Graph Convolutional Matrix Completion

Rianne van den Berg
University of Amsterdam
Amsterdam, The Netherlands
r.vandenberg2@uva.nl

Thomas N. Kipf
University of Amsterdam
Amsterdam, The Netherlands
t.n.kipf@uva.nl

Max Welling*
University of Amsterdam
Amsterdam, The Netherlands
m.welling@uva.nl

ABSTRACT

We consider matrix completion for recommender systems from the point of view of link prediction on graphs. Interaction data such as movie ratings can be represented by a bipartite user-item graph with labeled edges denoting observed ratings. This representation is especially useful in the setting where additional graph-based side information is present. Building on recent progress in deep learning on graph-structured data, we propose a graph auto-encoder framework based on differentiable message passing on the bipartite interaction graph. In settings where complimentary feature information or structured data such as a social network is available, our framework outperforms recent state-of-the-art methods. Furthermore, to validate the proposed message passing scheme, we test our model on standard collaborative filtering tasks and show competitive results.

CCS CONCEPTS

• **Computing methodologies** → *Learning latent representations*; *Machine learning*; *Neural networks*; *Factorization methods*; • **Human-centered computing** → *Collaborative filtering*; *Social networks*;

KEYWORDS

Graph neural networks, matrix completion, collaborative filtering

1 INTRODUCTION

With the explosive growth of e-commerce and social media platforms, recommendation algorithms have become indispensable tools for many businesses.

An important subtask of recommender systems is matrix completion. In this work, we view matrix completion as a link prediction problem on graphs: the interaction data between users and items can be represented by a bipartite graph between user and item nodes, with observed ratings/purchases represented by links. Predicting ratings then corresponds to predicting labeled links in the bipartite user-item graph.

In accordance with this point of view, we propose graph convolutional matrix completion (GC-MC): a graph-based auto-encoder framework for matrix completion, which builds on recent progress in deep learning on graph-structured data [1, 4, 5, 7, 13, 17]. The auto-encoder produces latent features of user and item nodes through a form of message passing on the bipartite interaction graph. These latent user and item representations are used to reconstruct the rating links through a bilinear decoder.

The benefit of formulating matrix completion as a link prediction task on a bipartite graph becomes especially apparent when recommender graphs are accompanied with structured external information such as social networks. Combining such external information with interaction data can alleviate performance bottlenecks related to the cold start problem. We demonstrate that our graph auto-encoder model efficiently combines interaction data with side information, without resorting to recurrent frameworks as in [20]. We furthermore show that in the pure collaborative filtering setting, our methods is able to compete with recent state of the art methods.

Our main contributions are as follows: (1) we apply graph neural networks to the task of matrix completion with structured side-information, and show that our simple message passing model outperforms more complicated graph-based approaches such as [20]. (2) we introduce node dropout, an effective regularization technique that drops out the entire set of all outgoing messages of a single node with a fixed probability.

The open source implementation of this work can be found at https://github.com/riannevdberg/gc-mc.

2 MATRIX COMPLETION AS LINK PREDICTION IN BIPARTITE GRAPHS

Consider a rating matrix M of dimensions $N_u \times N_v$, where N_u is the number of users and N_v is the number of items. A nonzero entry M_{ij} in this matrix represents an observed rating from user i for item j. $M_{ij} = 0$ reflects an unobserved rating. See Figure 1 for an illustration. The task of matrix completion consists of predicting the value of unobserved entries in M.

In an equivalent picture, matrix completion can be cast as a link prediction problem on a bipartite user-item interaction graph. More specifically, the interaction data can be represented by an undirected graph $G = (\mathcal{W}, \mathcal{E}, \mathcal{R})$ with entities consisting of a collection of user nodes $u_i \in \mathcal{W}_u$ with $i \in \{1, ..., N_u\}$, and item nodes $v_j \in \mathcal{W}_v$ with $j \in \{1, ..., N_v\}$, such that $\mathcal{W}_u \cup \mathcal{W}_v = \mathcal{W}$. The edges $(u_i, r, v_j) \in \mathcal{E}$ carry labels that represent ordinal rating levels, such as $r \in \{1, ..., R\} = \mathcal{R}$. This connection was previously explored in [16] and led to the development of graph-based methods for recommendation.

