Learning Graph Node Embeddings by Smooth Pair Sampling

AISTATS 2025

Konstantin Kutzkov

Archon Labs

Overview

Problem statement and DeepWalk

Problem

- Given a graph, we want to represent its nodes by continuous vectors, the so-called embeddings.
- The embeddings should capture the structural properties of the graph.
- The model for training embeddings should be intuitive and scalable.
- Many solutions have been presented in the last decade, most of them inspired by the DeepWalk algorithm.

DeepWalk

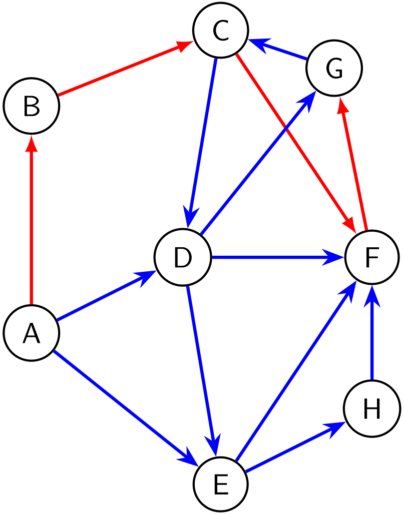
Input: Graph $G = (V, E)$, walk length ℓ, # negative samples k
for $u \in V$ **do**
  Start at node u a random walk T of length ℓ
  Generate positive node pairs P from the node sequence T using skip-gram
  for $u, v \in P$ **do**
    Generate k (random) negative node pairs u, w_i
    Feed $(u, v, +)$ and $(u, w_i, -)$ into a binary classifier that learns node embeddings $\vec{u} \in \mathbb{R}^d$ for $u \in V$

- One of the first algorithms for learning graph node embeddings.
- Generalizes word2vec by generating a corpus of node sequences using random walks which capture well the structural properties of the graph.
- For each positive pair of nodes generated by skip-gram on a random walk, we generate k negative pairs.

DeepWalk

Input: Graph $G = (V, E)$, walk length ℓ, # negative samples k
for $u \in V$ **do**
  Start at node u a random walk T of length ℓ
  Generate positive node pairs P from the node sequence T using skip-gram
  for $u, v \in P$ **do**
    Generate k (random) negative node pairs u, w_i
    Feed $(u, v, +)$ and $(u, w_i, -)$ into a binary classifier that learns node embeddings $\vec{u} \in \mathbb{R}^d$ for $u \in V$

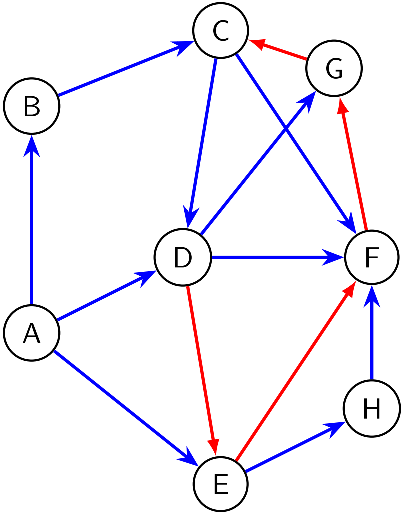
Sequence T: A, B, C, F, G
Pairs P: (A, B), (A, C), (B, C), (B, F), (C, F), (C, G), (F, G)



DeepWalk

1: **Input:** Graph $G = (V, E)$, walk length ℓ, # negative samples k
2: **for** $u \in V$ **do**
3: Start at node u a random walk T of length ℓ
4: Generate positive node pairs P from the node sequence T using skip-gram
5: **for** $u, v \in P$ **do**
6: Generate k (random) negative node pairs u, w_i
7: Feed $(u, v, +)$ and $(u, w_i, -)$ into a binary classifier that learns node embeddings $\vec{u} \in \mathbb{R}^d$ for $u \in V$

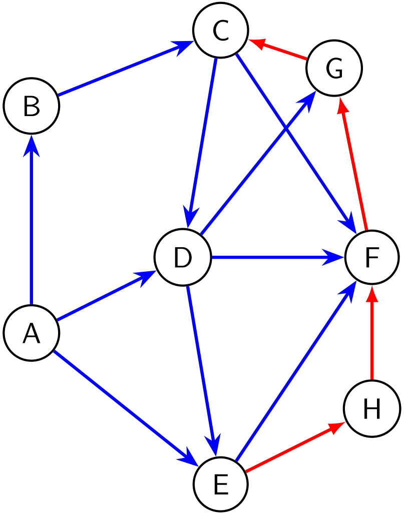
Sequence T: D, E, F, G, C
Pairs P: (D,E), (D, F), (E, F), (E, G), (F, G), (F, C), (G, C)



