node2vec: Scalable Feature Learning for Networks

Presenter: Tomáš Nováček, Faculty of information technology, CTU
Supervisor: doc. RNDr. Ing. Marcel Jiřina, Ph.D.

Authors: Aditya Grover, Jure Leskovec; Stanford University
Background
Tasks in network analysis

- Labels prediction
  - e.g. is user interested in Game of Thrones?
- Link prediction
  - e.g. are users real-life friends?
- Community detection
  - e.g. do characters in a book often meet?
Feature learning

1. Hand-engineering features
   ○ Based on expert knowledge
   ○ - Time-consuming
   ○ - Not generic enough

2. Solving optimization problem
   ○ Supervised
     ■ Good accuracy, high training time
   ○ Unsupervised
     ■ Efficient, hard to find the objective
   ○ Trade-off in efficiency and accuracy
Optimization problem

- Classic approach – linear and non-linear dimensionality reduction
- Alternative approach – preserving local neighbours
  - Most attempts rely on rigid notion
  - Insensitivity to connectivity patterns
    - Homophily
      - Based on communities
    - Structural equivalence
      - Roles in network
      - Equivalence does not emphasise connectivity
node2vec
node2vec

- Semi-supervised algorithm
- Generates sample network neighbours
  - Maximises likelihood of preserving neighborhood
  - Flexible notion of neighborhood of nodes
- Tunable parameters
  - Unsupervised
  - Semi-supervised
- Parallelizable
Skip-gram model

- Made for NLP (word2vec)
- Prediction of consecutive words
  - Similar context => similar meaning
- Learning feature representations
  - Optimizing likelihood objective
  - Neighborhood preserving
- Can we use it for networks?
  - Yes! We have to linearize the network.

### Source Text

- The quick brown fox jumps over the lazy dog.
- The quick brown fox jumps over the lazy dog.
- The quick brown fox jumps over the lazy dog.
- The quick brown fox jumps over the lazy dog.

### Training Samples

- (the, quick)
- (the, brown)
- (quick, the)
- (quick, brown)
- (quick, fox)
- (brown, the)
- (brown, quick)
- (brown, fox)
- (brown, jumps)
- (fox, quick)
- (fox, brown)
- (fox, jumps)
- (fox, over)
Feature learning in networks

- **G = (V, E)**
  - V – vertices (nodes)
  - E – edges (links)
  - (un)directed (un)weighted

- **f : V → R^d**
  - f – mapping func from nodes to feature representations
  - d – number of dimensions
  - matrix of size |V| × d parameters

- **∀ u ∈ V: N_S(u) ⊂ V**
  - N_S(u) – network neighborhood of u
  - S – sampling strategy
Optimizing objective function

- Maximizes the log-probability of observing a network neighborhood

\[
\max_f \sum_{u \in V} \log Pr(N_S(u) | f(u))
\]  

- \( N_S(u) \) is network neighborhood of node \( u \)
- Conditioned on its feature representation, given by \( f \)
Assumptions

● Conditional independence
  ○ Likelihood of observing a neighborhood node is independent of observing any other neighborhood node

\[ Pr(N_S(u) | f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i | f(u)) \]

● Symmetry in feature space
  ○ Source node and neighborhood node have a symmetric effect over each other

\[ Pr(n_i | f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))} \]
Optimizing objective function

- Thus we can simplify (1):

\[
\max_f \sum_{u \in V} \left[ -\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]
\]

- Where \( Z_u \) is the per-node partition function

\[
Z_u = \sum_{v \in V} \exp(f(u) \cdot f(v))
\]

- \( Z_u \) is approximated using negative sampling
Search strategies

- **Breadth-first sampling (BFS)**
  - Immediate neighbors
  - Small portion of the graph
  - Used by LINE algorithm

- **Depth-first sampling (DFS)**
  - Sequential nodes at increasing distances
  - Larger portion of the graph
  - Used by DeepWalk algorithm

