Deepwalk:
Online Learning of Social Representations

Summer Internship

2022.06.28

김지홍

Table of Contents

1. Introduction

- Deepwalk is an embedding algorithm
- It is very similar to Word2Vec
- Naively encoding text tokens
  i.e. [0, 0, ⋯ , 1, ⋯, 0, 0] = ‘cat’
- Naively encoding edges
  i.e. [0, 0, ⋯ , 1, ⋯, 0, 0] = ‘connected to vertex k’

- Graphs, $G = (V, E)$ are typically represented as sparse matrices

1. Introduction : Word2vec & Deepwalk

- Constructing dense representations via context prediction
- **Language**: 'the quick brown fox jumps over the lazy dog'
  Context pairs → (quick, fox), (brown, fox), (jumps, fox)
- **Graph:** Walk from vertices to construct a sequence of vertices
- V1 → V2 → V5 → V3 → V1
  Context pairs: (V2, V5), (V3, V5)

1. Introduction

- Its performance is evaluated by multilabel network classification problems in large heterogeneous graphs
- Unlike the old techniques, the paper approaches by learning label independent representations of the graph
- This enables the model to be shared among tasks

1. Introduction

- The contributions are as follows:

1. Introduce deep learning as a tool to analyze graph and to build robust representations
2. Showed good classification performance in the presence of label sparsity
3. Demonstrate the scalability of Deepwalk by applying parallel implementation

2. Problem Definition

- Problem: Classifying members of a social network into one or more categories

→ also known as ‘relational classification’

- Instead of mixing the label space as part of the feature space, we propose an unsupervised method that learns features that capture the graph structure independent of the labels’ distribution

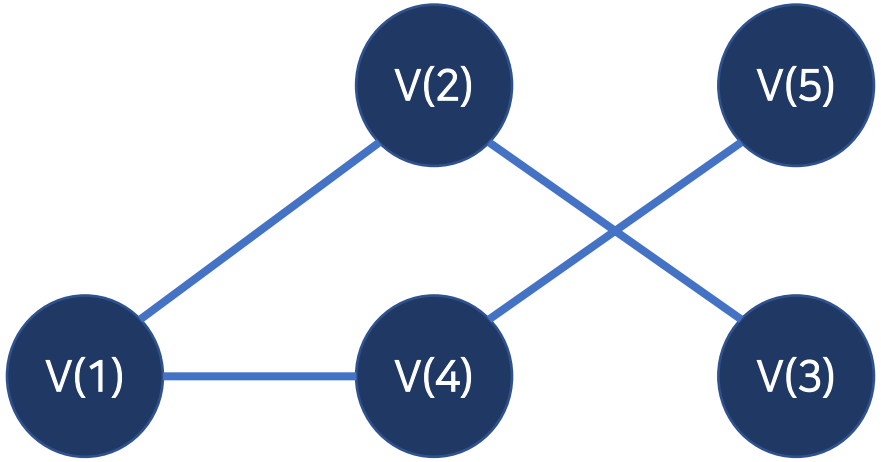
2. Problem Definition

- Input: $G_L = (V, E, X, Y)$
  - V: $members\ of\ the\ network$
  - E: $edges,\ E \subseteq (V \times V)$
  - $X \in \mathbb{R}^{|V| \times S}$ (S: $size\ of\ the\ feature\ space\ for\ each\ attribute\ vector$)
  - $Y \in \mathbb{R}^{|V| \times y}$ (y: $set\ of\ labels$)
- Goal: learn $X_E \in \mathbb{R}^{|V| \times d}$ (d: $number\ of\ latent\ dimensions$)

