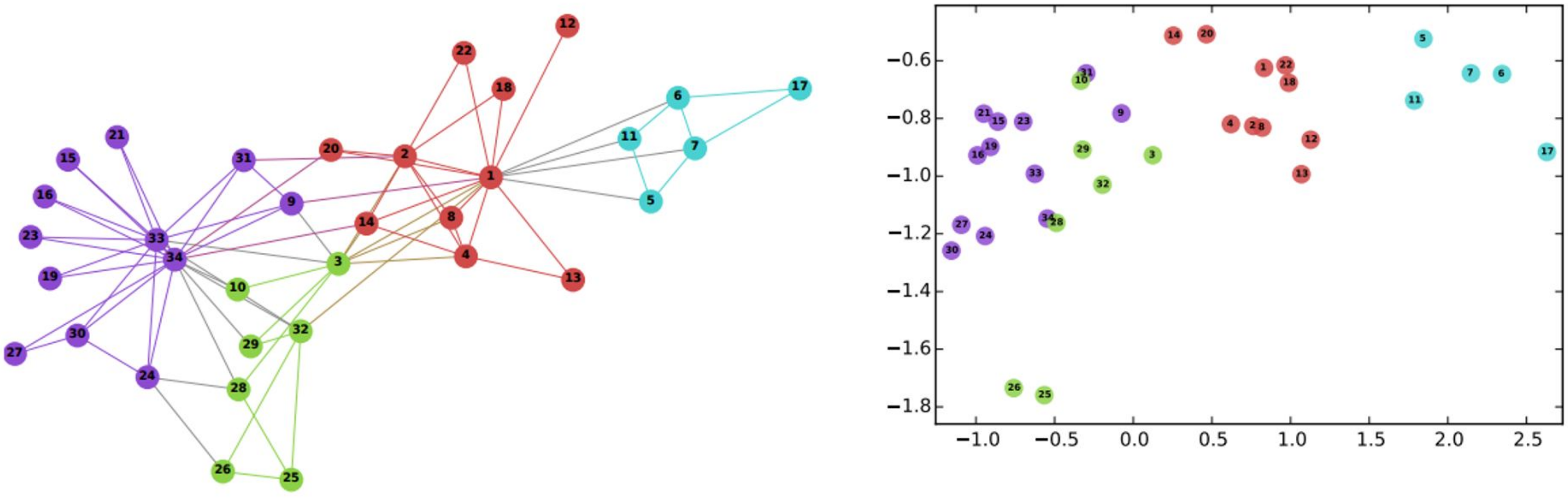
DeepWalk: Online Learning of Social Representations

