Understanding Large Social Networks

Information Retrieval and Extraction (Major Project)

Mentor

Ganesh Jawahar

Team Members (Team No.: 57)

Madan Jhawar	201202018
Raj Patel	201301048
Kishan Vadaliya	201505621

Problem Statement

To build a model that can capture the network information of a node in an efficient and scalable manner by representing every node into low-dimensional vector spaces.

Motivation

- Representing network nodes into low-dimensional vector spaces helps us in many tasks such as visualization, node classification, and link prediction, etc., using standard machine learning algorithms on top of that.

- The size of most real-world information networks such as YouTube, Flickr, Facebook, etc., ranges from hundreds of nodes to millions and billions of nodes. Scalable solutions to understand such networks are very useful in the industry.

- Deep Learning has worked wonders in Language Modelling and has provided commendable results in node representation tasks as well.

Dataset used

- **Blogcatalog** dataset is used for this project.
- It contains `10312` nodes, `333983` links.
- These nodes and links represent users and their friendship respectively.
- Dataset - http://socialcomputing.asu.edu/datasets/BlogCatalog3

Baseline Paper

LINE: LARGE-SCALE INFORMATION NETWORK EMBEDDING

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, Qiaozhu Mei

LINE: First-Order Proximity

- First-order proximity refers to the local pairwise proximity between the vertices in the network (i.e Only the direct neighbours).
- To model the first-order proximity between each undirected edge (i, j), we define ‘joint probability’ between vertex v_i and v_j

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)},$$

where the empirical probability is given by: $\hat{p}_1(i, j) = \frac{w_{ij}}{W}$

- Minimize the KL-divergence of two probability distributions, given by

$$O_1 = - \sum_{(i,j) \in E} w_{ij} \log p_1(v_i, v_j)$$

LINE: Second-Order Proximity

- The second-order proximity assumes that vertices sharing many connections to other vertices are similar to each other.
- In this model - Vertex plays two roles: the vertex itself($\vec{u}_i$) and a specific “context” ($\vec{u}'_i$) of other vertices.
- For each directed edge (i, j), we first define the probability of “context” *v$_j$* generated by vertex *v$_i$* as:

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}'^T_j \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}'^T_k \cdot \vec{u}_i)},$$

  where the empirical probability is given by: $\hat{p}_2(v_j|v_i) = \frac{w_{ij}}{d_i}$

- We take KL-divergence as the distance function: $O_2 = -\sum_{(i,j)\in E} w_{ij} \log p_2(v_j|v_i)$

LINE: Negative Sampling (Optimization)

- If we carefully analyze the denominator, it is quite expensive operation as it considers dot-product with every other node, where dot-product itself takes linear (depends on size of the vector) time.

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)}$$

- To optimize we use **Negative Sampling,** which samples multiple negative edges according to some noisy distribution for each edge (i, j)

$$p_2(v_j|v_i) = \log \sigma(\vec{u}_j'^{\,T} \cdot \vec{u}_i) + \sum_{i=1}^{K} E_{v_n \sim P_n(v)}[\log \sigma(-\vec{u}_n'^{\,T} \cdot \vec{u}_i)]$$

Novelty

- The authors train the neural networks for the First-Order Proximity and the Second-Order Proximity independently, and then concatenate the embeddings learned from both the models for a node to get the final embeddings of a node.

- In our project, we use the same lookup table for the node representation in both the neural networks, i.e., the node embeddings learned from the First-Order Proximity are used in the neural network for the Second-Order Proximity, instead of initializing them again to random values.

- The embeddings learned from the Second-Order Proximity deep-net are, then, taken as the final embeddings of a node.