3. Background

- Random Walk
- Power Law
- Language Modeling

3. Background : Random Walk



Random Walk

Number of random walks: 3
Root Vertex: V(1)
Length: 4

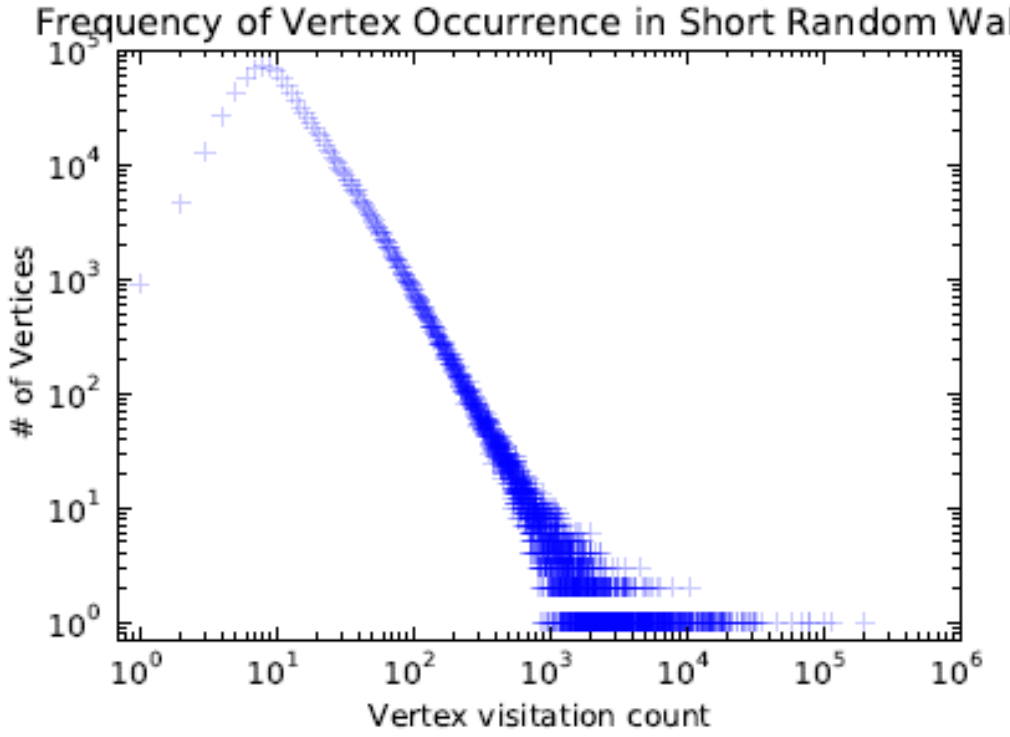
- [v(1), v(2), v(3), v(2)]
- [v(1), v(4), v(5), v(4)]
- [v(1), v(4), v(1), v(2)]

3. Background : Random Walk

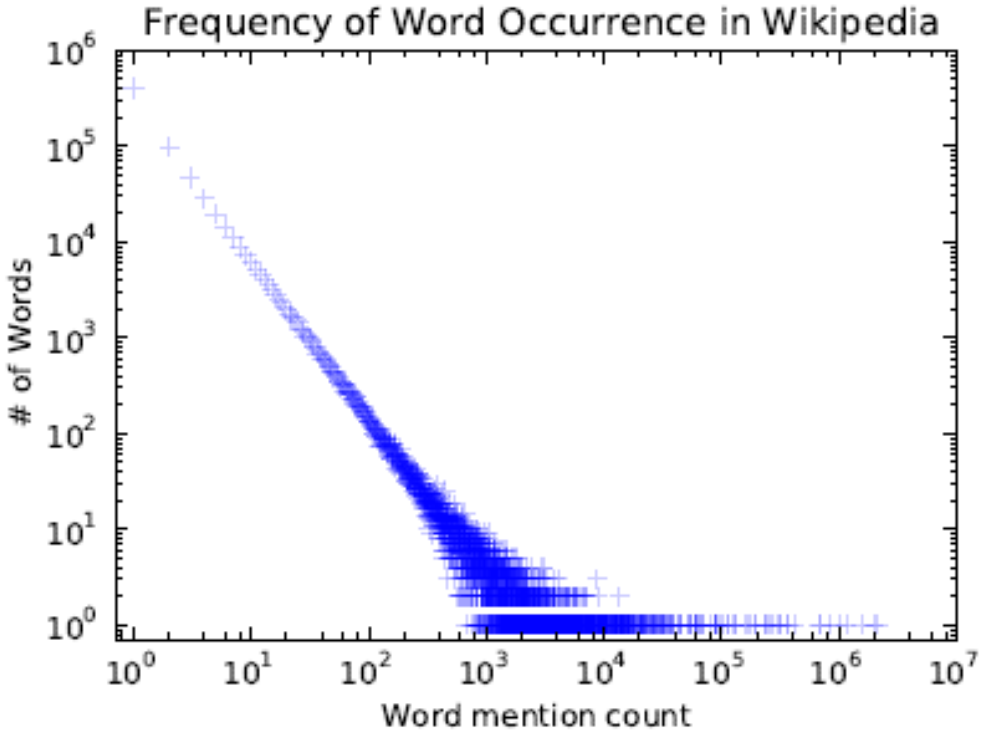
- w_{v_i}: random walk rooted at vertex v_i
- $w_{v_i}^{k+1}$: vertex chosen at random from the neighbors of vertex v_k

- It is the foundation of output sensitive algorithms
- Makes local exploration easy to parallelize
- Makes it possible to accommodate small changes in the graph structure

3. Background : Power Law



(a) YouTube Social Graph



(b) Wikipedia Article Text

3. Background : Language Modeling

- Original Goal(Language) :
  maximize $\Pr(w_n|w_0, w_1, \ldots, w_{n-1})$
  where $W_1^n = (w_0, w_1, w_2, \ldots, w_n)$ is given
- Original Goal(Graph) :
  maximize $\Pr(v_i|v_1, v_2, \ldots, v_{i-1})$
- Our goal is to learn a latent representation, not only a probability distribution of node co-occurrences, and so we introduce a mapping function $\Phi: v \in V \mapsto \mathbb{R}^{|V| \times d}$