Previous graph-based approaches for recommender systems (see [16] for an overview) typically employ a multi-stage pipeline, consisting of a graph feature extraction model and a link prediction model, all of which are trained separately. Recent results, however, have shown that results can often be significantly improved by modeling graph-structured data with end-to-end learning techniques [1, 4, 5, 13, 17, 19] and specifically with graph auto-encoders [12, 26]

*also affiliated with the Canadian Institute for Advanced Research (CIFAR)

KDD'18 Deep Learning Day, August 2018, London, UK

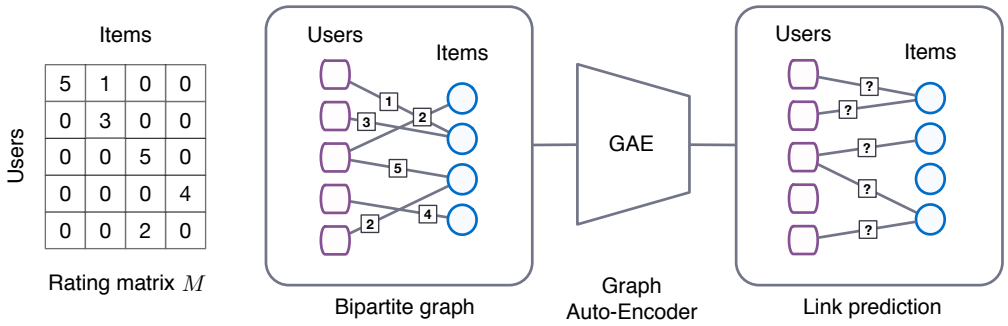

Figure 1: *Left*: Rating matrix M with entries that correspond to user-item interactions (ratings between 1-5) or missing observations (0). *Right*: User-item interaction graph with bipartite structure. Edges correspond to interaction events, numbers on edges denote the rating a user has given to a particular item. The matrix completion task (i.e. predictions for unobserved interactions) can be cast as a link prediction problem and modeled using an end-to-end trainable graph auto-encoder.

for unsupervised learning and link prediction. In what follows, we introduce a specific variant of graph auto-encoders for the task of matrix completion. We will show how graph-based side information can be incorporated naturally.

2.1 Revisiting graph auto-encoders

We revisit graph auto-encoders, which were originally introduced in [12, 26] as an end-to-end model for unsupervised learning [26] and link prediction [12] on undirected graphs. We consider the setup introduced in [12], where the graph encoder model $Z = f(X, A)$ takes as input an $N \times D$ feature matrix X and a graph adjacency matrix A, and produces an $N \times H$ node embedding matrix $Z = [z_1, \ldots, z_N]^T$. The decoder model is a pairwise decoder $\check{A} = g(Z)$, which takes pairs of node embeddings (z_i, z_j) and predicts entries $\check{A}_{ij}$ in the adjacency matrix. Note that N denotes the total number of nodes, D the number of input features, and H the embedding size.

For bipartite recommender graphs $G = (\mathcal{W}, \mathcal{E}, \mathcal{R})$, we can reformulate the encoder as $[Z_u, Z_v] = f(X_u, X_v, M_1, \ldots, M_R)$, where $M_r \in \{0, 1\}^{N_u \times N_v}$ is the adjacency matrix associated with rating type $r \in \mathcal{R}$, such that M_r contains 1's for those elements for which the original rating matrix M contains observed ratings with value r. Z_u and Z_v are now matrices of user and item embeddings with dimensions $N_u \times H$ and $N_v \times H$, respectively. A single user (item) embedding takes the form of a real-valued vector z_i^u (z_j^v) for user i (item j).

Analogously, we can reformulate the decoder as $\check{M} = g(Z_u, Z_v)$, i.e. as a function acting on the user and item embeddings and returning a (reconstructed) rating matrix $\check{M}$ of dimensions $N_u \times N_v$. We can train this graph auto-encoder by minimizing the reconstruction error between the predicted ratings in $\check{M}$ and the observed ground-truth ratings in M. Examples of metrics for the reconstruction error are the root mean square error, or the cross entropy when treating the rating levels as different classes.

Several recent state-of-the-art models for matrix completion [6, 15, 23, 28] can be cast into this framework and understood as a special case of our model. An overview of these models is provided in Section 3.

