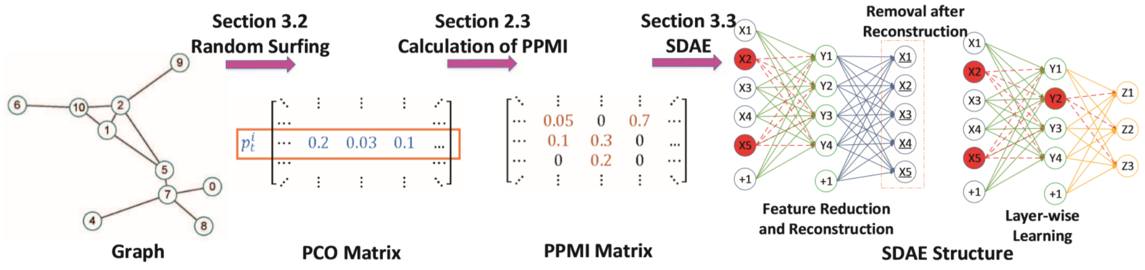
1 DNNs for Learning Graph Representations

1.1 Summary

The main work of [CLX16] was using random surfing to generate training and test dataset, or one can think this is data preprocess.



Given a graph, this algorithm not directly performed representation learning over it, rather, it first transformed this graph (matrix M) to positive pointwise mutual information (PPMI) matrix (M^{PPMI}), and then, performed denoise autoencoder over the PPMI matrix. Why did they calculate and perform operation on the PPMI matrix? It mainly inspired by [LG14], since many graph embedding algorithms were word2vec based, for example, performing random walk over a graph to generate training data, which corresponding to bag-of-words, and then, using skip-gram model to learn these node embeddings. However, [LG14] analyzed that word2vec algorithm was an implicit matrix factorization, shortly, if one input an original word-context matrix M, the word embedding matrix W and context embedding matrix C learnt by the word2vec is factorial of M^{PPMI}, i.e., $\boldsymbol{w}_i\boldsymbol{c}_j = M^{PPMI}_{i,j}$, this article wanted to directly learn non-linear factorizations of M^{PPMI}. So, one can just think it is non-linear factorization of M^{PPMI}.

1.2 Random surfing: where can I reach and what is the probability of that after some steps.

Assume we have a weighted graph with adjacent matrix A, where A is transition probability as well. The start position is node i, so, P_0 is one-hot vector, where $i-$th element of P_0 is 1. Therefore, the probability of next position it can arrive is $P_1 = P_0A$, and so on $P_k = P_0A^k$. So, P_k lists all possibilities if we start from node i and walk k steps; however, this has shortcoming, for example, consider this situation (fig.1), if we start from 1, and if k is odd, we can reach 2, if k is even we can reach 3 but we can't reach 2 ,so it is reasonable to summarize all p_k but gave long step a lower weight, one intuition is that you can't walk so far since your energy is limited.

$$r = \sum_{k=1}^{K} w(k) P_k$$

where r is the representation of node i.

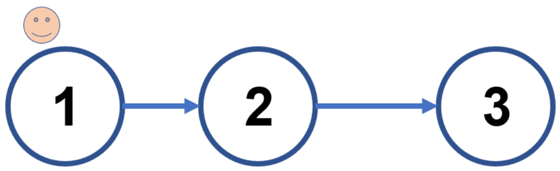


Figure 1: $k = 1 : 1 \to 2, \;\; k = 2 : 1 \to 2 \to 3$ (but it should include 2, i.e., $1 \to 2 \to$give up)

This article gave a assumption that wherever the current position was, there was some change to go back to the start point.

$$P_k = \alpha P_{k-1} + (1-\alpha) P_0$$

One can also think this is convex combination, ideally, one can reach B from A; however, actually, he only arrives at some point between A and B.

With these explanations we finally got:

$$r = \sum_{k=1}^{K} P_k, \;\; w(t) = \alpha^t + \alpha^t (1-\alpha)(K-t)$$

Once M^{PPMI} was computed, we have prepared the training data. Each row of M^{PPMI} then was a training example, and added some noise on it, we hoped that a autoencoder model can reconstruct it:

$$DEC(ENC(f_{noise}(\boldsymbol{x}_i))) = \boldsymbol{x}_i$$

References

[CLX16] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[LG14] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.