node2vec: Scalable Feature Learning for Networks

Authors: Aditya Grover and Jure Leskovec

Chenyu Shi and Shupei Li

Leiden Institute of Advanced Computer Science

November 18, 2022

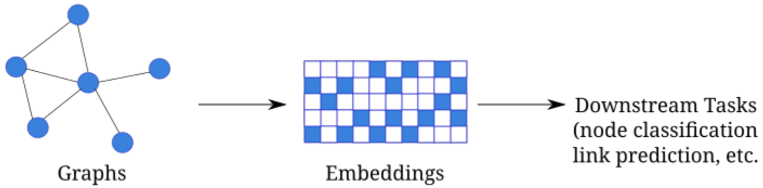
Universiteit Leiden
The Netherlands

Hello, everyone! We are group 43. The paper we are going to present today is node2vec: scalable feature learning for networks, written by Grover and Leskovec.

This is a brief content of our presentation. Firstly, we will introduce some background knowledge about graph embeddings, followed by a review of related work. Then, we will explain the node2vec algorithm in detail and present experimental results in the original paper. After that, we will introduce the idea of our course project. This presentation is concluded with the future work we plan to do in the project.

Introduction to Graph Embeddings

- Represent graph-structured data.
- Applications:
  Social network analysis, recommender systems, molecular structure modelling, etc.
- Challenge: Limitations of traditional methods.
- Development of techniques specially for graph representations.



- node2vec is a feature learning framework.

As we've seen in this course, complex relationships can be modelled by graphs. Graph-structured data appears in many modern applications like molecular structure modelling, social network analysis, recommender system, etc. However, if we want to leverage the information contained in graphs, we need to find a way to represent graph-structured data efficiently. Traditional statistical and machine learning methods are designed for extracting features from structured data, such as PCA, UMAP, t-SNE. Although these methods have achieved satisfactory performance on structured-data, they are hard to be generalized to graph-structured data, because they higly depend on properties of Eucildean space. This challenge led to the development of techniques specially for graph representation. Figure on the slide illustrates the general idea of graph embedding learning method. Firstly, we obtain embeddings by mapping graphs onto a low dimension space. And then, we can use these features in some downstream tasks like node classification and link prediction. From this perspective, node2vec algorithm that we address in presentation is a feature learning framework.

Related Work

A taxonomy of graph embedding techniques[1].

- Shallow Embedding Method
  - Supervised Learning: Label propagation
  - Unsupervised Learning / Semi-supervised Learning
    - Distance-based: Multi-dimensional scaling
    - Outer Product-based
      - Matrix Factorization
      - Skip-gram: DeepWalk, LINE, node2vec
- Deep Embedding Method: GCN

[1] Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2022. ISBN: 9780262046824.

This figure illustrates the taxonomy of graph embedding techniques, according to a textbook written by Murphy. There are two types of graph embedding techniques. They are shallow embedding method and deep embedding method. Shallow embedding methods use shallow encoding functions to map the original graph structure onto a Euclidean space and obtain the embedding matrix. If data is labelled, we can apply supervised algorithms to extract embeddings, for example, label propagation. However, labels are not avaliable or only partly avaliable in most cases, where we need to use unsupervised or semi-supervised learning. These methods can be divided into distance-based method and outer product method further. Some representative algorithms are given in the figure. It is worth noting that node2vec algorithm we emphasize in presentation belongs to outer product-based method using skip-gram architecture, which is inspired by skip-gram model in natral language processing field. Deep embedding method usually refers to algorithms that learn graph features via graph neural networks, or GNN. GNN is a special class of artificial neural networks designed for graph-structured data. In our project experiments, we mainly use a method called graph convolutional networks, which is a popular GNN architecture.

Feature Learning Framework

- node2vec is a feature learning framework in nature.
- Goal: Given a network $G = (V, E)$, find a projection $f: V \to R^d$.
- Generate a d-dimesion vector representation for each node.
- f can be formulated as a matrix of size $|V| \cdot d$.

