Yoram Louzoun's lab

RGCN Report

Knowledge graph analysis based on Kipf and Welling's Relational GCN method

Ori Cohen

5/2/2019

1 Table of contents

2 Introduction

2.1 Knowledge graphs

In trying to understand the world around us, specifically with the use of machine learning, we must find a way to represent both the objects of our study and their relationships with each other in a way that allows us both to describe as much of the information as possible and to analyze it properly.

One such representation, which has proven to be useful in many cases, is that of a knowledge graph. Simply put, a knowledge graph is a multigraph with both nodes and edges with different types. Such a graph allows us to represent multiple different entities with various relations between them in one concise structure.

Formally, a knowledge graph is a triplet $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ with nodes (entities) $v_i \in \mathcal{V}$ and labeled edges (relations) $(v_i, t, v_j) \in \mathcal{E}$, where $r \in \mathcal{R}$ is a relation type. It should be noted that $\mathcal{R}$ contains relations both in canonical directions (e.g. *born_in*) and the inverse direction (e.g. *born_in_inv*)[1].

2.2 Relational GCN

On way to analyze such networks is through the use of Kipf and Welling's GCN. Graph convolutions were shown to be effective tools when analyzing regular networks, and it comes as no surprise that such a tool, with minor modifications, could be used for knowledge graphs.

In their paper "Modeling Relational Data with Graph Convolutional Networks"[2], Kipf, Welling, and several others describe the application of the GCN to such graphs. The idea behind their modifications is simple enough – we just need to apply the GCN for every relation type. Such a rule allows us to propagate information through all relations and use them to produce an embedding for each of the nodes.

The paper further describes ways to regularize such a network, due to the fact that all the operations contains sparse matrices with large dimensions, which causes the network to quickly overfit.

Based on this work, we would like to be able to analyze our own networks with the intent of classifying entire graphs, and not only nodes in it.

[1] RGCN paper, page 2

[2] https://arxiv.org/pdf/1703.06103.pdf

3 Existing literature

This review of existing research focuses mainly on papers that worked on standard datasets and are connected to the RGCN paper. Further research could be found by searching for relevant keywords such as "knowledge graph", " knowledge graph embedding", " knowledge graph node classification", " knowledge graph link prediction" and so on.

3.1 Datasets for RGCN

Dataset	Paper	Description	Link	Notes
AIFB	1	Staff, research and publications of the AIFB institute	https://github.com/tkipf/relational-gcn/tree/master/rgcn/data/aifb	
MUTAG	1	Information about complex molecules that are potentially carcinogenic.	https://github.com/tkipf/relational-gcn/tree/master/rgcn/data/mutag	An example set of the DL-learner toolkit
British Geological Service	1	The dataset contains around 150 named rock units with a lithogenesis.	https://github.com/tkipf/relational-gcn/tree/master/rgcn/data/bgs	
Amsterdam Museum	1	information about artifacts in the Amsterdam Museum with production, material and content.	https://github.com/tkipf/relational-gcn/tree/master/rgcn/data/am	
Wordnet	2	A lexical database for English	https://wordnet.princeton.edu/related-projects	
Freebase	2	A knowledge graph of facts	https://developers.google.com/freebase/	The paper takes two subsets: 1) Entities which are also present in WikiLinks[1] 2) Most freqent 1 million entities
Wikidata	2*	A knowledge base of structured data	https://www.wikidata.org/wiki/Wikidata:Main_Page	Not directly mentioned, however freebase was deprecated in it's favor.
CMU NELL	3	A database of facts learned by a program reading the web	http://rtw.ml.cmu.edu/rtw/resources	
ClueWeb 09 related data	4	Entity annotation for web track queries	http://lemurproject.org/clueweb09/FACC1/	Based on the ClueWeb09 dataset: https://lemurproject.org/clueweb09.php/
Google knowledge graph	4	KG of entities in the google search engine	https://developers.google.com/knowledge-graph/	