3. Background : Language Modeling

- Changed Problem:

  estimate $\Pr(w_n|((\phi(V_1),(\phi(V_2),\ldots,(\phi(V_{i-1}))))$

- However, as the walk length grows, computing this objective function becomes unfeasible

3. Background : Language Modeling

- Recent relaxation in language modeling alters the prediction problem once again

1. instead of using the context to predict a missing word, it uses one word to predict the context
2. the context is composed of the words appearing to right side of the given word as well as the left side
3. it removes the ordering constraint on the problem

→ $\underset{\Phi}{\text{minimize}} \; -\log Pr(\{v_{i-w}, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{i+w}\} | \Phi(v_i))$

3. Random Walk + Language Modeling

- These methods generate representations of social networks that satisfy the following characteristics:
  - Adaptable
  - Community aware
  - Low dimensional
  - Continuous
- Its representations encode latent forms of community membership

4. Method

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. Random Walk Generator
2. Update Procedure

4. Method

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**

Window size (w)	Context to its left & right
Embedding size (d)	Size of latent vector
Walks per vertex (r)	Iteration
Walk length (t)	Length of a sentence

Ex) v1, ⋯, v8 → total 8 nodes
Random order
→ (v3, v1, v2, v5, v8, v4, v6, v7)

4. Method

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 $\Phi \in \mathbb{R}^{|V| \times d}$

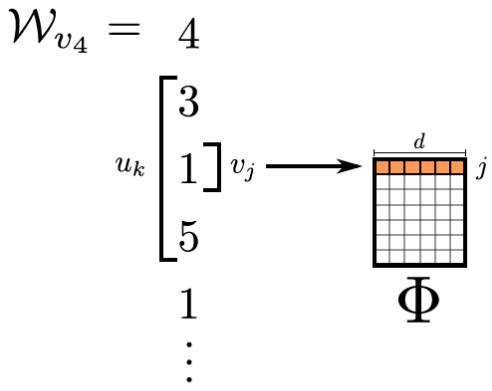
1: Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
2: Build a binary Tree T from V
3: **for** $i = 0$ to γ **do**
4: $\mathcal{O} = \text{Shuffle}(V)$
5: **for each** $v_i \in \mathcal{O}$ **do**
6: $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
7: SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
8: **end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

- Authors have defined the notion of running multiple walks from the same starting point
- Let's assume v3 is connected to v2, v7 ,v8
- We assume that the probability of v3 moving to v2, v7, v8 is uniform(=1/3)
- If t=5, $\mathcal{w}_{V_3}$=(v3, v7, v2, v4, v1)

4. Method : SkipGram

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

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



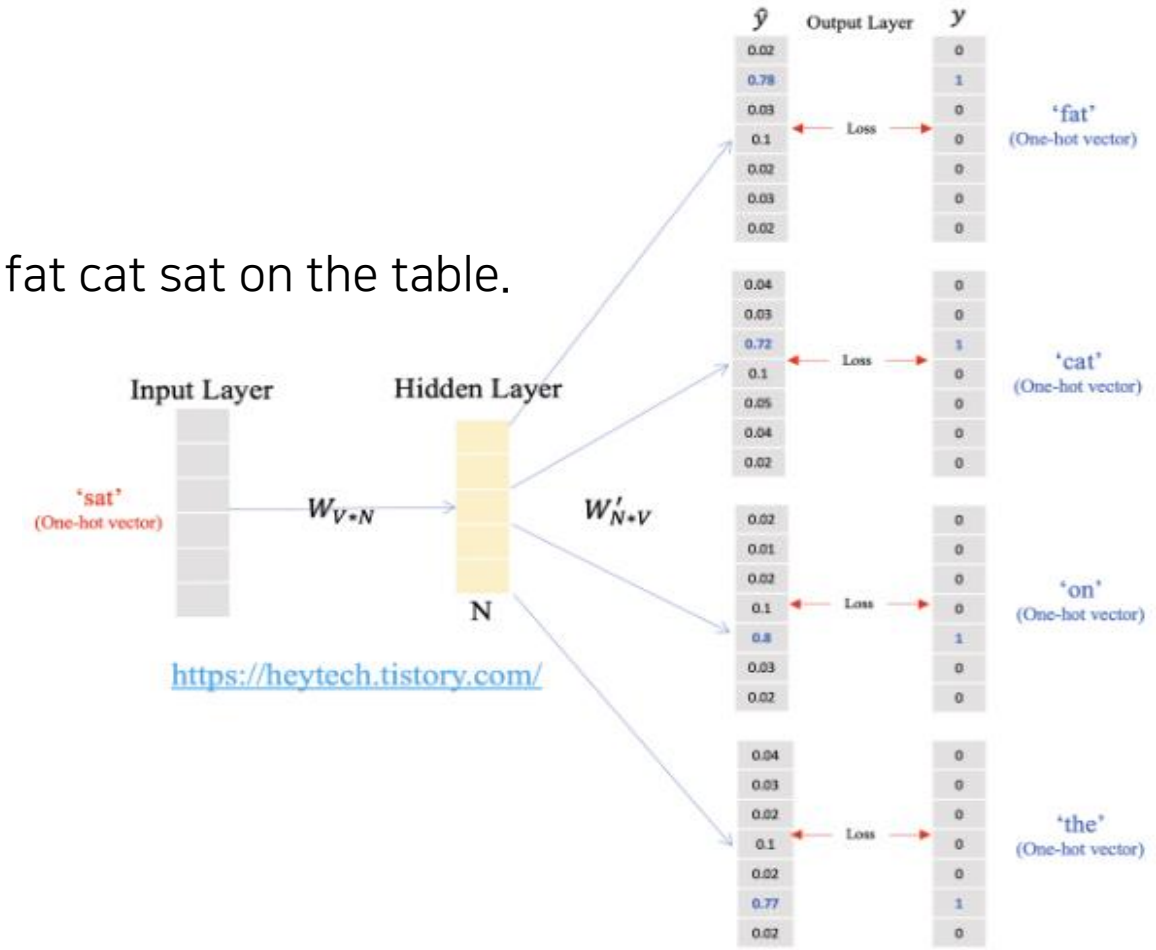
(b) Representation mapping.