Feature Learning Framework

Extending skip gram architecture to networks.

Formulate feature learning in networks as a maximum likelihood optimization problem:

$$\max_{f} \sum_{u \in V} \log Pr\left(N_S\left(u\right)|f(u)\right)$$

$N_S\left(a\right)$ is the network neighborhood set generated by neighborhood sampling strategy S for node a.

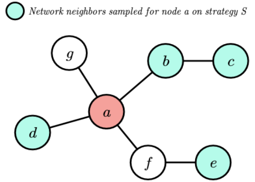
Important: $N_S\left(a\right)$ isn't equavalent to direct local neighborhood.

For NLP: Given a literal data: *This is a [feature learning framework social network].*

$Pr(\{$"feature","learning","social","network"$\}|$"framework"$)$

For Graph data:

$$N_S\left(a\right) = \{b, c, d, e\}$$
$$Pr\left(\{b, c, d, e\}|a\right) = Pr\left(N_s\left(a\right)|a\right)$$



$N_s(a) = \{b, c, d, e\}$

Feature Learning Framework

Extending skip gram architecture to networks.

Formulate feature learning in networks as a maximum likelihood optimization problem:

$$\max_{f} \sum_{u \in V} \log Pr\left(N_S\left(u\right) | f(u)\right)$$

$N_S\left(a\right)$ is the network neighborhood set generated by neighborhood sampling strategy S for node a.

Important: $N_S\left(a\right)$ isn't equavalent to direct local neighborhood.

For NLP: Given a literal data: *This is a [feature learning framework social network]*.

$Pr(\{\text{"feature","learning","social","network"}\}|\text{"framework"})$

For Graph data:

$$N_S\left(a\right) = \{b, c, d, e\}$$
$$Pr\left(\{b, c, d, e\} | a\right) = Pr\left(N_s\left(a\right) | a\right)$$

Two problems to be solved:

1. How to define $N_S\left(a\right)$?
2. How to compute $Pr\left(N_S\left(a\right) | a\right)$?

Maximum Likehood Optimization

Formulate feature learning in networks as a maximum likelihood optimization problem:

$$\max_{f} \sum_{u \in V} \log Pr\left(N_S\left(u\right) | f(u)\right)$$

Two standard assumptions:

1. Conditional independence:

$$Pr\left(N_S\left(u\right) | f\left(u\right)\right) = \prod_{n_i \in N_S(u)} Pr\left(n_i | f(u)\right)$$

2. Symmetry in feature space:

$$Pr\left(n_i | f\left(u\right)\right) = \frac{\exp\left(f\left(n_i\right) \cdot f\left(u\right)\right)}{\sum_{v \in V} \exp\left(f\left(v\right) \cdot f\left(u\right)\right)}$$

Maximum Likehood Optimization

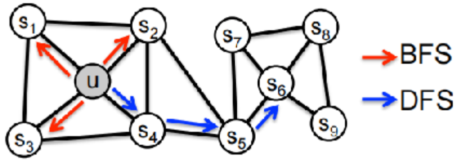
Finally, the optimization problem is converted into the form of:

$$\max_f \sum_{u \in V} \left[-\log \left(\sum_{u \in V} \exp\left(f(u) \cdot f(v)\right) \right) + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]$$

Use stochastic gradient decent to obtain projection f.

Network Neighborhood Sampling Strategy

Use classic search strategies:
Breadth-first Sampling (BFS) and Depth-first Sampling (DFS).



There are two kinds of similarities:

1. homophily (such as u and s_1)
2. structural equivalence (such as u and s_6)

DFS tends to discover homophily, BFS tends to discover structural equivalence.

How to discover both kinds of similarities?

Network Neighborhood Sampling Strategy

Use basic random walk to discover both homophily and structural equalvalence similarities.

