DeepWalk and node2vec

Gabriele Santin[1]

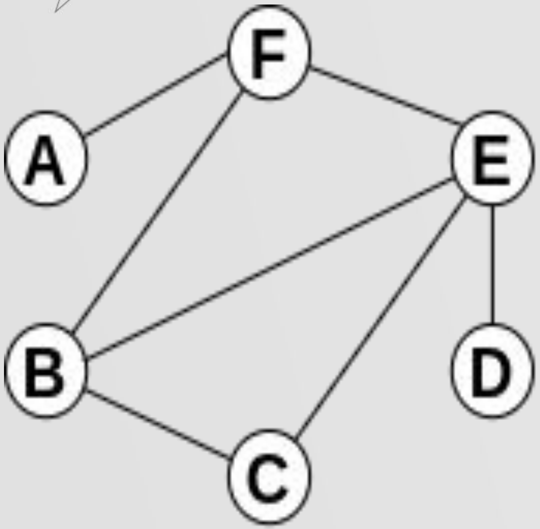
MobS[1] Lab, Fondazione Bruno Kessler,Trento, Italy

TABLE OF CONTENTS

01 Learning graph representations

Goal:

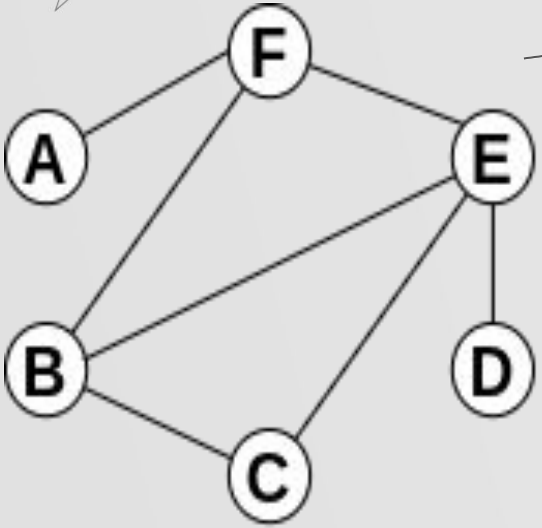
- Find a good representation of a graph G = (V, E)



01 Learning graph representations

Goal:

- Find a good representation of a graph G = (V, E)



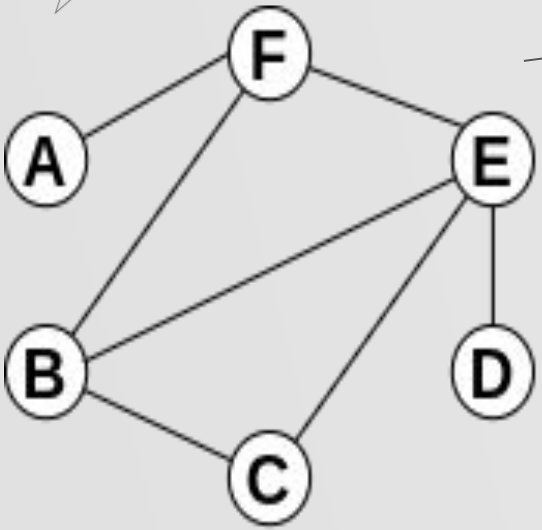
Node embedding:

$$v \mapsto [f_1(v), \ldots, f_d(v)]$$

01 Learning graph representations

Goal:

- Find a good representation of a graph G = (V, E)



Node embedding:

$$v \mapsto [f_1(v), \ldots, f_d(v)]$$

Edge embedding:

$$(u, v) \mapsto [g_1(u, v), \ldots, g_n(u, v)]$$

01 Learning graph representations

Usage of features: G=(V, E) with node features X

- Features f(V) that can be combined with the existing ones
- Any learning algorithm of [X, f(V)] -> structural features + original features
- (similarly for the edges)
- Task independent features (vs. GNN)

01 Learning graph representations

DeepWalk and node2vec:

- A good embedding preserves similarity between nodes (edges)
- The embedding is graph-dependent
- Based on a neighborhood preserving objective
- Based on a language model

DeepWalk: https://arxiv.org/pdf/1403.6652.pdf

node2vec: https://arxiv.org/pdf/1607.00653.pdf

01 Learning graph representations

DeepWalk and node2vec:

- A good embedding preserves similarity between nodes (edges)
- The embedding is graph-dependent
- Based on a neighborhood preserving objective
- Based on a language model

Here: Mix of the two, math mainly from node2vec

DeepWalk: https://arxiv.org/pdf/1403.6652.pdf

node2vec: https://arxiv.org/pdf/1607.00653.pdf

02 Ideas from language models: word2vec

Idea: Neighborhood preserving likelihood objective

- Vocabulary V (set of words v)
- sequence of words of fixed length (v_1, ..., v_n) from the corpus
- Learn function that maximizes P(v_n | (v_1, ..., v_{n-1})

02 Ideas from language models: word2vec

Idea: Neighborhood preserving likelihood objective

- Vocabulary V (set of words v)
- sequence of words of fixed length (v_1, ..., v_n) from the corpus
- Learn function that maximizes P(v_n | (v_1, ..., v_{n-1})

Use the context to predict a word

02 Ideas from language models: word2vec

Idea: Neighborhood preserving likelihood objective

- Vocabulary V (set of words v)
- sequence of words of fixed length (v_1, ..., v_n) from the corpus
- Learn function that maximizes P(v_n | (v_1, ..., v_{n-1})

Use the context to predict a word

More efficient: use a word to predict the context, forget order

- Window size w
- Predict P({w_{i-w}, \dots w_{-1}, w_{i+1}, \dots, w_{i+w}}| w_i)
- SkipGram

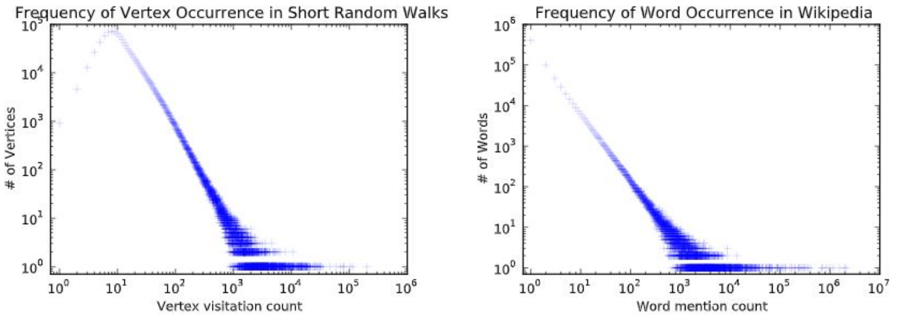
02 Ideas from language models: word2vec

Problem: graphs are not sequential

Idea: use random walks

- V = (v_1, ..., v_n) random walk
- the set of words (the context) is a neighborhood
- work on one vertex at a time

03 Making graphs sequential via random walks



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

“Similar power law distribution between occurrences of nodes in short random walks and frequency of words in texts if the degree distribution of a connected graph follows is scale-free

Figure from https://arxiv.org/pdf/1403.6652.pdf

03 Making graphs sequential via random walks

Advantages of random walks:

- high parallelization
- easy to recompute if short length and local modification

03 Making graphs sequential via random walks

Advantages of random walks:

- high parallelization
- easy to recompute if short length and local modification

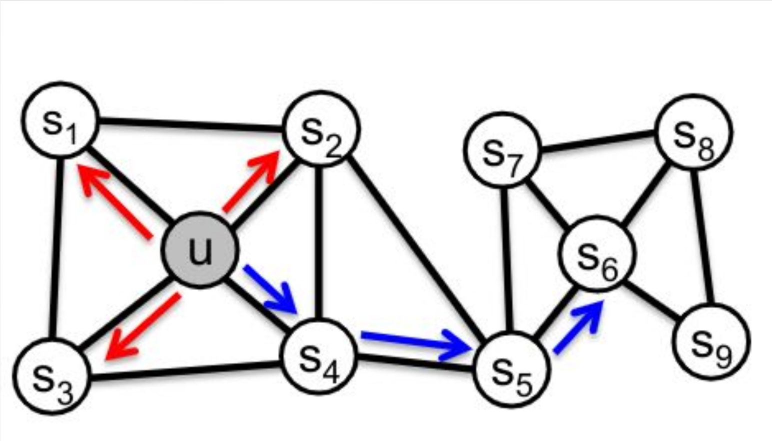
What similarities should be preserved?

