CSCE 496/896 Lecture 8: word2vec and node2vec

Stephen Scott

(Adapted from Haluk Dogan)

sscott@cse.unl.edu

Introduction

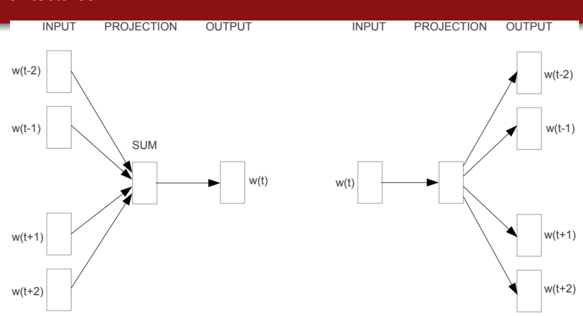
- To apply recurrent architectures to text (e.g., NLM), need numeric representation of words
  - The “Embedding lookup” block
- Where does the embedding come from?
  - Could train it along with the rest of the network
  - Or, could use “off-the-shelf” embedding
    - E.g., word2vec or GloVe
- Embeddings not limited to words: E.g., biological sequences, graphs, ...
  - Graphs: node2vec
- The xxxx2vec approach focuses on training embeddings based on **context**

Outline

Word2vec (Mikolov et al.)

- Training is a variation of autoencoding
- Rather than mapping a word to itself, learn to map between a word and its **context**
  - Context-to-word: **Continuous bag-of-words** (CBOW)
  - Word-to-context: **Skip-gram**

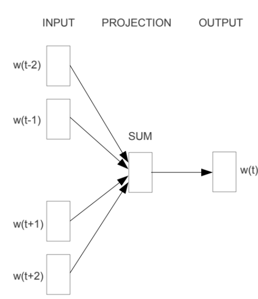
Word2vec (Mikolov et al.)
Architectures



CBOW **Skip-gram**

- CBOW: Predict current word $w(t)$ based on context
- Skip-gram: Predict context based on $w(t)$
- One-hot input, hidden linear activation, softmax output

Word2vec (Mikolov et al.)
CBOW

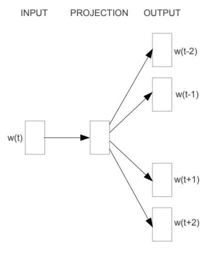


CBOW

- N = vocabulary size, d = embedding dimension
- $N \times d$ matrix W is shared weights from input to hidden
- $d \times N$ matrix W' is weights from hidden to output
- When one-hot context vectors $\boldsymbol{x}_{t-2}, \boldsymbol{x}_{t-1}, \ldots, \boldsymbol{x}_{t+2}$ input, corresponding rows from W are summed to $\hat{\boldsymbol{v}}$
- Then get **score vector** $\boldsymbol{v}'$ and softmax it
- Train with cross-entropy
- Use ith column of W' as embedding

Word2vec (Mikolov et al.)
Skip-gram

- Symmetric to CBOW: use ith row of W as embedding
- Goal is to maximize $P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$
- Same as minimizing $-\log P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t)$
- Assume words are independent given w_t: $P(w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2} \mid w_t) = \prod_{j \in \{-2,-1,1,2\}} P(w_{t+j} \mid w_t)$



Skip-gram

CSCE 496/896 Lecture 8: word2vec and node2vec
Stephen Scott
Introduction
word2vec
node2vec

Word2vec (Mikolov et al.)
Skip-gram

- Equivalent to maximizing log probability

$$\sum_{j \in \{-c,-(c-1),\ldots,(c-1),c\}, j \neq 0} \log P(w_{t+j} \mid w_t)$$

- Softmax output and linear activation imply

$$P(w_O \mid w_I) = \frac{\exp\left(\boldsymbol{v}_{w_O}'^{\top} \boldsymbol{v}_{w_I}\right)}{\sum_{i=1}^{N} \exp\left(\boldsymbol{v}_i'^{\top} \boldsymbol{v}_{w_I}\right)}$$

  where $\boldsymbol{v}_{w_I}$ is w_I's (input word) row from W and $\boldsymbol{v}_i'$ is w_i's (output word) column from W'
- I.e., trying to maximize dot product (similarity) between words in same context
- **Problem:** N is big ($\approx 10^5$–10^7)

