Using GraphSAGE to improve Link Prediction in Knowledge Bases

Neeraj Katiyar
McGill University
neeraj.katiyar@mail.mcgill.ca

Abstract

Knowledge graphs empower a wide variety of applications, including information retrieval and advice chatbots. Despite great efforts invested in their creation and maintenance, even some of the largest (e.g Yago, DBPedia or Wikidata) remain incomplete. GraphSAGE (SAmple and aggreGatE) by (Hamilton et al., 2017) is a recent general inductive framework that leverages node feature information (e.g., text attributes) to efficiently generate node embeddings for previously unseen data. Instead of training individual embeddings for each node, GraphSAGE learns a function that generates embeddings by sampling and aggregating features from a node's local neighbourhood. We hypothesised that using GraphSAGE in tasks involving large knowledge graphs would improve performance. To test this, we applied the GraphSAGE algorithm for the task of link prediction in large knowledge graphs (specifically, FB15K-237 by (Toutanova et al., 2015)) and compared the results with previous work of (Schlichtkrull et al., 2017).

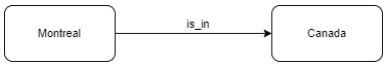
1 Introduction

Knowledge Graphs and R-GCNs

A knowledge graph, also known as a semantic network, represents a network of real-world entities—i.e. objects, events, situations, or concepts—and illustrates the relationship between them. In this work, we represent a graph as a directed multi-graph:

$$G = (V, E, R) \tag{1}$$

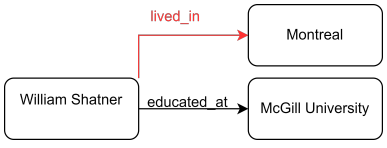
where V is the set of entities, E is the set of edges and R is a relation type. For example, a singular relationship/link between entities can be represented as:



where *Montreal, Canada* are entities and *is_in* is a relationship.

Knowledge graphs have important applications in search, analytics, recommendation, and data integration – however, they tend to suffer from incompleteness, i.e., missing links in the graph. Facts sizes, usually in millions, in knowledge graphs make such link predictions crucial and should scale in a manageable way with respect to both the number of parameters and computational costs to be applicable in real-world scenarios. Predicting missing information in knowledge bases is the main focus of statistical relational learning (SRL). In this work, we deal with the problem of predicting links (i.e. recovery of missing facts in the form of subject-predicate-object triplets) where many missing pieces of information can be expected to reside within the graph encoded through neighbourhood structure.

Following the previous work on SRL and R-GCN by (Schlichtkrull et al., 2017) our extended experimental focus is on one fundamental SRL task: link prediction (recovery of missing triples-subject, predicate and object). For example, it should be possible to predict the red link (provided there exists a relation between *McGill University* and *Montreal*):



Mathematically, if the knowledge base is represented by $G = (V, E, R)$, we are only given an incomplete subset e instead of the full set E. The task is to assign scores $f(s, r, o)$ to possible edges (s, r, o) in order to determine how likely those edges are to belong to E.

GraphSAGE

Recently, (Hamilton et al., 2017) introduced the GraphSAGE algorithm that leverages node features (e.g., text attributes, node profile information, node degrees) in order to learn an embedding function that generalizes to unseen nodes:

Algorithm 1 GraphSAGE

INPUT: Graph $G(V, E)$ where V are vertices and E edges; per-node input features $\{x_v, \forall_v \epsilon V\}$; depth K; per-depth weight matrices $W^k \forall k \epsilon \{1, ..., K\}$; per-depth non-linearity $\sigma_k, \forall k \epsilon \{1, ...K\}$; differentiable aggregation functions $AG_k, \forall k \epsilon \{1, ...K\}$; neighborhood function $N : v \rightarrow 2^V$
OUTPUT: Embedding representations $z_v \forall v \epsilon V$
$h_v^0 \leftarrow x_v, \forall v \epsilon V$
for $k = 1...K$ **do**
  for $v \epsilon V$ **do**
    $h_{N(v)}^k \leftarrow AG_k(\{h_u^{k-1}, \forall u \epsilon N(v)\})$
    $h_v^k \leftarrow \sigma_k(W_k.CNCT(h_v^{k-1}, h_{N(v)}^k))$
  end for
  $h_v^k \leftarrow h_v^k / ||h_v^k||_2, \forall v \epsilon V$
end for
$z_v \leftarrow h_v^K, \forall v \epsilon V$

Where CNCT stands for concatenation operation.

