HEBBIAN GRAPH EMBEDDINGS

Anonymous authors
Paper under double-blind review

ABSTRACT

Representation learning has recently been successfully used to create vector representations of entities in language learning, recommender systems and in similarity learning. Graph embeddings exploit the locality structure of a graph and generate embeddings for nodes which could be words in a language, products of a retail website; and the nodes are connected based on a context window. In this paper, we consider graph embeddings with an error-free associative learning update rule, which models the embedding vector of node as a non-convex Gaussian mixture of the embeddings of the nodes in its immediate vicinity with some constant variance that is reduced as iterations progress. It is very easy to parallelize our algorithm without any form of shared memory, which makes it possible to use it on very large graphs with a much higher dimensionality of the embeddings. We study the efficacy of proposed method on several benchmark data sets in Goyal & Ferrara (2018b) and favourably compare with state of the art methods. Further, proposed method is applied to generate relevant recommendations for a large retailer.

1 INTRODUCTION

Graph embeddings learn vector representations of nodes in a graph. [Cai et al. (2018)] and [Goyal & Ferrara (2018b)] give a comprehensive survey of graph embedding methods like node2vec [Grover & Leskovec (2016)] and also deep convolutional embeddings.

Our method uses error-free associative learning to learn the embeddings on graphs. The algorithm is quite simple, but very effective. We apply the learnt embeddings to the task of recommending items to users and to the task of link prediction and reconstruction.

Label propagation and message passing have been applied to many tasks like feature propagation [Heaukulani & Ghahramani (2013)], interest propagation, propagation of information in a population [Rapoport (1953)] and other network models of behavior like PageRank [Page et al. (1999)] and models of text like TextRank [Mihalcea & Tarau (2004)]. Instead of propagating a single unit of information, we propagate entire embeddings across the network. By propagating information on a graph iteratively, long distance similarities can also be learnt.

A recent paper [Nickel & Kiela (2017)] uses Hyperbolic geometry to construct embeddings in hierarchies and graphs. Their results show that on hierarchies and graphs, hyperbolic embeddings of a much smaller dimension can outperform Euclidean embeddings. We use some of the data sets which [Nickel & Kiela (2017)] and [Goyal & Ferrara (2018b)] use (it is difficult to say whether our results are comparable to [Nickel & Kiela (2017)] because of the difference in the way the test set is sampled before computing the mean average precision). For link prediction and reconstruction, our results are directly comparable to [Goyal & Ferrara (2018b)].

2 HEBBIAN GRAPH EMBEDDINGS

Hebbian learning is the simplest form of learning invented by Donald Hebb in 1949 in his book The organization of behavior [Hebb (1949)]. It is inspired by dynamics of biological systems. A synapse between two neurons is strengthened when the neurons on either side of the synapse (input and output) have highly correlated outputs. In essence, when an input neuron fires, if it frequently leads to the firing of the output neuron, the synapse is strengthened. In simple terms: ”neurons that fire together wire together” [Hebb (1949)]. Recently, there’s renewed interest in Hebbian learning. [Keysers & Gazzola (2014)] postulates that Hebbian learning predicts mirror-like neurons for

sensations and emotions. [Treur (2016)] applies Hebbian learning in modelling of temporal-causal network.

Hebbian learning consists of a parameter update rule which is based on the strength of connection between two nodes, as applied to neural networks (based on firing tendencies of neurons on the opposite ends of a synapse). We extend the idea to graphs. Based on a pre-computed transition probability between two nodes of a product graph, we update the parameters (the embeddings of a node) iteratively based on an error-free associative learning rule (nodes that are contextually connected should have similar embeddings, like word2vec for words [Mikolov et al. (2013)]). For a discussion on errorless learning, please see [McClelland (2006)].

