SCSE18-0168

# Experimental comparison of recommender systems

**See Jie Xun**

**U1522059A**

Supervisor: Associate Professor Xavier Bresson

*Submitted in Partial Fulfilment of the Requirements for the Degree of Bachelor of Engineering (Computer Science) from Nanyang Technological University*

**March 2019**

**School of Computer Science and Engineering**

# Abstract

Recommender systems seek to predict the rating that a user would give an item, given the data of the past ratings of all users and items and other side information. Traditionally, recommender system methods are split into two broad categories: content-based and collaborative filtering approaches. However, because of the graph-structured nature of data in recommender system tasks, graph neural networks hold much promise in pushing the state-of-the-art in recommender systems. This project aims to investigate the latest graph neural network approaches to recommender systems. In particular, it aims to incorporate the use of Residual Gated Graph ConvNets, an architecture that has proven effective on various graph learning tasks, into the recommender system task. We show that within a framework similar to collaborative filtering, using graph neural networks can produce competitive results across various benchmark datasets.

## Acknowledgments

I sincerely thank my project supervisor, Associate Professor Xavier Bresson for his meticulous guidance and insightful advice. This project would not have been possible without his mentorship.

Also, I would like to express my appreciation to my family for their love and support throughout the course of the project.

Last but not least, I am grateful to my course mates, Leow Yao Yang and Ong Jia Rui, for providing me with invaluable suggestions to improve the project, and for their willingness to discuss and generate ideas with me.

# Table of contents

## List of Figures

## List of Tables

# Introduction

## Motivation

With vast amounts of information and choice available for consumers these days, businesses stand to profit if they can recommend appropriate items for their customers. To thrive amid the competition, retailers need to retain customer interest and encourage multiple purchases by suggesting items that they are most likely to want. Thus, recommender systems hold the key to applying their access to big data to provide intelligent recommendations to consumers.

The recommender systems task can be reduced to a matrix completion task. Traditionally, recommender systems are built on a collaborative filtering or content-based approach, using ground truth ratings to predict future ratings and thus complete the user-item matrix. However, ratings data can also be represented by an undirected bipartite user-item graph, with weighted edges denoting the rating given by the user for the item. Furthermore, we can also represent user and item data in separate user and item graphs. Clearly, graph-structured data arises naturally in the recommendation task, and recent research has made use of graph convolutional neural network techniques to achieve state-of-the-art results on benchmark datasets.

## Purpose and scope

The primary focus of this project is to investigate the application of graph convolutional neural networks to recommender systems, considering how the non-Euclidean graph-structured data in the recommender system task lends itself suitably to their use.

In particular, the approach taken by Berg et al. in Graph Convolutional Matrix Completion (GCMC) [1] will be examined in detail. The architecture proposed by Bresson et al. in Residual Gated Graph ConvNets (RGGCN) [2] will be fused with the GCMC framework in attempt to further improve the recommendation results.  GCMC has produced state-of-the-art results on the Douban [9], Flixster [10], and YahooMusic [11] recommender system datasets. In the interest of depth, this report only details experiments on the aforementioned three datasets, while the popular MovieLens benchmark datasets are not investigated here.

Also, the framework of using graph convolutional networks for the matrix completion problem is applied to a domain outside of recommender systems, namely social network graphs.

## Contributions

This project was completed successfully and met its initial objectives. These are the main contributions of the project:

- Implemented the graph convolutional network models in TensorFlow [13], with code being open-sourced on GitHub[1]. (Instructions on reproducing the experiments in this report can be found in the Appendix.)
- Reproduced the results of state-of-the-art recommender systems applied to three benchmark recommender systems datasets.
- Obtained experimental results of our proposed model on the same benchmark datasets, and evaluated our results against those published in literature.
- Applied proposed model to the unrelated domain of social network graphs, achieving competitive results and demonstrating its general applicability to matrix completion problems.

## Project timeline

This project was undertaken in August 2018 and completed in March 2019. Progress on this project was update on a blog, which contains fortnightly summaries of progress and plans. Meetings with A/P Bresson were also conducted fortnightly.

[1] https://github.com/jiexunsee/graph-cnn-recommender-systems/

# Overview on Graph Convolutional Networks

Graph Convolutional Networks (GCNs) generalise the convolution operation to graph-structured data, beyond regular Euclidean data. The advent of Convolutional Neural Networks (CNNs) has led to breakthroughs in image, sound, and video processing and recognition among other tasks. Their effectiveness lies in their ability to learn multi-scale local stationary structures with shift and translation invariant filters, recognising features independent of spatial location and composing them to form expressive representations. GCNs retain the fundamental elements which make traditional CNNs so effective - local connections, shared weights, and the use of multiple layers.

The key function of a GCN is in generating a transformed representation of each node of a graph. Multiple GCN layers can also be stacked to produce progressively more expressive representations of the graph. The two broad categories of spectral and spatial GCNs are described below.

## Spectral Graph Convolutional Networks

Spectral approaches, first introduced by Bruna et al. [3], work with a spectral representation of the graphs. The convolution operates in the spectral domain, using the graph Laplacian **L**, which is analogous to the Fourier basis in signal processing. A signal x is filtered by $g_\theta$ as follows, with $g_\theta(\Lambda) = \text{diag}(\theta)$.

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = Ug_\theta(\Lambda)U^T x$$

This operation is potentially computationally expensive, and produces non-spatially localized filters. To address this, Defferrard et al. [4] extended spectral graph convolutions with fast localised convolutions. They proposed a polynomial filter, expressed in terms of Chebyshev polynomials up to the $K^{th}$ order. This results in a convolution that is K-localised; the operation for each node only depends on nodes that are at most K steps away from it. Their filter is shown below:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

Monti et al. [5] utilised the above filter, combined with recurrent neural networks in a matrix completion framework to obtain competitive results on various recommender system datasets.

As spectral methods often handle the whole graph simultaneously, they face problems of scalability and parallelisability, especially when larger graphs are in question. Consequently, spatial GCNs have garnered more research focus recently.