We hypothesized that these embeddings, when applied to link prediction tasks, should improve performance. To test this, we based our work on the RelationPrediction codebase provided by Schlichtkrull et al. Depending on time and ease of implementation, we planned to compare the training accuracy and convergence time for various combinations of datasets (FB15k-237, WN18). In essence we wanted to replace the embeddings generated by R-GCN with those generated by running GraphSAGE on the dataset. In terms of our contributions, to the best of our knowledge, ours is the first academic work that uses the GraphSAGE algorithm for enhancing link prediction tasks.

2 Related work

Our experiment draws inspiration both from the field of graph convolutional network, more specifically from the recent work by (Schlichtkrull et al., 2017) on R-GCN and (Hamilton et al., 2017) on GraphSAGE frameworks that operate on graphs. In what follows, we provide a brief overview on related work in the mentioned fields.

- (Kipf and Welling, 2017) The convolutional architecture presented by the author in the paper is one of the most popular works that introduced Graph Convolutional Networks (GCN), which propose a scalable approach for semi-supervised learning on graph-structured data based on a variant of convolutional neural networks that operates directly on graphs. The work contributed in this paper was considered as one of the base for novel research presented in (Schlichtkrull et al., 2017).

- (Schlichtkrull et al., 2017) Our work builds on this paper, where authors have introduced Relational GCNs for relational data. RGCNs, related to a recent class of neural networks operating on graphs, are developed specifically to deal with the highly multi-relational data characteristic of realistic knowledge bases. Our work is the extended experiment of one of the statistical relational learning (SRL) task: recovery of missing triples. We have adapted the link prediction model discussed in the paper with an R-GCN encoder and a DistMult decoder for our experiment.

- (Hamilton et al., 2017) As described in the introduction section, this paper proposes a method (graphSAGE) to overcome the need to have all nodes in the graph present during embedding training time. Their main idea is that a node can be represented as a feature vector made by aggregating features from adjacent nodes. Thus, when an unseen node is added to the graph, this method can generalize to unseen nodes.

3 MethodX

Replicating the algorithms for baselines

Since we only considered only the task of relation prediction, which deals with prediction of new facts (i.e. triplets of subject, relation, object), we used the provided RelationPrediction GitHub code as a starting point. This was written with Tensorflow v1.4 and hence was not compatible with the latest Tensorflow installation. We then migrated the code to Tensorflow v2.0 as Tensorflow v1.4 was not available in any of the public distribution packages (anaconda / pip) and fixed any associated errors during the process. Finally, we verified that there were no significant difference in the results from Schlichtkrull et al.'s reported results before

starting the next task of generating the embeddings for every node using the GraphSAGE algorithm.

Implementing GraphSAGE

We set out with the goal of generating GraphSAGE embeddings, which we would then plug into the RelationPrediction pipeline. We explored two approaches for doing this:

- **Generate GraphSAGE embeddings separately**: Hamilton et. al. provided a sample implementation of their algorithm in PyTorch. We considered running the PyTorch implementation on our datasets, storing the embeddings to file and then loading that from within the relation prediction pipeline. Implementing the algorithm would have been a straightforward effort here but integrating the embeddings into the RelationPrediction pipeline would be tricky since it would need an extra effort to maintain the metadata of the dataset and to maintain the tensors in the relevant shape.

- **Generate GraphSAGE embeddings within the RelationPrediction pipeline**: With this approach, we wanted to explore how RelationPrediction represents graphs and apply GraphSAGE directly to this format of data. Note that RelationPrediction does not use a standard library such as NetworkX, rather they use their own custom format for representing the graphs. With this method, the integration process would be straightforward but finding and understanding Schlichtkrull et al.'s custom format and implementing the GraphSAGE algorithm in the pipeline would be tricky.

