Cheap Node2vec

March 2020

Abstract

Our main goal is to do a projection on large graphs, i.e graphs with hundreds of thousands and even millions of nodes, in the shortest time we can. The basic and known projection methods such as Node2Vec, HOPE and GCN are struggeling to deal with big graphs and their time complexity is high. Therefore, we would like to find a cheaper way to do projection on big graphs. In this paper we will introduce the problem and our solution (that is based on the node2vec algorithm), followed by an explanation of why current ways are having a hard time dealing with big graphs.

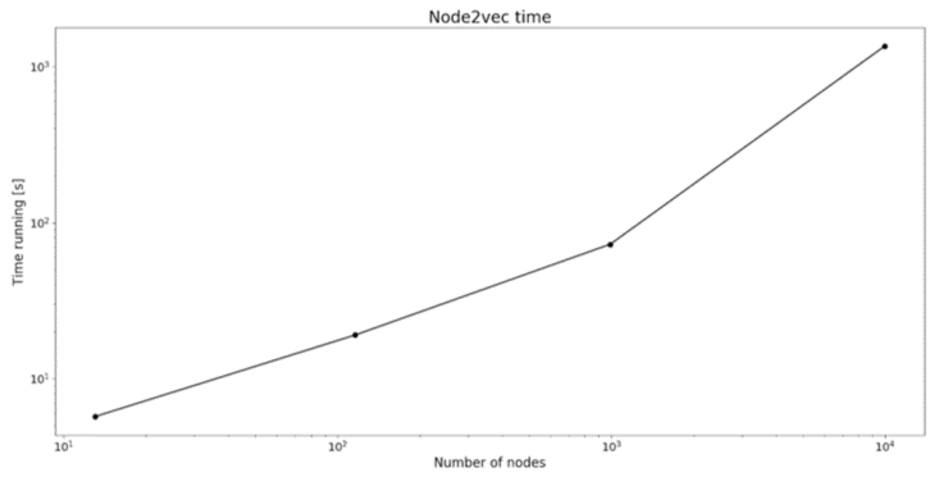
The Main Problem:

Given a large graph $G = (V, E, W)$, we would like to find the nodes' projection in the shortest time possible.

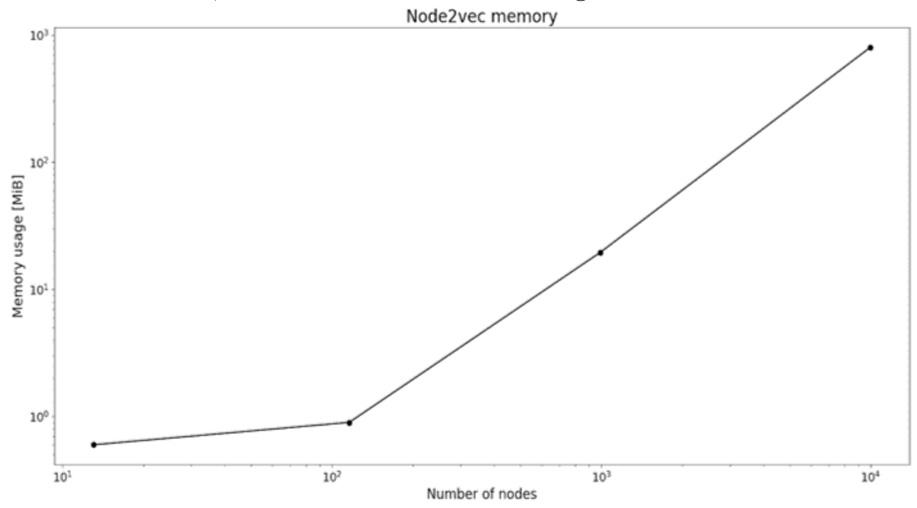
Current Solution to Graph Embedding - Node2Vec:

Node2Vec is an algoritmic framework for learning continuous featre representations for nodes in networks. In node2vec, we learn a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighberhoods of nodes. It is based on a random walk procedure, and seems to work very well on number of missions, such as link prediction and node classification, in comparision to state-of-the-art techniques.
However, its time complexity is $O(|V|d)$, meaning that in large networks it is going to be expansive to claculate the projection using it. We can see this

in the figure below:



We can see the running time is getting higher as the number of nodes is getting higher, and that's not good.
In addition, the memory usage of node2vec also increases as the number of nodes increases, as demonstrated in the fighure below:



With these two observations, we figured we need to look at this problem from a different angle- look on a sub-graph of the big graph, do a projection on it, and from this projection predict the projections of the other nodes of the graph. Thus, we have gotten a new problem to solve.

The New Problem:

Given a graph $G = (V, E, W)$, a group of nodes H such that $H \subseteq V$ and a projection of H - H^x, such that $H^x : i \to \overline{x}_i$ when $i \in H$ and $\overline{x}_i$ is its projection, we would like to find for every node $j \in G$ such that j is not in H, its projection $\overline{x}_j$.

