University of Hong Kong

Final Year Project Plan

Fast2Vec: Efficient and Scalable Information Network Embedding, FYP18058

MA Rutian
Department of Computer Science

Supervised by
Dr. Reynold C.K. Cheng
Department of Computer Science

September 29, 2018

Contents

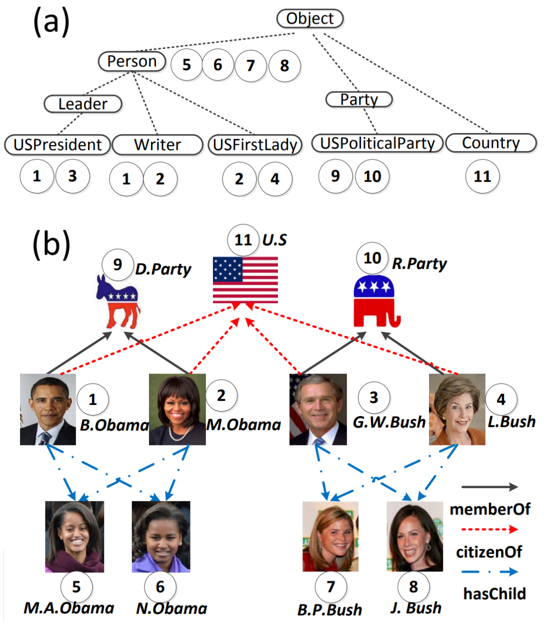
1 Introduction and Background

The final year project *FAST2VEC*proposed is an efficient, scalable and effective information network embedding toolkit. In this introduction section, key concepts related to the project, current works and initiatives of the project will be introduced.

1.1 Information Network

Information Network is a ubiquitous abstraction for large scale graph datasets in the forms of social media networks, scholarship networks, knowledge-base graph, protein-to-protein networks, etc[9] [2] [26] [7]. According to the number of types of edges and vertices in the network, research community further categorize them as Heterogeneous Information Networks and Homogeneous Information Networks. Scales of networks rise from hundreds to billions vertices [21] [22] and numbers of edges range from thousands to tens of billion as well. In addition, a information network may also carry auxiliary label and context information as shown in Figure 1.1.

Figure 1: Example of Information Network shown in Changping's Paper[14]

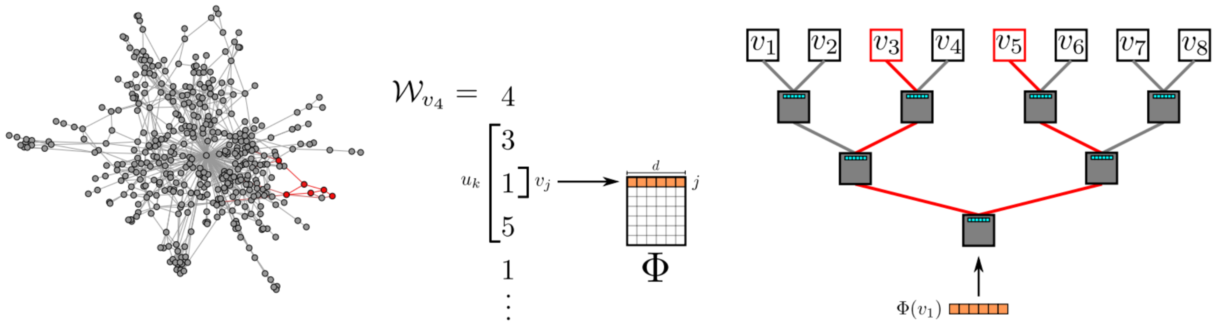


With blossom of social network, growth of computing power and development on machine learning algorithms, analysis on information network has drawn great attention. However, usual machine learning algorithms relies on numeric input, such as real number, vector or matrix while network data are of higher dimension and different type of information. Therefore the challenge of application of machine learning algorithms on network data lies on mapping information network to its numerical representation that fits those algorithms and such a mapping is also called *Network Representation Learning*

