An Illustrated Explanation of Using SkipGram To Encode The Structure of A Graph (DeepWalk)

M

Feb 23 · 5 min read ★

Introduction

In this article, I will walk through an example of applying DeepWalk to create embeddings of a graph. I assume the reader is already familiar with the SkipGram algorithm.

Motivation

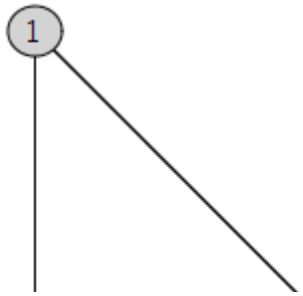
DeepWalk is an algorithm that is used to create embeddings of the nodes in a graph. The embeddings are meant to encode the community structure of the graph. It achieves this by using SkipGram to create the embeddings.

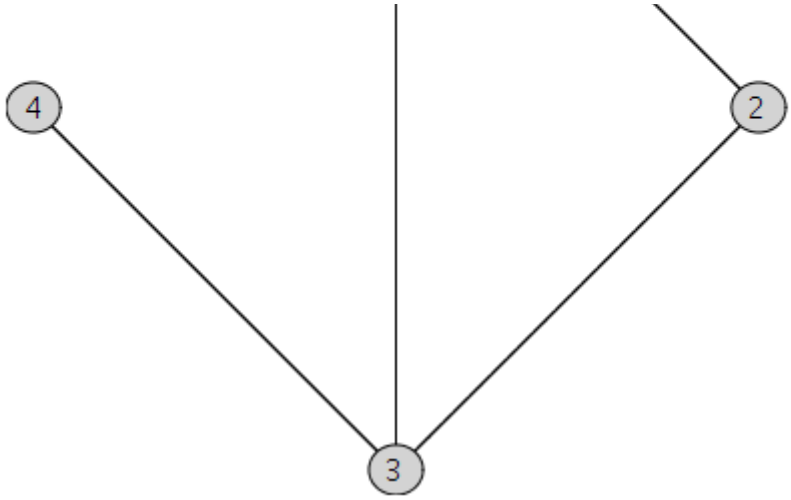
However, SkipGram is an algorithm in the Natural Language Processing (NLP) domain that is used to create word embeddings. How could such an algorithm be used to create graph embeddings? To understand the justification, we will examine how DeepWalk works by going through a small example.

Terminology

We will use the following informal definitions throughout this article:

A **graph** consists of circles that are joined by lines. For example:





We call the circles **nodes** and the lines connecting the circles **edges.**

The SkipGram Algorithm

SkipGram is an algorithm that is used to create word embeddings i.e. high-dimensional vector representation of words. These embeddings are meant to encode the semantic meaning of words such that words that are semantically similar will lie close to each other in that vector's space. It assumes words that are semantically similar are used in similar contexts.

For example, if your corpus consists of just the following sentences:

- I like dogs
- I like cats
- Dogs and cats make great pets
- I like dogs and cats

Then the vector representations for the word "cats" and "dogs" should be close to each other because the words surrounding them are similar.

Explanation

Goal

Suppose we have the following graph:

It is clear that this graph have two communities (call them community 1 and 2) in it, namely:

- Community 1: nodes 1, 2, 3, 4, 5, 6
- Community 2: nodes 8, 9, 10, 11, 12, 13

Our goal is to come up with good vector representations for each node. By good, we mean that the vector representations capture the graph's community structure. In this example, the vector representations are good if the vectors for nodes 1, 2, 3, 4, 5, 6 and nodes 8, 9, 10, 11, 12, 13 form distinct clusters.

How does DeepWalk achieve such a representation?

Solution

Since DeepWalk uses SkipGram to create the embeddings, the first step is to generate a corpus of the graph. In the NLP domain, a corpus is a collection of written texts. So how do we create a corpus for a given a graph?

To create a corpus from a graph, you simply perform multiple random walks on each node! A random walk means that you pick a starting node and then randomly pick an edge to move to that node's neighbor. You will repeat this step until you get a walk of some predefined length. For example, here's the result of performing 1 random walk of length 20 on each node in our example graph presented as a 13 x 20 matrix:

The i-th row refers to a random walk performed starting at node i. For example, the 8-th row reads:

This means that the random walk began at node 8, then it walked to node 9, then to node 10 then back to node 9 again, and then back to node 10, and then to node 11, and so on until in ended at node 9.

In NLP terms, we have 13 sentences and each sentence has 20 words in it. Our words are 1, 2, …, 13 and the size of the vocabulary is 13.

Now that we have a corpus, we can apply SkipGram to learn the embeddings of each node. To understand why the resulting embeddings will encode the graph's community structure, suppose for simplicity's sake that we will create the embeddings using a window of size 1 i.e. the context of a word are the words immediately to its left and right. For example:

In the diagram above I have highlighted the context for the word “2”. Notice that the context for the word “2” tend to be “1”, “3” and/or “6”. Therefore, the SkipGram embeddings for “1”, “2”, “3”, and “6” will be close to each other which is exactly what we want since these nodes are connected to each other.

In practice, DeepWalk will perform multiple random walks on each node to generate a large corpus. The following result is generated using DeepWalk with the following hyperparameters:

- Number of random walks per node: 80
- Length of 1 random walk: 40
- Window size: 10
- Embedding dimension: 128

The above diagram is the plot of the embeddings for each node using PCA (default arguments) to reduce the embedding dimension from 128 to 2. Notice that the embeddings clearly capture the community structure of the graph since nodes 1, 2, 3, 4, 5, 6 together form a distinct cluster from nodes 8, 9, 10, 11, 12, 13. As node 7 was in the middle of this two "communities", it makes sense that its embedding shows that it is somewhere in between these two clusters.

For comparison, here is what the embeddings look like using TSNE (perplexity = 5 since this is a small dataset):

The results are consistent with PCA.

Conclusion

In this article, we have seen that we can use SkipGram to capture the community structure of a graph by viewing the graph as a kind of language where:

- Each node is a unique word in the language
- Random walks of finite length on the graph constitutes a sentence

I hope this has improved your understanding of how DeepWalk works. Let me know in the comments if you have any questions.

References

DeepWalk: Online Learning of Social Representations; Perozzi et al. 2014

Word2Vec Tutorial — The Skip-Gram Model; McCormick. 2016

Machine Learning | Deep Learning | Knowledge Graph | Artificial Intelligence