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

DSAIL 2023 Winter Internship
JunYoung Kim

TL;DR

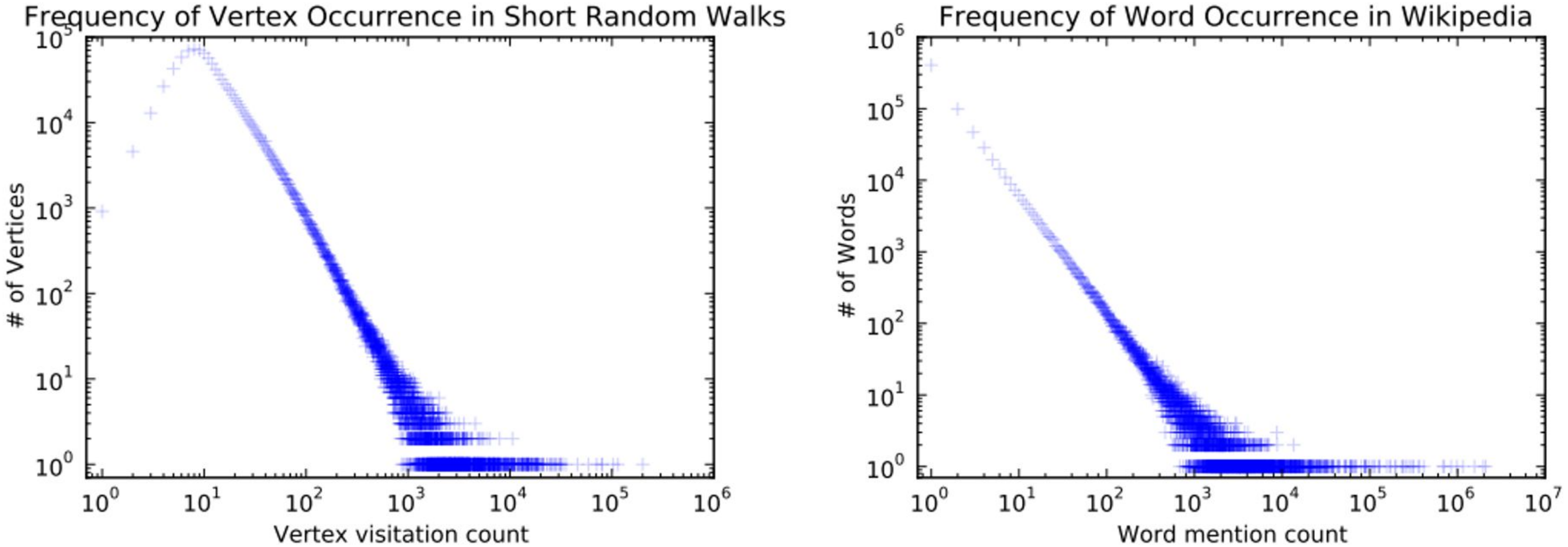
1. 이 논문에서는 기존에 자연어처리 과정에서 성공적으로 사용되었던 DeepWalk를 처음으로 그래프에 도입하였음
2. DeepWalk는 그래프의 레이블에 독립적인 표현을 학습하므로, 표현 품질이 레이블된 노드의 영향을 받지 않아서 멀티워커를 사용할 수 있음
3. DeepWalk는 특히 레이블이 희소한 상황에서 다른 방법들에 비해 더욱 우수한 성능을 보였음
4. DeepWalk는 온라인 알고리즘으로 병렬화가 용이하며, 따라서 모든 분류 방법과 결합하여 사용할 수 있음



(a) Input: Karate Graph

(b) Output: Representation

Deep walk is a graph embedding method



(a) YouTube Social Graph

(b) Wikipedia Article Text

Words frequency in a natural language corpus follows a power law.

Short random walks vs. sentences

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

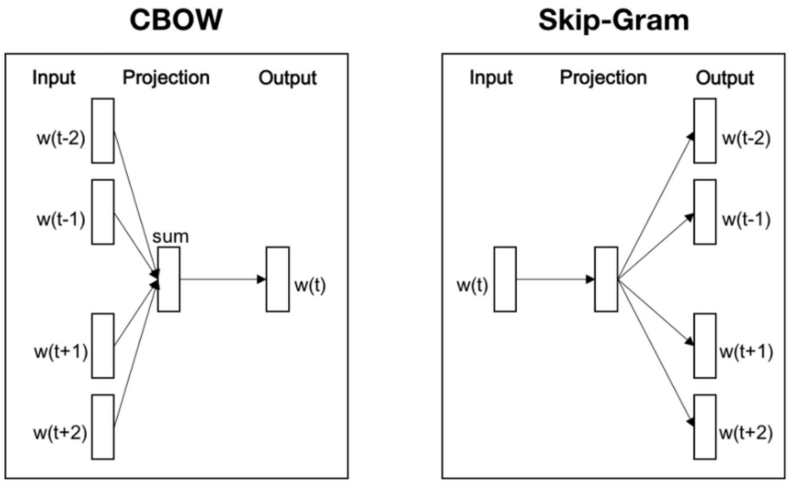
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

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

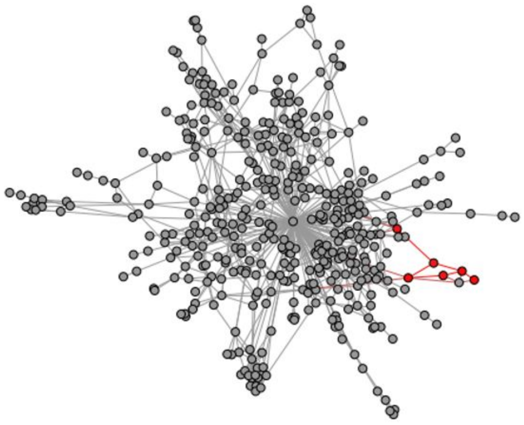
Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