Different notions of similarity:

homophily vs. **structural equivalence**

$$\{u, s_1, s_2, s_3, s_4\}$$
$$\{s_5, s_6, s_7, s_8, s_9\}$$

$$\{u, s_6\}$$



03 Making graphs sequential via random walks

Advantages of random walks:

- high parallelization
- easy to recompute if short length and local modification

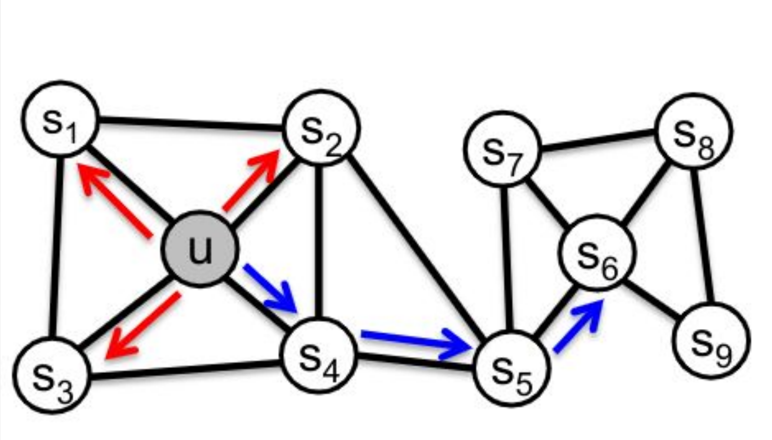
What similarities should be preserved?

Different notions of similarity:

homophily vs. **structural equivalence**

$$\{u, s_1, s_2, s_3, s_4\}$$
$$\{s_5, s_6, s_7, s_8, s_9\}$$

$$\{u, s_6\}$$



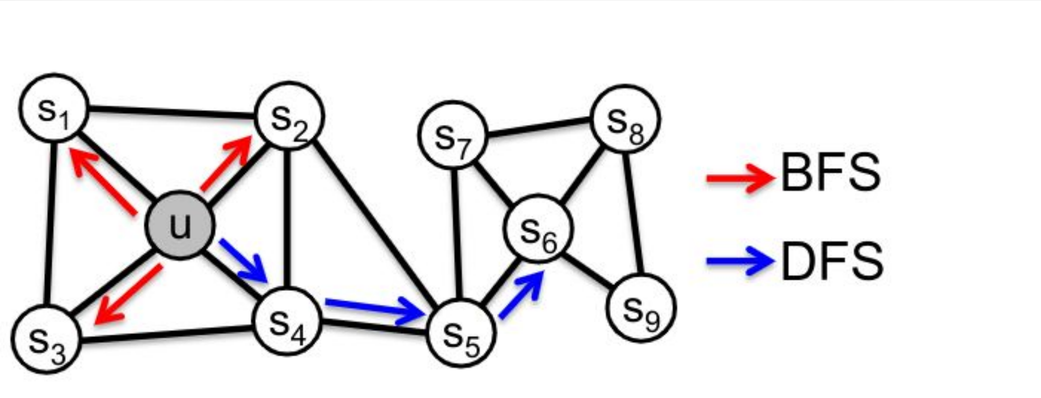
Sampling of random walks

- DeepWalk: use fixed sampling strategy
- Node2vec: use **parametric sampling strategy**

03 Making graphs sequential via random walks

Common sampling strategies:

- Breadth-first Sampling (**BFS**)
  - Good: for structural equivalence (it's a local property)
  - Bad: very small exploration