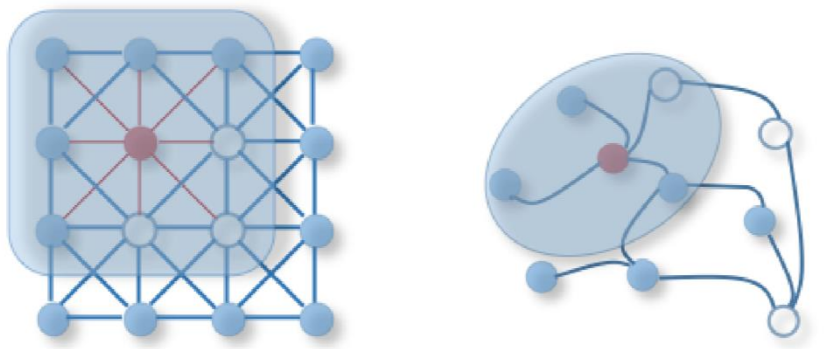
## Spatial Graph Convolutional Networks

Kipf and Welling [6] simplified the spectral methods, while improving scalability and classification performance in large-scale networks. They proposed a multi-layer GCN with a layer-wise propagation rule as follows:

Their convolutional architecture is a forerunner of spatial GCNs, and they showed that it can be motivated via a first-order approximation of localised spectral filters on graphs. By limiting the layer-wise convolution operation to K = 1, they recover a linear function on the graph Laplacian spectrum, and are not confined to the parameterisation given by Chebyshev polynomials as in the filter used by Defferrard et al. [4]. With further approximations, they derive the expression for the propagation rule above. Hence, they also demonstrated in a principled manner that spectral and spatial convolution approaches can be unified, with spatial convolutions as an approximation of spectral convolutions.

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

Spatial-based approaches formulate graph convolutions based on a node's spatial relations. This is done in a manner similar to the way 2D convolutions work, where filters are applied to each pixel and its neighbourhood.



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes a weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.

(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of graph convolution operation takes the average value of node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

**Figure 1:** 2D convolution vs graph convolution. Taken from [7].

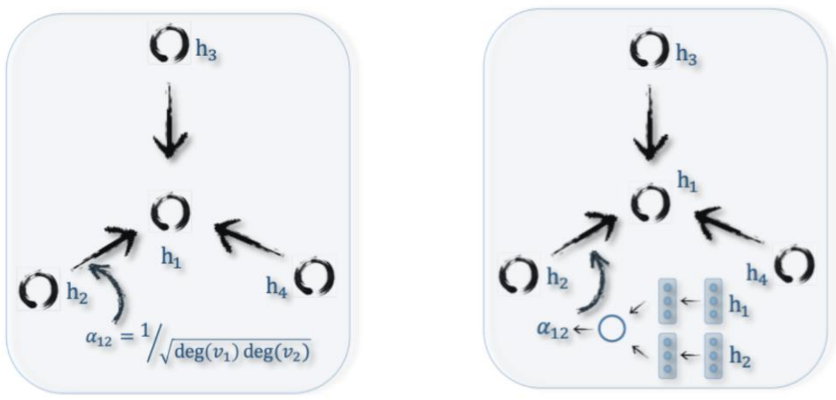
As illustrated in Figure 1a, a filter of size 3x3 is applied to 3x3 patches in the 2D image. Each pixel is connected to its eight neighbours in a fixed positional order. The trainable weights of the filter are shared across different locations, due to the regular structure of the nodes.

On the other hand, the spatial graph convolution aggregates the representations of a node and its neighbours to produce a new representation for the node, as depicted by Figure 1b.

GCN layers can thus be stacked on top of one another to produce progressively more complex hidden representations. Also, an additional GCN layer increases the receptive field of each node by one hop. With *k* GCN layers, we can effectively convolve the $k^{th}$-order neighbourhood of a node. Bresson et al. [2] improved on this basic form of GCNs by adding residual connections as well as edge gates, to improve their performance on various graph learning tasks.

### Graph Attention Networks

Graph attention networks can be classified as a type of spatial graph convolutional network. The attention mechanism that they introduce is similar to the gating mechanism as in RGGCN [2]. Just like the gating weights, the attention weights are learned end-to-end together with the other weight parameters. The key difference is that the attention mechanism can assign larger or smaller weights to more important nodes. Velickovic et al. [8] introduced this graph attention network framework and obtained state-of-the-art results across some transductive and inductive graph benchmarks. Figure 2 illustrates how graph attention networks aggregate the representation of neighbouring nodes, as compared to a normal graph convolution operation.



(a) Graph Convolution Networks [14] explicitly assign a non-parametric weight $a_{ij} = \frac{1}{\sqrt{deg(v_i)deg(v_j)}}$ to the neighbor $v_j$ of $v_i$ during the aggregation process.

(b) Graph Attention Networks [15] implicitly capture the weight $a_{ij}$ via an end to end neural network architecture, so that more important nodes receive larger weights.

**Figure 2:** Differences between graph convolutional networks and graph attention networks. Taken from [7].

Overall, GCNs can also be used as a core building block in forming more complex neural network models. For example, they could be incorporated into autoencoder models (as in GCMC), generative models, spatio-temporal models, and so on.

Spatial graph convolutional networks are the focus of this project, as they have been demonstrated to achieve competitive results on recommender system tasks [1]. Also, they are suitable for variable length graphs, which makes them more flexible and widely applicable than spectral graph convolutional networks.

# Models

## Traditional methods

### Content-based

Content-based recommenders suggest similar items based on a particular item. This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations. They operate on the premise that if a person liked a particular item, he or she will also like an item that is similar to it. Typically, features of items are abstracted using a vector-space representation and cosine similarity is used to determine the similarity of between items.

### Collaborative filtering

Collaborative filtering engines aim to predict the rating or preference that a user would give an item, based on past ratings and preferences of other users. The principle on which they function is that people like things similar to other things that they like, as well as things that other people with similar tastes like. Collaborative filters do not require item metadata like its content-based counterparts.