We tried both these approaches and found that generating GraphSAGE embeddings within the RelationPrediction pipeline would overall result in a better and easily extensible pipeline for the link prediction task. After attempting to implement GraphSAGE on our own, we found StellarGraph, which is an open-source implementation of GraphSAGE. More crucially, StellarGraph has methods to take raw numpy arrays as input.
Our implementation can be found in the file *graphsage.py* in our fork of the repository. The GraphSAGE model parameters we used for gathering results are as follows (please also refer to Algorithm 1 on page 2):

- **Depth = 2** Parameter K in Algorithm 1. Can also be thought of as the number of hops around a node that GS looks at. This also constraints the GS network to be 2 layers deep.

- **Number of samples for {hop 1, hop 2} = {20, 10}** The number of adjacent nodes looked at for this hop. Analogous to parameter in neighborhood function N.

- **Aggregrator = Mean** Aggregration method used in step 6 of the algorithm. It is important for this to be *order invariant*.

- **Activation for {layer 1, layer 2} = {ReLU, Softmax}** The per-layer (or per-depth) nonlinearity σ_k used in step 7.

- **Normalization = L2 norm** Normalization term used in step 9.

- **Minibatch size = 20** StellarGraph implements a modification of basic GraphSAGE called minibatch, where the main idea is to sample all the nodes needed for computation first in order to allow forward/backward passes using stochastic gradient descent.

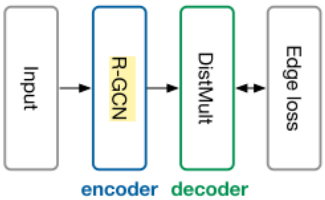
We also defined a custom node feature embedding to pass as input to GS. For each node $n \epsilon N$ (N is the set of all nodes), we defined a one hot vector:

$$V_n = \left[E_1 E_2 E_3 ... E_i\right]$$

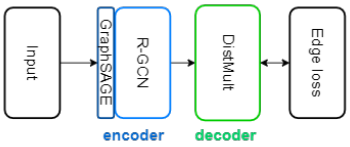
where i ranges from 0 to number of edges/relations in the dataset. An entry E_i is set to 1 if the node contains edge i.

Pipeline architecture

This is the pipeline architecture of the link prediction task from Schlichtkrull et al. that we modified:



We replaced the first layer of the encoder, R-GCN, with our model which initialises the nodes with the weights calculated using the GraphSAGE algorithm:



After examining the input/output shape of first layer of the R-GCN module, we replaced it with our GraphSAGE class that with the same shapes for input and output.

Hardware

We used tensorflow-gpu (CUDA) on two laptops with the following specs.

- Intel Core i7, NVIDIA RTX 2060
- Intel Core i7, NVIDIA Quadro P2000

Experiments

Our goal was to evaluate if using GraphSAGE embeddings provides an improvement over earlier methods used by Schlichtkrull et al., across multiple datasets (FB15-237k, WN18) using both weight regularization techniques described in the paper (*basis*-decomposition and *block-diagonal*-decomposition). We planned to compare the loss and Mean Reciprocal Rank (MRR) for each of our experiments, similar to the evaluation metric used by Schlichtkrull et al. We also wanted to evaluate if using GraphSAGE embeddings could provide faster convergence.

4 Results

Unfortunately, when we ran our experiments, our assumption about having enough compute hardware turned out to be incorrect. We had assumed that changing the graph batch size and the embedding size would be sufficient to run on our hardware. However, for each test, on both laptops, the program ran out of GPU memory after around 2000 iterations. We believe the graph stored in memory eventually grows too large regardless of the graph batch size. Due of this, we were not able to perform meaningful experiments on WN18 since the dataset itself is of larger size.

While we were unable to calculate the final MRR at convergence for FB15K-237 dataset either, we were able to gather promising data until about 2000 iterations (Table 1). Logs for these experiments on FB15k-237 are located in the folder comp_550_logs. The figures 1, 2 and 3 show reported loss v/s iteration count for each experiment.

