DeepWalk: Online Learning of Social Representations

Bryan Perozzi, Rami Al-Rfou, Steven Skiena
ACM SIG-KDD 2014

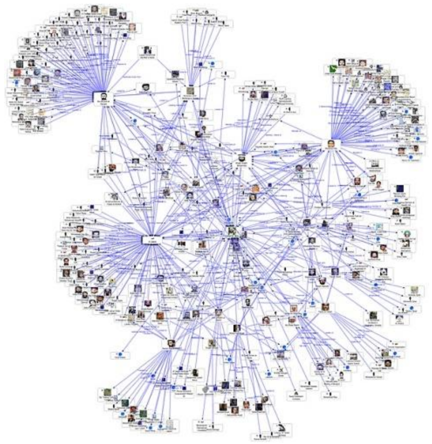
DSAIL 20222 Winter Lab Intern Seohyun Lim

목차

01 Introduction

Graph

There are many graphs used in real world. We may want to execute tasks in Machine Learning/Deep Learning with these graph data.



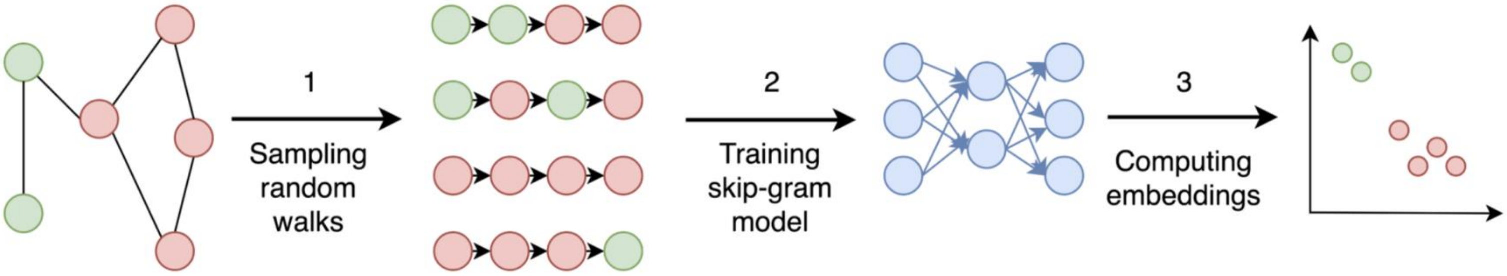
- network classification
- content recommendation
- anomaly detection

...

01 Introduction

DeepWalk

- Deep walk is a graph embedding method.
- It uses method called “Random Walk” to make node sequence and use it on logics of language model to express the node in vector type.



01 Introduction

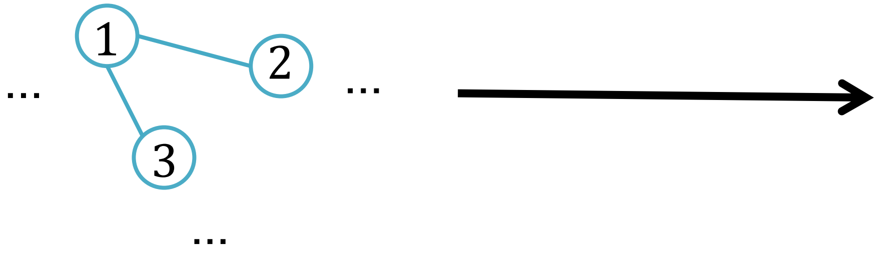
DeepWalk’s contribution

- learns the graph structure with short random walks.
- it’s representation outperform its competitors with even less data.
- is parallelizable and can represent large web-scale graphs.

02 Problem Definition

Goal

The goal of DeepWalk is to map nodes into an embedding space that has less dimension than the number of nodes in the graph.



Adjacency Matrix

1	0	1	1	
2	1	0	0	…
3	1	0	0	

⋮

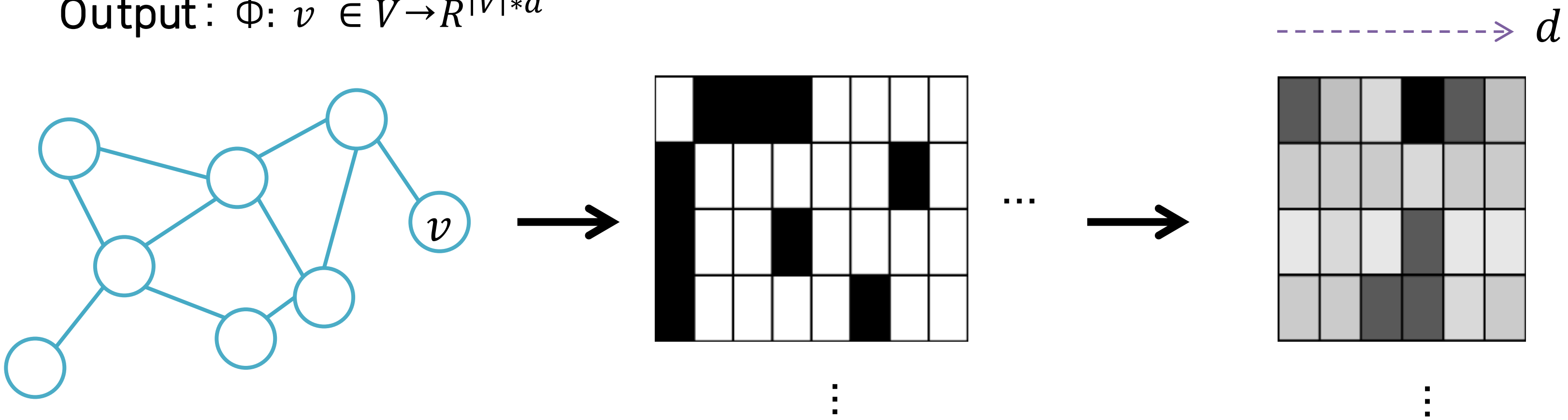
02 Problem Definition

Goal

The goal of DeepWalk is to map nodes into an embedding space that has less dimension than the number of nodes in the graph.