- Constrained size $k$
- Multiple sets for a node
Breadth-first sampling

- Samples correspond closely to structural equivalence
- Accurate characterization of the local neighborhoods
  - Bridges
  - Hubs
- Nodes tend to repeat
- Small graph is explored
  - Microscopic view of the neighborhood
Depth-first sampling

- Larger part is explored
  - Reflects the macroscopic view
- Can be user to infer homophily
- Need to infer dependencies and their nature
  - High variance
  - Complex dependencies
node2vec

- Flexible biased 2\textsuperscript{nd} order random walk
  - Can return to previously visited node
  - Time and space efficient
- Combines BFS and DFS
  - Controlled by parameters
Parameters

- **Parameter $p$ (return parameter)**
  - Likelihood of immediately revisiting a node
  - High value ($> \text{max}(q, 1)$) => less probability
  - Low value ($< \text{min}(q, 1)$) => local walk

- **Parameter $q$ (in-out parameter)**
  - Inward vs. outward nodes
  - $q > 1$
    - Biased to local view of graph
    - BFS-like behaviour
  - $q < 1$
    - Further nodes
    - DFS-like behaviour
Search bias

- Edge weights bias
  - Does not account structure
  - Does not combine BFS and DFS
- Parameters p and q

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

- \( \pi_{vx} = \alpha_{pq}(t, x) \times w_{vx} \)
node2vec phases

1. Preprocessing to compute transition probabilities
2. Random walk simulations
   ○ $r$ random walks of fixed length $l$ from every node
     ■ Offset of start node implicit bias
3. Optimization using SGD

- Phases executed sequentially
- Phases asynchronous and parallelizable
Learning edge features

- Binary operator $\circ$ over corresponding feature vectors $f(u)$ and $f(v)$
- $g(u, v)$ such that $g : V \times V \to \mathbb{R}^d$

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>$\oplus$</td>
<td>$[f(u) \oplus f(v)]_i = \frac{f_i(u) + f_i(v)}{2}$</td>
</tr>
<tr>
<td>Hadamard</td>
<td>$\odot$</td>
<td>$[f(u) \odot f(v)]_i = f_i(u) \cdot f_i(v)$</td>
</tr>
<tr>
<td>Weighted-L1</td>
<td>$| \cdot |_1$</td>
<td>$|f(u) \cdot f(v)|_{1i} =</td>
</tr>
<tr>
<td>Weighted-L2</td>
<td>$| \cdot |_2$</td>
<td>$|f(u) \cdot f(v)|_{2i} = (f_i(u) - f_i(v))^2$</td>
</tr>
</tbody>
</table>
Experiments
Les Misérables

- Victor Hugo novel (1862)
- 77 nodes
  - characters from the novel
- 254 edges
  - co-appearing characters
- $d = 16$
  - number of dimensions
Les Misérables – homophily

- $p = 1$
  - less likely to immediately return
- $q = 0.5$
  - DFS
Les Misérables – structural equivalence

- $p = 1$
  - likely return
- $q = 2$
  - BFS
Benchmark

- **Spectral clustering**
  - matrix factorization approach

- **DeepWalk**
  - simulating uniform random walks
  - special case of node2vec with $p = 1$ and $q = 1$

- **LINE**
  - first phase – $d/2$ dimensions, BFS-style simulations
  - second phase – $d/2$ dimensions, nodes at 2-hop distance from the source

- **node2vec**
  - $d = 128$, $r = 10$, $l = 80$, $k = 10$
  - $p$, $q$ learned on 10% labeled data from {0.25, 0.50, 1, 2, 4}
Datasets

- **BlogCatalog**
  - social relationships of bloggers
  - labels are interests of bloggers
  - 10,312 nodes, 333,983 edges, 39 different labels

- **Protein-Protein Interactions (PPI)**
  - PPI network for Homo sapiens
  - labels from the hallmark gene set
  - 3,890 nodes, 76,584 edges, 50 different labels

