node2vec: A Representation Learning Technique for Graph Data

Pritom Saha Akash (pakash2)

November 13, 2020

1 Introduction

One of the most important pre-steps for many machine learning tasks in various areas like natural language processing, computer vision, audio is to represent highly structured objects such as graphs into a continuous feature space. This representation, not only makes it easy for the computer to process the data but also improves the predictive capability of various tasks to a greater extend. node2vec is such a representational learning framework for graph data [1]. More specifically, node2vec takes a graph as input and for each node in the graph, it can learn a continuous feature vector representation. These continuous feature representations can then be used for many downstream machine learning applications.

2 node2vec Model

Now, it is time to put the details of how node2vec works. It is obvious from its name, node2vec embeds vector for nodes. This task of embedding for node2vec quite similar to that of the word2vec skip-gram model [2] for embedding words. In the word2vec skip-gram model, the words in a vocabulary are represented as a vector that is learned from the corpus of unstructured natural language text. More specifically, for a word, the vector representations are learned from the word's co-occurrence information with other words in the corpus. In the case of node2vec, the embedding is learned for each node in a graph instead of a word. However, there is no corpus for the graph and instead, we have the structured relations within nodes in the graph. From these graphs, the node2vec first generate a corpus like input for learning to the skip-gram model. The generation of the corpus from the graph is the most innovative part of the node2vec algorithm and it is done using a sampling strategy.

Given an input graph, to generate corpus from that graph, let's first consider a corpus as a group of directed acyclic graphs having a maximum out-degree of 1. Therefore, we can represent a text sentence as an acyclic graph with nodes as words like Figure 1. Therefore,

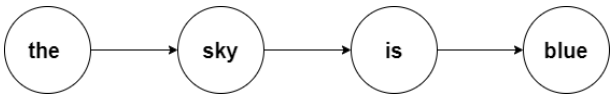

Figure 1: Word sequence as graph

we can see that this type of graph can easily be embedded by word2vec. However, the main

problem is that there not all graphs come as this type of simple graph structure, they can be complex like cyclic, weighted, undirected. To solve this problem, the node2vec algorithm uses a sampling strategy to generate multiple acyclic subgraphs from the origin graph. This task is done by generating biased random walks from each node of the graph. More specifically, it builds representations by maintaining a spectrum of equivalences from homophily to structural equivalence using a way of balancing the exploration-exploitation tradeoff. For example, Figure 2 shows an example of transitioning properties for the random walk. After reaching node v from t, the probability of a walk whether staying inward revisiting nodes (t) or staying near to the preceding nodes (x_1), or moving outward farther away (x_2, x_3) is controlled by two parameters that are called the return hyperparameter p and the inout hyperparameter q.

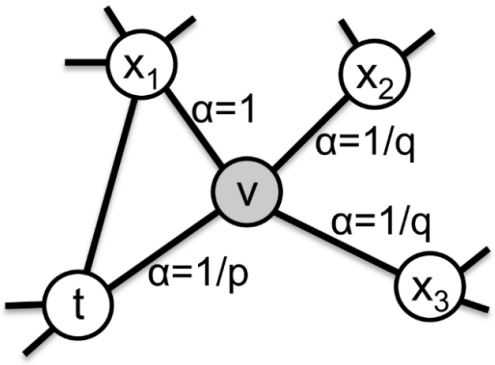

Figure 2: Biased random walk
Source: `https://snap.stanford.edu/node2vec/`

After simulating random walks, we get the list of node sequences represented graph like one shown in Figure 1. Then, this list of node sequences is fit to the skip-gram model for embedding vectors. The big picture of node2vec is shown in Figure 3.

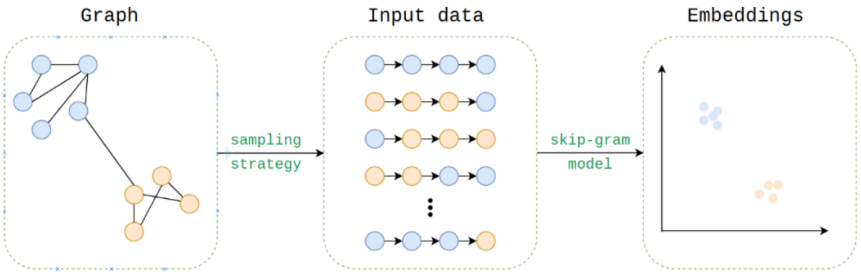

Figure 3: node2vec embedding process
Source: `https://towardsdatascience.com/node2vec-embeddings-for-graph-data-32a866340fef`

3 Conclusion

The current machine learning tasks are largely crowded with graph data. And converting the graph for making it suitable for learning many machine learning tasks is quite important. Therefore, after the introduction of the node2vec algorithm, it gains popularity in the machine learning arena. Additionally, this algorithm provides some flexibility for users by allowing

tweaking its hyperparameters. Hence, the user can decide what kind of information they intend to embed into the vector. A reference implementation of node2vec in Python is available on GitHub [1]. A high-performance implementation is included in SNAP and available on GitHub [2] as well.

References

[1] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[1] https://github.com/aditya-grover/node2vec

[2] https://github.com/snap-stanford/snap/tree/master/examples/node2vec