Within the collaborative filtering framework, either item-based, user-based, or both item and user-based methods are possible. Collaborative filtering methods produce predicted ratings through a matrix factorisation mechanism.

## Deep learning approaches

In this project, models based on deep learning are the focus of experimentation and discussion.

### Graph Convolutional Matrix Completion (GCMC)

In GCMC, Berg et al. [1] which achieved competitive results on the major collaborative filtering benchmarks, outperforming the spectral graph convolution method used by Monti et al. [5]. Their approach comprises a graph encoder that utilises a message passing GCN to produce embeddings and a decoder that translates the embeddings into predicted ratings. Figure 3 depicts the architecture of their model.

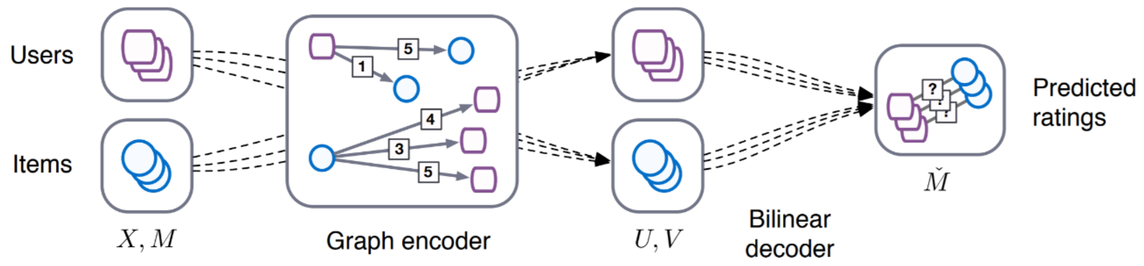


**Figure 3:** Schematic of a forward-pass through the GCMC model, which is comprised of a graph convolutional encoder [U, V] = f(X, $M_1$, . . . , $M_R$) that passes and transforms messages from user to item nodes, and vice versa, followed by a bilinear decoder model that predicts entries of the (reconstructed) rating matrix Mˇ = g(U, V), based on pairs of user and item embeddings. Taken from [1].

GCMC works by having a graph encoder which takes as input an N x D feature matrix and a graph adjacency matrix *A*, to produce an N x E embedding matrix. The bipartite user-item ratings graph is treated as a standard graph, with users and items constituting the nodes of the graph. Thus, the graph encoder transforms each user and item node to have a latent representation *Z*. The bilinear decoder model then takes pairs of user-item node embeddings and predicts the respective entries $A_{ij}$ in the adjacency matrix. The model is trained by minimising the error in this reconstruction of the adjacency matrix *A*.

GCMC's bilinear decoder is similar to the last matrix factorisation step in collaborative filtering which maps the latent representation of users and items to a predicted rating. The difference is that instead of doing a dot product of the two latent representation vectors to directly produce a rating, a probability distribution over possible ratings is produced by applying a bilinear operation followed by a softmax function. As a result the predicted ratings are also discrete rather than continuous. Nevertheless, both methods require a latent variable representation of the users and items that captures adequate information, so that a reasonable rating can be predicted.

## Residual Gated Graph ConvNets with bilinear decoder

This model, illustrated in Figure 4, is the main contribution of the project, and it combines RGGCN with the autoencoder framework of GCMC.

The graph encoder in GCMC is composed of a simple message passing step, which passes and transforms messages from user to item nodes and vice versa. On the other hand, RGGCN [2] can be said to perform a similar message passing process, but adds gates and residual connections.

Each RGGCN transforms the representation *h* of each node as follows. Note the residual connection maintained by the inclusion of the identity operator, the $h_i$ term.

$$h_i^{\ell+1} \quad = \quad f^\ell\left(\, h_i^\ell \,,\, \{h_j^\ell : j \to i\}\,\right) + h_i^\ell$$

The function f is defined as follows.

$$h_i^{\ell+1} = f^\ell_{\text{G-GCNN}}\left(\, h_i^\ell \,,\, \{h_j^\ell : j \to i\}\,\right) \quad = \quad \textbf{ReLU}\left(U^\ell h_i^\ell + \sum_{j \to i} \eta_{ij} \odot V^\ell h_j^\ell\right)$$

In the above equation, $\eta_{ij}$ act as edge gates which are computed as follows.

$$\eta_{ij} = \sigma\left(A^\ell h_i^\ell + B^\ell h_j^\ell\right)$$

In theory, using RGGCN as the graph encoder should enable the model to learn a more informative representation of the graph, as it is able to suppress or highlight certain activations, and maintain residual information. Hence, in an attempt to improve the GCMC model, the graph encoder is replaced with the RGGCN, with the bilinear decoder retained.

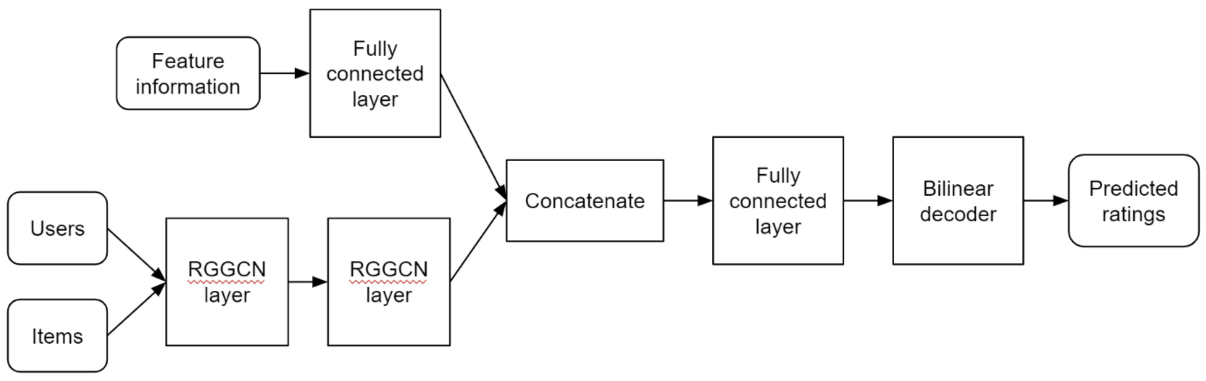


**Figure 4:** RGGCN layers combined with bilinear decoder, with feature information as additional input. In this diagram, two RGGCN layers are shown stacked on each other, but in our experiments any number of them can be used.