Basic random walk with length l from source node u:

$$P\left(c_i = x | c_{i-1} = v\right) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

c_i: the i-th node in the walk.

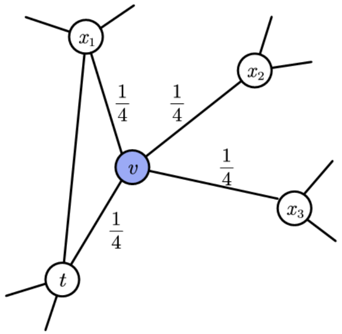
v: current node.

π_{vx}: unnormalized transition probability.

Z: normalization constant.

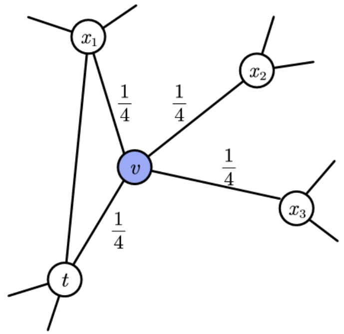
Network Neighborhood Sampling Strategy

π_{vx}: Often set $\pi_{vx} = w_{vx}$ in weighted graphs.
In unweighted graph: $\pi_{vx} = 1$.



Network Neighborhood Sampling Strategy

π_{vx}: Often set $\pi_{vx} = w_{vx}$ in weighted graphs.
In unweighted graph: $\pi_{vx} = 1$.



Random walk can combine features of DFS and BFS, and discovery both two kinds of similarities.

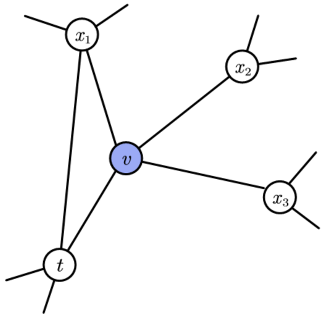
Still not enough:
It's hard for us to guide and control the walking process.

Network Neighborhood Sampling Strategy

Use the second order bias random walk to get control of the walking process.

$$\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$$

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$



v: current node.

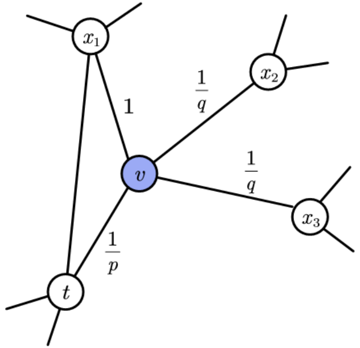
t: last node in the walk.

x: next node to be chosen.

Network Neighborhood Sampling Strategy

p: return parameter.

q: in-out parameter.



p :

- High value: Less likely to sample an already visited node.
- Low value: Likely to step back, then walk locally near the source node u.

q :

- High value: Biased towards nodes close to t, act more similarly to BFS.
- Low value: Biased towards nodes distant to t, act more similarly to DFS.

Learning Edge Features

We have found a projection $f : V \to R^d$ with node2vec, which allocates each node vector embedding representation.

These embedding vectors can be used in node-related downstream tasks.

But how to learn edge features and deal with edge-related downstream tasks?

Learning Edge Features

We have found a projection $f : V \to R^d$ with node2vec, which allocates each node vector embedding representation.

These embedding vectors can be used in node-related downstream tasks.

But how to learn edge features and deal with edge-related downstream tasks?

Operator	Symbol	Definition
Average	$\boxplus$	$[f(u) \boxplus f(v)]_i = \frac{f_i(u)+f_i(v)}{2}$
Hadamard	$\boxdot$	$[f(u) \boxdot f(v)]_i = f_i(u) * f_i(v)$
Weighted-L1	$\Vert\cdot\Vert_{\bar{1}}$	$\Vert f(u) \cdot f(v)\Vert_{\bar{1}_i} = \lvert f_i(u) - f_i(v)\rvert$
Weighted-L2	$\Vert\cdot\Vert_{\bar{2}}$	$\Vert f(u) \cdot f(v)\Vert_{\bar{2}_i} = \lvert f_i(u) - f_i(v)\rvert^2$

Given projection f obtained by node2vec and two nodes u, v along with edge (u, v), apply the binary operator on $f(u)$ and $f(v)$ to generate the representation $g(u, v)$, where $g : V \times V \to R^{d'}$.