We first initialize all embeddings to a multivariate normal distribution with mean 0 and variance σ^2.

$$j \sim N(0, \sigma^2 I) \tag{1}$$

We model the embedding at a node as a non-convex Gaussian mixture of the embeddings of the connected nodes. If there is an edge from node i to node j, the embedding of node j is modeled as follows:

$$j \sim N(i, \sigma^2 I) \tag{2}$$

The variance σ^2 starts off at a value of 10 and is divided by 1.1 every iteration in the spirit of simulated annealing [Kirkpatrick et al. (1983)]. The embedding of node j is updated as follows:

$$\delta_j = \sum_i (N(i, \sigma^2 I) * p_{ij} * \eta) \tag{3}$$

The δ_j are then simply added to the embedding at node j (where there is an edge from node i to node j). p_{ij} is the transition probability and η is the learning rate. The graph is weighted, asymmetric and undirected. Also, a random negative edge is selected at each node and the negative of the embeddings is propagated to both selected nodes with a small transition probability. This iterative procedure learns the embeddings of all nodes in the graph and is able to generate very effective embeddings, as the next section shows. As shown in figure 1, the embeddings get propagated across the graph through a Gaussian hierarchy across connected components of nodes.

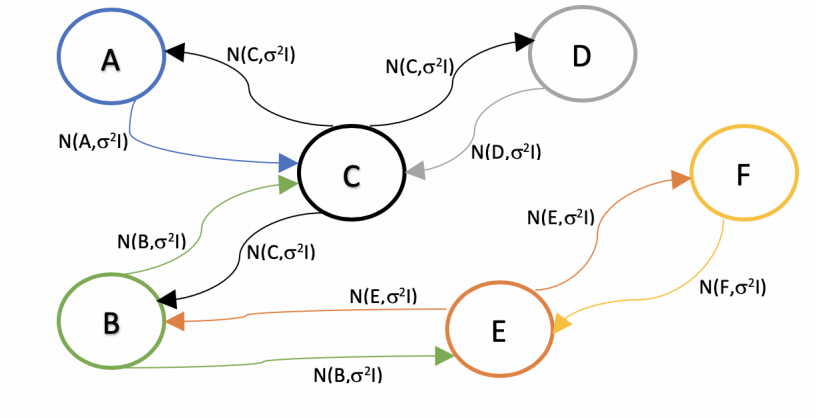

Figure 1: Propagation of Embeddings Across a Graph

Algorithm 1 Hebbian Graph Embeddings

procedure FINDEMBEDDINGS(G)
  Inputs: Weighted, asymmetric and undirected graph with nodes as products $(1, 2 \ldots, P)$ and edge weights as transition probabilities between products p_{ij}
  Hyper-parameters:
  σ^2 Variance of normal distribution
  N Number of iterations of Hebbian learning
  K Dimensionality of node representation
  τ Variance reduction factor
  Initialization: Initialize the nodes representation w_i by sampling from a zero mean multivariate normal distribution $N(0, \sigma^2 I)$ of dimensionality K
  for each integer m in N **do**
    for each node i in P **do**
      for each node j in $Adj(i)$ **do**

$$\tilde{\mathbf{w}}_i \sim N(\mathbf{w_i}, \boldsymbol{\sigma^2 \mathbf{I}}) \tag{4}$$

$$\mathbf{w_j} \leftarrow \mathbf{w_j} + \eta \tilde{\mathbf{w}}_i p_{ij} \tag{5}$$

      end for
    end for

$$\boldsymbol{\sigma^2} \leftarrow \boldsymbol{\sigma^2}/\tau \tag{6}$$

  end for
end procedure

Table 1: Mean Average Precision (MAP) results for network embeddings for Reconstruction of the entire graph