CSCE 496/896 Lecture 8: word2vec and node2vec
Stephen Scott
Introduction
word2vec
node2vec

Word2vec (Mikolov et al.)
Skip-gram

- Speed up evaluation via **negative sampling**
- Update the weight of each target word and only a small number (5–20) of **negative words**
- I.e., do not update for all N words
- To estimate $P(w_O \mid w_I)$, use

$$\log \sigma\left(\boldsymbol{v}_{w_O}'^{\top} \boldsymbol{v}_{w_I}\right) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma\left(-\boldsymbol{v}_{w_i}'^{\top} \boldsymbol{v}_{w_I}\right)\right]$$

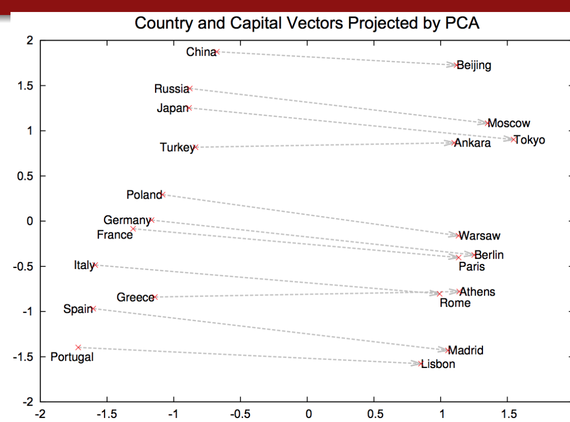
- I.e., learn to distinguish target word w_O from words drawn from **noise distribution**

$$P_n(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{N} f(w_j)^{3/4}} ,$$

  where $f(w_i)$ is frequency of word w_i in corpus
- I.e., $P_n(w_i)$ is a **unigram distribution**

CSCE 496/896 Lecture 8: word2vec and node2vec
Stephen Scott
Introduction
word2vec
node2vec

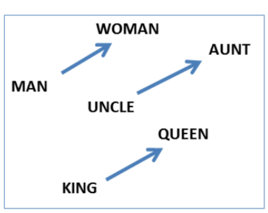
Word2vec (Mikolov et al.)
Semantics

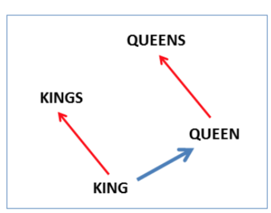


- Distances between countries and capitals similar

CSCE 496/896 Lecture 8: word2vec and node2vec
Stephen Scott
Introduction
word2vec
node2vec

Word2vec (Mikolov et al.)
Semantics





- Analogies: a is to b as c is to d
- Given normalized embeddings $\boldsymbol{x}_a$, $\boldsymbol{x}_b$, and $\boldsymbol{x}_c$, compute $\boldsymbol{y} = \boldsymbol{x}_b - \boldsymbol{x}_a + \boldsymbol{x}_c$
- Find d maximizing cosine: $\boldsymbol{x}_d \boldsymbol{y}^{\top} / (\|\boldsymbol{x}_d\| \|\boldsymbol{y}\|)$

CSCE 496/896 Lecture 8: word2vec and node2vec
Stephen Scott
Introduction
word2vec
node2vec

Node2vec (Grover and Leskovec, 2016)

- Word2vec's approach generalizes beyond text
- All we need to do is represent the context of an instance to embed together instances with similar contexts
  - E.g., biological sequences, nodes in a graph
- Node2vec defines its context for a node based on its local neighborhood, role in the graph, etc.