2.2 Graph convolutional encoder

In what follows, we propose a particular choice of encoder model that makes efficient use of weight sharing across locations in the graph and that assigns separate processing channels for each edge type (or rating type) $r \in \mathcal{R}$. The form of weight sharing is inspired by a recent class of convolutional neural networks that operate directly on graph-structured data [1, 4, 5, 13]. The graph convolutional layer performs local operations that only take the direct neighbors of a node into account, whereby the same transformation is applied across all locations in the graph.

This type of local graph convolution can be seen as a form of message passing [3, 7], where vector-valued messages are being passed and transformed across edges of the graph. In our case, we can assign a specific transformation for each rating level, resulting in edge-type specific messages $\mu_{j \to i, r}$ from items j to users i of the following form:

$$\mu_{j \to i, r} = \frac{1}{c_{ij}} W_r x_j^v \,. \tag{1}$$

Here, c_{ij} is a normalization constant, which we choose to either be $|\mathcal{N}(u_i)|$ (left normalization) or $\sqrt{|\mathcal{N}(u_i)||\mathcal{N}(v_j)|}$ (symmetric normalization), with $\mathcal{N}(u_i)$ denoting the set of neighbors of user node i. W_r is an edge-type specific parameter matrix and x_j^v is the (initial) feature vector of item node j. Messages $\mu_{i \to j, r}$ from users to items are processed in an analogous way. After the message passing step, we accumulate incoming messages at every node by summing over all neighbors $\mathcal{N}_r(u_i)$ connected by a specific edge-type r, and by accumulating the results for each edge type into a single vector representation:

$$h_i^u = \sigma\left[\text{accum}\left(\sum_{j \in \mathcal{N}_1(u_i)} \mu_{j \to i, 1}, \ldots, \sum_{j \in \mathcal{N}_R(u_i)} \mu_{j \to i, R}\right)\right]. \tag{2}$$

Here accum$(\cdot)$ denotes an accumulation operation, such as stack$(\cdot)$, i.e. a concatenation of vectors (or matrices along their first dimension), or sum$(\cdot)$, i.e. summation of all messages. $\sigma(\cdot)$ denotes an element-wise activation function such as the ReLU$(\cdot) = \max(0, \cdot)$. To arrive at the final embedding of user node i, we transform the

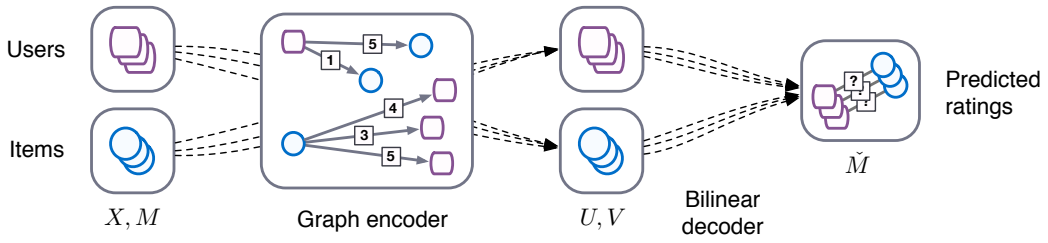

Figure 2: Schematic of a forward-pass through the GC-MC model, which is comprised of a graph convolutional encoder $[U, V] = f(X, M_1, \ldots, M_R)$ that passes and transforms messages from user to item nodes, and vice versa, followed by a bilinear decoder model that predicts entries of the (reconstructed) rating matrix $\check{M} = g(U, V)$, based on pairs of user and item embeddings.

intermediate output h_i as follows:

$$z_i^u = \sigma(W h_i^u) . \tag{3}$$

The item embedding z_i^v is computed analogously with the same parameter matrix W. In the presence of user- and item-specific side information we use separate parameter matrices for user and item embeddings. We will refer to (2) as a *graph convolution* layer and to (3) as a *dense* layer. Note that deeper models can be built by stacking several layers (in arbitrary combinations) with appropriate activation functions. In initial experiments, we found that stacking multiple convolutional layers did not improve performance and a simple combination of a convolutional layer followed by a dense layer worked best.