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

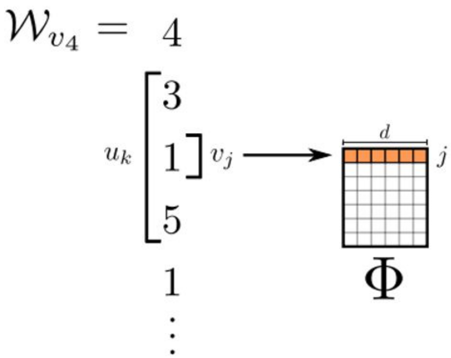
- Word2Vec : language model
  - CBOW method: predict the center word based on the source of the context words
  - Skip-gram: predicts the context words with the center word



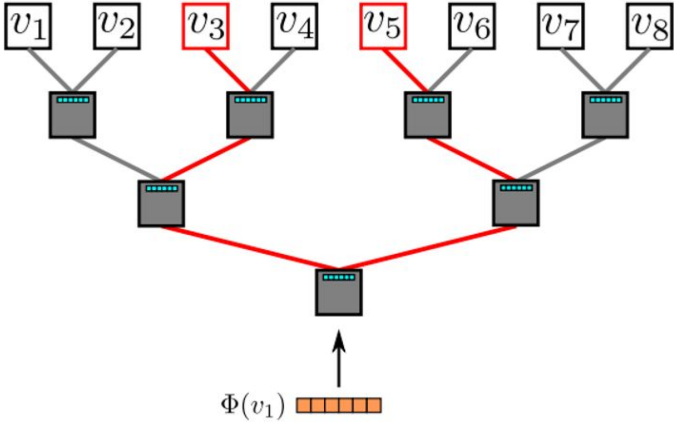
```python
def fit(self):
    walks = []
    nodes = list(self.G.nodes())
    for _ in tqdm(range(self.g)):
        random.shuffle(nodes)
        for node in nodes:
            walks.append(self.randomWalk(node))
    #SkipGram
    self.wvmodel = Word2Vec(walks, vector_size=self.d, window=self.w, sg=1, hs=1)
```



(a) Random walk generation.



(b) Representation mapping.



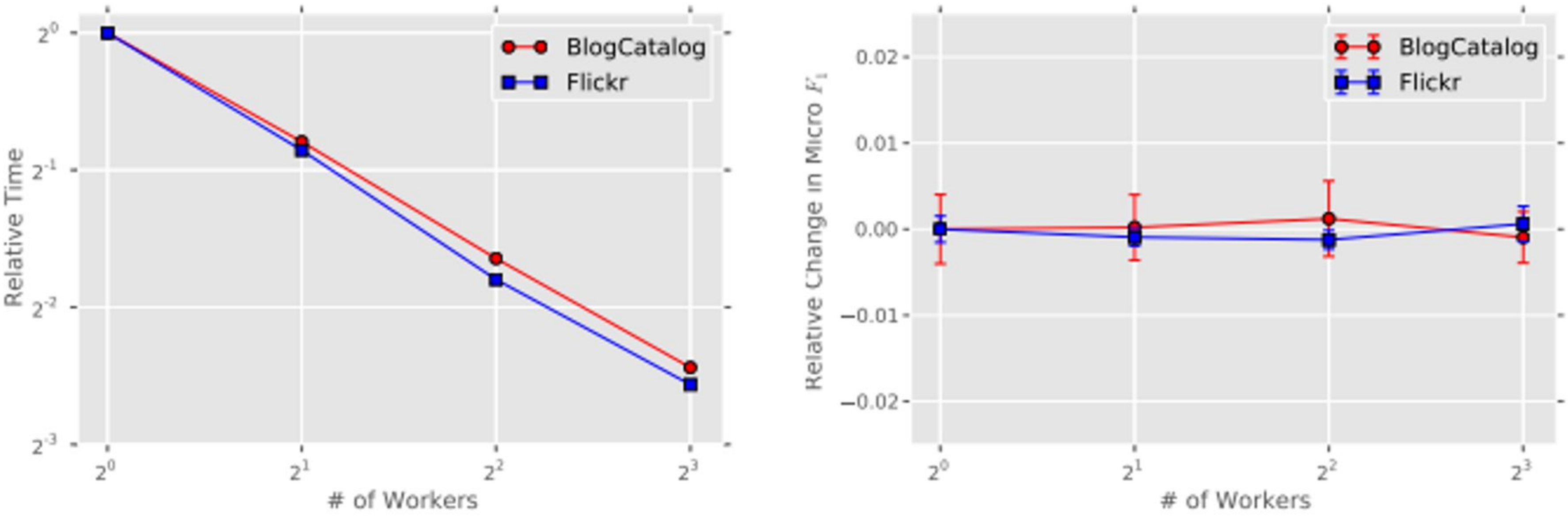
(c) Hierarchical Softmax.