Input : Graph = (V, E, X, Y) and HyperParameters (window size, embedding size ..)

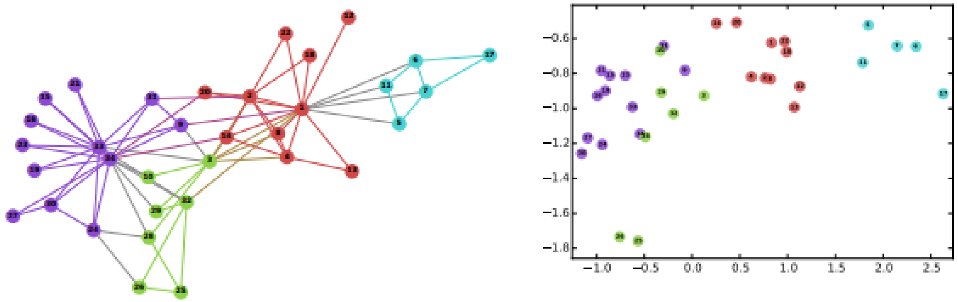
Output : $\Phi: v \in V \rightarrow R^{|V|*d}$



02 Problem Definition

Goal

The similarity of a pair of nodes in the graph should correspond to their similarity in the embedding space as well.

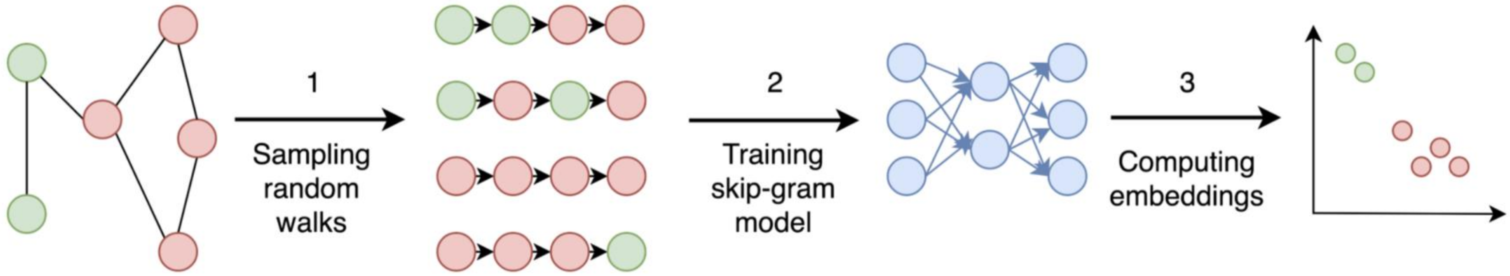


(a) Input: Karate Graph (b) Output: Representation

03 Background

Learning Social Representation

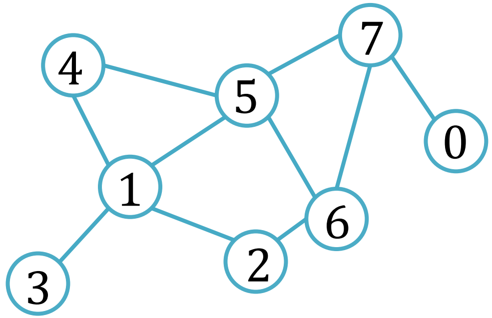
- RandomWalk
- Skip Gram
- Language Modeling



03 Background

Random walk

- Random walk is a series of nodes, where next node is chosen randomly from the adjacent nodes.
- We can express random walk with $W v_i$ meaning a random walk starting at v_i.

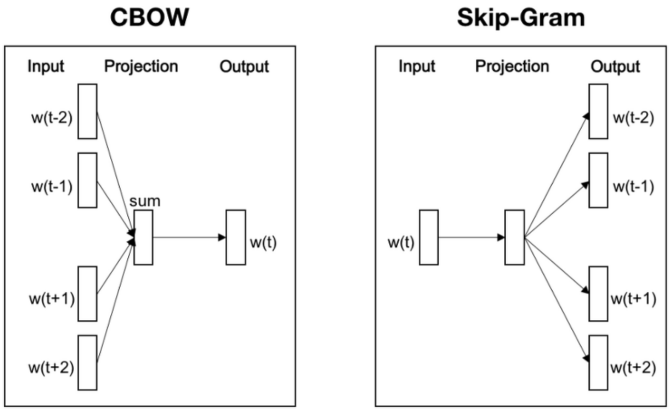


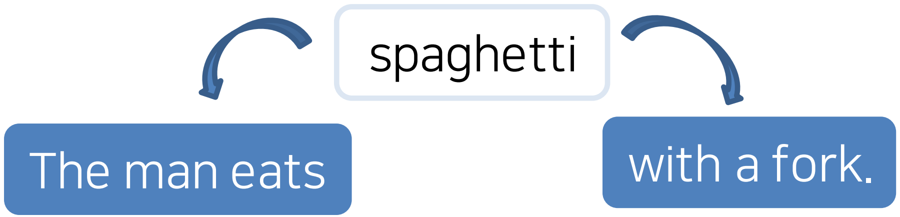
if walk length is 5 then..
[1, 2, 6, 5, 4]
[5, 4, 1, 2, 6]
[7, 6, 5, 1, 3]
...