Experiment 1: Multi-label Classification

- Task description
  - Labels from a finite set $\mathcal{L}$
  - Training: A fraction of nodes and all their labels.
  - Predict the labels for the remaining nodes.
- Data

Dataset	Nodes	Edges	Labels
BlogCatalog	10,312	333,983	39
Protein-Protein Interactions (PPI)	3,890	76,584	50
Wikipedia	4,777	184,812	40

- Metrics: Macro-F1 score.

Authors of the original paper conducted two experiments to verify the effectiveness of the node2vec algorithm. The first experiment is multi-label classification. Given a finite label set and a fraction of nodes with labels, the task is predicting the labels for the remaining nodes. They use three data sets: BlogCatalog, PPI, and Wikipedia. The table shows some statistics of data sets. Macro-F1 score is selected as metrics.

Experiment 1: Multi-label Classification

- Results

Algorithm		**Dataset**	
	BlogCatalog	PPI	Wikipedia
Spectral Clustering	0.0405	0.0681	0.0395
DeepWalk	0.2110	0.1768	0.1274
LINE	0.0784	0.1447	0.1164
node2vec	**0.2581**	**0.1791**	**0.1552**
node2vec settings (p, q)	0.25, 0.25	4, 1	4, 0.5
Gain of node2vec [%]	**22.3**	**1.3**	**21.8**

- node2vec outperforms the other benchmark algorithms.

Spectral clustering, DeepWalk, and LINE are selected as benchmarks. From the table, we can see that node2vec outperforms other algorithms on all datasets. Besides, its relative performance gain is significant.

Experiment 2: Link Prediction

- Task description
  - A network with a fraction of edges removed.
  - Predict these missing edges.
- Data

Dataset	Nodes	Edges
Facebook	4,039	88,234
Protein-Protein Interactions (PPI)	19,706	390,633
arXiv ASTRO-PH	18,722	198,110

- Metrics: Area Under Curve (AUC) score.

The second task is link prediction. Given a network with a fraction of edges removed, the task is predicting these missing edges. They also use three datasets, Facebook, PPI, and arXiv, in this experiment. Statistics of data sets are shown in the table. AUC score is used as metrics.

Experiment 2: Link Prediction

- Results

Algorithm	Dataset		
	Facebook	PPI	arXiv
Common Neighbors	0.8100	0.7142	0.8153
Jaccard's Coefficient	0.8880	0.7018	0.8067
Adamic-Adar	0.8289	0.7126	0.8315
Pref. Attachment	0.7137	0.6670	0.6996
Spectral Clustering	0.6192	0.4920	0.5740
DeepWalk	**0.9680**	0.7441	0.9340
LINE	0.9490	0.7249	0.8902
node2vec	**0.9680**	**0.7719**	**0.9366**

- The learned feature representations outperform heuristic scores. node2vec achieves the best AUC.

When the original paper was published, there were no feature learning algorithms that had been used for link prediction. Therefore, authors set four heuristic scores as benchmarks. The first four lines of table shows the performance of heuristic scores. The other three benchmarks are the same as in the experiment 1. Authors tested average, hadamard, weighted-L1, and weighted-L2 operations on these three benchmarks as well as node2vec. Due to the limited space, table on the slide only shows the result of hadamard operation. We can see that the learned feature representations outperform heuristic scores. And node2vec achieves the best AUC score on all datasets.