- SkipGram is a language model that maximizes the cooccurrence probability among the words that appear within a window in a sentence

- We map each vertex v_j to its current representation vector $\Phi(v_j)$
- Given the representation of v_j, we would like to maximize the probability of its neighbors in the walk

4. Method : SkipGram

The fat cat sat on the table.

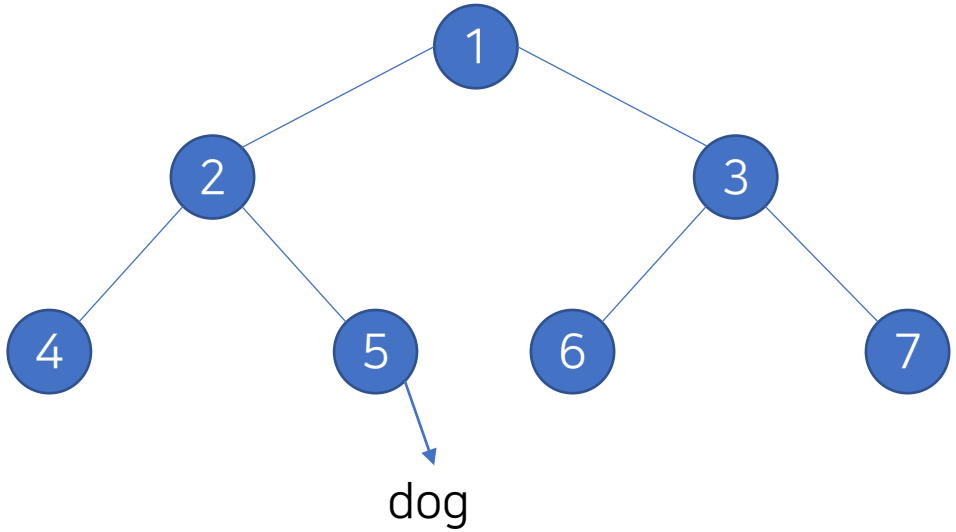


Softmax Function

: $\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

4. Method : Hierarchical Softmax

- Hierarchical Softmax reduces time complexity of Skipgram
- If we assign the vertices to the leaves of a binary tree, the prediction problem turns into maximizing the probability of a specific path in the tree



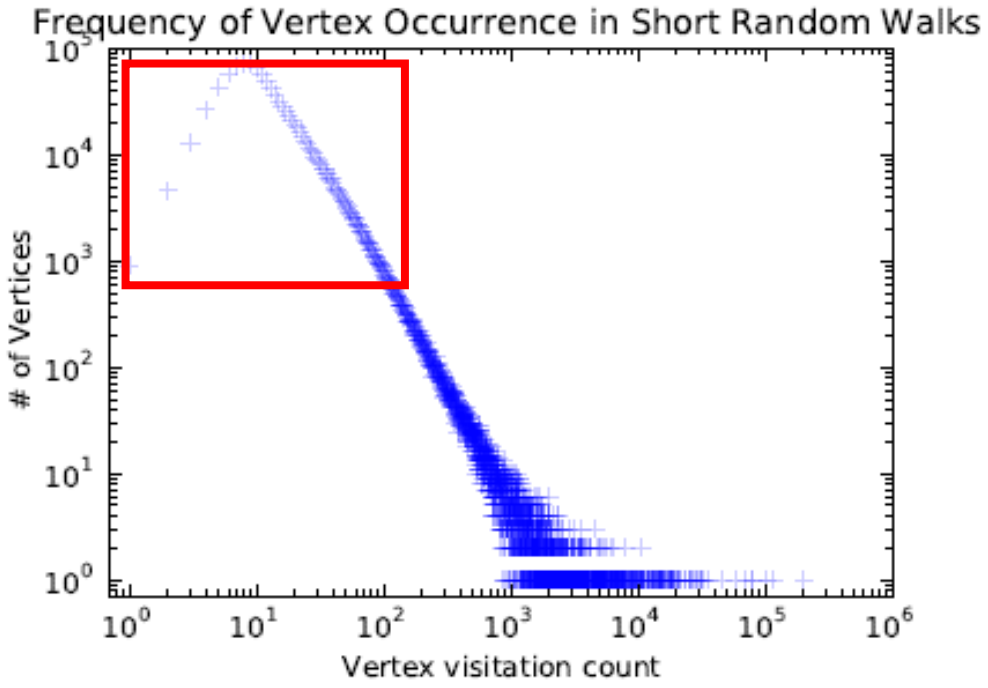
$$P(\text{"}dog\text{"}|context)$$
$$= P(left\ branch\ at\ node\ 1|context)$$
$$\times P(right\ branch\ at\ node\ 2|context)$$
$$\times P(right\ branch\ at\ node\ 5|context)$$