03 Background

Skip-gram

- Word2Vec language model has two embedding algorithm methods.
- **CBOW** method tries to predict the center word based on the source of the context words.
- On the other hand, **Skip-gram** predicts the context words with the center word.



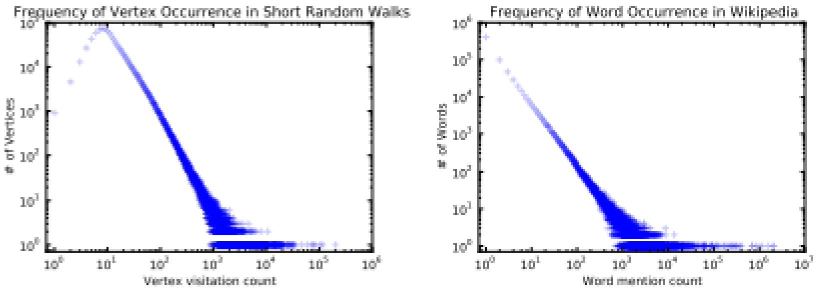


03 Background

Skip-gram

Language Model : Word – Sentence - Corpus ex. ['The', 'man', 'eats', … 'fork']
DeepWalk : Vertex – Random walk – Random walks ex. [1, 3, 5, 4, … Vt]

The language corpus word frequency follows power law and vertex frequency also follows the power law.



(a) YouTube Social Graph (b) Wikipedia Article Text

03 Background

Language Model

Language Model : $maximize\ \Pr(\ wn \mid w0, w1, \ldots\ , wn - 1)$

DeepWalk : $maximize\ \Pr(\ vi \mid \Phi(v1), \Phi(v2),\ \cdots, \Phi(vi - 1))$

However, when there are lots of vertex, the computational cost grow exponentially.

Therefore, the function changes to calculating probability of context words of v_i.

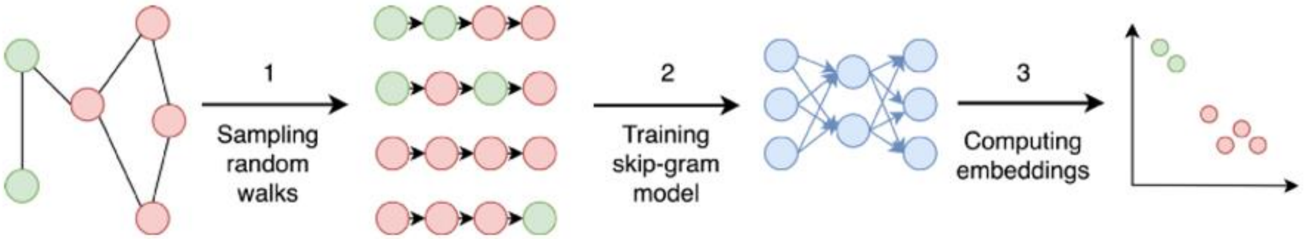
$$minimize\ \Phi \quad -\log \Pr(\{v_{i-w}, \cdots, v_{i-1}, v_{i+1}, \cdots, v_{i+w}\} \mid \Phi(v_i))$$

04 Method

DeepWalk Algorithm

DeepWalk consists of two parts :

1. Random walk generator
2. Skip-gram



04 Method

DeepWalk Algorithm

Input : $Graph = (V, E, X, Y)$ and HyperParameters

- window size w
- embedding size d
- walks per vertex γ
- walk length t

Output : Φ: $v \in V \rightarrow R^{|V| * d}$

Algorithm 1 DEEPWALK(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

```
Initialization: Sample Φ from U^{|V|×d}
Build a binary Tree T from V
for i = 0 to γ do
    O = Shuffle(V)
    for each v_i ∈ O do
        W_{v_i} = RandomWalk(G, v_i, t)
        SkipGram(Φ, W_{v_i}, w)
    end for
end for
```

04 Method

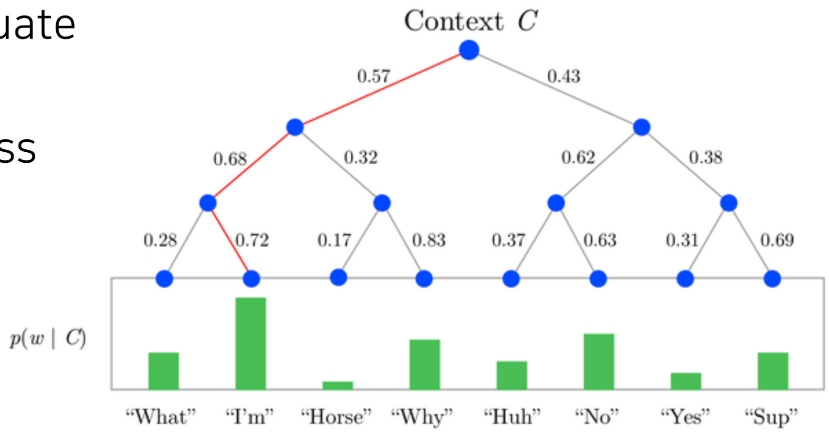
Hierarchical Softmax

- utilizes a multi-layer binary tree where the probability of a word is calculated on whether it goes to left or right edge
- an alternative to *softmax* but faster to evaluate
- O(log n) time compared to O(n) for *softmax*
- can use Huffman coding to reduce the access time of frequent elements in the tree