DataSet	Nodes	Edges	Reconstruction Results of Varying Dimensionality							
			10	20	50	100	200	300	400	500
CondMat	23,133	93,497	0.192	0.304	0.495	0.649	0.778	0.838	0.873	0.895
GrQc	5,242	14,496	0.245	0.407	0.625	0.763	0.860	0.894	0.910	0.918
HepPh	12,008	118,521	0.196	0.293	0.455	0.586	0.698	0.755	0.789	0.814
AstroPh	18,772	198,110	0.181	0.245	0.362	0.461	0.573	0.635	0.675	0.707
HepTh	27,770	352,807	0.188	0.261	0.402	0.509	0.619	0.679	0.709	0.732
BlogCatalog	88,784	4,186,390	0.438	0.503	0.584	0.646	0.704	0.735	0.753	0.763

3 EXPERIMENTS AND RESULTS

3.1 RESULTS ON RECONSTRUCTION

We ran our algorithm for reconstruction on publicly available data sets. Reconstruction tries to reconstruct the entire original graph (without splitting into train/test). As in [Goyal & Ferrara (2018b)], we sample 1024 nodes for calculation of the MAP. We run the algorithm for 10 iterations with a learning rate 1.0. The results in table 1, table 2 and figure 2 show that our algorithm is able to achieve good results on reconstruction when the dimensionality is large. As benchmarks, we use three data sets that [Goyal & Ferrara (2018b)] uses for reconstruction. Our results are favorably comparable on those three data sets. The other data sets are not used by [Goyal & Ferrara (2018b)] but the supporting code base as in [Goyal & Ferrara (2018a)] can be used to compare.

3.2 RESULTS ON THE RECOMMENDER SYSTEM OF A LARGE RETAILER

Also, in the recommender system at a large retailer, we used a sample of 200 thousand items as our population for training and measurement. 10% of the users are held out as the test set. The number of nodes in the graph is 200,000 and the number of edges is about 13.1 billion (note that the weight of an incoming edge might be different from an outgoing edge between any two nodes).

Table 2: Random Mean Average Precision (MAP) results (no training) for network embeddings for Reconstruction

DataSet	Nodes	Edges	Random (no training)
			500 (Dimension)
CondMat	23,133	93,497	0.0139
GrQc	5,242	14,496	0.0126
HepPh	12,008	118,521	0.0233
AstroPh	18,772	198,110	0.0255
HepTh	27,770	352,807	0.0292
BlogCatalog	88,784	4,186,390	0.0364

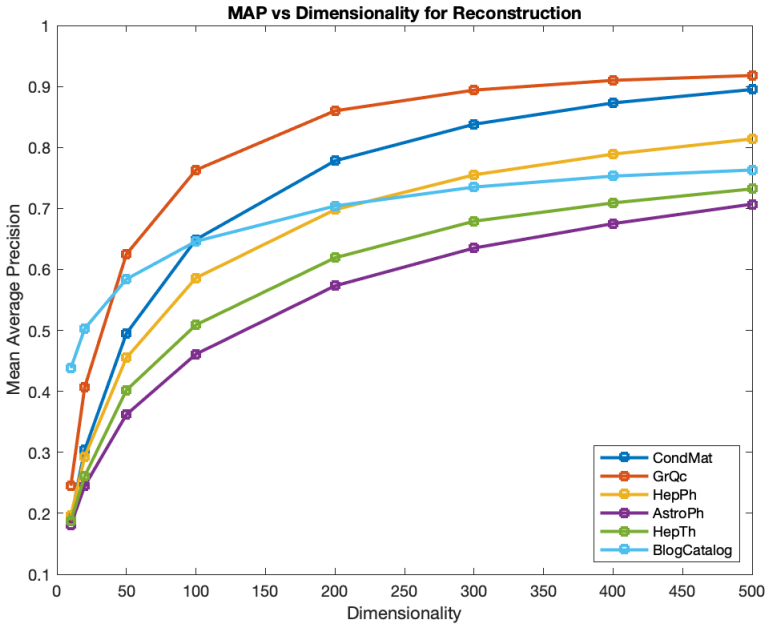

Figure 2: Mean Average Precision for Reconstruction with Varying Dimensionality.