DeepWalk

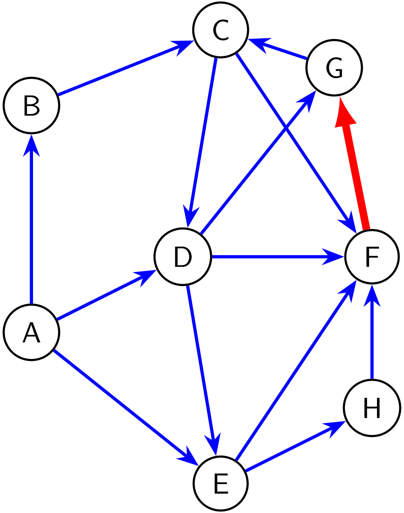
1: **Input:** Graph $G = (V, E)$, walk length ℓ, $\#$ negative samples k
2: **for** $u \in V$ **do**
3: Start at node u a random walk T of length ℓ
4: Generate positive node pairs P from the node sequence T using skip-gram
5: **for** $u, v \in P$ **do**
6: Generate k (random) negative node pairs u, w_i
7: Feed $(u, v, +)$ and $(u, w_i, -)$ into a binary classifier that learns node embeddings $\vec{u} \in \mathbb{R}^d$ for $u \in V$

Sequence T: E, H, F, G, C
Pairs P: (E, H), (E, F), (H, F), (H, G), (F, G), (F, C), (G, C)



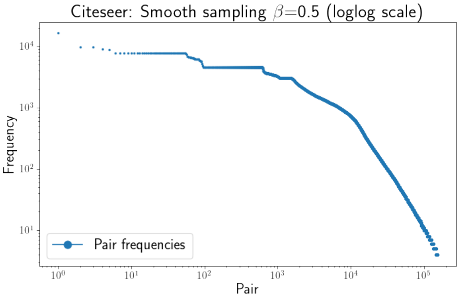
DeepWalk

- We see that the pair F, G in all sequences.
- It turns out that such heavy hitters are common in real graphs.
- We show theoretically and experimentally that the existence of such pairs affects negatively the learning process.



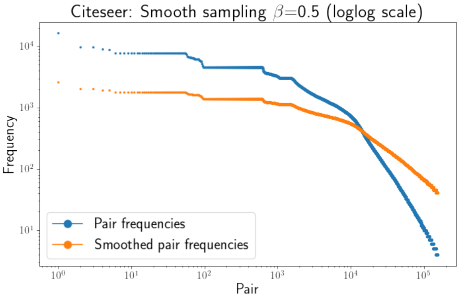
Pair frequency smoothing

Frequency distribution



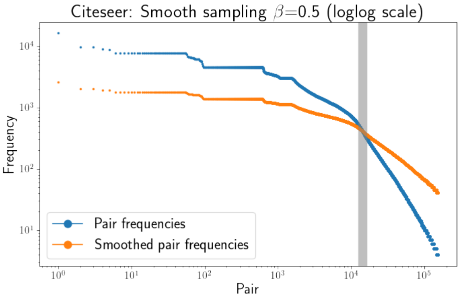
- The above plot shows the frequency distribution of positive pairs generated by DeepWalk.
- The distribution is highly skewed and a fraction of the pairs dominate the learning process (note the doubly-log scale).
- Previous works have indirectly approached the issue using approaches such as different random walk algorithms or negative sampling strategies.

Frequency distribution



- Our approach is to directly adjust the frequency of the positive pairs.
- If a pair u, v occurs $\#(u, v)$ times in the original random walk corpus, we update its frequency to $T_\beta \#(u, v)^\beta$ for $\beta \in (0, 1)$.
- T_β is a constant that depends on the total number of pairs in the random walk corpus.
- The orange dots show the smoothed frequency of the positive pairs.

Frequency distribution



- The gray vertical line shows the transition point after which smoothing leads to more positive samples for the corresponding pairs.
- We decrease the frequency only for a fraction of the pairs (note the doubly-log scale).
- Therefore, smoothing serves as a progressive tax-like wealth regularizer.