2.3 Bilinear decoder

For reconstructing links in the bipartite interaction graph we consider a bilinear decoder, and treat each rating level as a separate class. Indicating the reconstructed rating between user i and item j with $\check{M}_{ij}$, the decoder produces a probability distribution over possible rating levels through a bilinear operation followed by the application of a softmax function:

$$p(\check{M}_{ij} = r) = \frac{e^{(z_i^u)^T Q_r z_j^v}}{\sum_{s=1}^{R} e^{(z_i^u)^T Q_s v_j}} , \tag{4}$$

with Q_r a trainable parameter matrix of dimensions $H \times H$, and H the dimensionality of hidden user (item) representations z_i^u (z_j^v). The predicted rating is computed as

$$\check{M}_{ij} = g(u_i, v_j) = \mathbb{E}_{p(\check{M}_{ij}=r)}[r] = \sum_{r \in R} r \, p(\check{M}_{ij} = r) . \tag{5}$$

2.4 Model training

Loss function. During model training, we minimize the following negative log likelihood of the predicted ratings $\check{M}_{ij}$:

$$\mathcal{L} = - \sum_{i,j;\Omega_{ij}=1} \sum_{r=1}^{R} I[M_{ij} = r] \log p(\check{M}_{ij} = r) , \tag{6}$$

with $I[k = l] = 1$ when $k = l$ and zero otherwise. The matrix $\Omega \in \{0, 1\}^{N_u \times N_v}$ serves as a mask for unobserved ratings, such that ones occur for elements corresponding to observed ratings in M, and zeros for unobserved ratings. Hence, we only optimize over observed ratings.

Mini-batching. We introduce mini-batching by sampling contributions to the loss function in Eq. (6) from different observed ratings. That is, we sample only a fixed number of contributions from the sum over user and item pairs. This serves both as an effective means of regularization, and reduces the memory requirement to train the model, which is necessary to fit the full model for MovieLens-10M into GPU memory. We experimentally verified that training with mini-batches and full batches leads to comparable results for the MovieLens-1M dataset while adjusting for regularization parameters. For all datasets except for the MovieLens-10M, we opt for full-batch training since it leads to faster convergence than training with mini-batches in this particular setting.

Node dropout. In order for the model to generalize well to unobserved ratings, it is trained in a denoising setup by randomly dropping out all outgoing messages of a particular node, with a probability p_{dropout}, which we will refer to as *node dropout*. Messages are rescaled after dropout as in [24]. In initial experiments we found that node dropout was more efficient in regularizing than message dropout. In the latter case individual outgoing messages are dropped out independently, making embeddings more robust against the presence or absence of single edges. In contrast, node dropout also causes embeddings to be more independent of particular user or item influences. We furthermore also apply regular dropout [24] to the hidden layer units (3).

Weight sharing. Not all users and items have an equal number of ratings for each rating level. In the graph convolution layer, this results in certain columns of the weight matrices W_r to be optimized significantly less frequently than others. Therefore, some form of weight sharing between the matrices W_r for different r is desirable to alleviate this optimization problem. Following [28], we implement the following weight sharing setup:

$$W_r = \sum_{s=1}^{r} T_s . \tag{7}$$

We will refer to this type of weight sharing as ordinal weight sharing due to the increasing number of weight matrices included for higher rating levels.

As an effective means of regularization of the pairwise bilinear decoder, we resort to weight sharing in the form of a linear

combination of a set of basis weight matrices P_s:

$$Q_r = \sum_{s=1}^{n_b} a_{rs} P_s \,, \tag{8}$$

with $s \in (1, ..., n_b)$ and n_b being the number of basis weight matrices. Here, a_{rs} are the learnable coefficients that determine the linear combination for each decoder weight matrix Q_r. Note that in order to avoid overfitting and to reduce the number of parameters, the number of basis weight matrices n_b should naturally be lower than the number of rating levels.