Algorithm 1 DEEPWALK(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

```
Initialization: Sample Φ from U^{|V|×d}
Build a binary Tree T from V
for i = 0 to γ do
    O = Shuffle(V)
    for each v_i ∈ O do
        W_{v_i} = RandomWalk(G, v_i, t)
        SkipGram(Φ, W_{v_i}, w)
    end for
end for
```



https://paperswithcode.com/method/hierarchical-softmax

04 Method

Outer loop with γ

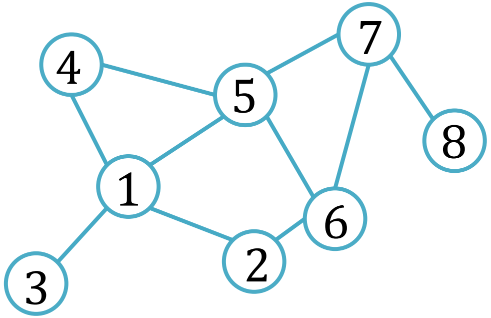
- Outer loop depends on walk per vertex γ
- Shuffle the node order of original graph

Algorithm 1 DEEPWALK(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t

Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
    $\mathcal{O} = \text{Shuffle}(V)$
    for each $v_i \in \mathcal{O}$ **do**
        $\mathcal{W}_{v_i} = RandomWalk(G, v_i, \text{t})$
        SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
    end for
end for



[1, 5, 7, ⋯]

04 Method

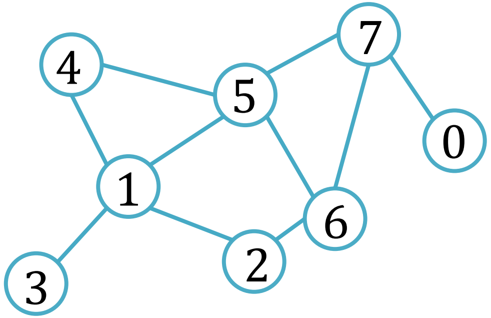
Inner loop with RandomWalk

- Inner loop depends on the number of node
- It makes a random walk with length t for v_i

Algorithm 1 DEEPWALK(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

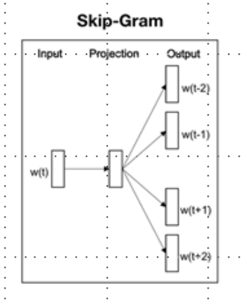


if walk length is 5 then..
[1, 2, 6, 5, 4]
[5, 4, 1, 2, 6]
[7, 6, 5, 1, 3]
...

04 Method

SkipGram

- analyze a single vertex to maximize the probability of the surrounding nodes
- In general, it has to modify all node vectors in the vocabulary but if the vocabulary size is huge, it will take a long time to run
- There fore, we can use methods like Hierarchical Softmax or negative sampling.



Algorithm 1 DEEPWALK(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
    SkipGram$(\Phi, \mathcal{W}_{v_i}, w)$
  end for
end for

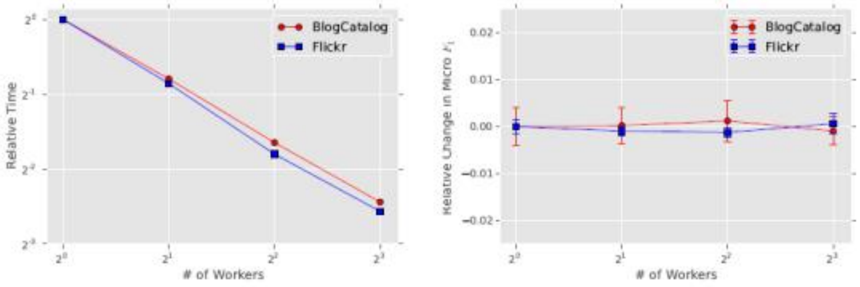
Algorithm 2 SkipGram$(\Phi, \mathcal{W}_{v_i}, w)$

for each $v_j \in \mathcal{W}_{v_i}$ **do**
  for each $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do**
    $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
    $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
  end for
end for

04 Method

Parallelizability

- Researcher found out that DeepWalk is parallelizable



(a) Running Time (b) Performance

Figure 4: Effects of parallelizing DEEPWALK

05 Experiments

Task “Node Classification”

- a task to predict unlabeled nodes.
- Datasets are **BlogCatalog, Flickr, YouTube**.
- For baseline models, they used
  - SpectralClustering
  - MaxModularity
  - EdgeCluster(K-means)
  - weighted vote Relational Neighbor(wvRN)
  - Majority

Name	BlogCatalog	Flickr	YouTube
$\lvert V \rvert$	10,312	80,513	1,138,499
$\lvert E \rvert$	333,983	5,899,882	2,990,443
$\lvert \mathcal{Y} \rvert$	39	195	47
Labels	Interests	Groups	Groups

Table 1: Graphs used in our experiments.

05 Experiments

BlogCatalog

- In this experiment, they varied the labeled nodes from 10~90%.
- DeepWalk showed better performance where there were few labeled nodes.