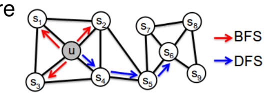
Node2vec (Grover and Leskovec, 2016)
Notation

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- $\mathcal{A}$ is a $|\mathcal{V}| \times |\mathcal{V}|$ adjacency matrix
- $f : \mathcal{V} \to \mathbb{R}^d$ is a mapping function from individual nodes to feature representations
  - $|\mathcal{V}| \times d$ matrix
- $N_S(u) \subset \mathcal{V}$ denotes a neighborhood of node u generated through a **neighborhood sampling strategy** S
- **Objective:** Preserve local neighborhoods of nodes

Node2vec (Grover and Leskovec, 2016)

Organization of nodes is based on:

- **Homophily:** Nodes that are highly interconnected and cluster together should embed near each other



- **Structural roles:** Nodes with similar roles in the graph (e.g., hubs) should embed near each other
- u and s_1 belong to the same community of nodes
- u and s_6 in two distinct communities share same structural role of a hub node

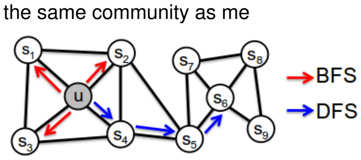
Goal

- Embed nodes from the same network community closely together
- Nodes that share similar roles have similar embeddings

node2vec

Key Contribution: Defining a flexible notion of a node's network neighborhood.

1. **BFS:** role of the vertex
   - far apart from each other but share similar kind of vertices
2. **DFS:** community
   - reachability/closeness of the two nodes
   - my friend's friend's friend has a higher chance to belong to the same community as me



node2vec

Objective function

$$\max_f \sum_{u \in \mathcal{V}} \log P\left(N_S(u) \mid f(u)\right)$$

Assumptions:

- Conditional independence:
  $$P\left(N_S(u) \mid f(u)\right) = \prod_{n_i \in N_S(u)} P(n_i \mid f(u))$$
- Symmetry in feature space:
  $$P\left(n_i \mid f(u)\right) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in \mathcal{V}} \exp(f(v) \cdot f(u))}$$

Objective function simplifies to:

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

Node2vec (Grover and Leskovec, 2016)
Neighborhood Sampling

Given a source node u, we simulate a random walk of fixed length ℓ:

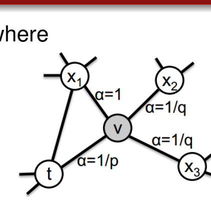
$$P\left(c_i = x \mid c_{i-1} = v\right) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

- $c_0 = u$
- π_{vx} is the unnormalized transition probability
- Z is the normalization constant.
- 2^{nd} order Markovian

Node2vec (Grover and Leskovec, 2016)
Neighborhood Sampling

Search bias α**:** $\pi_{vx} = \alpha_{pq}(t, x) w_{vx}$ where

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



Return parameter p:

- Controls the likelihood of immediately revisiting a node in the walk
- If $p > \max(q, 1)$
  - less likely to sample an already visited node
  - avoids 2-hop redundancy in sampling
- If $p < \min(q, 1)$
  - backtrack a step
  - keep the walk local

Node2vec (Grover and Leskovec, 2016)
Neighborhood Sampling

In-out parameter q:

- If $q > 1$ inward exploration
  - Local view
  - BFS behavior
- If $q < 1$ outward exploration
  - Global view
  - DFS behavior

Node2vec (Grover and Leskovec, 2016)
Algorithm

Algorithm 1 The *node2vec* algorithm.

```
LearnFeatures (Graph G = (V, E, W), Dimensions d, Walks per
    node r, Walk length l, Context size k, Return p, In-out q)
  π = PreprocessModifiedWeights(G, p, q)
  G' = (V, E, π)
  Initialize walks to Empty
  for iter = 1 to r do
    for all nodes u ∈ V do
      walk = node2vecWalk(G', u, l)
      Append walk to walks
  f = StochasticGradientDescent(k, d, walks)
  return f

node2vecWalk (Graph G' = (V, E, π), Start node u, Length l)
  Inititalize walk to [u]
  for walk_iter = 1 to l do
    curr = walk[−1]
    V_curr = GetNeighbors(curr, G')
    s = AliasSample(V_curr, π)
    Append s to walk
  return walk
```

- Implicit bias due to choice of the start node u
  - Simulating r random walks of fixed length ℓ starting from every node

Phases:

1. Preprocessing to compute transition probabilities
2. Random walks
3. Optimization using SGD

Each phase is parallelizable and executed asynchronously