1.2 Network Representation Learning

As introduced previously, *Network Representation Learning*(*NLR*), which is also called *embedding*, is a mapping from complex network data to easy-to-use, lower dimensional real matrix which can be used for downstream machine learning algorithms. To define an *embedding*process, people first design an objective function evaluating information loss, the gap between real data and its matrix representation, through the conversion and then *embedding*becomes training a model, which is essentially a real-value matrix, that minimizes the selected objective function. After training the model, developers can leverage it to a collection of algorithms. Taking friend recommendation as an example, after embedding a social network, say Facebook, developers will have a real-value vector for each user and collectively a real-value matrix U will be generated for all the user. To decide whether or not to recommend user A to user B, developers can simply train a logistic-classifier based on the matrix U. Additionally, example work flow for *DeepWalk* is also shown below

Figure 2: Example of embedding work flow taking from *DeepWalk*[17] paper. First it generates random walks from information network. Second, it learns vector representation of vertices based on random walks. Last, it decodes information using *softmax* function



(a) Random walk generation. (b) Representation mapping. (c) Hierarchical Softmax.

As shown in the example above, embedding provides a general framework for complex handling network data and its downstream applications are sensitive to embedding quality. Therefore, scholars spend great effort designing effective and efficient algorithms which will be discussed in subsequent sub-section.

1.3 Current Works

Current focus on embedding is to encapsulate structural information of network data. By saying structural information, people emphasize how are vertices in the network connected rather than what a single vertex contains. Taking Google Scholar data as an example, most embedding techniques leverage the fact that who published which paper, which paper was published in which venue and which paper contains certain topics instead of content of a paper, *etc.*. Inspired by *word2vec*[15], a state-of-art word embedding algorithm, lots of *random-walks-based NLR*algorithms have been proposed.

In the reset of this sub-section, general framework of *random-walks-based NLR*algorithm will be first presented and then details of specific algorithms so as their pros and cons will be discussed.

1.3.1 Random-Walk-Based Network Representation Learning

word2vec algorithm [15] has been commonly adopted in *NLR*algorithms so it is critical to understand it before advancing to *NLR*algorithms. Similar to *NLR*, word embedding aims for finding a real value representation for each word. Insight of *word2vec* is that co-occurrence of two words within a sentence is an evidence of semantic similarity between them and lack of co-occurrence between two word implies semantic differences. For example, (*Father*, *Mother*) appear more often than (*Father*, *Ocean*) and therefore it is more convincing to conclude that *Father* is more similar to *Mother* than *Ocean* semantically. More formally, by defining co-occurrence probability by softmax function which takes two vectors as input, the objective of *word2vec* becomes find a representation that maximizes softmax function for co-occurred words and minimizes for non-co-occurred words.

Inspired by *skip-gram* and *word2vec* algorithm[15], lots of graph embedding algorithms leverage random walk to serve as analogy of sentence in *word2vec* and then train model against a collection of random walks while *word2vec* trains against a corpora. However, drawing random walk from information network is more difficult then tokenizing a document to generate sentences as:

- Vertices can be of different types and carrying different attributes[25]. Therefore we should decide whether to treat vertices of different types equally when draw a random walk.

- Sentences is a solid data source while random walk generation is algorithm specific and involves lots of design decision.

Consequently, we can categorize current embedding technologies into two sub-categories:

- Homogeneous Random Walk: When draw a random walke, it won't take vertex type nor value into account.
- Heterogeneous Random Walk: Unlike homogeneous, it will consider vertex type and value and assign probability accordingly. These leverage additional information in heterogeneous information network and hence called heterogeneous random walk.

In general, both types of algorithms will consist three components as shown below [9]

- Random Walker: a walker w generate a collection of random walks from a information network. It will determine whether a *NLR*algorithm is homogeneous or heterogeneous.
- Encoder: a mapping function between vertex in information network and its real-value vector representation.
- Decoder: a decoder takes two real-value vectors which are representation of corresponding pair of vertices as input and outputs a real value. Normally, it serves as the objective function of the algorithm and developers want to maximize output for co-occurred vertices and minimize for non-co-occurred vertices.

In the rest of sub-section both types of random-walk-based *NLR*algorithms will be discussed

1.3.2 DeepWalk

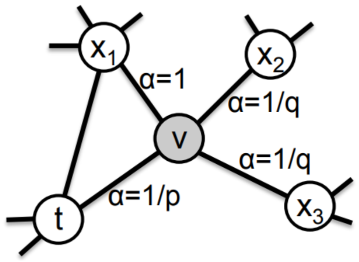
DeepWalk[17] is the first paper introducing *word2vec*[15] to *NLR*. By regarding each random-walk as a sentence, *DeepWalk* generates vector representation for every vertex in random walk. Formally speaking, its *Random Walker* generate fixed number of random walks with fixed length for all vertices and it use *softmax* as its *Decoder* function.