	% Labeled Nodes	10%	20%	30%	40%	50%	60%	70%	80%	90%
Micro-F1(%)	DeepWalk	36.00	38.20	39.60	40.30	41.00	41.30	41.50	41.50	42.00
	SpectralClustering	31.06	34.95	37.27	38.93	39.97	40.99	41.66	42.42	42.62
	EdgeCluster	27.94	30.76	31.85	32.99	34.12	35.00	34.63	35.99	36.29
	Modularity	27.35	30.74	31.77	32.97	34.09	36.13	36.08	37.23	38.18
	wvRN	19.51	24.34	25.62	28.82	30.37	31.81	32.19	33.33	34.28
	Majority	16.51	16.66	16.61	16.70	16.91	16.99	16.92	16.49	17.26
Macro-F1(%)	DeepWalk	21.30	23.80	25.30	26.30	27.30	27.60	27.90	28.20	28.90
	SpectralClustering	19.14	23.57	25.97	27.46	28.31	29.46	30.13	31.38	31.78
	EdgeCluster	16.16	19.16	20.48	22.00	23.00	23.64	23.82	24.61	24.92
	Modularity	17.36	20.00	20.80	21.85	22.65	23.41	23.89	24.20	24.97
	wvRN	6.25	10.13	11.64	14.24	15.86	17.18	17.98	18.86	19.57
	Majority	2.52	2.55	2.52	2.58	2.58	2.63	2.61	2.48	2.62

Table 2: Multi-label classification results in BlogCatalog

05 Experiments

Flickr

- In this experiment, they varied the labeled nodes from 1~10%.
- DeepWalk showed better performance on cases on Flickr.

	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DeepWalk	**32.4**	**34.6**	**35.9**	**36.7**	**37.2**	**37.7**	**38.1**	**38.3**	**38.5**	**38.7**
	SpectralClustering	27.43	30.11	31.63	32.69	33.31	33.95	34.46	34.81	35.14	35.41
	EdgeCluster	25.75	28.53	29.14	30.31	30.85	31.53	31.75	31.76	32.19	32.84
	Modularity	22.75	25.29	27.3	27.6	28.05	29.33	29.43	28.89	29.17	29.2
	wvRN	17.7	14.43	15.72	20.97	19.83	19.42	19.22	21.25	22.51	22.73
	Majority	16.34	16.31	16.34	16.46	16.65	16.44	16.38	16.62	16.67	16.71
Macro-F1(%)	DeepWalk	**14.0**	17.3	**19.6**	**21.1**	**22.1**	**22.9**	**23.6**	**24.1**	**24.6**	**25.0**
	SpectralClustering	13.84	**17.49**	19.44	20.75	21.60	22.36	23.01	23.36	23.82	24.05
	EdgeCluster	10.52	14.10	15.91	16.72	18.01	18.54	19.54	20.18	20.78	20.85
	Modularity	10.21	13.37	15.24	15.11	16.14	16.64	17.02	17.1	17.14	17.12
	wvRN	1.53	2.46	2.91	3.47	4.95	5.56	5.82	6.59	8.00	7.26
	Majority	0.45	0.44	0.45	0.46	0.47	0.44	0.45	0.47	0.47	0.47

Table 3: Multi-label classification results in Flickr

05 Experiments

YouTube

- YouTube dataset is close to a real world graph.
- It is so huge that SpectralClustering and Modularity couldn't train on it
- Result shows that DeepWalk can scale to large graphs and perform exceedingly well in sparsely labeled graph

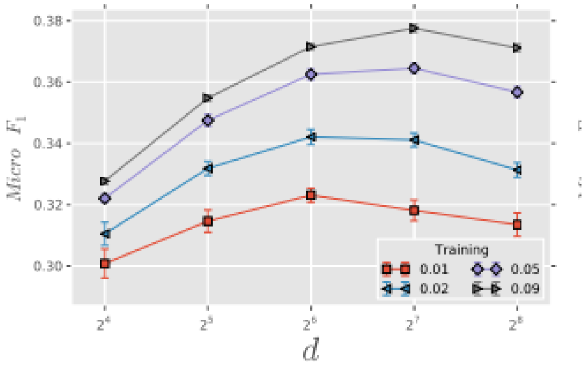
	% Labeled Nodes	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Micro-F1(%)	DeepWalk	**37.95**	**39.28**	**40.08**	**40.78**	**41.32**	**41.72**	**42.12**	**42.48**	**42.78**	**43.05**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
	EdgeCluster	23.90	31.68	35.53	36.76	37.81	38.63	38.94	39.46	39.92	40.07
	Modularity	—	—	—	—	—	—	—	—	—	—
	wvRN	26.79	29.18	33.1	32.88	35.76	37.38	38.21	37.75	38.68	39.42
	Majority	24.90	24.84	25.25	25.23	25.22	25.33	25.31	25.34	25.38	25.38
Macro-F1(%)	DeepWalk	**29.22**	**31.83**	**33.06**	**33.90**	**34.35**	**34.66**	**34.96**	**35.22**	**35.42**	**35.67**
	SpectralClustering	—	—	—	—	—	—	—	—	—	—
	EdgeCluster	19.48	25.01	28.15	29.17	29.82	30.65	30.75	31.23	31.45	31.54
	Modularity	—	—	—	—	—	—	—	—	—	—
	wvRN	13.15	15.78	19.66	20.9	23.31	25.43	27.08	26.48	28.33	28.89
	Majority	6.12	5.86	6.21	6.1	6.07	6.19	6.17	6.16	6.18	6.19

Table 4: Multi-label classification results in YouTube

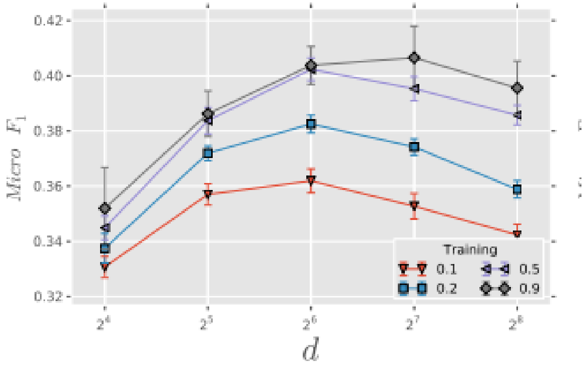
05 Experiments

Parameter sensitivity

- Test on parameters (number of walk per vertex γ, amount of training data T_R) and dimensions d on x-axis
- More data, the better