## Simple fully connected model with bilinear decoder

As a sanity check to confirm the efficacy of the graph encoder, a simple fully connected neural network is used as the encoder. The bilinear decoder then decodes the representation produced by the fully connected encoder, to produce predicted ratings.

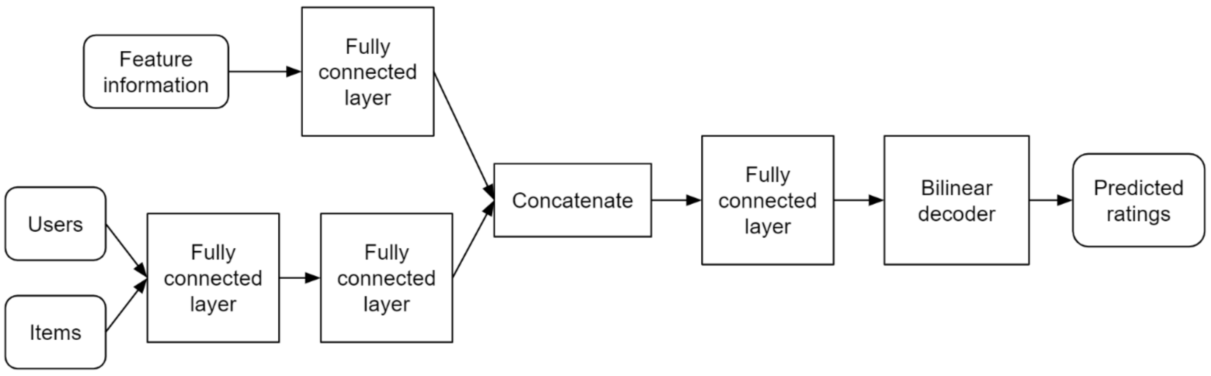


**Figure 5:** Simple fully connected model with bilinear decoder, with feature information as additional input.

# Experiments

## Dataset information

In our experiments so far, we used three benchmark recommender system datasets: Douban, Flixster, and YahooMusic. The datasets are composed of 3,000 users by 3,000 items subsets of their original datasets, preprocessed and provided by Monti et al. [5] for evaluating performance.

| **Dataset** | **Users** | **Items** | **Features** | **Ratings** | **Density** | **Rating levels** |
|---|---|---|---|---|---|---|
| Douban | 3,000 | 3,000 | Users | 136,891 | 0.0152 | 1, 2, …, 5 |
| Flixster | 3,000 | 3,000 | Users, items | 26,173 | 0.0029 | 0.5, 1, …, 5 |
| YahooMusic | 3,000 | 3,000 | Items | 5,335 | 0.0006 | 1, 2, …, 100 |

**Table 1:** Information on the datasets used in our experiments (Statistical figures taken from [1])

| **Dataset** | **Training set ratings** | **Test set ratings** |
|---|---|---|
| Douban | 123,202 | 13,689 |
| Flixster | 23,556 | 2,617 |
| YahooMusic | 4,802 | 533 |

**Table 2:** Number of ratings in training and test set splits for the three datasets. Test set comprises 10% of total ratings.

In line with machine learning best practices, we sought to leave the test set unused, in order to carry out model architecture and hyperparameter iteration. Thus, we split the official training set into a training and validation set, with the validation set making up 20% of the training set. For this train/validation split, the same random seed is applied as in the experiments by van den Berg et al. [1], such that the dataset used in our experiments is exactly the same as that used in their experiments.

## Experimental settings

For all the three models, the ADAM [14] optimiser was used. Dropout was applied on nodes and edges, at the start of each layer before the transformations are applied. In order to speed up the tensor operations, sparse tensors are used where possible. For example, the edge to node matrices are fed as sparse tensors to the model.

## Results

The results below are the accuracies obtained by the model on the validation set, with the model having been trained on the train set. The test set is untouched. See the appendix for full grid search results in determining the best hyperparameters to use.

| **Model** | **Dataset** | | |
|---|---|---|---|
| | **Douban** | **Flixster** | **YahooMusic** |
| GCMC | 0.753 | **0.889** | **21.0** |
| RGGCN with bilinear decoder | **0.735** | 0.894 | 21.6 |
| Simple fully connected with bilinear decoder | 0.797 | 0.918 | 23.1 |

**Table 3:** Results in terms of RMSE for the various models, averaged over 5 runs. The hyperparameters used for RGGCN were found using grid search. For GCMC, the hyperparameters used were the same as in the original paper. For the exact results of each run, refer to the Appendix.

As seen from the results in Table 3, RGGCN with bilinear decoder shows performance that is competitive with that of GCMC.

Interestingly, a simple fully connected model with bilinear decoder does not perform much worse than RGGCN with bilinear decoder or GCMC. This result is achieved even without performing a thorough grid search to find the best hyperparameters for the simple fully connected model. This suggests that both architectures may not be fully taking advantage of the graph structure of the data, and could potentially be further improved.

# Discussion

## Comparison of models used in experiments

| Model | Number of parameters | Training time |
|---|---|---|
| GCMC | 3,863,989 | 29.0 s |
| RGGCN with bilinear decoder | 2,287,989 | 66.9 s |
| Simple fully connected with bilinear decoder | 1,414,189 | 37.9 s |

**Table 4:** Comparison of models used on Douban dataset

| Model | Number of parameters | Training time |
|---|---|---|
| GCMC | 3,863,999 | 20.7 s |
| RGGCN with bilinear decoder | 2,287,999 | 61.0 s |
| Simple fully connected with bilinear decoder | 1,414,199 | 16.5 s |

**Table 5:** Comparison of models used on Flixster dataset

| Model | Number of parameters | Training time |
|---|---|---|
| GCMC | 3,845,671 | 38.3 s |
| RGGCN with bilinear decoder | 2,942,544 | 174.2 s |
| Simple fully connected with bilinear decoder | 1,414,321 | 6.3 s |

**Table 6:** Comparison of models used on YahooMusic dataset

As can be seen, using RGGCN with bilinear decoder requires a smaller number of parameters, but a significantly longer training time. Although the RGGCN module is more complex than the straightforward message passing module in GCMC, the hidden dimensions we used for RGGCN are much smaller. For instance, on the Douban dataset, we use a hidden dimension of 50, compared to the 500 used by GCMC.