Maximize: $\Pr(v_3|\Phi(v_1))$

$\Pr(v_5|\Phi(v_1))$

Overview of DeepWalk

```python
def randomWalk(self, start_node):
    walk = []
    current_node = start_node
    walk.append(str(start_node))
    for _ in range(self.t - 1):
        neighbors = list(self.G.edges([current_node]))
        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]
        walk.append(str(current_node))
    return walk
```



(a) Running Time

(b) Performance

Effects of parallelizing DeepWalk

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

Table 1: Graphs used in our experiments.

Results: BlogCatalog

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

Table 1: Graphs used in our experiments.

	% 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

Results: Flickr

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

Table 1: Graphs used in our experiments.

	% 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

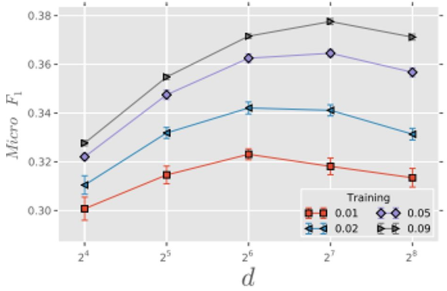
Results: YouTube

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

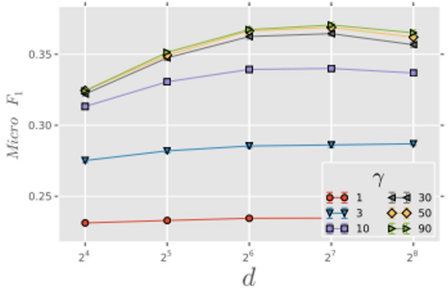
Table 1: Graphs used in our experiments.

	% 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

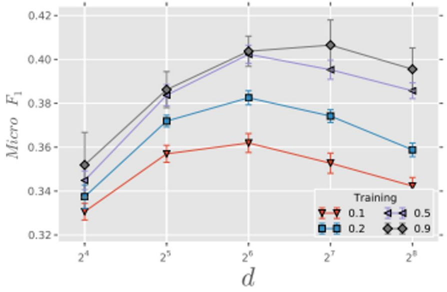
Table 4: Multi-label classification results in YouTube