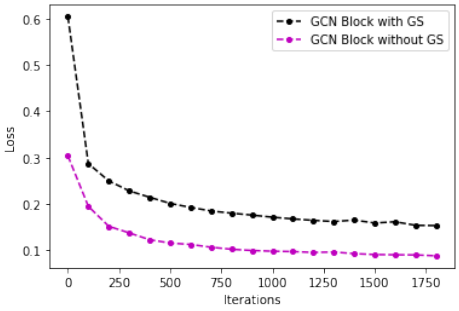


Figure 1: Loss for GCN block with and without GS

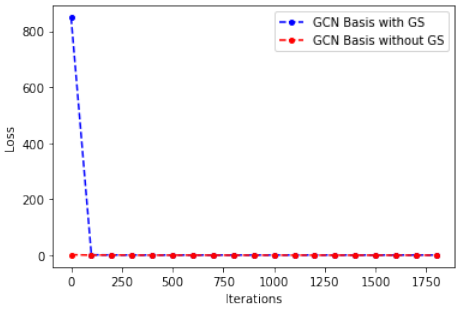


Figure 2: Loss for GCN basis with and without GS

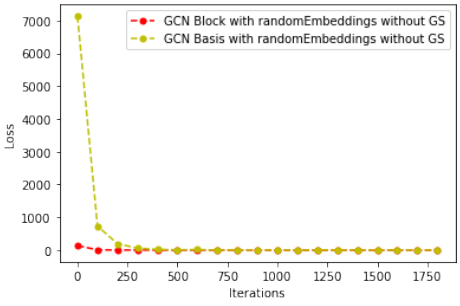


Figure 3: Loss for GCN random with block and basis

5 Discussion and Conclusion

We observed that the technique used in Schlichtkrull et al.'s work initialised embeddings for nodes in multiple ways, one of them being a random generation. Based on our understanding of the GraphSAGE algorithm, we theorised that it is possible to generate *more* meaningful embeddings than random for these nodes. The reason for choosing the GraphSAGE algorithm for the task of generating the initial embeddings was because the algorithm works well for dynamic graphs and is able to generalize well even for the nodes that are not seen by the training data. While the existing R-GCN architecture needs to observe all the nodes and the edges of the graph to accurately predict the relation between

Decomposition	FB15k-237		
	Without GraphSAGE	With Random Embeddings	With GraphSAGE
Basis	0.1695	0.7015	0.6989
Block	0.0884	0.2933	0.1536

Table 1: Experiment Results: Training Loss at 1900 iterations

the node, the GraphSAGE algorithm starts with understanding the link relations for only one or two hops from the existing node before it accurately models the links between the nodes. We wanted to observe if augmenting the initial embeddings with GraphSAGE enables faster convergence.

Based on our results (Table 1), we observe that initialising the relation prediction task with embeddings generated from GraphSAGE indeed gives better results when compared to random embeddings, even when there is no hyperparameter tuning done for the GraphSAGE model or any adjustements done to the relation prediction pipeline. While this remains to be validated against other datasets, which we were unable to perform due to hardware constraints, we believe these results could enable some future work where inductive algorithms can be used to augment the relation prediction task.

The GraphSAGE algorithm requires the node features to be initialized so that it can learn the embeddings using aggregation functions. Since the FB15k-237 dataset that we used does not contain node features, we initialised these as one-hot vectors as detailed in section 3. Reflecting on this choice, one other way of initialising the vector could be to group the nodes based on the relation data that is mentioned in the FB15k-Toutanova dataset, which might have captured the node properties in a better way.

We were unable to conclusively test our hypothesis that using GraphSAGE might allow faster convergence since we faced out of memory issues. Based on the initial loss, we can assume that the existing plugged in GraphSAGE implementation could converge faster than one initialized with random embeddings but poorly when compared to the original relation prediction technique outlined in Schlichtkrull et al. However, we admit that this is just a speculation and extensive empirical results would help us back this theory.

References

William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2017. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Lisbon, Portugal. Association for Computational Linguistics.