The effect of smoothing

Theorem (vanilla version)

- Let P the total number of positive pairs. If the frequencies adhere to a Zipfian distribution with $z > 1$, smoothing decreases the frequency only for $o(P)$ pairs and increases the frequency for the rest.
- Let $\vec{u}$ be the embedding vector of node u and $\mu : V \to [0, 1]$ be the negative sampling probability distribution. For $\#u^{(\beta)} = \sum_{v \in V} \#(u, v)^\beta$, SmoothDeepWalk optimizes the objective

$$\vec{u}^T \vec{v} = \log \frac{\#(u, v)^\beta}{\#u^{(\beta)} \mu(v)}$$

The effect of smoothing

Theorem (vanilla version)

- Let P the total number of positive pairs. If the frequencies adhere to a Zipfian distribution with $z > 1$, smoothing decreases the frequency only for $o(P)$ pairs and increases the frequency for the rest.
- Let $\vec{u}$ be the embedding vector of node u and $\mu : V \to [0, 1]$ be the negative sampling probability distribution. For $\#u^{(\beta)} = \sum_{v \in V} \#(u, v)^{\beta}$, SmoothDeepWalk optimizes the objective

$$\vec{u}^T \vec{v} = \log \frac{\#(u, v)^{\beta}}{\#u^{(\beta)} \mu(v)}$$

SmoothDeepWalk

SmoothDeepWalk

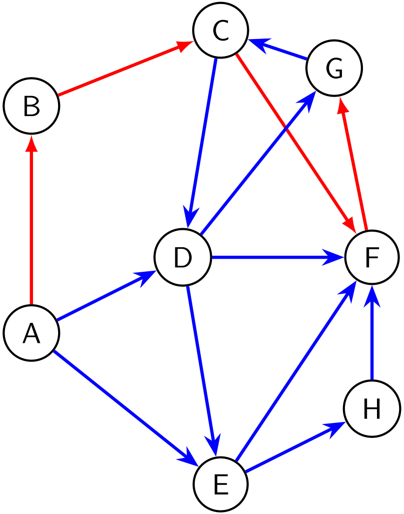
Input: Graph G, walk length ℓ, smoothing parameter $\beta \in (0,1)$, number of pairs M
samples $= 0$
while *samples* $< M$ **do**
  for $u \in G$ **do**
    Start from u a random walk T of length ℓ
    Generate positive node pairs P from T
    for $u, v \in P$ **do**
      Generate $r \in \mathbb{U}[0,1)$
      if $r \leq \#(u,v)^{\beta-1}$ **then**
        samples $+= 1$
        Generate k negative node pairs u, w_i
        Feed $(u, v, +)$ and $(u, w_i, -)$ into a binary classifier

- The only difference to DeepWalk is the sampling step.
- We iterate over the graph nodes until we have sampled the desired number of positive pairs M.
- M is the number of positive pairs that would be generated by the standard DeepWalk.

SmoothDeepWalk

```
Input: Graph G, walk length ℓ, smoothing parameter β ∈ (0,1), number of pairs M
samples = 0
while samples < M do
    for u ∈ G do
        Start from u a random walk T of length ℓ
        Generate positive node pairs P from T
        for u, v ∈ P do
            Generate r ∈ U[0,1)
            if r ≤ #(u,v)^(β−1) then
                samples += 1
                Generate k negative node pairs u, w_i
                Feed (u, v, +) and (u, w_i, −) into a binary classifier
```

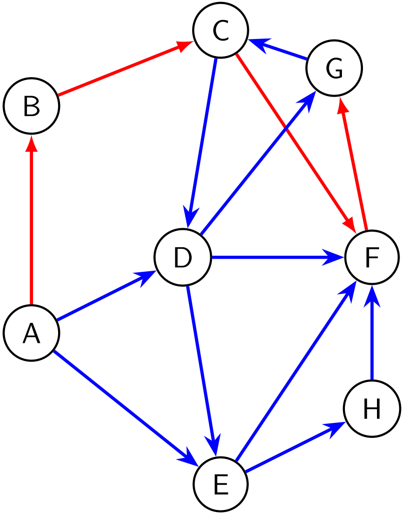
Let $\beta = 0.7$. We have #(A, B)=3. We generate $r = 0.65$. Since $0.65 < 3^{-0.3} \approx 0.72$ we sample (A, B).



SmoothDeepWalk

Input: Graph G, walk length ℓ, smoothing parameter $\beta \in (0, 1)$, number of pairs M
$samples = 0$
while $samples < M$ **do**
  for $u \in G$ **do**
    Start from u a random walk T of length ℓ
    Generate positive node pairs P from T
    for $u, v \in P$ **do**
      Generate $r \in \mathbb{U}[0, 1)$
      if $r \leq \#(u, v)^{\beta - 1}$ **then**
        $samples \mathrel{+}= 1$
        Generate k negative node pairs u, w_i
        Feed $(u, v, +)$ and $(u, w_i, -)$ into a binary classifier

Let $\beta = 0.7$. We have #(F, G)=10. We generate $r = 0.51$. Since $0.51 > 10^{-0.3} \approx 0.5$ we don't sample (F, G).