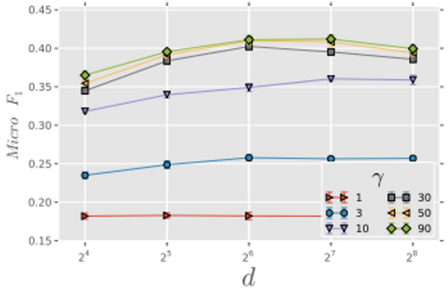
(a1) FLICKR, $\gamma = 30$



(a2) FLICKR, $T_R = 0.05$

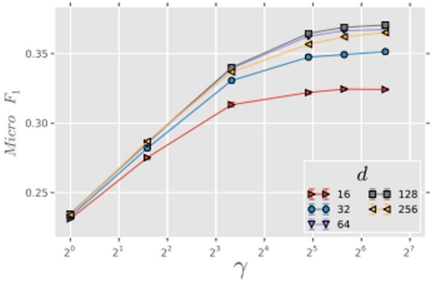


(a3) BLOGCATALOG, $\gamma = 30$

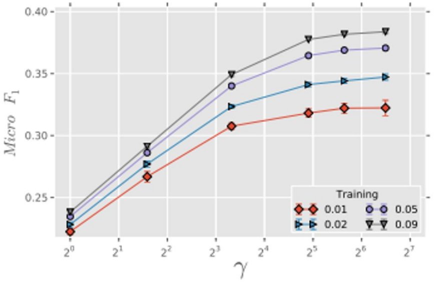


(a4) BLOGCATALOG, $T_R = 0.5$

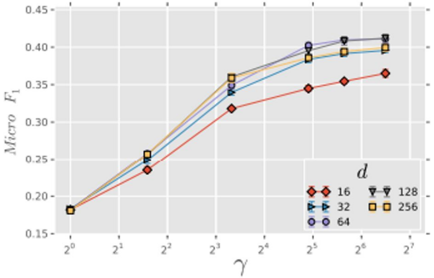
(a) Stability over dimensions, d



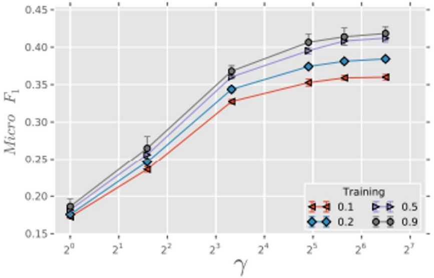
(b1) FLICKR, $T_R = 0.05$



(b2) FLICKR, $d = 128$



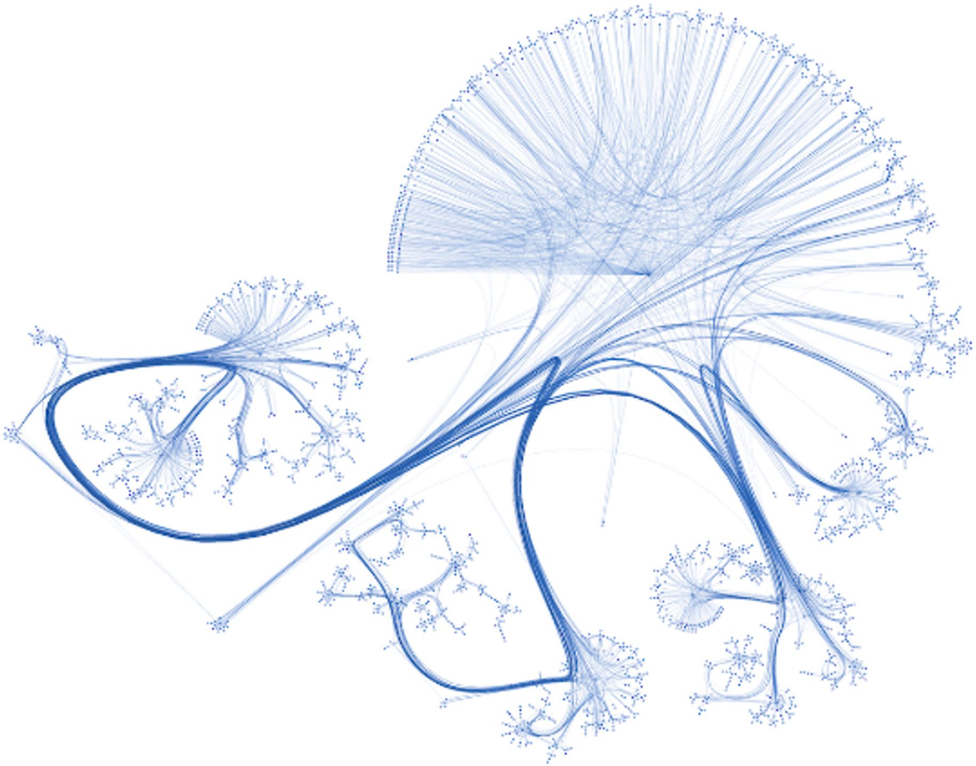
(b3) BLOGCATALOG, $T_R = 0.5$



(b4) BLOGCATALOG, $d = 128$

(a) Stability over number of walks, γ

Parameter Sensitivity Study

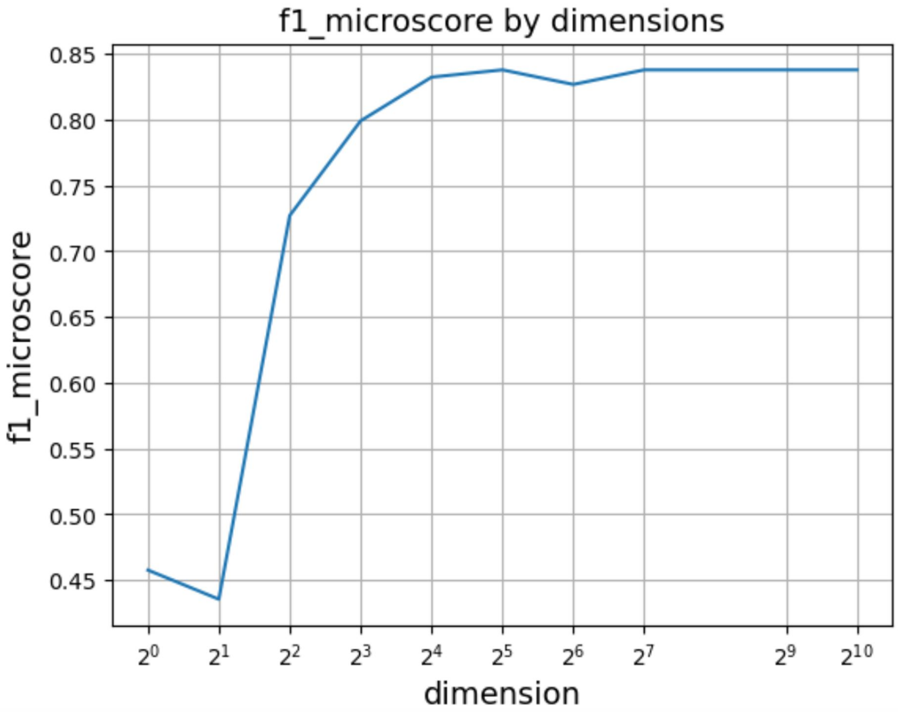


	target	source	label
0	35	1033	cites
1	35	103482	cites
2	35	103515	cites
3	35	1050679	cites
4	35	1103960	cites
...	...	...	...
5424	853116	19621	cites
5425	853116	853155	cites
5426	853118	1140289	cites
5427	853155	853118	cites
5428	954315	1155073	cites

```
31336               Neural_Networks
1061127               Rule_Learning
1106406      Reinforcement_Learning