Summary of node2vec

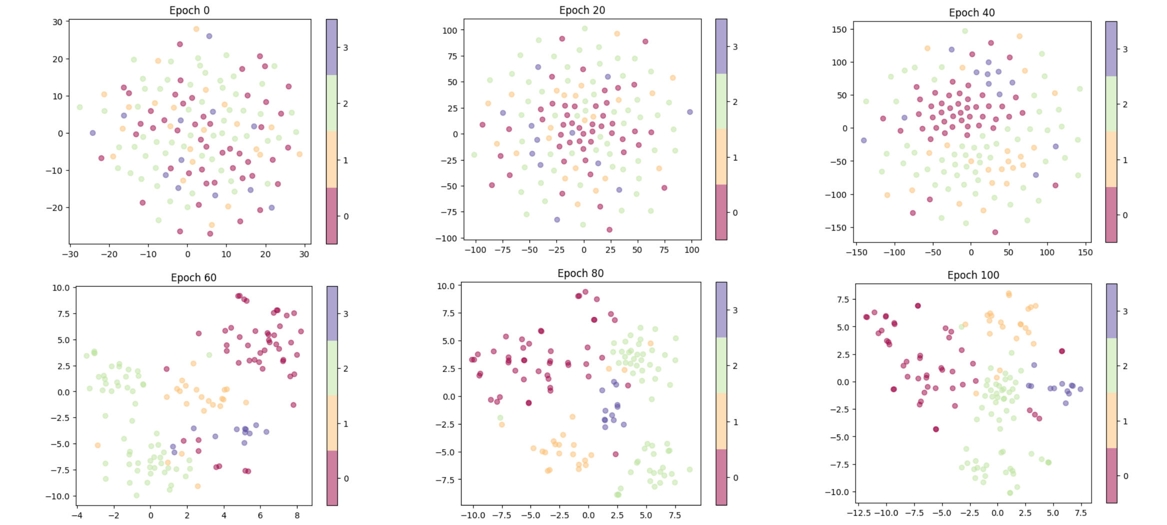
- An efficient graph embedding learning algorithm.
- Search strategy: Both flexible and controllable exploring network neighborhoods.

A brief summary of node2vec. It is an algorithm designed for extracting embedidngs from graphs. Results of experiments support the effectiveness of the algorithm. Its neighborhood search strategy is flexible and controllable. We can easily change the behavior of search by adjusting hyperparameter p and q.

Our Contributions

1. During the training of node2vec, the intermediate state of node embeddings is a black box.
   Our solution: Visualize the node embeddings during the training of node2vec with t-SNE technique.
2. Randomly initialized inputs in GNN affect the robustness of model performance and extend the model training time.
   Our solution: Propose a novel method that combines node2vec and GNN.
3. Effectiveness of algorithms.
   Our solution: Evaluate node2vec, GNN, and our proposed method on five real-world data sets with metrics that are different from the original paper.

Next, I will talk about our contributions in the project. Firstly, the intermediate state of node embeddings during node2vec training is a black box. We try to visualize the node embeddings with t-SNE technique to provide some insights into node2vec. Secondly, we find that researchers in graph embedding learning field have focused more on deep embedding methods these years. However, randomly initialized inputs in GNN affect the robustness of model performance and extend the training time. To address the problem, we proposed a novel method that combines node2vec and GNN. Thirdly, to further test the effectiveness of algorithms, we evaluate node2vec, GNN, and our proposed method on five real world datasets and select different metrics.



AIFB dataset: Each node has a category label. There are 4 classes in total.

During the training of node2vec, nodes embedding are changed from chaos into order.

We apply node2vec algorithm on an open source dataset AIFB. Results of visualization are showed on the slide. Each node in AIFB belongs to a class. The total number of classes is four. We set the number of epochs to 100. It is easy to see that nodes embeddings are changed from chaos into order during the training of node2vec.