(a1) FLICKR, $\gamma = 30$

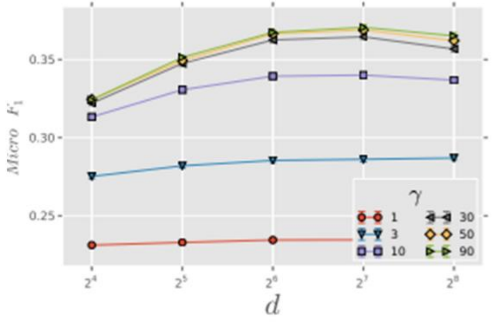


(a3) BLOGCATALOG, $\gamma = 30$

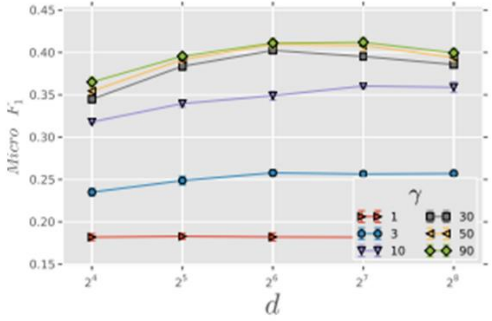
05 Experiments

Parameter sensitivity

- In Flickr and BlogCatalog dataset, when γ was 30, it shows most efficient performance
- When the value γ is fixed, the performance between different dimensions is relatively stable.



(a2) FLICKR, $T_R = 0.05$

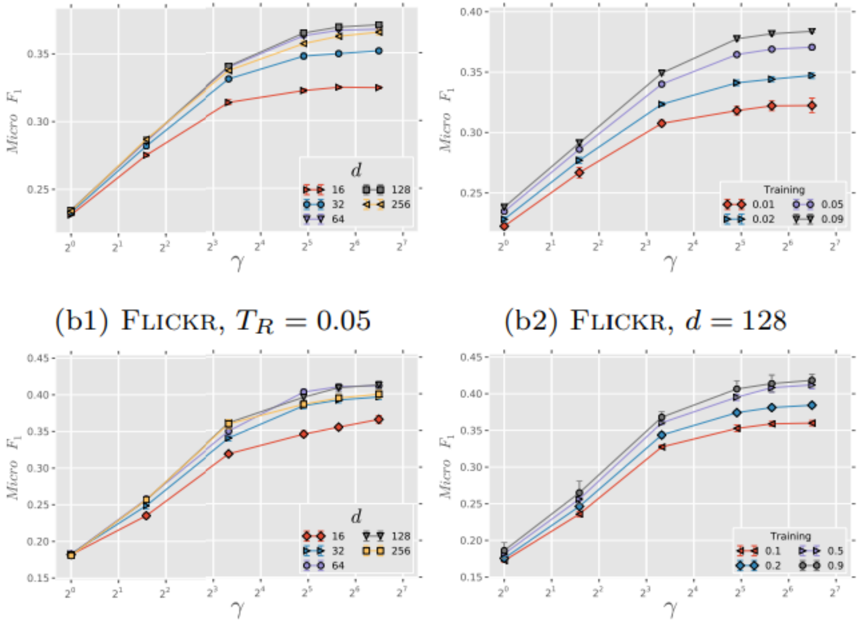


(a4) BLOGCATALOG, $T_R = 0.5$

05 Experiments

Sampling frequency

- Increasing γ has effects but quickly slows down when γ is over 10.



(b1) FLICKR, $T_R = 0.05$

(b2) FLICKR, $d = 128$

(b3) BLOGCATALOG, $T_R = 0.5$

(b4) BLOGCATALOG, $d = 128$

(a) Stability over number of walks, γ

06 Code

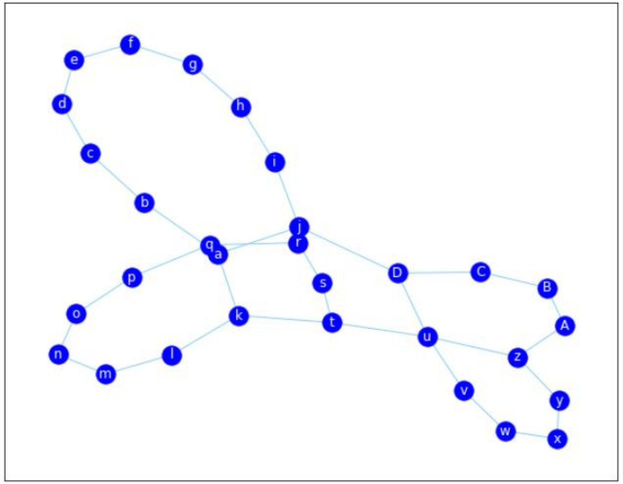
(1) init

Input : $Graph = (V, E, X, Y)$ and HyperParameters

- window size w
- embedding size d
- walks per vertex γ
- walk length t

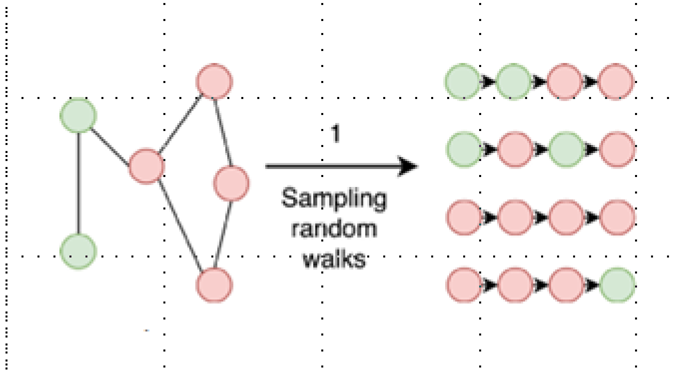
Output : $\Phi\colon\ v\ \in V \rightarrow R^{|V|*d}$

```python
class DeepWalk(torch.nn.Module):
    def __init__(self, G, w= 10, d = 2, gamma = 20, t = 6):
      super(DeepWalk, self).__init__()
      self.G = G
      self.w = w      #window size
      self.d = d      # embedding size
      self.gamma = gamma      # walks per vertex
      self.t = t    # walk length
      self.embedding = None
```