The drawback of this algorithm is that users and developers have limited control on drawing random walks from dataset.

1.3.3 Node2Vec

Node2Vec improves *DeepWalk* by adding customizable *random walker*. It has two parameter p and q interpolate between *BFS-like* and *DFS-like* random walk. A *BFS-like* random walk tend to explore neighbors around a vertex while *DFS-like* random walk will try to explore vertices far away from the source vertex. To better explain the process, an example has been shown below Figure 1.3.3

Figure 3: Example of random walks generation for *Node2Vec*. The walk just transited from t to v and α is the weight factor for computing next transition probability.



However, both methods above fail to leverage heterogeneous information network as their *random walkers* don't take vertices types and edges types into account.

1.3.4 Metapath2Vec and Metapath2Vec++

Metapath2Vec leverages heterogeneity of certain information network with an improved *random walker*. It first asks user to provide a *meta-path*, which is essentially a subset of schema of the information network, and then when generate random walk according to *meta-path*. Taking Google Scholar as an example again, first user sets meta-path to be *Author-Paper-Author* and then *Metapath2Vec* will generate a collection random walks that are instances of the selected meta-path, such as $author_1 - paper_2 - author_2 - paper_1 - \dots$. Similar to algorithms above, it will use *softmax* as its decoder.

[metapath2vec figure]

However, experiments have shown that *meta-path* is task-driven and may not cover all vertices. In the example above, *Author-Paper-Author meta-path* cannot cover *Venue* and *Topic* vertices.

1.4 Initiatives

Network Representation Learning is an emerging field of study with exciting applications. But current trend is more on the *effectiveness* of *NLR*, people may thereafter find below drawbacks

- Lack of efficient studies and implementation. Current open source codes for *NLR* are more for elaboration purpose and cannot handle large scale data efficiently. To boost computation, people may find better integration of *NLR*with real life applications.
- Lack of standard dataset format and collections. Researchers have their own preferences on dataset selection and the way to process it. Consequently, it is difficult to compare result in different papers and develop new algorithms. It is desirable to have a standard collection of data.

Therefore, *FAST2VEC* has been proposed and it will serve as an extensive and efficient collection of several *NLR* algorithms and developers may also use it to investigate more *NLR* algorithms.

2 Objectives

FAST2VEC will consist of four components: an efficient efficient *NLR* framework, a standard collection of dataset, a pipeline for common applications and benchmark report for current *NLR* algorithm under *FAST2VEC*. In this section, all these components will be presented.

2.1 Efficient NLR Framework

As stated in introduction section, performance and scalability is bottleneck for existing *NLR* tools. *FAST2VEC* will optimize for parallel computing and also leverage existing high performance computing library.

The framework will have both data processing module and training module. Data processing module will be responsible for prepare training data. It will draw various random walks for information network according to the selected algorithm. Training module will take in a collection of random walks and output trained model.

As stated in previous section, most *NLR* algorithms will share same training module and *random walker* can also be derived from other. Therefore, developers will not only enjoy an efficient implementation of existing *NLR* algorithms but also be able to fast prototype their on algorithms.

2.2 Collection of Datasets with Unified Format

Companies are providing increasing amount of open datasets which are of different format to best describe itself. Within *NLR* research community, their have been recognized dataset such as *Flickr*, *DBLP* and *BlogCatalog* but a standard format is still in absent. Researchers have processed and pruned data differently to fit their usage in papers which brings difficulties in comparing result from different papers.

To address this un-unified issue, *FAST2VEC* will also unify dataset format for *NLR* research. It will automatically download recognized data from public website and unify them to the proposed format. This makes result from *FAST2VEC* comparable which allows easy comparison and benchmark.

2.3 Pipeline for Common Application

As stated before, *NLR* has a broad range of applications and *FAST2VEC* will demonstrate two of them under its framework. The general workflow will be:

1. Download and pre-prossess data from the standard collection as described above.
2. Generate random walks according to the selected algorithms.
3. Train the model against generated random walks.
4. Feed downstream applications with trained model.

