Relational Graph Convolutional Networks with Sampling and Batching

Lab Rotation Report

by Johannes Rieke

(johannes.rieke@gmail.com)

01 December 2018

AMLab, University of Amsterdam

Bernstein Center for Computational Neuroscience Berlin

External Supervisors: Prof. Dr. Max Welling, Thomas Kipf

BCCN Supervisor: Prof. Dr. Klaus Obermayer

Abstract

Graph Convolutional Networks (GCNs) are a novel class of machine learning models that operate on top of graphs. However, these networks are difficult to scale to large and dense graphs because of their memory requirements during training. Recently, some strategies were introduced to overcome this scalability problem, namely sub-sampling of neighboring nodes and mini-batching. In this work, we apply such strategies to the Relational Graph Convolutional Network (R-GCN), a variant of the GCN, which explicitly models relational graphs. Additionally, we introduce a simplified version of the R-GCN. We test our models on link prediction with the FB15k-237 dataset. While we were successful in lowering the memory requirements during training, we could not achieve link prediction results on par with the original R-GCN implementation or other models from the literature.

1. Introduction

Graph data is ubiquitous in today's world: May it be Facebook's social graph or large knowledge bases like Google Knowledge Graph - a large amount of world knowledge is stored as graphs. Making use of this information is a promising, yet difficult task for machine learning. Recently, the Graph Convolutional Network (GCN; Kipf & Welling 2017) and related models (Duvenaud et al. 2015, Li et al. 2016, Defferrard et al. 2016) were proposed as a way of creating neural network models on top of graphs. It builds on the success of traditional convolutional neural networks (CNNs), which had a huge impact on computer vision. GCN extends the filter-based approach of CNNs to non-Euclidean data like graphs. Just like CNNs, GCNs are not designed with a specific task in mind but serve as a general framework to build applications on top of graphs. Examples include node-level classification (Kipf & Welling 2017), inference for molecular graphs (Duvenaud et al. 2015), and relational reasoning (Santoro et al. 2017).

One of the main issues of GCN-like models is their scalability: Because these models aggregate data across all neighboring graph nodes, making them work on large and dense graphs requires a lot of memory and compute time. This is a serious problem for real-world applications, where graphs can easily contain millions of nodes. Recently, some approaches were introduced to overcome this scalability problem by using sub-sampling of neighbors and mini-batching (Hamilton et al. 2017, Ying et al. 2018, Chen et al. 2018).

Here, we apply such strategies to the Relational Graph Convolutional Network (R-GCN; Schlichtkrull et al. 2018). This network is an extension of the original GCN, which explicitly models different relation types in a graph. We introduce two variants of our model: One is formally equivalent to the original R-GCN model but includes sub-sampling and mini-batching. The other one is a simplified version of the model, which trains significantly faster. We test these models on the link prediction task (i.e. predicting whether a given edge in the graph exists or not) for the FB15k-237 dataset (Toutanova & Chen 2015). While our implementations are in fact easier to scale than the original R-GCN model (i.e. they require less memory and compute time), they do not quite reach the results of the original model. PyTorch implementations of our models are available at https://github.com/jrieke/sampled-rgcn.

2. Related Work

Recently, several studies have introduced modifications of GCN-like models: GraphSage (Hamilton et al. 2017) uses sub-sampling of neighbors and mini-batching quite similar to our approach, albeit on a simplified version of (non-relational) GCN. PinSage (Ying et al. 2018) is an extension of GraphSage that is applied to a large-scale recommender system. FastGCN (Chen et al. 2018) uses importance sampling, i.e. they do not sample neighboring nodes from a uniform distribution (like we do), but based on how important the nodes are for training. To the best of our knowledge, none of these approaches has been applied to the R-GCN model so far.

3. Model

We briefly review the original R-GCN model (Schlichtkrull et al. 2018) and then describe the modifications we introduced. Also, we present a simplified version of our model (*Additive R-GCN*).

3.1 Background: Original R-GCN model

R-GCN is an extension of the Graph Convolutional Network, which is specifically aimed at relational graphs. In such graphs, each edge has a relation type: E.g. the two nodes "*Barack Obama*" and "*Michelle Obama*" might be connected by the relation type "*married to*". The R-GCN models these relations explicitly by having a separate weight matrix for each relation type.

