Network Embedding with TensorFlow

(网络嵌入在TensorFlow中的实践)

Xihan Li

School of EECS, Peking University

xihanli@pku.edu.cn

https://snowkylin.github.io

2017/11/19

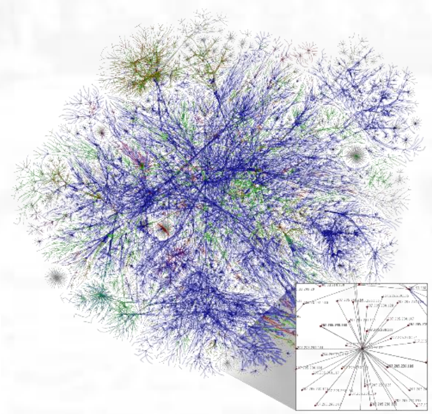
GDG DevFest 2017 Beijing

Network is ubiquitous

- Networks naturally exist in a wide diversity of real-world scenarios.
  - Social Network
  - Knowledge Network
  - Internet
  - User Interest Network
  - Etc.

Social network

Citation network



World Wide Web

Internet of Things (IOT)

…

(images from slide of Jian Tang's LINE paper)

Applications on Network

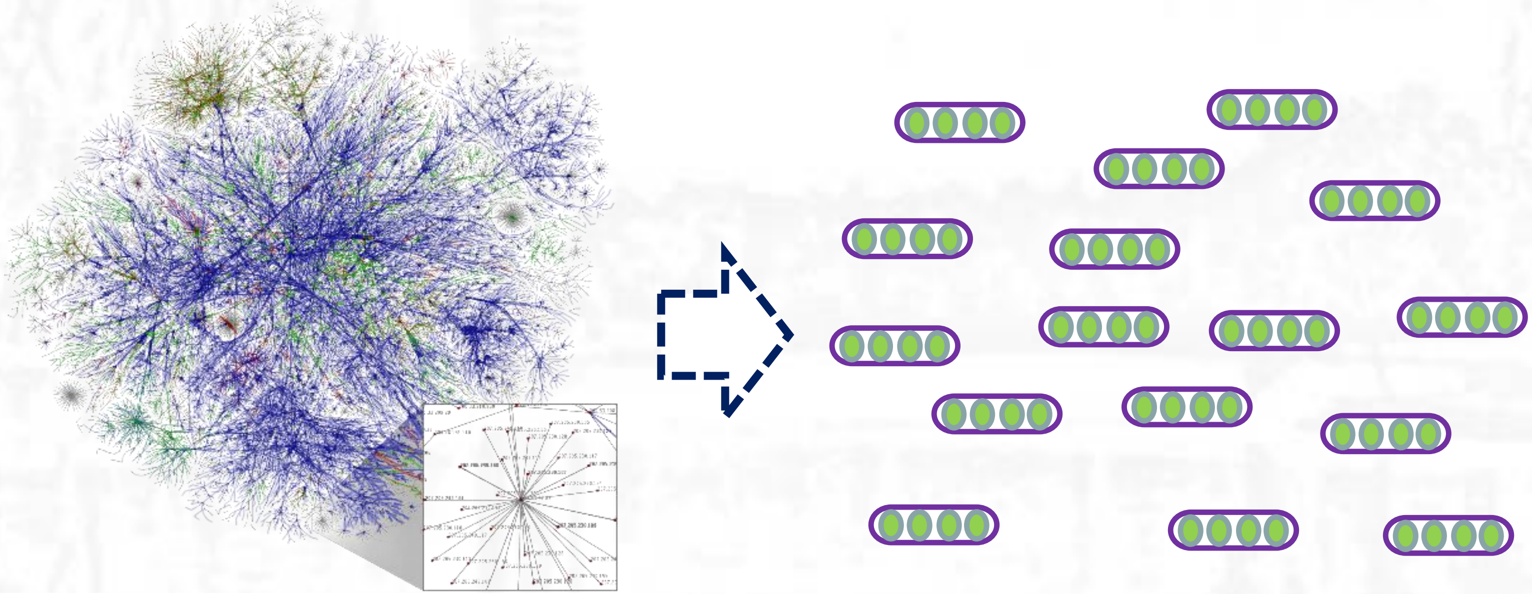
- Node classification / clustering
- Node recommendation
- Link prediction
- etc.