2.5 Input feature representation and side information

Features containing information for each node, such as content information, can in principle be injected into the graph encoder directly at the input-level (i.e. in the form of input feature matrices X_u and X_v). However, when the content information does not contain enough information to distinguish different users (or items) and their interests, this leads to a severe bottleneck of information flow. Therefore, in this work, we choose to include side information in the form of user and item feature vectors $x_i^{u,f}$ and $x_j^{v,f}$ (for user node i and item node j) via a separate processing channel directly into the the dense hidden layer:

$$z_i^u = \sigma(W h_i^u + W_2^{u,f} f_i^u) \quad \text{with} \quad f_i^u = \sigma(W_1^{u,f} x_i^{u,f} + b^u)\,, \tag{9}$$

where $W_1^{u,f}$ and $W_2^{u,f}$ are trainable weight matrices, and b^u is a bias. The latent item vectors are obtained similarly with different weight matrices $W_1^{v,f}$ and $W_2^{v,f}$ and b^v. The node input feature matrices X_u and X_v for the graph convolution layer are then chosen to contain unique one-hot vector for every node in the graph.

In [25] the authors propose to include content information along similar lines, although in their case the proposed model is strictly user- or item-based, and thus only supports side information for either users or items.

3 RELATED WORK

3.1 Auto-encoders

User- or item-based auto-encoders [23, 25, 28] are a recent class of state-of-the-art collaborative filtering models that can be seen as a special case of our graph auto-encoder model, where only either user or item embeddings are considered in the encoder. AutoRec by [23] is the first such model, where the user's (or item's) partially observed rating vector is projected onto a latent space through an encoder layer, and reconstructed using a decoder layer with mean squared reconstruction error loss.

The CF-NADE algorithm by [28] can be considered as a special case of the above auto-encoder architecture. In the user-based setting, messages are only passed from items to users, and in the item-based case the reverse holds. Note that in contrast to our model, unrated items are assigned a default rating of 3 in the encoder, thereby creating a fully-connected interaction graph. CF-NADE imposes a random ordering on nodes, and splits incoming messages into two sets via a random cut, only one of which is kept. This model can therefore be seen as a denoising auto-encoder, where part of the input space is dropped out at random in every iteration.

3.2 Factorization models

Our model is related to a number of matrix factorization (MF) techniques: Probabilistic matrix factorization (PMF) [18] takes a probabilistc approach in solving the MF problem $M \approx UV^T$. BiasedMF [14] improves upon PMF by incorporating a user and item specific bias, as well as a global bias. Neural network matrix factorization (NNMF) [6] extends the MF approach by passing the latent user and item features through a feed forward neural network. Local low rank matrix approximation [15], introduces the idea of reconstructing rating matrix entries using different (entry dependent) combinations of low rank approximations.

3.3 Matrix completion with side information

In matrix completion (MC) [2], the objective is to approximate the rating matrix with a low-rank rating matrix. Rank minimization, however, is an intractable problem, and [2] replaced the rank minimization with a minimization of the nuclear norm (the sum of the singular values of a matrix), turning the objective function into a tractable convex one. Inductive matrix completion (IMC) [9] incorporates content information of users and items in feature vectors and approximates the observed elements of the rating matrix as $M_{ij} = x_i^T U V^T y_j$, with x_i and y_j representing the feature vector of user i and item j respectively.

The geometric matrix completion (GMC) model proposed by [10] introduces a regularization of the MC model by adding side information in the form of user and item graphs. In [21], a more efficient alternating least squares optimization optimization method (GRALS) is introduced to the graph-regularized matrix completion problem.

Closely related to our work is RGCNN [20]. They explore the application of spectral graph filters based on Chebyshev polynomials [4] of the k-nearest neighbor graphs of users and items. This is combined with a recurrent estimation of the interaction matrix. In contrast, our model is based on neural message passing directly on the interaction graph (which is related to using a first order expansion of spectral filters [13]). Furthermore, we model the rating graph directly in a single encoder-decoder step instead of using a recurrent estimation, which leads to significant speed-ups.

Lastly, we note that concurrently to our work Ying et al. [27] developed PinSage, a highly scalable graph convolutional network for recommendation on web-scale graphs based on the GraphSAGE [8] framework, where neighborhoods are subsampled to enhance scalability. In contrast to their work, we focus on the inclusion of graph-based side information, e.g. in the form of social network graphs, and further introduce regularization techniques that improve generalization.