4. Method : Hierarchical Softmax

- It reduces the computational complexity from $\mathrm{O}(|V|)$ to $\mathrm{O}(\log|V|)$
- Huffman coding is used to reduce the access time of frequent elements in the tree

4. Method : Parallelizability

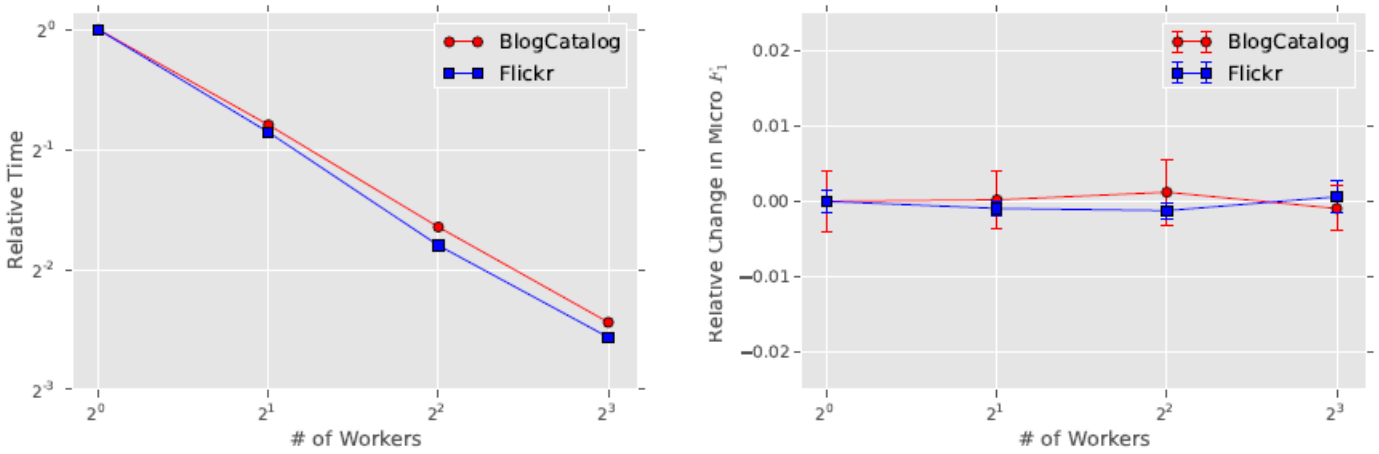
- As the number of vertices increases, the frequency of vertex occurrence becomes very infrequent
- Therefore, the updates that affect Φ will be sparse in nature



(a) YouTube Social Graph

4. Method : Parallelizability

- Updates are sparse and do not acquire a lock to access the model shared parameters
- This allows us to use ASGD in the multi worker case
- ASGD: multiple workers processing data in parallel and communicating with the parameter servers. Each worker processes a mini-batch of data independently of the other ones



(a) Running Time (b) Performance

Figure 4: Effects of parallelizing DeepWalk

5. Experiment : Multi-label Classification

- Randomly sample a portion of the labeled nodes and use them as training data
- Rest of the nodes are used as test

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

5. Experiment

Deepwalk

- Window Size(w)= 10
- Number of walks started per vertex(γ) = 80
- Latent Dimension = 128
- Portion of labeled nodes (T_R)
- BlogCatalog : 10% ~ 90%
- Flickr, YouTube : 1% ~ 10%
- One-vs-rest logistic regression for classification method