Formally, the model operates on a graph G with nodes V and labeled edges with relation types R (same notation as in Schlichtkrull et al. 2018). Each node i has a feature vector h_i, which can either be inferred from some underlying node data or simply be a one-hot vector. In each layer l of the network, this node vector is updated by aggregating feature vectors from neighboring nodes:

$$h_i^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_i^{(l)}\right)$$

Here, σ is a nonlinearity, N_i^r is the neighborhood of node i for relation type r, $c_{i,r}$ is a normalization constant and W are weight matrices (for details see Schlichtkrull et al. 2018).

In this paper, we solely deal with the link prediction task. The aim is to predict whether an edge with relation type r exists between a given subject node s and object node o. We can describe the setup for this task as a graph auto-encoder: The R-GCN serves as an encoder, i.e. it creates embedding vectors for s and o. The decoder (DistMult; Yang et al. 2015) takes these two node embeddings and outputs a score for a given triple (s, o, r), which describes the probability that the edge exists. The model is trained with a binary cross-entropy loss, i.e. it classifies between triples that actually exist in the graph and negative samples (subject or object of a valid triple replaced by a random node).

3.2 Sampling and Batching

The original R-GCN model is difficult to scale to large and dense graphs. There are two main problems: First, to update the feature vector of a node, the algorithm sums over all neighbors of this node. Especially in densely-connected graphs, this requires a lot of memory. Also, the amount of required memory varies a lot from node to node (in the most extreme case, a node might be connected to all other nodes in the graph). Second, in each layer of the

network, the feature vectors of all nodes in the graph are updated before the computation moves on to the next layer. To overcome these issues, we introduce the following two modifications.

Sub-sampling of neighbors: Instead of summing over all neighboring nodes during the update step, we only sum over a sampled subset of neighbors. For a given node i that we want to update, we take all edges that originate from this node and sample uniformly with replacement a fixed number n of them, regardless of their relation type. If i has fewer than n edges, we oversample. In order to propagate the previous feature vector of node i, we include a self-connection to it during the sampling process (i.e. we sample from the neighborhood $N_{0,i}$ which includes all neighboring nodes of i and the node i itself).

Mini-batching: With this sampling approach, we can efficiently mini-batch the algorithm. Instead of updating the feature vectors of all nodes in the graph before moving on to the next layer, we choose a random subset of nodes for each batch and calculate their feature vectors across all layers.

All in all, our modified update step takes the following form:

$$h_i^{(l+1)} = \sigma\left(\frac{1}{n}\sum_{j\sim\mathcal{N}_{0,i}} W_{r_j}^{(l)} h_j^{(l)}\right)$$

3.3 Additive Model

In our experiments, we found that training the R-GCN model takes a considerable amount of time, even with our modifications. This is due to the relation-specific weight matrices W_{r_j} of the model. Especially in graphs with many different relation types, carrying out the required matrix multiplications is time-consuming. Therefore, we introduce a simplified version of the model, which does not use relation-specific weights. Instead, we use one weight matrix W that is shared across all relation types (just like in the non-relational version of GCN; Kipf & Welling 2017) and add a weight-specific embedding vector e_{r_j}. The update rule takes the form:

$$h_i^{(l+1)} = \sigma\left(\frac{1}{n}\sum_{j\sim\mathcal{N}_{0,i}} (W^{(l)} h_j^{(l)} + e_{r_j}^{(l)})\right)$$

3.4 Other changes

In addition to sampling and batching, our model has a few other minor differences to the original R-GCN implementation:

- The original R-GCN paper uses two specific regularization techniques for the encoder (basis and block regularization) and l2-regularization for the decoder. Instead, our implementation uses dropout between the encoder and decoder stage.
- We use the ELU nonlinearity instead of ReLU.
- We put an additional dense layer in front of the network (which maps the one-hot embeddings of the nodes to dense embeddings), similar to the block-regularized version of the original model.
- We use only one convolutional layer, while the original paper uses one (block model) or two (basis model) layers.

4. Results

We trained our modified R-GCN models on the FB15k-237 dataset (Toutanova & Chen 2015) for link prediction. Additionally, we trained DistMult (Yang et al. 2015) to test our framework. To tune the hyperparameters of our models, we performed a grid search (except for the batch size, which was fixed to 128). Table 1 shows the details of this grid search and the best parameter combination for each model. In the grid search, we chose the best performing model based on the mean reciprocal rank (MRR) on the validation set and evaluated this model on the test set. The results (along with training times and some values from the literature) are given in table 2. Due to the sampling process, the exact values differ slightly between runs, but we found that this effect is negligible.