4 EXPERIMENTS

We evaluate our model on a number of common collaborative filtering benchmark datasets: MovieLens[1] (100K, 1M, and 10M), Flixster, Douban, and YahooMusic. The datasets consist of user ratings for items (such as movies) and optionally incorporate additional user/item information in the form of features. For Flixster, Douban,

[1] https://grouplens.org/datasets/movielens/

Dataset	Users	Items	Features	Ratings	Density	Rating levels
Flixster	3,000	3,000	Users/Items	26,173	0.0029	0.5, 1, ..., 5
Douban	3,000	3,000	Users	136,891	0.0152	1, 2, ..., 5
YahooMusic	3,000	3,000	Items	5,335	0.0006	1, 2, ..., 100
MovieLens 100K (ML-100K)	943	1,682	Users/Items	100,000	0.0630	1, 2, ..., 5
MovieLens 1M (ML-1M)	6,040	3,706	—	1,000,209	0.0447	1, 2, ..., 5
MovieLens 10M (ML-10M)	69,878	10,677	—	10,000,054	0.0134	0.5, 1, ..., 5

Table 1: Number of users, items and ratings for each of the MovieLens datasets used in our experiments. We further indicate rating density and rating levels.

and YahooMusic we use preprocessed subsets of these datasets provided by [20][2]. These datasets contain sub-graphs of 3000 users and 3000 items and their respective user-user and item-item interaction graphs (if available). Dataset statistics are summarized in Table 1.

For all experiments, we choose from the following settings based on validation performance: accumulation function (stack vs. sum), whether to use ordinal weight sharing in the encoder, left vs. symmetric normalization, and dropout rate $p \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$. Unless otherwise noted, we use the Adam optimizer [11] with a learning rate of 0.01, weight sharing in the decoder with 2 basis weight matrices, and layer sizes of 500 and 75 for the graph convolution (with ReLU) and dense layer (no activation function), respectively. We evaluate our model on the held out test sets using an exponential moving average of the learned model parameters with a decay factor set to 0.995.

4.1 MovieLens 100K

For this task, we compare against matrix completion baselines that make use of side information in the form of user/item features. We report performance on the canonical u1.base/u1.test train/test split. Hyperparameters are optimized on a 80/20 train/validation split of the original training set. Side information is present both for users (e.g. age, gender, and occupation) and movies (genres). Following [21], we map the additional information onto feature vectors for users and movies, and compare the performance of our model with (GC-MC+Feat) and without the inclusion of (GC-MC). Note that GMC [10], GRALS [21] and sRGCNN [20] represent user/item features via a k-nearest-neighbor graph. We use stacking as an accumulation function in the graph convolution layer in Eq. (2), set dropout to 0.7, and use left normalization. GC-MC+Feat uses 10 hidden units for the dense side information layer (with ReLU activation) as described in Eq. 9. We train both models for 1,000 full-batch epochs. We report RMSE scores averaged over 5 runs with random initializations[3]. Results are summarized in Table 2.

Model	ML-100K + Feat
MC [2]	0.973
IMC [9]	1.653
GMC [10]	0.996
GRALS [21]	0.945
sRGCNN [20]	0.929
GC-MC (Ours)	0.910
GC-MC+Feat (Ours)	**0.905**

Table 2: RMSE scores, for the MovieLens 100K task with side information on a canonical 80/20 training/test set split. Side information is either presented as a nearest-neighbor graph in user/item feature space or as raw feature vectors. Baseline numbers are taken from [20].

4.2 MovieLens 1M and 10M

We compare against current state-of-the-art collaborative filtering algorithms, such as AutoRec [23], LLorma [15], and CF-NADE [28]. Results are reported as averages over the same five 90/10 training/test set splits as in [28] and summarized in Table 3. Model choices are validated on an internal 95/5 split of the training set. For ML-1M we use accumulate messages through summation in Eq. (2), use a dropout rate of 0.7, and symmetric normalization. As ML-10M has twice the number of rating classes, we use twice the number of basis function matrices in the decoder. Furthermore, we use stacking accumulation, a dropout of 0.3 and symmetric normalization. We train for 3,500 full-batch epochs, and 18,000 mini-batch iterations (20 epochs with batch size 10,000) on the ML-1M and ML-10M dataset, respectively.