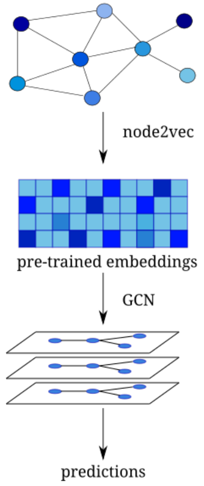
Proposed Method

node2vec + GNN

- Inspired by the concept of meta learning.
- An improved version of GNN.
- Choose the graph convolutional network (GCN) in our experiments.
    - GCN iteration formula:

$$h^{(k)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}h^{(k-1)}W^{(k)})$$

with $\hat{A} = A + I$, where A is the adjacency matrix and $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$.



As mentioned before, random initialization in GNN may not be a good strategy. Inspired by the concept of meta learning, we use the pretrained embeddings from node2vec as the meta information for GNN. Figure on the right illustrates the architecture of our proposed model. We expect our method will accelerate the training stage of GNN and improve the model performance. We choose graph convolutional network or GCN in our experiments. The iteration formula of GCN is given on the slide. Preliminary results we have obtained so far support the effectiveness of our proposed method.

Future work

1. Apply node2vec, GCN, and our proposed model to node classification and link prediction.
2. Try different strategies of hyperparameter tuning.

Our future work will mainly focus on two tasks. Task one, apply node2vec, GCN, and our proposed model to node classification and link prediction. Task two, try different strategies of hyperparameter tuning. Due to the time limit, we omit technical details of GNN. If you are interested in GNN, we can discuss the principle of GNN together. Thank you. This's all of our presentation. Any questions?

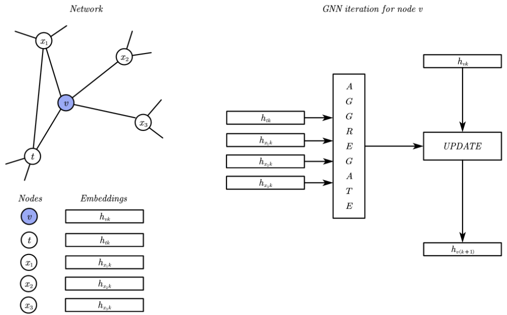
Graph Neural Network

Graph Neural Network is a deep learning framework for graph.

General GNN iteration formula:

$$h_u^{(k)} = \sigma(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_n^{(k)} \sum_{v \in N(u)} h_v^{(k-1)} + b^{(k)})$$

- $h_u^{(k)}$: k-th layer output embedding of node u.
- $W^{(k)}$: weights of k-th layer (trainable).
- $b^{(k)}$: bias of k-th layer (trainable).
- σ: activation function.



Aggregate: To aggregate embeddings of u's neighborhood.

Update: To update u's embeddings using aggregation result and previous u's embeddings.

Graph Neural Network

$$h_u^{(k)} = \sigma(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_n^{(k)} \sum_{v \in N(u)} h_v^{(k-1)} + b^{(k)})$$

Final output embeddings can be used for calculating predictions. Then, we can use these predictions to compute the loss and optimize parameters with back propagation.

Problem: How to produce h_u^0 for each node?

1. Use one-hot vector for each node.
   - Drawback: The total number of nodes is large. Using one-hot vector for each node will cause the input tensor to be very sparse.
2. Transfer pretrained embeddings from other similar tasks.
   - Drawback: There can't always be a similar task with pretrained embeddings for every network.
3. Use a trainable embedding layer to allocate randomly initialized embeddings for each node.
   - Drawback: Randomly initialized embeddings could have negative impact on training process.

Combine node2vec and GNN

node2vec can produce embeddings for each node with graph information.

GNN lacks a good general method to initialize its input nodes' embeddings.

We can use node2vec to produce initial input nodes' embeddings for GNN to improve training process of GNN and obtain better performance.

It's a general method because there is no specific requirement of graphs when applying node2vec and GNN.