Baseline Methods

- SpectrualClustering
- Modularity
- EdgeCluster
- wvRN
- Majority

5. Experiment : Results in BlogCatalog

	% 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

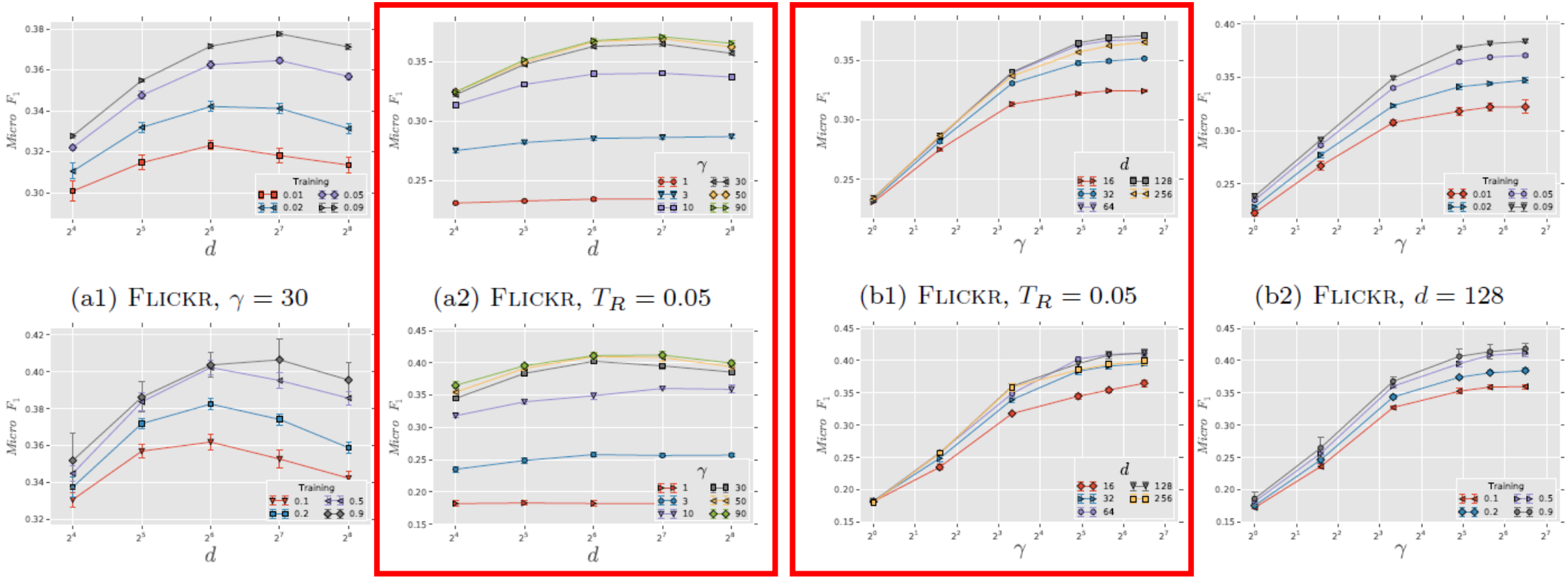
5. Experiment : Results in Flickr

	% 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

5. Experiment : Results in Youtube

	% 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

5. Experiment : Parameter Sensitivity