However, most network analytics methods suffer the high computation and space cost.

- Networks in real world can have billions of edges and millions of nodes
- Many problems on networks are NP-hard

Network Embedding

- Effective & efficient way to solve the problem
- Convert a network into a low-dimensional space in which the graph information is preserved



Sparse, high-dimension

Dense, low-dimension

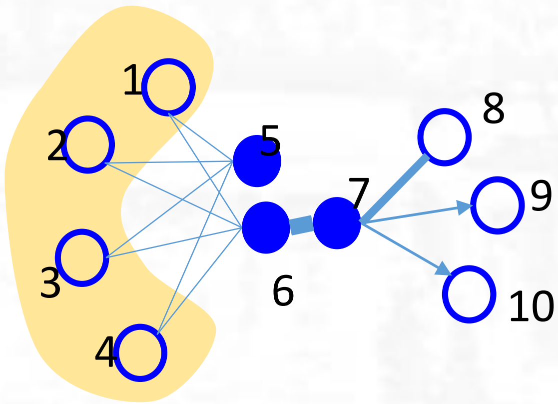
(images from slide of Jian Tang's LINE paper)

LINE

- Preserve first-order proximity and second-order proximity of network
  - **First-order proximity:** local pairwise proximity between the vertices
  - **Second-order Proximity:** proximity between the neighborhood structures of the vertices

Vertex **6** and **7** have a large first-order proximity

Vertex **5** and **6** have a large second-order proximity



(images from slide of Jian Tang's LINE paper)

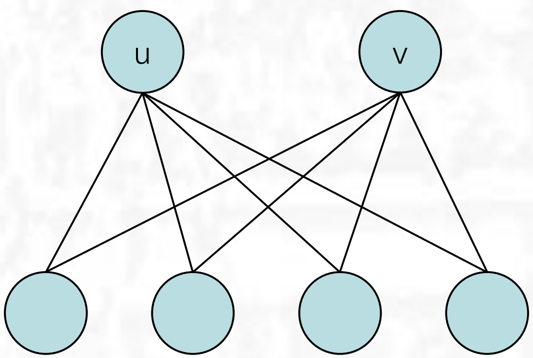
LINE

First-order proximity

$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$ and $\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$ close to each other

$$\hat{p}_1(u,v) = \frac{w_{uv}}{\sum_{(i',j')} w_{i'j'}} \approx p_1(u,v) = \frac{1}{1+\exp(-\vec{u}^T \cdot \vec{v})}$$

Second-order Proximity



$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$ and $\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$ close to each other

$$\hat{p}_2(v|u) = \frac{w_{uv}}{\sum_{k \in N(u)} w_{uk}} \approx p_2(v|u) = \frac{\exp(\vec{u}^T \cdot \vec{v})}{\sum_{p \in V} \exp(\vec{p}^T \cdot \vec{v})}$$

Objective function of LINE

Objective function for each edge $(i, j) \in E$

First-order:

$$\log \sigma\left(u_j^T \cdot u_i\right) + \Sigma_{i=1}^{K} E_{v_n \sim Pn(v)}[log\sigma(-u_n^T \cdot u_i]$$

Second-order:

$$\log \sigma\left(u'^T_j \cdot u_i\right) + \Sigma_{i=1}^{K} E_{v_n \sim Pn(v)}[log\sigma(-u'^T_n \cdot u_i]$$

Negative Sampling

First-order:
Given node list V, edge list E, solve embedding vectors $u_i, i \in V$
s.t.

$$\Sigma_{(i,j)\in E} log\sigma\left(u_j^T \cdot u_i\right) + (Negative\ sampling)$$

Is minimized

TensorFlow

- An open source software library for numerical computation originally developed by Google Brain Team.
- Main Features
  - Speed
    - GPUs
    - Distributed computation
  - Convenience
    - Automatic derivation
    - Built-in models with reliability
    - Single API

Three Types of "Values"

