

# node2vec: Scalable Feature Learning for Networks

Authors: Aditya Grover and Jure Leskovec

Chenyu Shi and Shupeí Li

Leiden Institute of Advanced Computer Science  
November 18, 2022



**Universiteit  
Leiden**  
The Netherlands

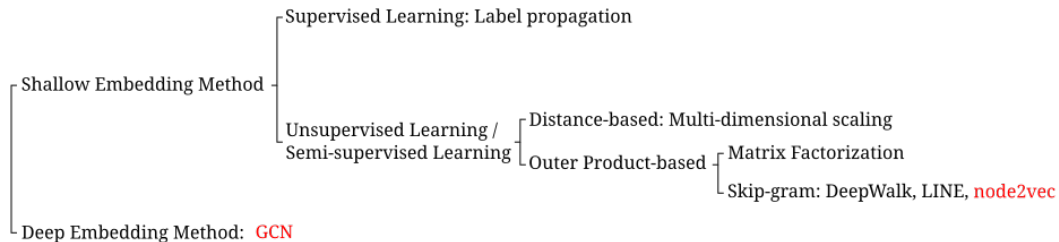
- ① Introduction
- ② Related Work
- ③ Methodology
- ④ Experiments
- ⑤ Our work
- ⑥ Future work

# 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.

# Related Work

A taxonomy of graph embedding techniques<sup>1</sup>.



<sup>1</sup>Kevin P. Murphy. *Probabilistic Machine Learning: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2022. ISBN: 9780262046824.

# Feature Learning Framework

- Goal: Given a network  $G = (V, E)$ , find a projection  $f : V \rightarrow R^d$ .
- 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:

$$\max_f \sum_{u \in V} \log Pr(N_S(u) | f(u))$$

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

Important:  $N_S(a)$  isn't equivalent to direct local neighborhood.

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

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

For Graph:

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

# Feature Learning Framework

Extending skip gram architecture to networks.

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

$$\max_f \sum_{u \in V} \log Pr(N_S(u) | f(u))$$

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

Important:  $N_S(a)$  isn't equivalent to direct local neighborhood.

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

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

For Graph:

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

Two problems to be solved:

- ① How to define  $N_S(a)$ ?
- ② How to compute  $Pr(N_S(a) | a)$ ?

# Maximum Likelihood Optimization

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

$$\max_f \sum_{u \in V} \log \Pr(N_S(u) | f(u))$$

Two standard assumptions:

① Conditional independence:

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

② Symmetry in feature space:

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



# Maximum Likelihood Optimization

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

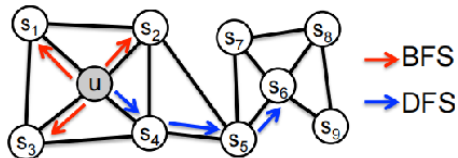
$$\max_f \sum_{u \in V} \left[ -\log \left( \sum_{v \in V} \exp(f(u) \cdot f(v)) \right) + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]$$

Use gradient decent stochastic 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  $u$ :

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

$c_i$ : the  $i$ -th node in the walk.

$v$ : current node.

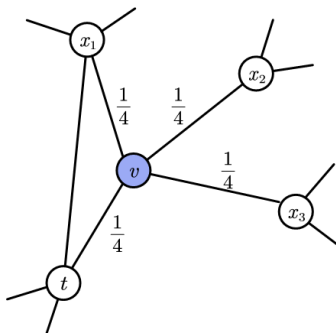
$\pi_{vx}$ : unnormalized transition probability.

$Z$ : normalization constant.

# Network Neighborhood Sampling Strategy

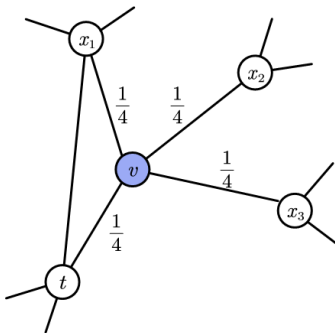
$\pi_{vx}$ : Often set  $\pi_{vx} = w_{vx}$  in weighted graphs.

In unweighted graph:  $\pi_{vx} = 1$ .



# Network Neighborhood Sampling Strategy

$\pi_{vx}$ : 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 discover both two kinds of similarities.