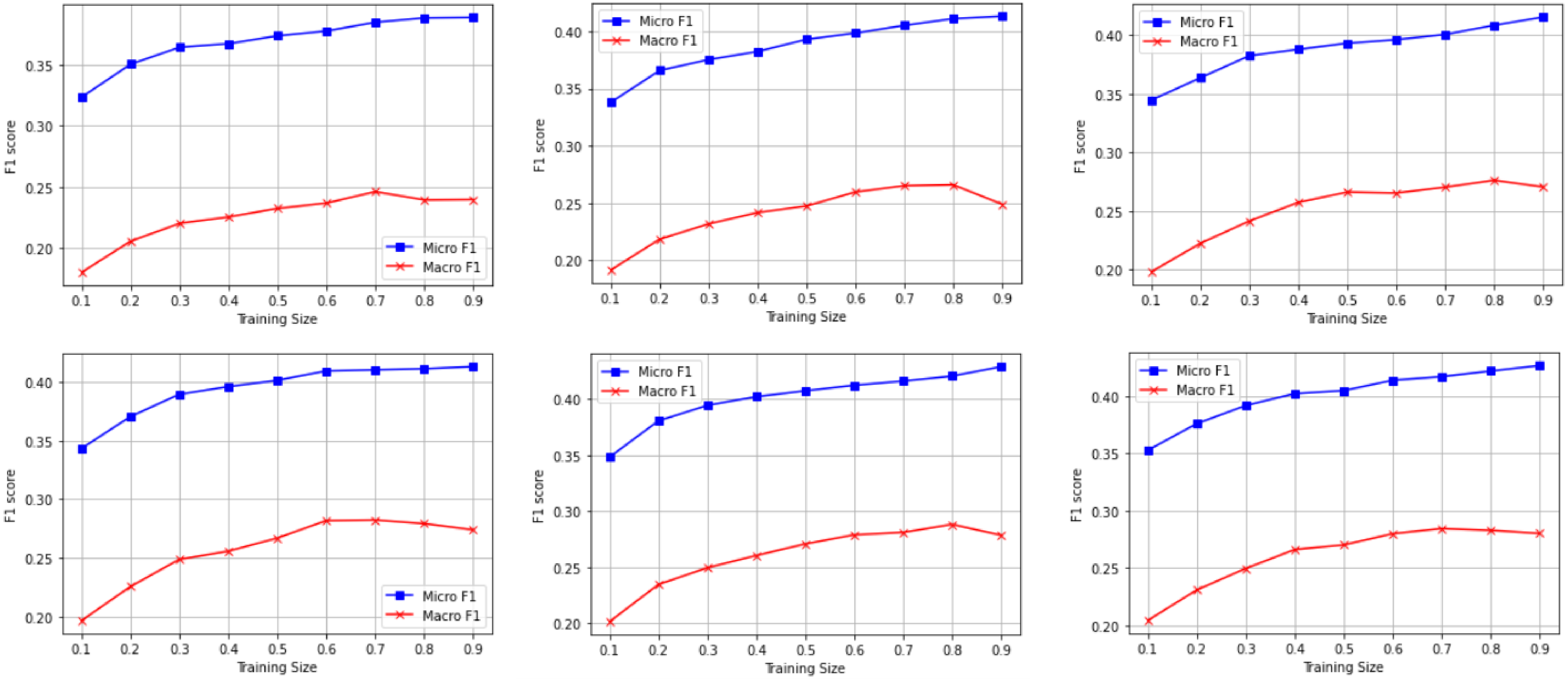


(a1) FLICKR, $\gamma = 30$ (a2) FLICKR, $T_R = 0.05$ (b1) FLICKR, $T_R = 0.05$ (b2) FLICKR, $d = 128$

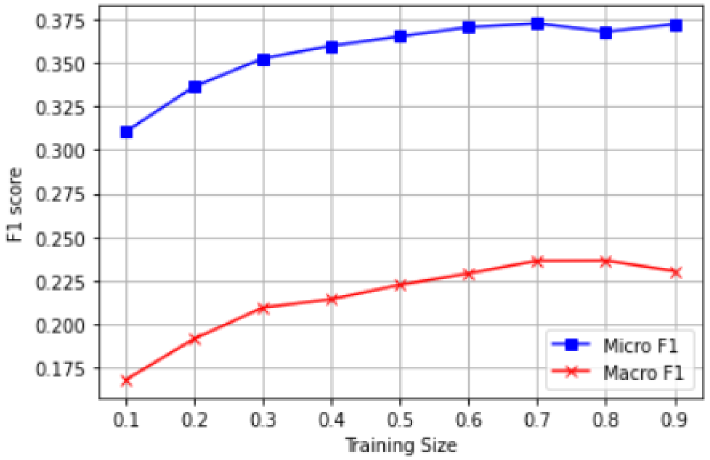
(a3) BLOGCATALOG, $\gamma = 30$ (a4) BLOGCATALOG, $T_R = 0.5$ (b3) BLOGCATALOG, $T_R = 0.5$ (b4) BLOGCATALOG, $d = 128$

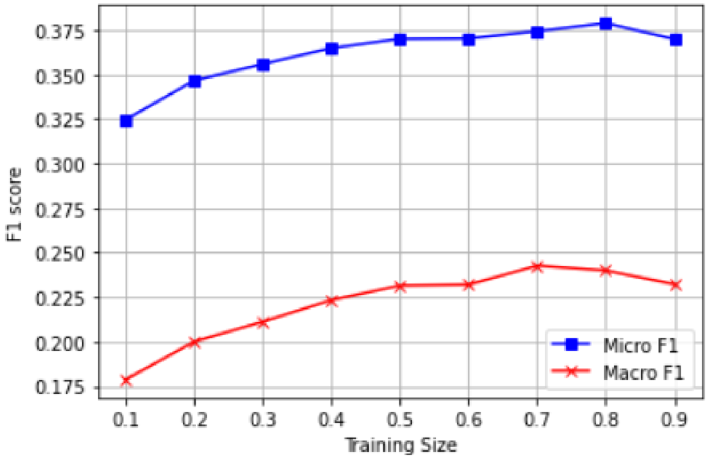
(a) Stability over dimensions, d (a) Stability over number of walks, γ