We measure the performance of our algorithm on the hit rate. Top 10 recommendations are generated per item based on the nearest neighbors of the generated embeddings based on an inner product (using all 200,000 items). Then, one random item from the users entire interaction history is chosen. Recommendations for this random item are computed. If any of the top 10 recommended items (other than the seed item) also occurs in the users interaction history, it is considered a hit. Otherwise a fail. The average hit rate is then the number of successes divided by the number of users in the test set. Results are shown in table 3. We use 10 iterations and a learning rate of 1.0.

3.3 Results on Link Prediction

For link prediction, we use some of the data sets used in [Nickel & Kiela (2017)] and [Goyal & Ferrara (2018b)]. As in [Goyal & Ferrara (2018b)], we sample 1024 nodes for calculation of the MAP. We keep 10% of the edges as a held out test set. We run the algorithm for 10 iterations with a learning rate 1.0. The results in table 4, table 5 and figure 3 show that our algorithm is able to achieve good results on link prediction when the dimensionality is large. (It is not completely clear on how [Nickel & Kiela (2017)] samples the test and validation sets). As benchmarks, we use three data sets that [Goyal & Ferrara (2018b)] uses for link prediction. Our results are favorably comparable on those three data sets for link prediction. [Goyal & Ferrara (2018b)] also has a supporting code base [Goyal & Ferrara (2018a)] which can be used to compare on other data sets.

Table 3: Results on a very large graph for recommender systems at a large retailer

Dimensionality	**HitRate@10**
100	24.2%
200	30.1%
250	31.1%

Table 4: Mean Average Precision (MAP) results for network embeddings for Link Prediction (10% randomly chosen edges are held out as the test set)

DataSet	**Nodes**	**Edges**	**Link Prediction Results for Varying Dimensionality**							
			10	20	50	100	200	300	400	500
CondMat	23,133	93,497	0.070	0.130	0.251	0.350	0.450	0.507	0.531	0.544
GrQc	5,242	14,496	0.064	0.129	0.233	0.292	0.332	0.348	0.363	0.383
HepPh	12,008	118,521	0.065	0.121	0.213	0.289	0.346	0.384	0.401	0.424
AstroPh	18,772	198,110	0.060	0.092	0.179	0.235	0.317	0.357	0.388	0.409
HepTh	27,770	352,807	0.070	0.120	0.203	0.259	0.339	0.370	0.383	0.407
BlogCatalog	88,784	4,186,390	0.199	0.252	0.286	0.322	0.353	0.367	0.383	0.396

It is quite easy to parallelize the algorithm, and we implement it on Apache Spark. We run the algorithm for 10 iterations (which takes about 3 hours on the parallel implementation on recommender system data and from 5 minutes to 2 hours (depending on the dimensionality) on the publicly available data). We found that the learning rate does not affect the results in any significant way (we use 1.0).

4 Conclusion

In this paper, we described a simple, but very effective algorithm to learn the embeddings on a graph. The results show that the algorithm, as applied to the tasks of link prediction and reconstruction, is able to perform well when the dimensionality of the embeddings is large. This shows the effectiveness of learning on graphs using iterative methods. Its a useful experiment of error-free (errorless) learning on graphs. Our method can learn long distance similarities because of the iterative nature of the algorithm which percolates the embeddings on the weighted graph.

A distinctive advantage of our approach is that it is very easy to parallelize the algorithm without any need for shared memory. It is quite easy to implement the algorithm on platforms like Apache Spark, which makes the algorithm amenable to very large graphs which cannot be processed on one machine.

Our recommender system work was tested live and it did very well. But because our item graph has a very large number of nodes and edges, we omit the implementation of [Nickel & Kiela (2017)] and [Goyal & Ferrara (2018b)] for our recommender system.

Other algorithms like in [Vinh et al. (2018)] and [Chamberlain et al. (2019)] could be compared with our work. There is still an opportunity to improve the algorithm through hyperparameter tuning. It might be interesting to measure the algorithm with a much higher dimensionality of the embeddings.