The RGGCN module requires input edge information in the form of two special matrices: an “edge to starting vertex” matrix and an “edge to ending vertex” matrix. The number of rows of the these matrices corresponds to the number of edges in the dataset, and the number of columns corresponds to the number of nodes in the dataset. Because there can be a huge number of edges in the dataset, these matrices are very large, and are likely to be a main cause of the longer computation time for predictions, leading to a longer training time. Conversely, GCMC takes in edge information in the form of a "support" matrix. This "support" matrix has rows corresponding to the number of users and items and columns corresponding to the rating types per user/item. This makes it much smaller than the "edge to starting/ending vertex" matrices, and leads to faster computation of predictions.

Also, it is possible that this project's implementation of RGGCN with bilinear decoder is not perfectly optimised, further increasing its training time.

## Notes on variable length graphs

Recommender systems datasets typically have user and item nodes as unique, individual nodes without additional information. In other words, the primary source of information we know about them is from the ratings they have given/received. Thus, they can only be represented as a one-hot vector. This also means that when adding new users/items into the task, every other user/item representation has to be changed to a longer one-hot vector.

Because of this limitation, the typical recommender system setting is not very compatible with having variable length graphs. This hampers us from truly harnessing the flexibility of spatial graph convolutions in handling variable length graphs.

## Data use at validation/test time

At validation and test time, all the ratings, users and items of the training data has to be input in order for a GCN model to make predictions. This may pose a storage and memory hassle in practice, as all the training data has to be kept and consumed by the model when making predictions. Nevertheless, this is to be expected, as it is the ratings made by other users that will help inform the model's prediction of a specific user's rating of a specific item.

## Caveat on results of RGGCN vs GCMC

The hyperparameters for the model when using RGGCN have been carefully tuned so as to extract maximum performance from the model, on each specific dataset. On the other hand, the authors of GCMC used the same hyperparameter setting across all three datasets, Douban, Flixster, and YahooMusic, while still achieving state-of-the-art results. Although RGGCN marginally beats GCMC on some datasets, GCMC has is a simpler model and will be favoured if we follow the Occam's razor principle.

# Application to other domains

## Facebook social graph dataset

This is a Facebook social network dataset taken from the Stanford Network Analysis Project [12]. It is made of 'friends lists' from Facebook, and comprises node features (profiles), circles, and 10 ego networks. Each ego network contains a number of 'circles', composed of friends of the ego user that belong to the same social circle, as manually categorised by the ego user. The typical use case of this dataset involves the prediction of the ego user's social circles, so as to alleviate the time-consuming process of curating and updating one's social circles on social networks like Facebook and Google+.

### Dataset details

By combining ego networks into a single graph, we get a social network graph consisting of 4,039 nodes, and 88,234 edges, where the presence of an edge represents friendship between two nodes. Each node is described by features from their social profile, which includes attributes such as the birthdays, hometowns, political affiliations, work locations, and so on. We consolidate each node's features into a 1406-dimensional one-hot feature vector, with each dimension representing a unique attribute.

### Problem formulation

This Facebook dataset can be appropriately framed as an adjacency matrix completion problem. We take the social network to be an undirected unweighted graph $G = (V, E)$ on $n = |V|$ distinguishable users as nodes, with an adjacency matrix $A \in \{0, 1, 2\}^{n \times n}$. In the adjacency matrix, "0" denotes a missing edge between a pair of nodes (not known if the two nodes are 'friends'), "1" denotes a known absent edge between a pair of nodes (the two users are not 'friends'), and "2" denotes a known present edge between a pair of nodes (the two users are 'friends').

There are two alternative ways of formulating this problem: as a network completion problem, or a link prediction problem.

1. **Network completion**

In a network completion problem, the edge information about a subset of nodes is completely removed. Consequently, the corresponding row of these unobserved nodes in the adjacency matrix will be missing entirely, and the entries would all be "0". Formally, the network's set of N nodes is composed of subsets of observed $N_o$ and unobserved $N_u$ nodes, with $N_o \cup N_u \equiv N$. For the unobserved nodes, $A_{i,j}$ is set to "0", if $i, j \in N_u$. The problem objective is then to predict the missing edges of the unobserved nodes, to complete the adjacency matrix A, based on the observed nodes and the feature vectors of the unobserved nodes.

**2. Link prediction**

In a link prediction problem, all nodes are observed, but random entries of adjacency matrix A are missing. The problem objective is to predict the missing edges to complete the adjacency matrix A, based on the feature vectors and the known graph structure of all the nodes.

We find that the graph autoencoder method does not generalise well to unobserved nodes with unseen feature vectors without information on their neighbouring graph structure, when only trained on observed nodes, as in the network completion scenario. This is likely to be because the graph autoencoder method only receives as input the raw feature vectors of unobserved nodes when attempting to predict the edges of the unobserved nodes. As a graph algorithm, the graph autoencoder naturally relies heavily on the graph structure of nodes as a source of information, and thus performs poorly without access to it.

Therefore, in this project, we use the link prediction problem formulation.

### Related work on this dataset

Masrour et al. [15] investigated the network completion problem, on this Facebook dataset. Using the provided node attributes, they obtain the pairwise similarity between nodes, and store this in a similarity matrix. They then make use of the assumption that the similarity matrix shares latent information with the structure of the network. Their method completes the adjacency sub-matrix of the observed nodes first and then utilises the similarity matrix to complete the missing edges of the unobserved nodes.