2.3.1 Multi-class Classification

It is a common requirement to categorize users within a social network to different user groups and researchers have further generalize this application to multi-class classification. For example, people may want to find a professor's research interest based on his or her publication records. Common classification algorithms like multi-class logistic regression ask for vector input so people can feed them with *NLR* result. To train the classifier, user will use both *NLR* result and vertex label file. With a trained model, if a user want to categorize a scholar then he or she will first index scholar's vector representation and then put it into classifier.

2.3.2 Link Prediction

It is another common requirement to make recommendation for a user and it can be regarded as a binary problem that between two vertices whether certain link should exist or not. Similar to the procedure above, pipe line for link prediction is as shown below

1. Train an *NLR* model.
2. Prepare training data for the binary classifier, entry of which is a pair of vertices share certain link. Prepare test data of same format too.
3. Train binary classifier with *NLR* model and pairs of vertices prepared above.
4. Test result against test data

2.4 Extensive Benchmark

Both *effectiveness* and *efficiency* will be benchmarked under *FAST2VEC* framework. First, performance comparison between *FAST2VEC* and other implementations will be conducted. Experiments will be carried out against different number of computation nodes, various combination of parameters and different datasets. Second, effectiveness of each algorithm will be tested on standard collection of data and applications under *FAST2VEC*. Objective of the second part of the experiment is to examine current state-of-art algorithms.

To present experiments result, an extensive experiment report will be written.

3 Methodology

3.1 Parallelization

To boost *NLR* computation, *FAST2VEC* will leverage various parallel computing techniques. All these features below will be implemented incrementally during the development.

3.1.1 Multi-Threading

Multi-threading computation on a single machine is the most approachable methods for parallel computing. *NLR* computation is trivially parallelize-able but it is up to developer's design on lock mechanism and scheduler. Open source library such as *OpenMP* and *Intel Thread Building Block* will be used.

3.1.2 Locality

Due to the nature of *NLR* algorithms, lots of random access will be carried out, which is not friendly to cache. To make use of locality, regroup of random walks and improvement of negative sampling will be studied. The objective is to speed up training by minimizing overhead for memory access.

3.1.3 Distributed Computing

One of the most cost effective way to increase computing power is to run on a cluster of computation nodes. However, current implementations of *NLR* mostly run only on a single machine. To meet performance requirement of real life application, it is desirable to introduce distributed computing techniques into implementation. Open source library *OpenMPI* will be used for message passing between different nodes.

3.1.4 GPU Computing

Profiling on existing implementations have shown that matrix computation contributes most of the work load and can be boosted by GPU. Open source library *CUDA* will be used.

3.2 Integration of Static Language and Scripting Language

Static languages like *C++* are of better performance at a price of flexibility and simplicity while vice versa for scripting languages. *FAST2VEC* will be implemented in various programming languages. The decision philosophy is to write performance critical code in static languages and write data processor and helper functions in scripting language. If time allowed, wrapper for performance critical code will be provided for calling from easy-to-use scripting language.

3.3 Data Driven Evaluation

FAST2VEC is not just about implementation, it is also a framework for benchmarking *NLR* algorithms. Therefore, it is exciting to use *FAST2VEC* to evaluate current methods. Both performance and effectiveness metric will be reported against collections of dataset and applications for each of the algorithms. Common metric like *macro-F1*, *NMI*, *micro-F1* will be recorded.

4 Schedule and Milestones

Schedule below is independent of standard final year project timetable and will leave a month of buffer time.

4.1 Single Machine Multi-Threaded Prototype

This is the *proof-of-concept (POC)* and prototype phase of the project. Objective of this phase is to verify implementations of *random-walk-based NLR* in depth and explore the bottleneck of current implementation. Deliverables of this phase includes

- Trivially parallelized *DeepWalk* and *Node2Vec* implementation
- Pipeline for both multi-class classification and link prediction
- Data Collection for *DBLP* and *ACM*

This phase should be completed on or before October 14th, 2018.

4.2 Locality Optimization

Deliverables of this phase is an improvement of *FAST2VEC* for locality and random access and should be completed on or before November 4th, 2018.

4.3 GPU Accelerated Computation Support

Deliverables of this phase is an improvement of *FAST2VEC* to accelerate computation with GPU and random access and should be completed on or before December 2nd, 2018.