- **Wikipedia**
  - co-occurrence of words the first million bytes of the Wikipedia dump
  - labels represent the Part-of-Speech (POS) tags
  - 4,777 nodes, 184,812 edges, 40 different labels
Link prediction

● Generated dataset
  ○ Positive sample generation
    ■ randomly removing 50% of edges
    ■ network stays connected
  ○ Negative sample generation
    ■ 50% node pairs
    ■ no edge between them

● Benchmarks
  ○ Facebook users (4 039 nodes, 88 234 edges)
  ○ Protein-Protein Interactions (19 706 nodes and 390 633 edges)
  ○ arXiv ASTRO-PH (18 722 nodes and 198 110)

<table>
<thead>
<tr>
<th>Op</th>
<th>Algorithm</th>
<th>Facebook</th>
<th>Dataset</th>
<th>arXiv</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>Spectral Clustering</td>
<td>0.5960</td>
<td>0.6588</td>
<td>0.5812</td>
</tr>
<tr>
<td></td>
<td>DeepWalk</td>
<td>0.7238</td>
<td>0.6923</td>
<td>0.7066</td>
</tr>
<tr>
<td></td>
<td>LINE</td>
<td>0.7029</td>
<td>0.6330</td>
<td>0.6516</td>
</tr>
<tr>
<td></td>
<td>node2vec</td>
<td>0.7266</td>
<td>0.7543</td>
<td>0.7221</td>
</tr>
<tr>
<td>(b)</td>
<td>Spectral Clustering</td>
<td>0.6192</td>
<td>0.4920</td>
<td>0.5740</td>
</tr>
<tr>
<td></td>
<td>DeepWalk</td>
<td>0.9680</td>
<td>0.7441</td>
<td>0.9340</td>
</tr>
<tr>
<td></td>
<td>LINE</td>
<td>0.9490</td>
<td>0.7249</td>
<td>0.8902</td>
</tr>
<tr>
<td></td>
<td>node2vec</td>
<td>0.9680</td>
<td>0.7719</td>
<td>0.9366</td>
</tr>
<tr>
<td>(c)</td>
<td>Spectral Clustering</td>
<td>0.7200</td>
<td>0.6356</td>
<td>0.7099</td>
</tr>
<tr>
<td></td>
<td>DeepWalk</td>
<td>0.9574</td>
<td>0.6026</td>
<td>0.8282</td>
</tr>
<tr>
<td></td>
<td>LINE</td>
<td>0.9483</td>
<td>0.7024</td>
<td>0.8809</td>
</tr>
<tr>
<td></td>
<td>node2vec</td>
<td>0.9602</td>
<td>0.6292</td>
<td>0.8468</td>
</tr>
<tr>
<td>(d)</td>
<td>Spectral Clustering</td>
<td>0.7107</td>
<td>0.6026</td>
<td>0.6765</td>
</tr>
<tr>
<td></td>
<td>DeepWalk</td>
<td>0.9584</td>
<td>0.6118</td>
<td>0.8305</td>
</tr>
<tr>
<td></td>
<td>LINE</td>
<td>0.9460</td>
<td>0.7106</td>
<td>0.8862</td>
</tr>
<tr>
<td></td>
<td>node2vec</td>
<td>0.9606</td>
<td>0.6236</td>
<td>0.8477</td>
</tr>
</tbody>
</table>
Conclusion

- Efficient scalable algorithm for feature learning
  - both nodes and edges between them
- Network-aware
  - homophily and structural equivalence
- Parameterizable
  - dimensions, length of walk, number of walks, sample size
  - return parameter
  - inward-outward parameter
- Parallelizable
- Link prediction
Drawbacks

- Vague definitions
- Only works for single-layered networks
- Worse results in dense graphs
- Unanswered questions
  - What if the graph changes?
  - How about featureless nodes?