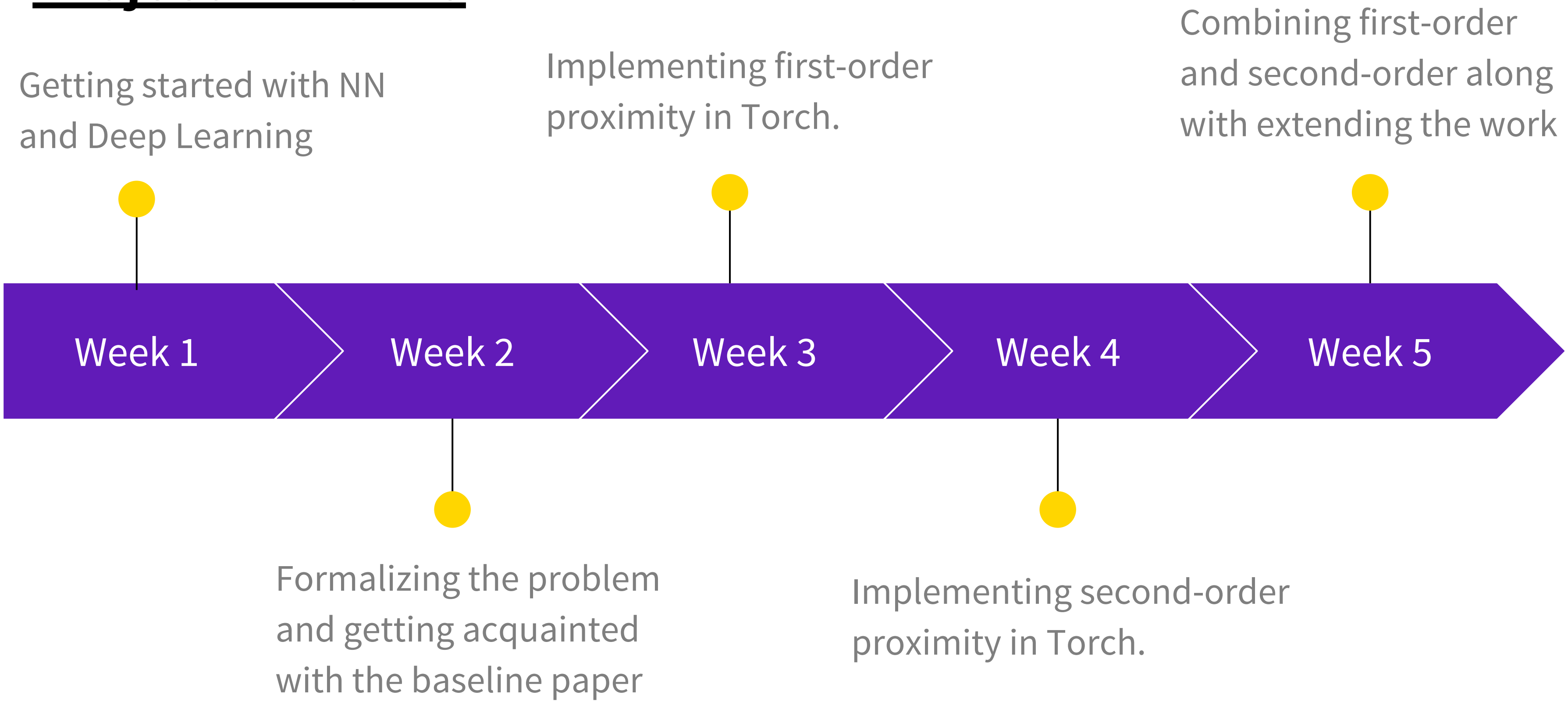
Languages used

- The complete code is written in **Lua** language.
- For building Neural Networks, **Torch**, a framework written in Lua, is used.
- We use the **scikit-learn** library in **Python** to test the accuracy of our model.

Experimentation and Results

“scoring.py” script is not working. Will be updated soon.

Project Timeline



Week 1: Getting started with NN and Deep Learning

Week 2: Formalizing the problem and getting acquainted with the baseline paper

Week 3: Implementing first-order proximity in Torch.

Week 4: Implementing second-order proximity in Torch.

Week 5: Combining first-order and second-order along with extending the work

Challenges Faced

- Understanding the mathematics of the baseline paper.
- Referring to other research articles related to the baseline paper.
- Getting around with Lua and Torch.
- Not everything could be printed and debugged while coding neural nets in Lua.
- The dataset is very large, so we had to make several adjustments with the data structures used in our code to reduce the time complexity.

References

- http://www.www2015.it/documents/proceedings/proceedings/p1067.pdf
- http://arxiv.org/pdf/1403.6652.pdf
- http://cs224d.stanford.edu/lecture_notes/LectureNotes1.pdf
- https://github.com/torch/nn
- https://github.com/gjwhr/dl4nlp-made-easy/blob/master/word2vec/cbow.lua
- https://github.com/gjwhr/dl4nlp-made-easy/blob/master/word2vec/skipgram.lua

Other links

YouTube	https://youtu.be/2Ifxt9nlzvk
GitHub	https:/github.com/raj454raj/Graph2vec/
github:pages	http://raj454raj.github.io/Graph2vec/

Thank you!