DBkWik		KG from wiki links	http://dbkwik.webdatacommons.org/	
WebIsALOD		An "Is a" relation dataset from the web	http://webisa.webdatacommons.org/	

3.2 Papers

Index	Name	Link	Description	Note
1	Modeling Relational Data with Graph Convolutional Networks	https://arxiv.org/pdf/1703.06103.pdf	Entity classification and link prediction using RGCN	Kipf and Welling
2	Translating Embeddings for Modeling Multi-relational Data	https://www.utc.fr/~bordesan/dokuwiki/_media/en/transe_nips13.pdf	Embedding a multigraph to a low-dimensional vector	
3	Typed Tensor Decomposition of Knowledge Bases for Relation Extraction	https://pdfs.semanticscholar.org/2a5d/9bfc8b1eb7cc19b611b17b81d6ce97f056ca.pdf	Relation extraction between entities	
4	Entity Query Feature Expansion using Knowledge Base Links	http://maroo.cs.umass.edu/getpdf.php?id=1143	Use KG to build better features for queries.	
5	A Review of Relational Machine Learning for Knowledge Graphs	https://ieeexplore.ieee.org/document/7358050	An overview of existing techniques for knowledge graphs.	

4 Setup

The code can be cloned using Git from the GitHub page: https://github.com/oricc/relational-gcn.

Running the code requires installing several libraries. The recommended way to do this is using the Conda package manager inside an environment.

The libraries required are as follows:

- Tensorflow (version 1.12 was used)
- Keras (version 2.2.4 was used)
- Pandas
- Rdflib
- Networkx
- Numpy
- Scipy

Additionally, for those using Pycharm, both the *graph_measures* and the *rgcn* directories should be marked as source roots. For running on the command line, the directories should be added to the sys path (using the sys module), examples of this can be found in the top of several files.

5 Code overview

This section is intended for whomever continues the work on this code in the future. The section aims to describe the general purpose of the various code sections and should help navigate the code.

The code was originally forked from Kipf's repository and can be found at https://github.com/oricc/relational-gcn.

5.1 File structure

The code specific for this project is contained in the *rgcn* directory, which itself has several sub-directories:

1) Data – contains the original datasets used by the RGCN paper
2) Layers – contains the RGCN layers used in the models
3) Models – contains model objects (more on them below) which are used to load the data, build the RGCN model and train it.
4) Utils – scripts with various functions, mainly used for data preprocessing and file I\O.

5.2 Model objects

The model objects are the main objects used for the training of the RGCN. The model objects contain three main functions:

1) *_get_data* – the method loads the relevant data from a file\files into the class variables.
2) *_build_model* – the method describes the structure of the model we will train, including which layers to use, their sizes, etc. The function is expected to return a Keras *model* object.
3) *Train* – the method used to run the training loop itself.

The constructor runs both *_get_data* and *_build_model*, guaranteeing that both will run before the *train* method can be called.

When writing new Model objects, you should inherit from the Model object that most closely resembles the one you intend to write, which ensures that you only write the minimal modifications you require.

Grids of different configurations are supported using the *args* argument give to the *BasicRGCN* model class. As can be seen in the constructor of the class, *args* is a dictionary that contains various parameters related to the model building and training. Further configuration options could be easily added by inheriting from the class and overriding the constructor with one of your own.

5.3 Data

The basic keras model (such as the one in *BasicRGCN*) used expects two kinds of inputs:

a) A feature matrix
b) A list of adjacency matrices (one for each relation, including the inverse ones)

All matrices are expected to be sparse matrices in csr format (from the scipy.sparse module).

The original datasets used in the paper can be downloaded and built using the *utils.prepare_dataset* script. An example of running the script is in the main README file.

Further datasets can be used, providing that they can provide the necessary data:

- A list of adjacency matrices
- A vector of labels
- A list of indices which are the nodes we know the label for (for both the train and test sets).

It is recommended to save all the data, after building it in the appropriate structure, in a single pickle using a dictionary, which will allow for faster iterations of training.