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

# Network Neighborhood Sampling Strategy

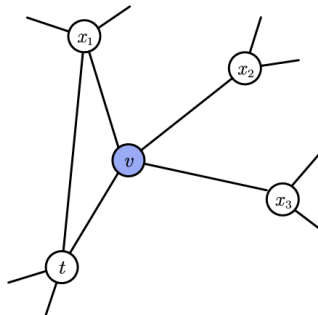
Use the second order bias random walk to get control of the walking process.

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

$v$ : current node.

$t$ : last node in the walk.

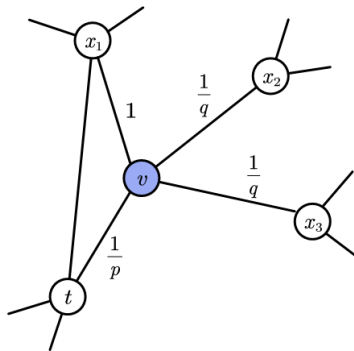
$x$ : next node to be chosen.



# Network Neighborhood Sampling Strategy

$p$ : return parameter.

$q$ : in-out parameter.



$p$  :

- when set a high value: less likely to sample an already visited node.
- when set a low value: likely to step back, then walk locally near the source node  $u$ .

$q$  :

- when set a high value: biased towards nodes close to  $t$ , act more similarly to BFS.
- when set a low value: biased towards nodes distant to  $t$ , act more similarly to DFS.

# Learning Edge Features

We have found a projection  $f : V \rightarrow R^d$  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  $f : V \rightarrow R^d$  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	$\ \cdot\ _1$	$\ f(u) \cdot f(v)\ _{1_i} =  f_i(u) - f_i(v) $
Weighted-L2	$\ \cdot\ _2$	$\ f(u) \cdot f(v)\ _{2_i} =  f_i(u) - f_i(v) ^2$

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  $g : V \times V \rightarrow R^{d'}$ .

# 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.

# 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
node2vc	<b>0.2581</b>	<b>0.1791</b>	<b>0.1552</b>
node2vec settings (p, q)	0.25, 0.25	4, 1	4, 0.5
Gain of node2vec [%]	<b>22.3</b>	<b>1.3</b>	<b>21.8</b>

## Experiment 2: Link Prediction

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

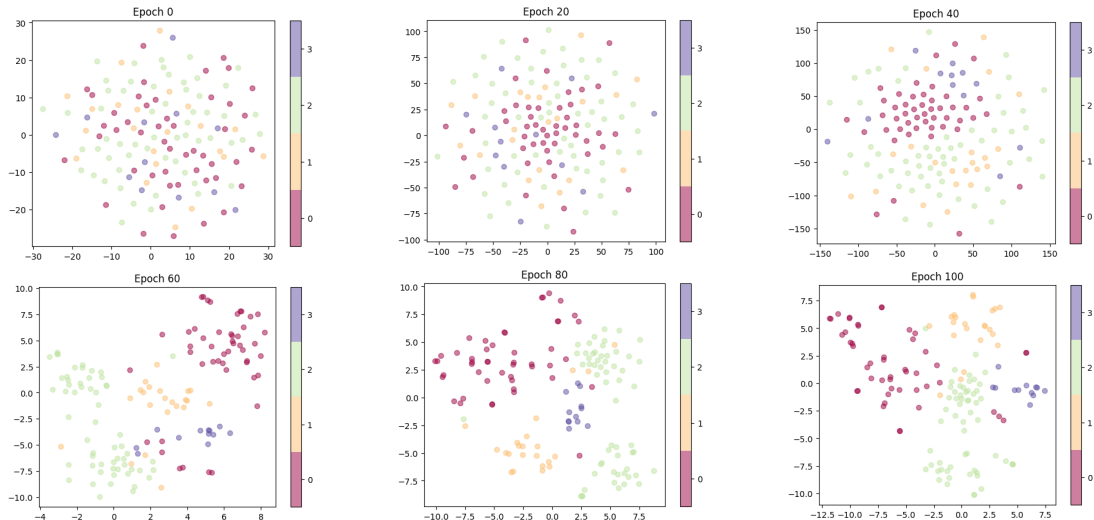
Dataset	Nodes	Edges
Facebook	4,039	88,234
Protein-Protein Interactions (PPI)	19,706	390,633
arXiv ASTRO-PH	18,722	198,110

- Metrics: Area Under Curve (AUC) score.