Rafailidis et al. [16] also investigated the problem of network completion, with a proposed method based on joint clustering and similarity learning. Instead of considering attribute-based similarities at the node level, they generate clusters based on node attributes, and use a joint objective function to jointly factorise the adjacency matrix of the observed edges with the cluster-based similarities of the node attributes. They then use an alternating minimisation optimisation algorithm to complete the network.

Cao et al. [17] formulated the problem as a link prediction task. Their method, termed SEMAC, induces a representation of each node by learning embeddings of subgraphs around given nodes in the graph, capturing neighbourhood information as well as global community structure. SEMAC proceeds with a form of convex matrix completion to lower the rank of the subgraph embeddings in order to remove noise and improve its generalisation abilities. Finally, the representations for different subgraphs of various depths associated with a given node are combined to form the node's overall representation, which can subsequently be used to predict new links in the graph.

### Suitability of graph autoencoder method

Although used above for recommender system tasks, the graph autoencoder method can be applied in general to any matrix completion problem. Both network completion and link prediction problems are in essence matrix completion problems, hence the graph autoencoder method is appropriate for use. However, instead of having to predict a variety of different 'ratings' between nodes, here we only need to predict if an edge is "1" (two nodes are not 'friends') or "2" (two nodes are 'friends').

The feature vectors in this Facebook dataset are of a similar dimensionality to the feature vectors in the Douban and Flixster datasets. This further supports the suitability of applying the same graph autoencoder method to this dataset, with similar hyperparameters.

### Experiments

In the following experiments, we recorded the accuracy, precision and recall of each model's predictions, as these metrics allow us to better understand the model's performance, compared to using just RMSE. Each experiment was run 5 times and the average result is reported.

We also examined the effect of varying the percentage of observed links provided to the models as a training set. We always use 10% of the links in the validation set, and leave the remaining links for the test set.

The edges data we use comprises the 'friend' connection edges that are provided in the original dataset, as well as randomly selected null edges between users that have no 'friend' connection. The number of null edges selected is roughly the same as the number of 'friend' connection edges, to make for a balanced dataset.

| Model | Observed links | RMSE | Accuracy | Precision | Recall |
|---|---|---|---|---|---|
| GCMC | 25% | **0.22916** | 93.494% | 92.212% | 95.622% |
| | 50% | 0.18380 | 95.867% | 94.381% | 97.905% |
| | 75% | 0.16566 | 96.672% | 95.266% | 98.524% |
| RGGCN with bilinear decoder | 25% | 0.24291 | 93.000% | 92.032% | 94.807% |
| | 50% | **0.18117** | 96.107% | 94.980% | 97.699% |
| | 75% | **0.16530** | 96.751% | 95.469% | 98.452% |
| Simple fully connected with bilinear decoder | 25% | 0.27564 | 90.270% | 88.575% | 93.429% |
| | 50% | 0.20807 | 94.588% | 93.586% | 96.217% |
| | 75% | 0.41540 | 72.253% | 73.827% | 66.226% |

**Table 7:** Experimental results for link prediction on Facebook dataset, with additional profile information features

| Model | Observed links | RMSE | Accuracy | Precision | Recall |
|---|---|---|---|---|---|
| GCMC | 25% | **0.25021** | 91.926% | 90.886% | 93.967% |
| | 50% | 0.20914 | 94.415% | 92.535% | 97.130% |
| | 75% | **0.18643** | 95.617% | 94.062% | 97.781% |
| RGGCN with bilinear decoder | 25% | 0.27579 | 91.502% | 90.746% | 93.253% |
| | 50% | **0.20473** | 94.838% | 93.310% | 97.105% |
| | 75% | 0.19146 | 95.538% | 94.237% | 97.415% |

**Table 8:** Experimental results for link prediction on Facebook dataset, without additional profile information features

| Model | AUROC score |
|---|---|
| SEMAC | **98.2** |
| GCMC | 96.5 |
| RGGCN with bilinear decoder | 96.5 |

**Table 9:** Comparison of experimental AUROC results for link prediction on Facebook dataset, with 80% of links observed

## Discussion on results

From Tables 7 and 8, we see that RGGCN with bilinear decoder tends to perform best when more information is available. When profile information features are supplied as input, RGGCN achieves better results GCMC. However, without profile information features, GCMC bests RGGCN. Nevertheless, the difference is again small, especially for when 50% and 75% of the links are observed. When only 25% of the links are observed, GCMC produces better results than RGGCN. These results are likely to be due to RGGCN being much more expressive than GCMC, leading it to overfit on the small training set.

The models perform well even when the percentage of observed links is reduced from 75% to 50%. This suggests that there is some critical mass of links where it is able to make accurate predictions. Also, the models are able to perform well even without the inclusion of profile information features, as seen in Table 8. There is a drop of only about 1-2% in accuracy when features are withheld. This demonstrates that the graph autoencoder method is able to make use of just the graph structure of the social network to form informative representations for each user. Unlike methods like [16], [17] and [18], the graph autoencoder method is not reliant on features, and can make good predictions solely with the graph structure, without the need for any modification to the algorithm.

Although GCMC and RGGCN with bilinear decoder both perform worse than SEMAC in terms of their area under receiver operating characteristic curve (AUROC) score (Table 9), the difference is not big. Also, it is unclear from [17] as to the composition of the test set of data used by SEMAC. In both SEMAC and our experiments, 20% of the ‘friend’ edges are allocated to the test set. For us, as mentioned previously, our test set comprises roughly the same number of ‘friend’ edges and null edges. However, we do not know how many null edges are included in the test set in the SEMAC experiments. Using a different number of null edges would result in a different AUROC score, and thus the comparison in Table 9 may not be a fair one.

# Conclusion

We have incorporated the general, expressive Residual Gated Graph ConvNet architecture into a graph autoencoder framework, for a highly flexible GCN-based matrix completion algorithm that can be applied across different domains. In recommender system tasks, our algorithm of RGGCN with bilinear decoder achieves similar performance to the state-of-the-art.

With the advent of larger datasets, it is likely that GCN-based deep learning matrix completion methods like ours will gain more prominence. They have hyperparameters that can be flexibly adjusted to handle large volumes of data and are not constrained by the need to compute time-consuming matrix inversion and eigendecomposition operations.