Accuracy of smoothing

We see that we sample the less frequent pair A, B with higher probability than the frequent pair F, G. More formally, $\#(u, v)$ is the frequency of pair u, v in the original DeepWalk corpus. We sample a pair u, v $T_\beta \#(u, v)$ times with probability $\#(u, v)^{\beta-1}$, thus we show the following result:

Theorem (vanilla version)

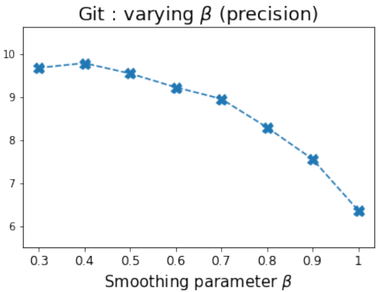
Let $S_{u,v}$ be the number of positive samples of pair u, v returned by SmoothDeepWalk. It holds $\mathbb{E}(S_{u,v}) = T_\beta \#(u, v)^\beta$ where T_β is a constant that increases with decreasing $\beta \in [0, 1]$. Further, $S_{u,v}$ is concentrated around the expected value with high probability.

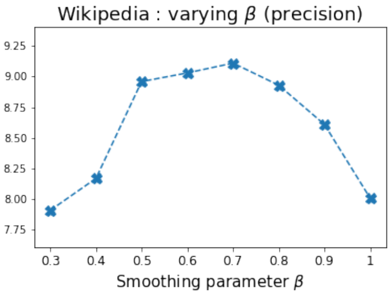
Space efficient smoothing

- In SmoothDeepWalk we assume we know the frequency of the pair u, v in the sampling step. $\texttt{If}\ \ r \leq \#(u, v)^{\beta - 1}$
- Exactly counting the frequencies of all pairs is clearly prohibitive for large graphs.
- For example, for the Flickr graph with 90K nodes and 450K edges, there are more than 200 million positive node pairs.
- We use a space-efficient sketching algorithm in order to estimate the frequency of *all* pairs.

Experimental evaluation

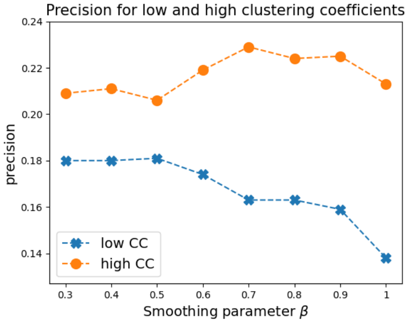
- Embeddings trained on a graph with 20% of edges removed.
- Nearest neighbors according to inner products.
- Compute precision and recall for the top 100 predicted edges w.r.t. removed edges.





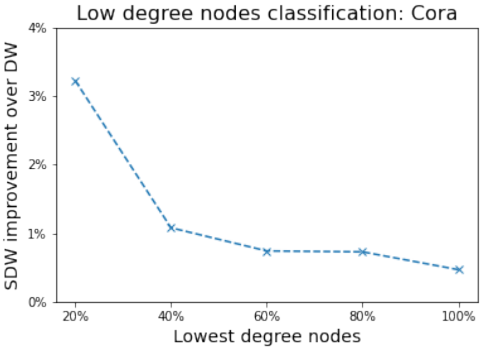
How strong should be smoothing?

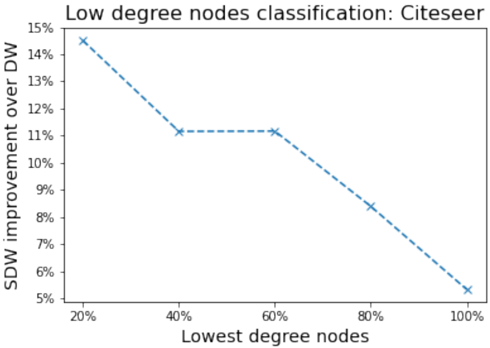
- The plot shows two synthetic graphs with a large and low clustering coefficients?
- A high clustering coefficients indicates the existence of local densely connected communities.
- Too aggressive smoothing (low β values) destroys such communities.
- Recommendation: check the clustering coefficient of the graph and adjust β accordingly.



Node classification

- We train a logistic regression model using the embeddings as features.
- We observe more significant accuracy gains when training and evaluating the model on lower degree nodes.