- Placeholder
- Variable
- Constant

Example:

$$L(i,j) = \log \sigma\left(u_j^T \cdot u_i\right)$$

- i, j: Placeholder
- $u = (u_1, \dots, u_{|V|})$: Variable
- Sorry, no constant here

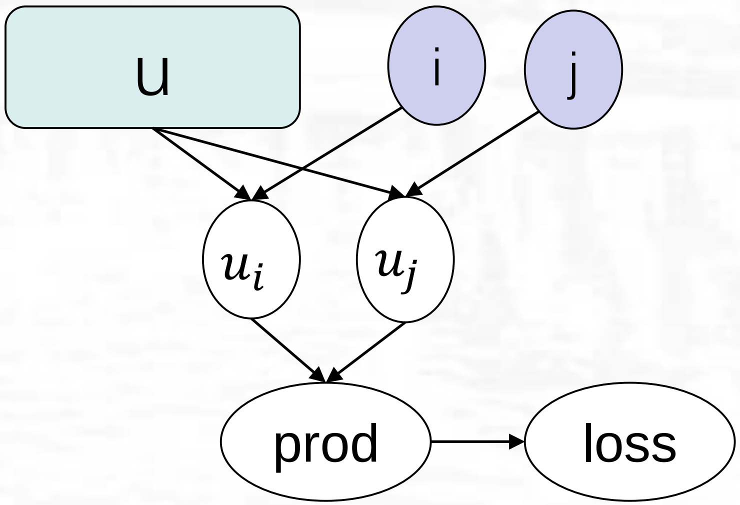
TensorFlow: Data Flow Model

$$L(i,j) = \log \sigma\left(u_j^T \cdot u_i\right)$$

TensorFlow Pseudocode

(minibatch and negative sampling are omitted here):

```
import tensorflow as tf
i = tf.placeholder(name='i', dtype=tf.int32)
j = tf.placeholder(name='j', dtype=tf.int32)
u = tf.get_variable('u', [num_of_nodes, embedding_dim],
    initializer=tf.random_uniform_initializer(minval=-1., maxval=1.))
u_i = u[i, :]
u_j = u[j, :]
prod = tf.reduce_sum(u_i * u_j)
loss = tf.log_sigmoid(prod)
```



TensorFlow: Data Feeding

$$L(i,j) = \log \sigma\left(u_j^T \cdot u_i\right)$$

We have an edge (1, 2) and we want to get $L(1,2)$

Pseudocode:

```
i = tf.placeholder(name='i', dtype=tf.int32)
j = tf.placeholder(name='j', dtype=tf.int32)
u = tf.get_variable('u', [num_of_nodes, embedding_dim],
    initializer=tf.random_uniform_initializer(minval=-1., maxval=1.))
u_i = u[i, :]
u_j = u[j, :]
prod = tf.reduce_sum(u_i * u_j)
loss = tf.log_sigmoid(prod)
with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        sess.run(loss, {i: 1, j: 2})      # if there is an edge (1, 2)
```

Initialize all variables

Training Variable

$$\min_{(i,j)\in E} L(i,j) = \min_{(i,j)\in E} \log \sigma\left(u_j^T \cdot u_i\right)$$

Pseudocode:

```
i = tf.placeholder(name='i', dtype=tf.int32)
j = tf.placeholder(name='j', dtype=tf.int32)
u = tf.get_variable('u', [num_of_nodes, embedding_dim],
    initializer=tf.random_uniform_initializer(minval=-1., maxval=1.))
u_i = u[i, :]
u_j = u[j, :]
prod = tf.reduce_sum(u_i * u_j)
loss = tf.log_sigmoid(prod)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
train_op = optimizer.minimize(loss)
with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for epoch in range(num_epoch)
                for u, v in edge:
                        sess.run(train_op, {i: u, j: v}
```

Summary

- TensorFlow version
  - Automatic derivation
  - All in one (with numpy)
  - Same code on all platform
  - No compilation
  - GPU acceleration
- C++ version
  - Manual derivation
  - External library to speed up calculation
  - Different code on different platform
  - Need compilation
  - Hard to use GPUs

Thank you!

Code & Slide at

https://github.com/snowkylin/line