We have also shown the generality of this method by applying it to the entirely different domain of social network graphs. On SNAP's Facebook dataset, the graph autoencoder obtains competitive results, without the need for any modification to the algorithm. As such it can potentially be used by social networks to make friend suggestions or detect spurious connections. The generality of this method also opens up the possibility of its application to other link prediction problem settings in network analysis. For example, there could be practical use cases such as predicting future credit card faults or future associations between terror suspects, or in biology, predicting protein interactions [18] or gene-disease associations [19].

# Future work

Below are some suggested directions of future research work in the field of GCN-based models applied for matrix completion purposes.

## Compare against traditional recommender system algorithms

Thus far, we have benchmarked our recommender systems results against those obtained by GCMC. However, to further investigate the efficacy of a graph neural network approach to recommender systems, we could compare our results against those of traditional content-based and collaborative filtering algorithms.

## Ablation study

At validation and test time, we could reduce the number of users or items available to the model, to see how it performs. In order for the model to work, the weights have to be contracted accordingly - rows corresponding to the user/item being removed have to be dropped.

To investigate the importance of various components of the model, we could remove the gating mechanism, or the residual connection, etc. and see how much performance declines.

## Apply to more matrix completion problems

Our models have been applied to recommender system tasks and to one social network graph dataset. There are numerous applications of matrix completion outside of these, and so a possible future direction would be to apply the same RGGCN and autoencoder framework to other matrix completion tasks. Some possible application domains include biology, computer vision and even national defence.

## Handling multiple graphs/multiplex networks

An interesting research direction would be to analyse problem settings with more than one graph. However, the current design of our method does not handle multiple graphs. Our method is able to process a single bipartite graph, where there can be two different types of nodes - in our case, user and item nodes. There may be cases in which two or more separate graphs contain information for the same nodes, and for which we want to do some multiplex network analysis. For example, Jalili et al. [20] proposed an algorithm that handles a multiplex network of Twitter and Foursquare social networks, considering the same users in these two platforms, to predict links.

## References

[1] R. van den Berg, T.N. Kipf and M. Welling. Graph Convolutional Matrix Completion. arXiv preprint arXiv:1706.02263, 2017.

[2] X. Bresson and T. Laurent. Residual Gated Graph ConvNets. International Conference for Learning Representations, 2018.

[3] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun. Spectral networks and locally connected networks on graphs. International Conference on Learning Representations, 2014.

[4] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. NIPS, 2016.

[5] F. Monti, M.M. Bronstein, and X. Bresson. Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks. Advances in Neural Information Processing Systems, 2017b.

[6] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. International Conference on Learning Representations, 2017.

[7] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596, 2019.

[8] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. International Conference on Learning Representations, 2017.

[9] Ma, H., Zhou, D., Liu, C., Lyu, M., and King, I. Recommender systems with social regularization. In Proc. Web Search and Data Mining, 2011.

[10] Jamali, M. and Ester, M. A matrix factorization technique with trust propagation for recommendation in social networks. In Proc. Recommender Systems, 2010.

[11] Dror, G., Koenigstein, N., Koren, Y., and Weimer, M. The Yahoo! music dataset and KDD-Cup’11. In KDD Cup, 2012.

[12] McAuley, J. and Leskovec, J. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.

[13] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org.

[14] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. arXiv:1412.6980, 2014.

[15] Masrour, F., Barjesteh, I., Forsati R., Esfahanian, A.H., and Radha, H. Network Completion with Node Similarity: A Matrix Completion Approach with Provable Guarantees. IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, 2015.

[16] Rafailidis, D. and Crestani, F. Network Completion via Joint Node Clustering and Similarity Learning. IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, 2016.

[17] Cao, Z., Wang, L., and de Melo, G. Link Prediction via Subgraph Embedding-Based Convex Matrix Completion. The Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[18] Breitkreutz, B.-J., Stark, C., Reguly, T., Boucher, L., Breitkreutz, A., Livstone, M., Oughtred, R., Lackner, D. H., Bahler, J., Wood, V., et al. The biogrid interaction database: 2008 update. Nucleic acids research 36 (suppl 1): D637–D640, 2008.

[19] Natarajan, N. and Dhillon, I. S. Inductive matrix completion for predicting gene–disease associations. Bioinformatics, Volume 30, Issue 12, Pages i60–i68, 2014.

[20] Jalili, M., Orouskhani, Y., Asgari Mehrabadi, M., Alipourfard, N., and Perc, M. Link prediction in multiplex online social networks. Royal Society Open Science 4(2):160863, 2017.

# Appendix

## Instructions on reproducing experiments

### Packages required (Python 3)

- tensorflow >= 1.13 (If possible, use tensorflow-GPU for faster training of models)
- numpy >= 1.14
- scipy >= 1.1
- scikit-learn >= 0.20
- pandas >= 0.23
- h5py >= 2.7

### Data

The datasets are already contained in the repository, under the 'gcmc_adaptation/data' folder. The Douban, Flixster, and YahooMusic datasets are taken from the repository for the Graph Convolutional Matrix Completion method by van den Berg et al. [1].

The Facebook dataset needs to be processed to generate an adjacency matrix and user features, for us to be able to train a GCN on it. To do this, run the 'gcmc_adaptation/data/facebook/processing.py' script, which will save the adjacency matrix as 'adjacency2.npy' in the same folder. The user features have already been generated and are saved as 'user_features.npy'.

### Scripts to use

To run a single experiment, use the script 'gcmc_adaptation/train.py'. Hyperparameter and experimental settings need to be specified at the command line when running the script. For example, use the command to run an experiment on the Facebook dataset:

```
python train.py -d facebook --accum stackRGGCN -do 0.7 -e 1000 --hidden 50 25 --
num_layers 2 --features --testing
```

To evaluate the results of a set of hyperparameters, averaged over a number of iterations, use the script 'gcmc_adaptation/evaluate_hyperparameters.py'. The hyperparameters and number of iterations can be edited within this script.