References

Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

Benjamin Paul Chamberlain, Stephen R Hardwick, David R Wardrope, Fabon Dzogang, Fabio Daolio, and Saúl Vargas. Scalable hyperbolic recommender systems. *arXiv preprint arXiv:1902.08648*, 2019.

Table 5: Random Mean Average Precision (MAP) results (no training) for network embeddings for Link Prediction

DataSet	Nodes	Edges	Random (no training)
			500 (Dimension)
CondMat	23,133	93,497	0.007
GrQc	5,242	14,496	0.007
HepPh	12,008	118,521	0.010
AstroPh	18,772	198,110	0.009
HepTh	27,770	352,807	0.009
BlogCatalog	88,784	4,186,390	0.014

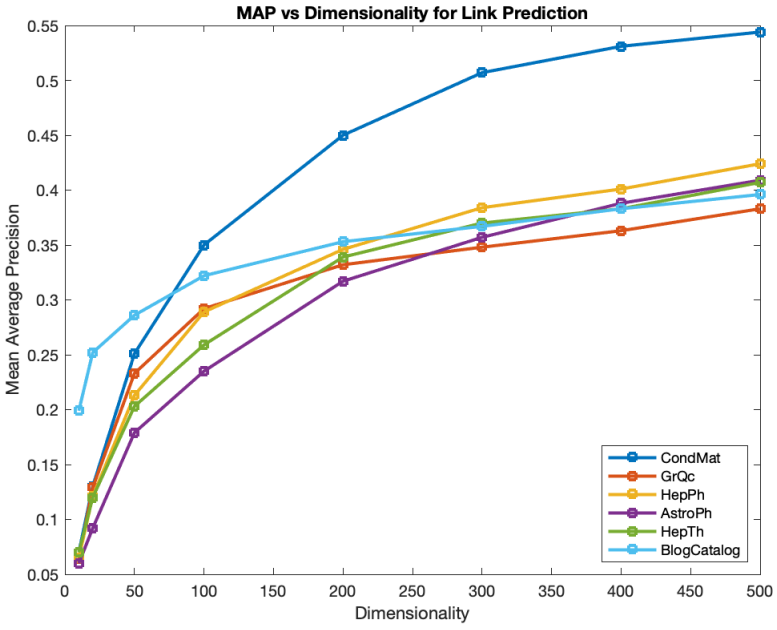

Figure 3: Mean Average Precision for Link Prediction with Varying Dimensionality.

Palash Goyal and Emilio Ferrara. Gem: A python package for graph embedding methods. *J. Open Source Software*, 3(29):876, 2018a.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018b.

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*, pp. 855–864. ACM, 2016.

Creighton Heaukulani and Zoubin Ghahramani. Dynamic probabilistic models for latent feature propagation in social networks. In *International Conference on Machine Learning*, pp. 275–283, 2013.

Donald Olding Hebb. *The Organization of Behavior*. Wiley & Sons, 1949.

Christian Keysers and Valeria Gazzola. Hebbian learning and predictive mirror neurons for actions, sensations and emotions. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1644):20130175, 2014.

Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

James L McClelland. How far can you go with hebbian learning, and when does it lead you astray? *Processes of change in brain and cognitive development: Attention and performance xxi*, 21: 33–69, 2006.

Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pp. 404–411, 2004.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pp. 6338–6347, 2017.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

Anatol Rapoport. Spread of information through a population with socio-structural bias: I. assumption of transitivity. *The bulletin of mathematical biophysics*, 15(4):523–533, 1953.

Jan Treur. *Network-oriented modeling*. Springer, 2016.

Tran Dang Quang Vinh, Yi Tay, Shuai Zhang, Gao Cong, and Xiao-Li Li. Hyperbolic recommender systems. *arXiv preprint arXiv:1809.01703*, 2018.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 5165–5175, 2018.