## 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	<b>0.9680</b>	0.7441	0.9340
LINE	0.9490	0.7249	0.8902
mode2vec	<b>0.9680</b>	<b>0.7719</b>	<b>0.9366</b>



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.

# Graph Neural Network

Graph Neural Network is a deep learning framework for graph.

General GNN iteration formula:

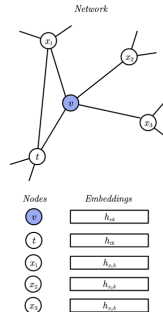
$$h_u^{(k)} = \sigma(W_{\text{self}}^{(k)} h_u^{(k-1)} + W_n^{(k)} \sum_{v \in N(u)} h_v^{(k-1)} + b^{(k)})$$

$h_u^{(k)}$ :  $k$ -th layer output embedding of node  $u$ .

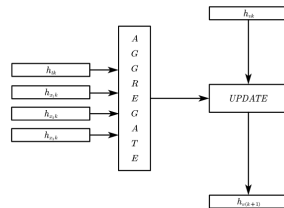
$W^{(k)}$ : weights of  $k$ -th layer (trainable).

$b^{(k)}$ : bias of  $k$ -th layer (trainable).

$\sigma$ : activation function.



GNN iteration for node  $v$



**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.



# 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.

# 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.

# 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.

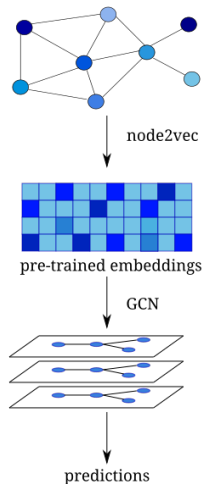
# Proposed Method

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  $\hat{A}$ .



# Preliminary Results

- Node classification: Given a fraction of nodes and their classes, predict the classes of the remaining nodes.
- Data

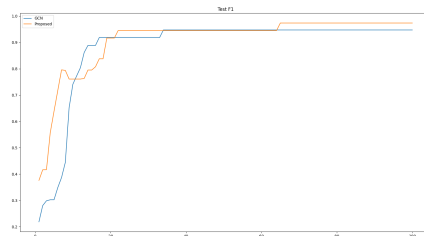
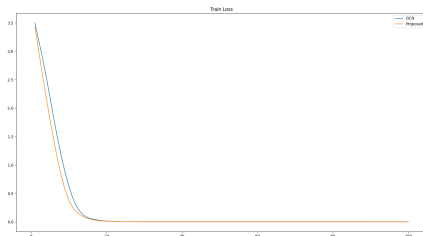
Dataset	Nodes	Edges	Classes
AIFB	8,285	58,086	4
MUTAG	23,644	148,454	2
PubMed	19,717	88,648	3

- Metrics: Accuracy, recall, and weighted F1 score.

# Preliminary Results

- Results: AIFB

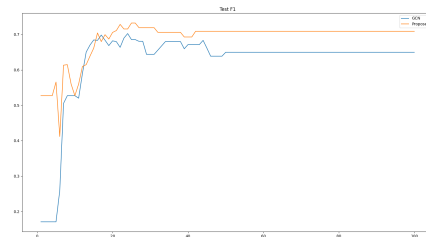
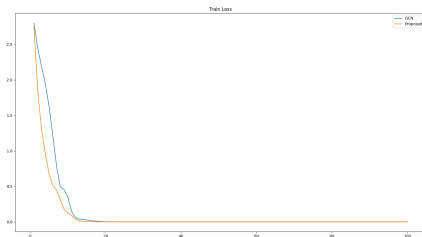
Algorithm	acc	recall	F1
node2vec	0.9444	0.9444	0.9421
GCN	0.9444	0.9444	0.9473
Proposed	<b>0.9722</b>	<b>0.9722</b>	<b>0.9737</b>



# Preliminary Results

- Results: MUTAG

Algorithm	acc	recall	F1
node2vec	<b>0.7941</b>	0.5652	0.6500
GCN	0.7206	0.7206	0.7024
Proposed	0.7353	<b>0.7353</b>	<b>0.7322</b>





# Preliminary Results

- Results: PubMed

Algorithm	acc	recall	F1
node2vec			
GCN			
Proposed			

# Future work

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

Thanks!