## Grid search results for finding best hyperparameters for RGGCN with bilinear decoder

In performing grid search, each hyperparameter configuration was tested by running the experiment three times and taking the average result.

### Douban

| Layers | Hidden dimensions | Dropout | Best epoch out of 500 epochs (average of 3) | Best validation accuracy (average of 3) |
|---|---|---|---|---|
| 1 | 50, 25 | 0.5 | 148 | 0.759 |
| 2 | 50, 25 | 0.5 | 144 | 0.731 |
| 3 | 50, 25 | 0.5 | 371 | 0.735 |
| 1 | 75, 50 | 0.5 | 249 | 0.772 |
| 2 | 75, 50 | 0.5 | 197 | 0.733 |
| 3 | 75, 50 | 0.5 | 360 | 0.735 |
| 1 | 100, 75 | 0.5 | 310 | 0.771 |
| 2 | 100, 75 | 0.5 | 227 | 0.733 |
| 3 | 100, 75 | 0.5 | 497 | 0.737 |

**Table 10:** Grid search results on Douban dataset

Based on the above grid search results, the following hyperparameter setting was used for the Douban dataset.

Layers: 2

Hidden dimensions: 50, 25

Dropout: 0.5

Epochs: 150

## Flixster

| Layers | Hidden dimensions | Dropout | Best epoch out of 500 epochs (average of 3) | Best validation accuracy (average of 3) |
|---|---|---|---|---|
| 1 | 40, 20 | 0.5 | 65 | 0.885 |
| 2 | 40, 20 | 0.5 | 89 | 0.883 |
| 3 | 40, 20 | 0.5 | 114 | 0.881 |
| 1 | 50, 25 | 0.5 | 48 | 0.878 |
| 2 | 50, 25 | 0.5 | 92 | 0.877 |
| 3 | 50, 25 | 0.5 | 111 | 0.881 |
| 1 | 70, 50 | 0.5 | 77 | 0.881 |
| 2 | 70, 50 | 0.5 | 118 | 0.891 |
| 3 | 70, 50 | 0.5 | 124 | 0.878 |

**Table 11:** Grid search results on Flixster dataset

Based on the above grid search results, the following hyperparameter setting was used for the Flixster dataset.

Layers: 2

Hidden dimensions: 50, 25

Dropout: 0.5

Epochs: 100

## YahooMusic

| **Layers** | **Hidden dimensions** | **Dropout** | **Best epoch out of 300 epochs (average of 3)** | **Best validation accuracy (average of 3)** |
|---|---|---|---|---|
| 1 | 71, 50 | 0.5 | 19 | 22.5 |
| 2 | 71, 50 | 0.5 | 69 | 20.4 |
| 3 | 71, 50 | 0.5 | 118 | 21.6 |
| 1 | 142, 75 | 0.5 | 15 | 22.6 |
| 2 | 142, 75 | 0.5 | 53 | 20.5 |
| 3 | 142, 75 | 0.5 | 102 | 21.9 |
| 1 | 142, 100 | 0.5 | 16 | 22.4 |
| 2 | 142, 100 | 0.5 | 97 | 20.4 |
| 3 | 142, 100 | 0.5 | 114 | 21.4 |

**Table 12:** Grid search results on YahooMusic dataset

Based on the above grid search results, the following hyperparameter setting was used for the YahooMusic dataset.

Layers: 2

Hidden dimensions: 71, 50

Dropout: 0.5

Epochs: 100

## Evaluation results

| **Dataset** | **Run** | | | | | |
|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **Mean** |
| Douban | 0.7538138 | 0.73138344 | 0.77874786 | 0.7694111 | 0.7299185 | 0.75265494 |
| Flixster | 0.884326 | 0.9003968 | 0.88057697 | 0.9018328 | 0.8764609 | 0.888718694 |
| YahooMusic | 20.259003 | 20.34255 | 22.202051 | 20.819937 | 21.393513 | 21.0034108 |

**Table 13:** Evaluation results on Douban dataset using best hyperparameters

| **Dataset** | **Run** | | | | | |
|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **Mean** |
| Douban | 0.7348396 | 0.73175544 | 0.7370063 | 0.7348891 | 0.73765165 | 0.735228418 |
| Flixster | 0.910286 | 0.8979351 | 0.8909421 | 0.8773042 | 0.89271826 | 0.893837132 |
| YahooMusic | 20.95075 | 22.338522 | 21.095638 | 21.010122 | 22.703297 | 21.6196658 |

**Table 14:** Evaluation results on Flixster dataset using best hyperparameters

| **Dataset** | **Run** | | | | | |
|---|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** | **Mean** |
| Douban | 0.75962436 | 0.8586997 | 0.7476196 | 0.7437411 | 0.87287116 | 0.796511184 |
| Flixster | 0.9002978 | 0.89896756 | 0.93641967 | 0.9256791 | 0.9273772 | 0.917748266 |
| YahooMusic | 22.086542 | 24.514278 | 23.504765 | 22.796432 | 22.43763 | 23.0679294 |

**Table 15:** Evaluation results on YahooMusic dataset using best hyperparameters

## Hyperparameters used

| Dataset | Epochs | Hidden dimensions | Dropout | Layers |
|---|---|---|---|---|
| Douban | 200 | 100, 75 | 0.5 | 2 |
| Flixster | 100 | 100, 75 | 0.5 | 2 |
| YahooMusic | 30 | 100, 75 | 0.5 | 2 |

**Table 16:** Hyperparameters used for the simple fully connected model on the three recommender systems datasets

| Model | Layers | Hidden dimensions | Dropout | Epochs | | |
|---|---|---|---|---|---|---|
| | | | | 25% links observed | 50% links observed | 75% links observed |
| GCMC | 2 | 100, 25 | 0.5 | 1000 | 2000 | 3000 |
| RGGCN with bilinear decoder | 2 | 100, 25 | 0.5 | 1000 | 3000 | 3000 |
| Simple fully connected with bilinear decoder | 2 | 100, 75 | 0.5 | 1000 | 2000 | 3000 |

**Table 17:** Hyperparameters used for the different models in Facebook social graph dataset experiments