06 Code

(2) randomWalk




```python
def randomWalk(self, st_v):
  one_walk = []
  current_node = st_v
  one_walk.append(str(st_v))
  for j in range(self.t - 1): # self.t = walk_length
    neighbors = list(self.G.edges([st_v]))
    if (len(neighbors) > 0):
      random_edge = random.choice(neighbors)
      if (random_edge[0] == current_node):
        current_node = random_edge[1]
      else :
        current_node = random_edge[0]
    one_walk.append(str(current_node))
  return one_walk
```

06 Code

(3) train

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

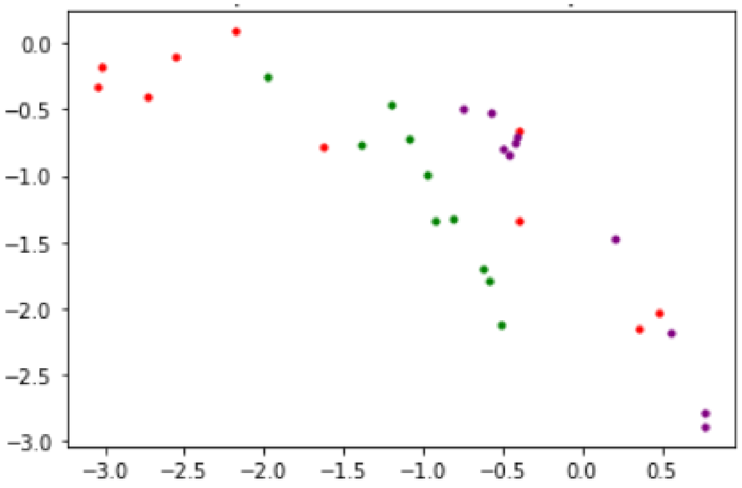
1: Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
2: Build a binary Tree T from V
3: **for** $i = 0$ to γ **do**
4: $\mathcal{O} = \text{Shuffle}(V)$
5: **for each** $v_i \in \mathcal{O}$ **do**
6: $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
7: SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
8: **end for**
9: **end for**

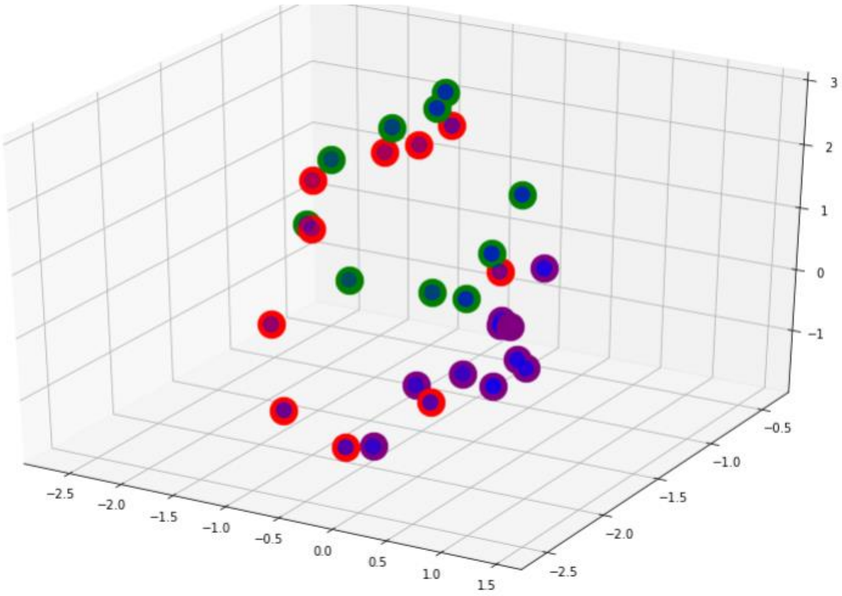
```
def train(self):
  walks = []
  nodes = list(self.G.nodes())
  print('starting', len(self.G.nodes))
  for i in range(self.gamma):
    random.shuffle(nodes)
    for node in nodes:
      walks.append(self.randomWalk(node))

  skipgrammodelresult = Word2Vec(walks, size=self.d, window=self.w, sg=1, hs=1)
  self.embedding = skipgrammodelresult
  return self.embedding
```