13195        Reinforcement_Learning
37879         Probabilistic_Methods
                        ...
1128975          Genetic_Algorithms
1128977          Genetic_Algorithms
1128978          Genetic_Algorithms
117328                   Case_Based
24043               Neural_Networks
Name: subject, Length: 2708, dtype: object
```

Cora dataset with labels

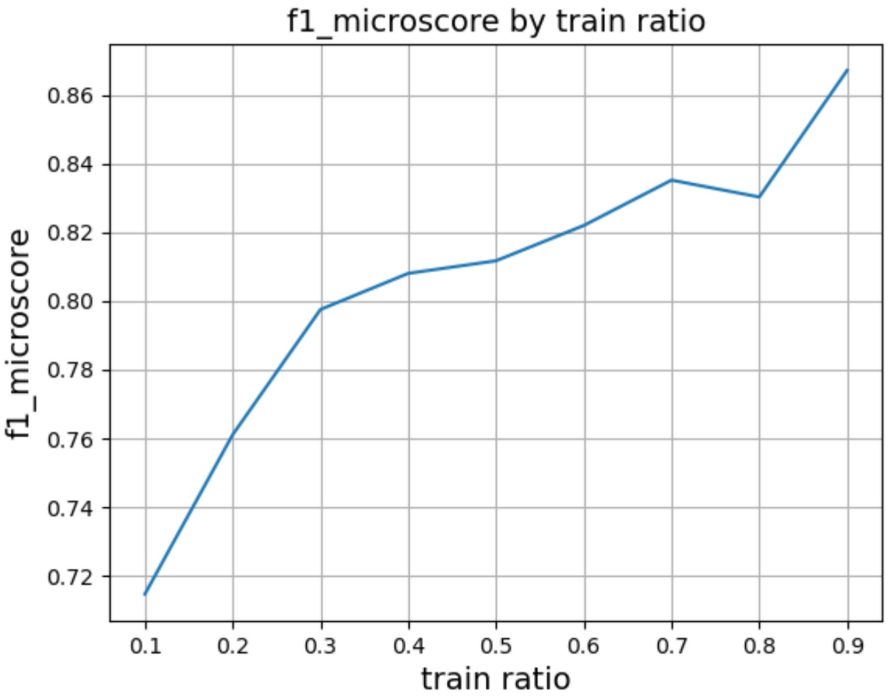
```python
def classify(word_vectors, tr=0.8):   # tr : Training ratio
    y = node_data.loc[[int(x) for x in targets],'subject']
    X = [word_vectors[str(idx)] for idx in y.index]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=(1-tr), random_state=5)
    rf = KNeighborsClassifier()
    rf.fit(X_train, y_train)
    y_predict = rf.predict(X_test)

    #evaluate
    f1_microscore = f1_score(y_test, y_predict, average='micro')
    return f1_microscore

def test(dim=64, tr=0.8, gamma=10):
    w = DeepWalk(Gnx, 5, dim, gamma, 10)
    w.fit()
    word_vectors = w.get_wvmodel().wv
    return classify(word_vectors, tr)
```