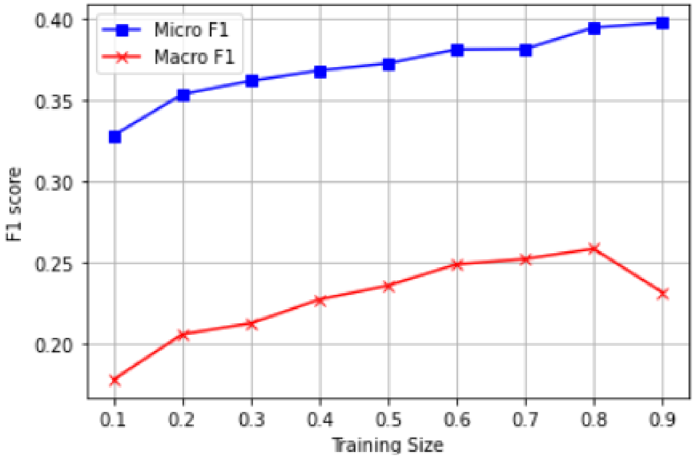
6. Code Implementation : walk length

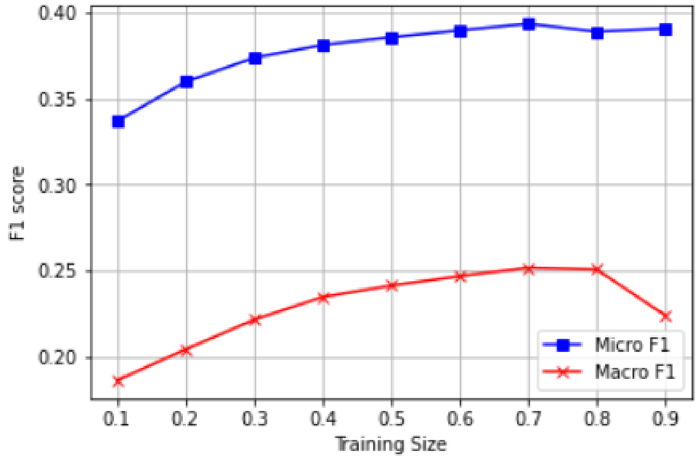


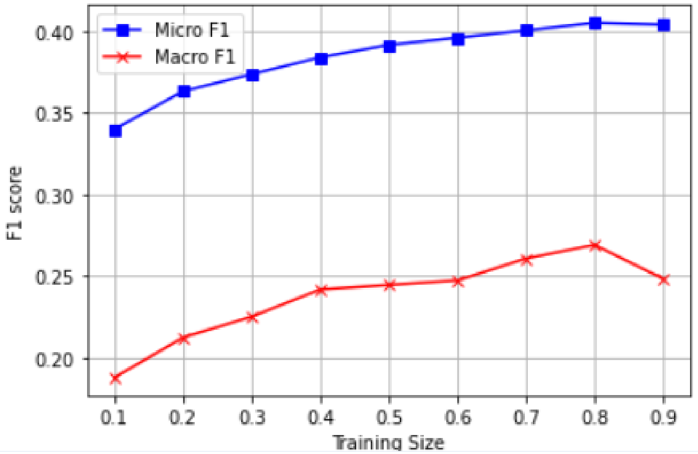
6. Code Implementation : window size

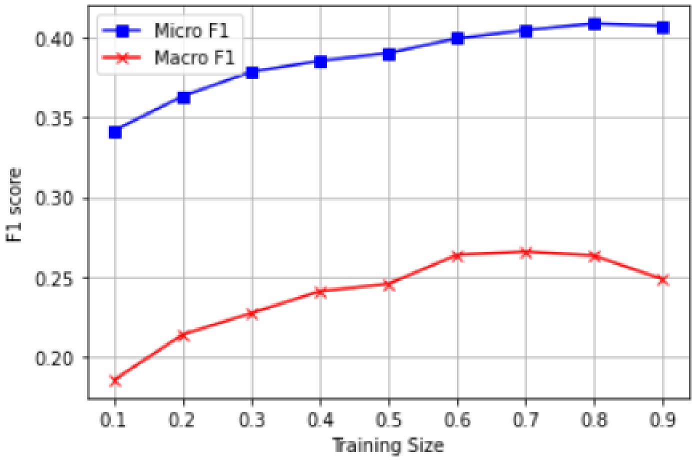




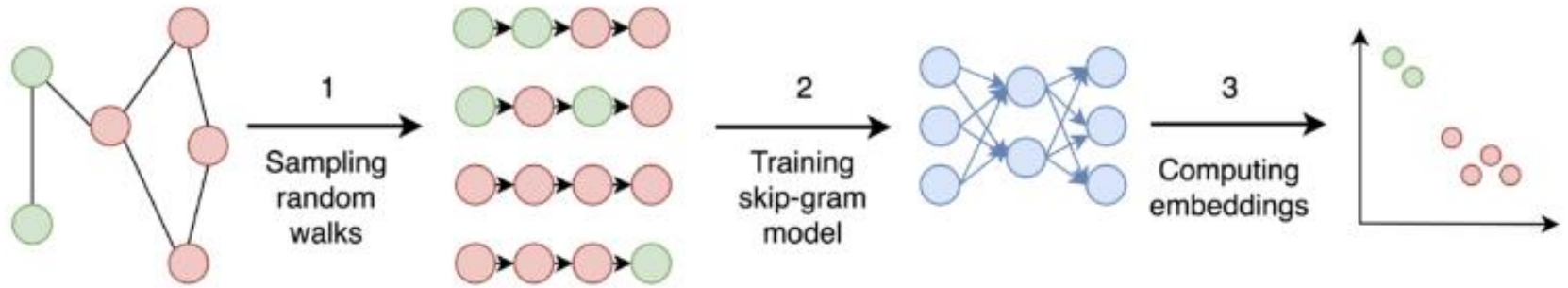








7. Conclusion



- Limitations: Deepwalk uses uniform random walks, which gives us no control over the explored neighborhoods
- Node2Vec overcomes this limitation by creating random walks with parameter p and q