As table 2 shows, our DistMult implementation is more or less on par with the results of other papers for the same model. Unfortunately, both variations of our R-GCN model perform worse than DistMult, the original R-GCN implementation and the current state of the art on this dataset (ConvE; Dettmers et al. 2018). It seems like the sampling procedure we introduced impairs the model's ability to learn meaningful features, even though it successfully lowered the memory requirements of the algorithm. We want to note that due to time restrictions, some minor differences exist between our implementation and the original R-GCN implementation, which may affect the results (see 3.4; especially, our regularization strategy is quite different).

Table 1: Parameter values for grid search and best parameter combination for each model (based on mean reciprocal rank on the validation set). Batch size was fixed to 128 in all runs.

	Embedding size	Dropout	Learning rate	Sampled neighbors
Values tested in grid search	200, 500	0.2, 0.5, 0.8	0.0005, 0.001	10, 20
DistMult	500	0.8	0.0005	-
R-GCN w/ sampling & batching	200	0.2	0.0005	20
Additive R-GCN w/ sampling & batching	500	0.2	0.0005	20

Table 2: Results for the link prediction task on FB15k-237. All metrics are evaluated on the test set in the filtered setting of Bordes et al. (2013). MR = Mean Rank, MRR = Mean Reciprocal Rank. Best values are in bold.

	MR	MRR	Hits@1	Hits@3	Hits@10	Training time (min / epoch)
DistMult (Dettmers et al. 2018)	254	0.240	0.155	0.263	0.419	
DistMult (ours)	533	0.231	0.165	0.252	0.363	1:30
R-GCN (Schlichtkrull et al. 2018)		0.248	0.153	0.258	0.414	
R-GCN w/ sampling & batching (ours)	236	0.167	0.093	0.174	0.320	9:30
Additive R-GCN w/ sampling & batching (ours)	**230**	0.178	0.100	0.187	0.339	5:30
ConvE (Dettmers et al. 2018)	330	**0.301**	**0.220**	**0.330**	**0.458**	

5. Conclusion

In this study, we implemented two modified versions of the relational graph convolutional network, which introduce sub-sampling of neighbors and mini-batching. While we achieved our initial goal of making the training process more scalable to large graphs (i.e. fewer memory requirements), we failed to reach link prediction results comparable to the original model. Some differences between our implementation and the original R-GCN model might affect the results, but we had not enough time in this project to test them thoroughly. Going further, one might also want to look at other sampling strategies, e.g. importance sampling, which was recently implemented for non-relational graph convolutions by Chen et al. (2018).

References

Chen, J., Ma, T., & Xiao, C. (2018). FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. ICLR. Retrieved from http://arxiv.org/abs/1801.10247

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. NIPS.

Dettmers, T., Minervini, P., Stenetorp, P., & Riedel, S. (2018). Convolutional 2D Knowledge Graph Embeddings. AAAI.

Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A., & Adams, R. P. (2015). Convolutional Networks on Graphs for Learning Molecular Fingerprints. NIPS. https://doi.org/10.1021/acs.jcim.5b00572

Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. NIPS. Retrieved from http://arxiv.org/abs/1706.02216

Kipf, T. N., & Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. ICLR. https://doi.org/10.1051/0004-6361/201527329

Li, Y., Zemel, R., Brockschmidt, M., & Tarlow, D. (2016). Gated Graph Sequence Neural Networks. ICLR.

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P., London, D. (2017). A simple neural network module for relational reasoning. ArXiv Preprint. Retrieved from https://arxiv.org/pdf/1706.01427.pdf

Schlichtkrull, M., Kipf, T. N., Bloem, P., Berg, R. van den, Titov, I., & Welling, M. (2018). Modeling Relational Data with Graph Convolutional Networks. ESWC. Retrieved from http://arxiv.org/abs/1703.06103

Toutanova, K., & Chen, D. (2015). Observed versus latent features for knowledge base and text inference. CVSC.

Yang, B., Yih, W., He, X., Gao, J., & Deng, L. (2015). Embedding Entities and Relations for Learning and Inference in Knowledge Bases. ICLR. Retrieved from http://arxiv.org/abs/1412.6575

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., & Leskovec, J. (2018). Graph Convolutional Neural Networks for Web-Scale Recommender Systems. KDD. https://doi.org/10.1145/3219819.3219890