Solution for Undirected Graphs:

Let H^x be the projection of a given group of nodes $H \subseteq V$.
For every node $j \in V$ we will find the first order neighbours set of j, N_{1j}, such that: $\forall n \in N_{1j} : n \in H$.
Let $\overline{x}_n$ be the projection of $n \in H$ (i.e $\overline{x}_n \in H^x$) and $|N_{1j}| = k_1$. Then the average projection of the first order neighbours of j is:

$$\overline{x}_1 = \frac{1}{k_1} \sum \overline{x}_n$$

For every node $j \in V$ we will find the second order neighbours set of j, N_{2j}, such that: $\forall a \in N_{2j} : a \in H$ and a is a neighbour of at least one of the first order neighbours of j.
Let $\overline{x}_a$ be the projection of $a \in H$ (i.e $\overline{x}_a \in H^x$) and $|N_{2j}| = k_2$. Then the average projection of the second order neighbours of j is:

$$\overline{x}_2 = \frac{1}{k_2} \sum \overline{x}_a$$

Now that we have these two average projections, we can use them in calculating $\overline{x}_j$, the projection of j:

$$\overline{x}_j = \overline{x}_1 + \epsilon * \frac{k_2}{k_1} * (\overline{x}_1 - \overline{x}_2)$$

when ϵ is a small parameter representing the weight we give to the influence of the second order neighbours.
Then, we get that $j \in H$ and $\overline{x}_j \in H^x$.
It is important to mention that after we find N_{1j} we order the nodes that

aren't in the initial projection (i.e not in H) by the size of N_{1j}, meaning we first calculate projection of nodes with the maximum number of first order neighbours that are already in the projection. After every calculation, we update its neighbours and re-order.
We will keep doing all that iteratively until we stay only with nodes that for them $|N_{1j}| = 0$, meaning they are not in the central connected component of G, and they are not needed in the final projection.
At the end, we get H- the group of nodes that are in the final projection, and H^x- the projection of all nodes that are in H, therefore we get the whole projection of G.

How to choose the nodes that are in the initial projection?

Choosing the number of nodes and the actual nodes that are in the initial projection is critical. The size of this group was chosen to be abput 5% of the size of the graph. The nodes were chosen to be the nodes with the maximum degree and maximum average degree because it seemed to be work well. However, one can try choose them differently, e.g by k-core score and more.

Solution for Directed Graphs:

Let H^x be the projection of a given group of nodes $H \subseteq V$.
For every node $j \in V$ we will find the set of first order nodes incoming to j, $N_{1,j}$, such that:

$$N_{1,j} = \{\, i \in V | (i,j) \in E\}$$

Let $\overline{x}_i$ be the projection of $i \in H$ (i.e $\overline{x}_i \in H^x$) and $|N_{1,j}| = k_1$. Then the average projection of the first order nodes incoming to j is:

$$\overline{x}_{j,1} = \frac{1}{k_1} \sum \overline{x}_i$$

For every node $j \in V$ we will find the set of first order nodes outgoing from j, $N_{2,j}$, such that:

$$N_{2,j} = \{\, i \in V | (j,i) \in E\}$$

Let $\overline{x}_i$ be the projection of $i \in H$ (i.e $\overline{x}_i \in H^x$) and $|N_{2,j}| = k_2$. Then the average projection of the first order nodes outgoing from j is:

$$\overline{x}_{j,2} = \frac{1}{k_2} \sum \overline{x}_i$$

Then we can define for every $j \in V$: $z_{j,kl}$,where $z_{j,11}$ is the average projection (i.e position) of nodes incoming to nodes incoming to j, $z_{j,12}$ is the average projection of nodes outgoing from nodes in coming to j, $z_{j,21}$ is the average projection of nodes incoming to nodes outgoing from j, $z_{j,22}$ is the average projection of nodes outfoing from nodes outgoing from j. With all these definitions, we can define the projection of the node $j \in V$ as:

$$w_j = \alpha_1 * x_{j,1} + \alpha_2 * x_{j,2} - \sum \beta_{k,l} * z_{j,kl}$$

where α_j , $\beta_{j,kl}$ are parameters with values we need to extract from the positions already in the graph (i.e from the position of nodes that are in the initial projection) using linear regression. Note that one can also have each beta multipillied by a function of k when k is the number of such nodes.
The values of α and β determine the influence of $\overline{x}_{j,1}$, $\overline{x}_{j,2}$ and $z_{j,kl}$ on the final projection of node j. Positive parameters mean we want the final projection to be in the same direction as the given variable, while negative parameters mean we want it in the oppisite direction. Big values mean more consideration in the variable and its direction, while smaller ones mean less consideration.

Visualization Task:

since embedding is a low-dimensional vector representation of nodes in the graph, it allows us to visualize the nodes to understand the network topology. As different embedding methods preserve different structures in the network, their ability and interpretation of node visualization differ. For instance, embeddings learned by node2vec with parameters set to prefer BFS random walk would cluster structurally equivalent nodes together. We compare the ability of regular node2vec method and our method, both for undirected and directed graphs, to visualize nodes on a few dataset attached in this git. We learn a 10-dimensional embedding for each method and input it to t-SNE to rduce dimensionality to 2 and visualize nodes in a 2-dimensional space. For labeled graph we also color the nodes by them.

Node Classification Task:

Predicting node labels using network topology is widely popular in network analysis and has variety of applications, including document classification and interest prediction. A good network embedding should capture the network structure and hence be useful for node classification. We compare the effectiveness of the regular node2vec and our suggested methods, for both directed and undirected graphs, on this task, by using the generated embedding as node features to classify the nodes. The node features are input to a one-vs-rest logistic regression using the LIBLINEAR library.
Then we commit two kinds of missions:
1. We compare between regular node2vec and our suggested methods. For each data set, we randomly sample 10% to 90% of nodes as training data and evaluate the performance on the remaining nodes. We perform this split 3 times and report the mean with confidence interval. We evaluate the preformance by 3 scores- micro-F1, macro-F1 and accuracy.
2. Only for our suggested methods, we plot graphs of each score as a function of number of nodes in the initial projection to see what influence it has on the success of the node classification mission.