06 Code

(4) plot





06 Code

(5) dataset

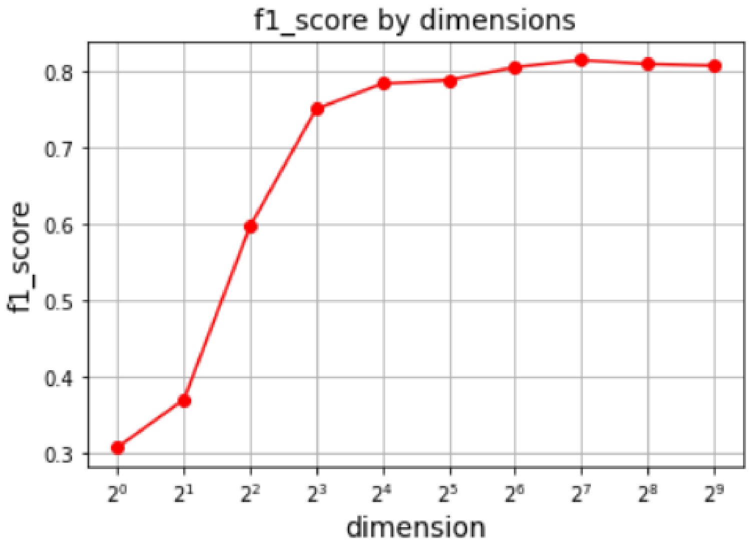
- Facebook Large Page-Page Network
- Each node represent official Facebook pages while the link are mutual likes between sites.
- The nodes are divided into 4 categories which are defined by Facebook: politicians, governmental organizations, television shows, and companies.

	Facebook
Nodes	22,470
Edges	171,002
Density	0.001
Transitvity	0.232

06 Code

(6) Experiment on dimensions

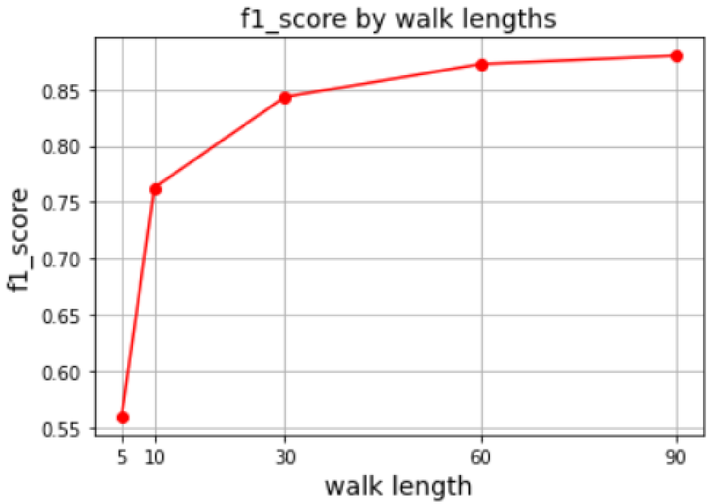
- window_size = 5
- gamma = 5
- walk_length = 10
- dimension =[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
- Result : In this dataset, 128 dimensions worked best.
- After 8-dimension, the f1 score did not change dramatically.



06 Code

(6) Experiment on walk lengths

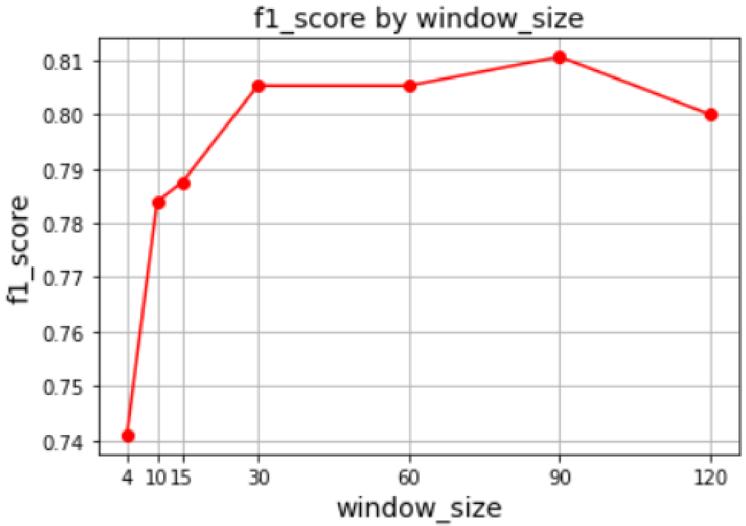
- window_size = 5
- dimension = 8
- gamma = 5
- walk length = [5, 10, 30, 60, 90]
- Result : In this dataset, if the walking length exceeds 30, it is effective.
- Walk length had the greatest effect on calculation time.



06 Code

(7) Experiment on window size

- dimension = 8
- gamma = 5
- walk length = 10
- window_size = [4, 10, 15, 30, 60, 90, 120]
- Result : In this dataset, the best value was 90.



07 Conclusions

Conclusion

- Deepwalk is appealing generalization of language modeling.
- It outperforms other methods on creating meaningful representations with large graphs.
- The approach is parallelizable, allowing workers to update different parts of the model concurrently.

07 Conclusions

Future works

Since it is unweighted random walk, the frequency or importance does not influence embeddings.

Node2Vec address some of the limitation of DeepWalk.

감사합니다

참고문헌

https://www.slideshare.net/bperz/14-kdddeep-walk-2
https://github.com/maziarraissi/Applied-Deep-Learning
https://dsgiitr.com/blogs/deepwalk/
https://github.com/AntonsRuberts/graph_ml/blob/main/Facebook%20-%20DeepWalk%20and%20Node2Vec.ipynb
https://paperswithcode.com/method/hierarchical-softmax
https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec
https://github.com/rlagywns0213/2021_Summer_Internship/blob/main/Graph%20Neural%20Network/RandomWalk/Deepwalk/train.py
https://github.com/Manu-Fraile/Network-Representation-Learning/blob/master/DeepWalk/main.py
https://plotly.com/python/t-sne-and-umap-projections/
https://velog.io/@minjung-s/GNN-Graph-Representation-Learning-Deep-Walk-node2vec