잘 안된 점

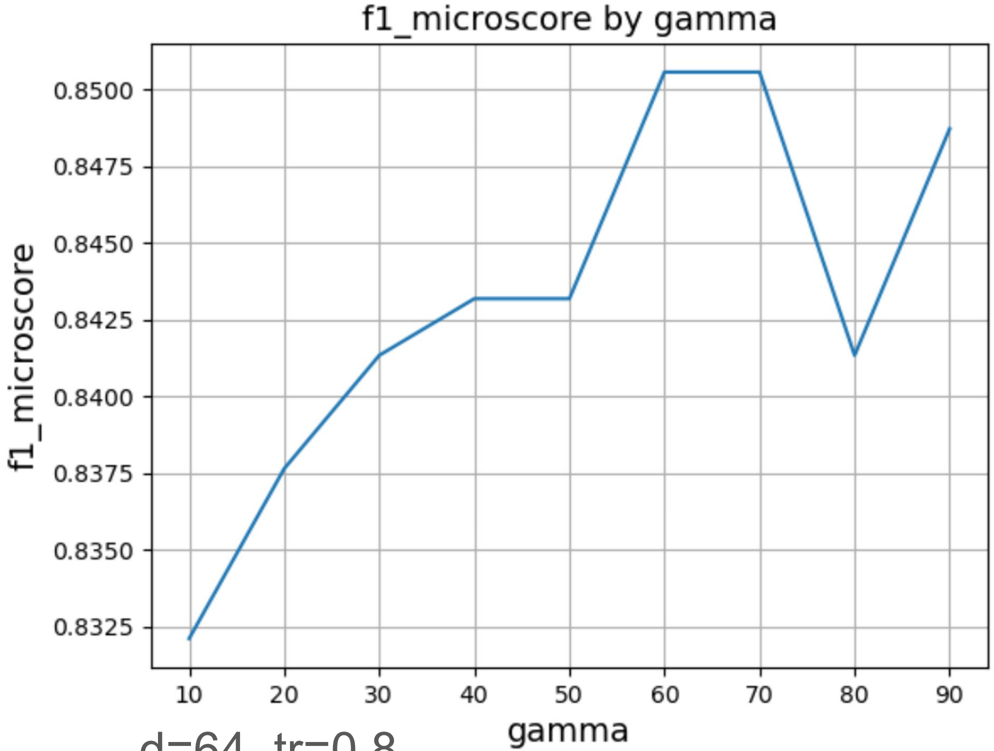
d 값이 어느정도(d=2^7) 이상부터는 오히려 감소하였는데 실제 구현해본 실험에서는 그렇지 않음

- tr=0.8, gamma=30
- d 을 변화하며 f1스코어를 측정함(d = 2^0, 2^1, … , 2^10)
- d 값이 증가할수록 micro-f1 score 증가



잘 안된 점
tr=0.8에서는 감소하였는데 오히려
tr=0.9에서는 큰 폭으로 증가함

- d=64, gamma =30
- train ratio 를 변화하며 f1스코어를 측정함(tr = 0.1, 0.2, … , 0.9)
- tr = 0.7까지는 train ratio 값이 증가할수록 micro-f1 score 증가



잘 안된 점
논문 실험에서는 gamma=30 이상부터는 큰 차이 없는 값을 보였는데, 실제로 실행한 실험에서는 gamma=60부터 설명되지 않는 패턴을 보임

- d=64, tr=0.8
- gamma 을 변화하며 f1스코어를 측정함(gamma = 10, 20, …, 90)
- gamma=60까지는 값이 증가할수록 micro-f1 score 증가

Conclusion

1. 이 논문에서는 기존에 자연어처리 과정에서 성공적으로 사용되었던 DeepWalk를 처음으로 그래프에 도입하였음
2. DeepWalk는 그래프의 레이블에 독립적인 표현을 학습하므로, 표현 품질이 레이블된 노드의 영향을 받지 않아서 멀티워커를 사용할 수 있음
3. DeepWalk는 특히 레이블이 희소한 상황에서 다른 방법들에 비해 더욱 우수한 성능을 보였음
4. DeepWalk는 온라인 알고리즘으로 병렬화가 용이하며, 따라서 모든 분류 방법과 결합하여 사용할 수 있음

감사합니다:)