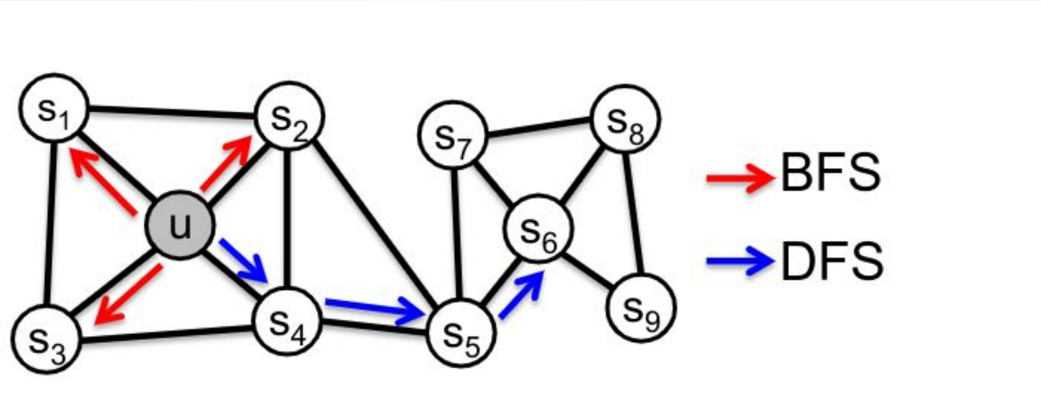


03 Making graphs sequential via random walks

Common sampling strategies:

- Breadth-first Sampling (**BFS**)
  - Good: for structural equivalence (it's a local property)
  - Bad: very small exploration

- Depth-first Sampling (**DFS**)
  - Good: for communities/homophily, large exploration
  - Bad: possible excessive distance

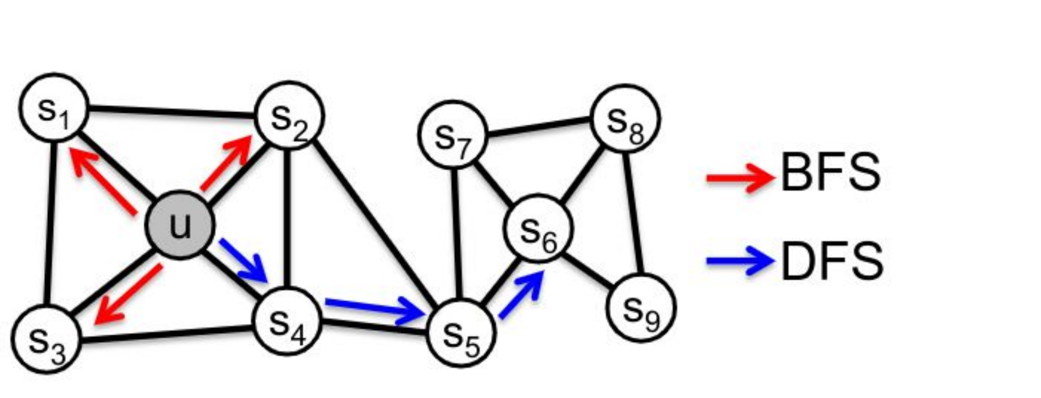


03 Making graphs sequential via random walks

Common sampling strategies: Bad: both require long memory!

- Breadth-first Sampling (**BFS**)
  - Good: for structural equivalence (it's a local property)
  - Bad: very small exploration

- Depth-first Sampling (**DFS**)
  - Good: for communities/homophily, large exploration
  - Bad: possible excessive distance



03 Making graphs sequential via random walks

Unbiased random walk:

1. Start from random node u
2. Move to v with probability

$$P(N_{i+1} = v | N_i = u) = \begin{cases} \frac{\pi_{vu}}{Z} \text{ if } (v, u) \in E \\ 0 \text{ otherwise} \end{cases}$$

03 Making graphs sequential via random walks

Unbiased random walk:

1. Start from random node u
2. Move to v with probability

$$P(N_{i+1} = v | N_i = u) = \begin{cases} \frac{\pi_{vu}}{Z} \text{ if } (v,u) \in E \\ 0 \text{ otherwise} \end{cases}$$

$$\pi_{uv}$$

Transition probability, e.g.

$$\pi_{uv} = w_{vu}$$

For a weighted graph

03 Making graphs sequential via random walks

Unbiased random walk:

1. Start from random node u
2. Move to v with probability

$$P(N_{i+1} = v | N_i = u) = \begin{cases} \frac{\pi_{vu}}{Z} \text{ if } (v,u) \in E \\ 0 \text{ otherwise} \end{cases}$$

$$\pi_{uv}$$

Transition probability, e.g.

$$\pi_{uv} = w_{vu}$$

For a weighted graph

$$Z$$

Normalization factor

03 Making graphs sequential via random walks

Unbiased random walk:

1. Start from random node u
2. Move to v with probability

$$P(N_{i+1} = v | N_i = u) = \begin{cases} \frac{\pi_{vu}}{Z} \text{ if } (v, u) \in E \\ 0 \text{ otherwise} \end{cases}$$

$$\pi_{uv}$$

Transition probability, e.g.