Summary

- Pair frequency smoothing appears to make the learning process more robust.
- The presentation focuses on DeepWalk but the approach applies to arbitrary (random) walk-based node embedding approaches, see the paper for examples.
- Note that aggressive smoothing (low β) increases the running time as we need several passes over the graph nodes in order to generate the desired number of positive pairs.
- Also, a very small sketch size leads to filtering of infrequent pairs, and thus increased running time.
- In the paper we recommend default values for the smoothing parameter β and the sketch size.

Open questions

- Study the combination of smooth pair sampling with different approaches to negative pair generation.
- Can we provide more insights when and how smoothing works?
- Does the approach yield advantages for Graph Neural Networks?

Thank you!

The FREQUENT algorithm

Frequent pair mining

1: **Input:** Stream of items $\mathcal{S}$, capacity k
2: sketch $H = \emptyset$
3: **for** $u \in \mathcal{S}$ **do**
4: **if** $u \in H$ **then**
5: $H[u] += 1$
6: **else**
7: **if** $|H| \leq k$ **then**
8: $H[u] = 1$
9: **if** $|H| > k$ **then**
10: Decrease by 1 the counter for all items in H
11: Remove from H items whose count is 0

New item ♣

Item	♣		
Counter	1		

The Frequent algorithm

Frequent pair mining

```
Input: Stream of items S, capacity k
sketch H = ∅
for u ∈ S do
    if u ∈ H then
        H[u] += 1
    else
        if |H| ≤ k then
            H[u] = 1
        if |H| > k then
            Decrease by 1 the counter for all items in H
            Remove from H items whose count is 0
```

New item $\diamondsuit$

Item	$\clubsuit$	$\diamondsuit$	
Counter	1	1	

The Frequent algorithm

Frequent pair mining

Input: Stream of items $\mathcal{S}$, capacity k
sketch $H = \emptyset$
for $u \in \mathcal{S}$ **do**
    if $u \in H$ **then**
        $H[u] \mathrel{+}= 1$
    else
        if $|H| \leq k$ **then**
            $H[u] = 1$
        if $|H| > k$ **then**
            Decrease by 1 the counter for all items in H
            Remove from H items whose count is 0

New item ♣

Item	♣	♢	
Counter	2	1	

The FREQUENT algorithm

Frequent pair mining

1: **Input:** Stream of items $\mathcal{S}$, capacity k
2: sketch $H = \emptyset$
3: **for** $u \in \mathcal{S}$ **do**
4:     **if** $u \in H$ **then**
5:         $H[u] += 1$
6:     **else**
7:         **if** $|H| \leq k$ **then**
8:             $H[u] = 1$
9:         **if** $|H| > k$ **then**
10:             Decrease by 1 the counter for all items in H
11:             Remove from H items whose count is 0

New item ♠

Item	♣	♢	♠
Counter	2	1	1

The FREQUENT algorithm

Frequent pair mining

```
Input: Stream of items 𝒮, capacity k
sketch H = ∅
for u ∈ 𝒮 do
    if  u ∈ H then
        H[u] += 1
    else
        if |H| ≤ k then
            H[u] = 1
        if |H| > k then
            Decrease by 1 the counter for all items in H
            Remove from H items whose count is 0
```

New item ♡

Item	♣	♢	♠
Counter	1	0	0

The FREQUENT algorithm

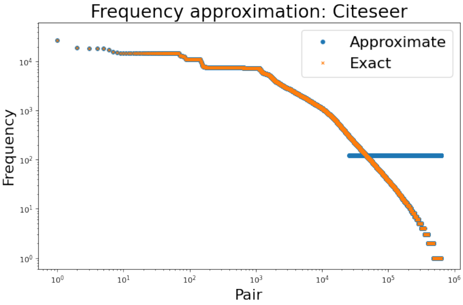
Frequent pair mining

Input: Stream of items $\mathcal{S}$, capacity k
sketch $H = \emptyset$
for $u \in \mathcal{S}$ **do**
  if $u \in H$ **then**
    $H[u] \mathrel{+}= 1$
  else
    if $|H| \leq k$ **then**
      $H[u] = 1$
    if $|H| > k$ **then**
      Decrease by 1 the counter for all items in H
      Remove from H items whose count is 0

New item ♡

Item	♣		
Counter	1		

Approximation quality



- For Zipfian distribution with $z > 1$, Frequent provably detect the k most frequent node pairs using $O(k)$ memory.
- In a second pass over the corpus we compute the exact frequency of pairs in the sketch.
- The frequency of pairs not in the sketch is approximated as `# pairs_not_in_sketch / sketch_size`.