 0$ & $\delta \in [0,1]$

ε is error which must be greater than 0

δ value must be in between 0 & 1 as this is probability that error will be greater than or equal to ε.

Example: Let $l=7$ then $\eta=877$

if $\varepsilon=0.1$ & $\delta=0.01$ then we need to generate m sample which will be equal to $\Rightarrow m=\left[\frac{2}{(0.1)^2}(\log(2^{877}-2)-\log(0.01))\right]=122500$

means with 122500 number of AW probability distribution will stable with error γ 0.1 or more & probability of this error will be 0.01

we can generate anonymous walk probability distribution using below technique:

① Co-occuring AW: when we generate AW from particular node which is randomly selected is called co-occuring anonymous walk.

② We can assume AW as single word if we conclude this with NLP. co-occuring AW as sentence.

③ when we generate these sentences from all node is called corpora.

like in NLP we predict next word when we have train of co-occuring words, same with conditional probability, same way here also we will predict co-occuring AW.

① Let Z_G is graph embeding represented using AW,

② Run T different- random walk from u each of length l:

$$N_\Delta(u)=\{w_1^u, w_2^u, w_3^u \cdots w_T^u\}$$

③ We will predict z_i AW that will happen in the interval Δ for random walk w_i:

$$=P(w_t|\{w_{t-\Delta}\cdots w_{t+\Delta}, Z_G\})$$

④ We will maximize average log probability (Objective function)

$$\frac{1}{T}\sum_{t=\Delta}^{T-\Delta}\log P(w_t|\{w_{t-\Delta}\cdots w_{t+\Delta}, Z_G\})$$

⑤ Let η is number of all possible walk then:

$$P(w_t|\{w_{t-\Delta}\cdots w_{t+\Delta}, Z_G\})=\frac{\exp(y(w_t))}{\sum_{i=1}^{\eta}\exp(y(w_i))}$$

$$\Rightarrow y(w_t)=b+U\cdot \text{cat}\left(\frac{1}{2\Delta}\sum_{i=-\Delta}^{\Delta} z_i, Z_G\right)$$

where $\text{cat}\left(\frac{1}{2\Delta}\sum_{i=-\Delta}^{\Delta} z_i, Z_G\right)$ is an avg. anonymous walk embeding in window concatinated with graph embeding Z_G

$b \in$ Real number, $U \in R^D$ where D is length of co-occuring AW.

(*) b & U is learnable parameter.

(*) We obtain Z_q (learnable parameter) after optimization.

(*) Use Z_q as graph kernel $Z_{q_1}^T \cdot Z_{q_2} \Rightarrow$ Inner product

(*) We can use Z_q in neural network as input to classify.