$$\pi_{uv} = w_{vu}$$

For a weighted graph

$$Z$$

Normalization factor

Node2vec idea: Biased random walks

- Parametrize BFS vs DFS
- Parametrize “stay local” vs “explore”
- Second order random walk

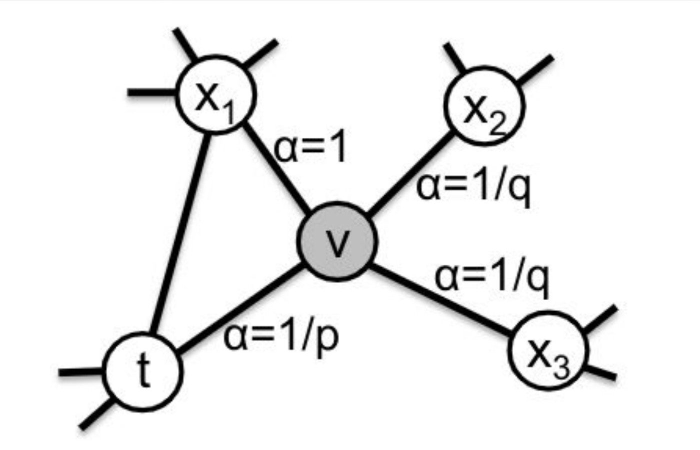
03 Making graphs sequential via random walks

Biased random walk:

Assume the walk is in **t,** moves to **v**, and decides the **next move**:

Define the **search bias**

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



03 Making graphs sequential via random walks

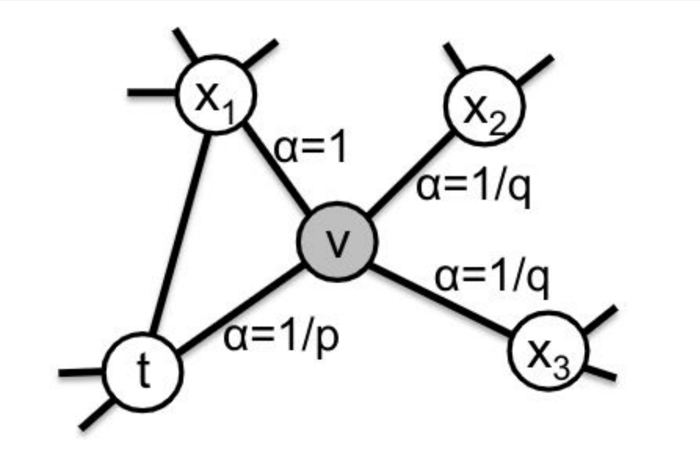
Biased random walk:

Assume the walk is in **t,** moves to **v**, and decides the **next move**:

Define the **search bias**

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

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



return parameter p:

- large -> exploration
- small -> backtrack, local

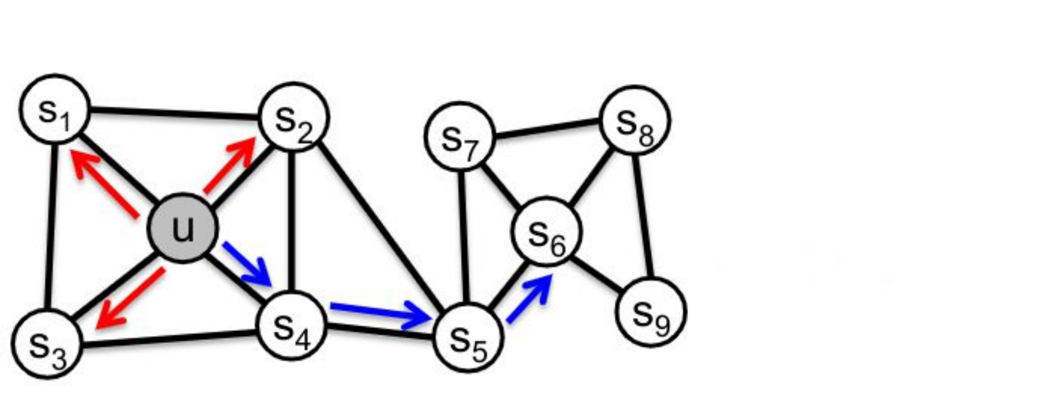
in-out parameter q:

- large -> stay close to t
- small -> exploration

03 Making graphs sequential via random walks

Details:

- **2nd order Markovian:** small memory requirements
- Sample length l, extract l-k walks
- Deep walk: p=1, q=1



Sample **l=6**, **k=3**: {u, s4, s5, s6, s8, s9}

1. **u:** s4,s5,s6
2. **s4:** s5,s6,s8
3. **s5:** s6,s8,s9

04 The learning problem

Given an **embedding function** $f : V \mapsto \mathbb{R}^d$ represented by a $|V| \times d$ matrix

Define the **similarity** between v and u:

$$P_f(v|u) := \frac{\exp(f(v)^T f(u))}{\sum_{w \in V} \exp(f(w)^T f(u))}$$

$\in [0, 1]$ probability

Symmetric in u, v

Dependent on f

04 The learning problem

Given a **neighborhood** $N_s(v)$ according to the **sampling strategy S**

Define of a probability of the neighborhood of u given u:

$$P_f(N_s(u)|u) := \prod_{v \in N_s(u)} P_f(v|u)$$

Similarity between v and u

04 The learning problem

Given a **neighborhood** $N_s(v)$ according to the **sampling strategy S**

Define of a global **neighborhood likelihood** given f:

$$\sum_{u \in V} P_f(N_s(u)|u)$$

04 The learning problem

Given a **neighborhood** $N_s(v)$ according to the **sampling strategy S**

Define of a global **neighborhood likelihood** given f:

$$\sum_{u \in V} \log P_f(N_s(u)|u)$$

04 The learning problem

$$\max_f \sum_{u \in V} \log P_f(N_s(u)|u)$$

04 The learning problem

$$\sum_{u \in V} \log P_f(N_s(u)|u) = \sum_{u \in V} \log \left(\prod_{v \in N_s(u)} P_f(v|u) \right) =$$

$$= \sum_{u \in V} \log \left(\prod_{v \in N_s(u)} \frac{\exp(f(v)^T f(u))}{\sum_{w \in V} \exp(f(w)^T f(u))} \right)$$

04 The learning problem

$$\sum_{u \in V} \log P_f(N_s(u)|u) = \sum_{u \in V} \log \left(\prod_{v \in N_s(u)} P_f(v|u) \right) =$$

$$= \sum_{u \in V} \log \left(\prod_{v \in N_s(u)} \frac{\exp(f(v)^T f(u))}{\sum_{w \in V} \exp(f(w)^T f(u))} \right)$$

$$\sum_{w \in V} \exp(f(w)^T f(u))$$

04 The learning problem

$$\sum_{u \in V} \log P_f(N_s(u)|u) = \sum_{u \in V} \log \left(\prod_{v \in N_s(u)} P_f(v|u) \right) =$$

$$= \sum_{u \in V} \log \left(\prod_{v \in N_s(u)} \frac{\exp(f(v)^T f(u))}{\sum_{w \in V} \exp(f(w)^T f(u))} \right)$$

$$\sum_{w \in V} \exp(f(w)^T f(u))$$

Hard to compute -> all the graph is required!

DeepWalk:
hierarchical softmax

node2vec:
negative sampling

04 The learning problem

$$\sum_{u\in V} \log P_f(N_s(u)|u) = \sum_{u\in V}\log\left(\prod_{v\in N_s(u)} P_f(v|u)\right) =$$

$$= \sum_{u\in V}\log\left(\prod_{v\in N_s(u)} \frac{\exp(f(v)^T f(u))}{\sum_{w\in V}\exp(f(w)^T f(u))}\right)$$

$$\sum_{w\in V}\exp(f(w)^T f(u))$$

Hard to compute -> all the graph is required!

COMING SOON

DeepWalk:
hierarchical softmax

node2vec:
negative sampling

05 Extension to edges

Based on aggregation of the node embedding f(u)

Define edge embedding

$$g : V \times V \to \mathbb{R}^{'}$$

$$(u, v) \mapsto B(f(u), f(v))$$

Aggregation function B:

- Average: $B(f(u), f(v)) = \dfrac{f(u) + f(v)}{2}$
- Hadamard: $B(f(u), f(v)) = f(u) \odot f(v)$
- Component-wise distance
- Component-wise squared distance
- ....