4.4 Distributed Computing Support

Deliverables of this phase is an improvement of *FAST2VEC* to allow distributed training on cluster and should be completed on or before January 27th, 2019.

4.5 Data Collection

After implementation of *FAST2VEC*, more dataset will be on boarded and more experiments will be carried out and it should be completed on or before February 10th, 2019.

4.6 Evaluation and Report

Experiments will be conducted throughout all the phases of implementation and all the result will be recorded. A report of current *NLR* algorithms together with benchmark of *FAST2VEC* against current implementation will be written and this should be completed on or before March 10th, 2019.

References

[1] Marcel Beishuizen. "Structural graph learning in real-world digital forensics". In: (2018).

[2] Hongyun Cai, Vincent W Zheng, and Kevin Chang. "A comprehensive survey of graph embedding: problems, techniques and applications". In: *IEEE Transactions on Knowledge and Data Engineering* (2018).

[3] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. "metapath2vec: Scalable representation learning for heterogeneous networks". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM. 2017, pp. 135–144.

[4] Bradley Efron et al. "Least angle regression". In: *The Annals of statistics* 32.2 (2004), pp. 407–499.

[5] Gang Fu et al. "Predicting drug target interactions using meta-path-based semantic network analysis". In: *BMC bioinformatics* 17.1 (2016), p. 160.

[6] Tao-yang Fu, Wang-Chien Lee, and Zhen Lei. "HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management.* ACM. 2017, pp. 1797–1806.

[7] Palash Goyal and Emilio Ferrara. "Graph embedding techniques, applications, and performance: A survey". In: *Knowledge-Based Systems* 151 (2018), pp. 78–94.

[8] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM. 2016, pp. 855–864.

[9] William L Hamilton, Rex Ying, and Jure Leskovec. "Representation Learning on Graphs: Methods and Applications". In: *arXiv preprint arXiv:1709.05584* (2017).

[10] Binbin Hu et al. "Leveraging Meta-path based Context for Top-N Recommendation with A Neural Co-Attention Model". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* ACM. 2018, pp. 1531–1540.

[11] Zhipeng Huang et al. "Meta structure: Computing relevance in large heterogeneous information networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2016, pp. 1595–1604.

[12] Yann Jacob, Ludovic Denoyer, and Patrick Gallinari. "Learning latent representations of nodes for classifying in heterogeneous social networks". In: *Proceedings of the 7th ACM international conference on Web search and data mining*. ACM. 2014, pp. 373–382.

[13] Jure Leskovec and Andrej Krevl. "{SNAP Datasets}:{Stanford} Large Network Dataset Collection". In: (2015).

[14] Changping Meng et al. "Discovering meta-paths in large heterogeneous information networks". In: *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2015, pp. 754–764.

[15] Tomas Mikolov et al. "Distributed representations of words and phrases and their compositionality". In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[16] Mingdong Ou et al. "Asymmetric transitivity preserving graph embedding". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2016, pp. 1105–1114.

[17] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 701–710.

[18] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. "struc2vec: Learning node representations from structural identity". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2017, pp. 385–394.

[19] Yu Shi et al. "Easing Embedding Learning by Comprehensive Transcription of Heterogeneous Information Networks". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2018, pp. 2190–2199.

[20] Yizhou Sun et al. "Pathsim: Meta path-based top-k similarity search in heterogeneous information networks". In: *Proceedings of the VLDB Endowment* 4.11 (2011), pp. 992–1003.

[21] Jian Tang et al. "Line: Large-scale information network embedding". In: *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2015, pp. 1067–1077.

[22] Jie Tang et al. "Arnetminer: extraction and mining of academic social networks". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 990–998.

[23] S Vichy N Vishwanathan et al. "Graph kernels". In: *Journal of Machine Learning Research* 11.Apr (2010), pp. 1201–1242.

[24] Daixin Wang, Peng Cui, and Wenwu Zhu. "Structural deep network embedding". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2016, pp. 1225–1234.

[25] Chuxu Zhang et al. "Task-Guided and Semantic-Aware Ranking for Academic Author-Paper Correlation Inference." In: *IJCAI*. 2018, pp. 3641–3647.

[26] Daokun Zhang et al. "Network representation learning: A survey". In: *IEEE Transactions on Big Data* (2018).