Model	ML-1M	ML-10M
PMF [18]	0.883	–
I-RBM [22]	0.854	0.825
BiasMF [14]	0.845	0.803
NNMF [6]	0.843	–
LLORMA-Local [15]	0.833	0.782
I-AUTOREC [23]	0.831	0.782
CF-NADE [28]	**0.829**	**0.771**
GC-MC (Ours)	0.832	0.777

Table 3: Comparison of average test RMSE scores on five 90/10 training/test set splits (as in [28]) without the use of side information. Baseline scores are taken from [28]. For CF-NADE, we report the best-performing model variant.

[2] https://github.com/fmonti/mgcnn
[3] Standard error less than 0.001.

4.3 Flixster, Douban and YahooMusic

These datasets contain user and item side information in the form of graphs. We integrate this graph-based side information into our framework by using the adjacency vector (normalized by degree) as a feature vector for the respective user/item. For a single dense feature embedding layer, this is equivalent to performing a graph convolution akin to [13] on the user-user or item-item graph.[4] We use a dropout rate of 0.7, and 64 hidden units for the dense side information layer (with ReLU activation) as described in Eq. 9. We use a left normalization, and messages in the graph convolution layer are accumulated by concatenation (as opposed to summation). All models are trained for 200 epochs. For hyperparameter selection, we set aside a separate 80/20 train/validation split from the original training set in [20]. For final model evaluation, we train on the full original training set from [20] and report test set performance. Results are summarized in Table 4.

Model	Flixster	Douban	YahooMusic
GRALS	1.313/1.245	0.833	38.0
sRGCNN	1.179/0.926	0.801	22.4
GC-MC	**0.941/0.917**	**0.734**	**20.5**

Table 4: Average RMSE test set scores for 5 runs on Flixster, Douban, and YahooMusic, all of which include side information in the form of user and/or item graphs. We replicate the benchmark setting as in [20]. For Flixster, we show results for both user/item graphs (right number) and user graph only (left number). Baseline numbers are taken from [20].

4.4 Cold-start analysis

To gain insight into the use of side information by the GC-MC model, we study the performance of our model in the presence of users with only very few ratings (cold-start users). We adapt the ML-100K benchmark dataset, so that for a fixed number of cold-start users N_c all ratings except for a minimum number N_r are removed from the training set (chosen at random with a fixed seed across experiments). Note that ML-100K in its original form contains only users with at least 20 ratings.

We analyze model performance for $N_r \in \{1, 5, 10\}$ and $N_c \in \{0, 50, 100, 150\}$, both with and without using user/item features as side information (see Figure 3). Hyperparameters and test set are chosen as before, i.e. we report RMSE on the complete canonical test set split. The benefit of incorporating side information, such as user and item features, is especially pronounced in the presence of many users with only a single rating.

4.5 Discussion

On the tasks with both user and item side information, our model outperforms related methods. Most related to our method is sRGCNN by [20] that uses graph convolutions on the k-nearest neighbor graphs of users and items, and learns representations in an iterative manner using recurrent neural networks. Our results demonstrate that a simple auto-encoder model with message passing instead on the bipartite interaction graph can outperform a more complicated recurrent estimation.

A possible reason for the increased performance can be the difference in the graph on which the message passing occurs. In sRGCNN, the k-nearest neighbour graphs for users and items respectively are used for message passing. Therefore, messages are only passed among users and between items. In contrast, our method uses the graph of observed ratings for message passing. As a result, messages are sent from users to items and items to users. Note that in the side information setting, we also use the k-nearest neighbor graphs as provided by Monti et al. [20] to compute side information features.

A second difference occurs in the approximation of the corresponding graph Laplacian: sRGCNN uses a Chebyshev expansion (of the user and item k-nearest neighbor graphs), which for a given order p takes into account messages from neighboring nodes up to p hops away. Our method is related to using a first-order approximation (of the bipartite interaction graph for each rating type), such that only the direct neighbors of each node are accessed. This first-order approximation scheme has been shown to improve performance [13].

Our results on ML-1M and ML-10M demonstrate that it is possible to scale our method to larger datasets, putting it into the vicinity of recent state-of-the-art collaborative filtering user- or item-based methods in terms of predictive performance. At this point, it is important to note that several techniques introduced in CF-NADE [28], such as layer-specific learning rates, a special ordinal loss function, and the auto-regressive modeling of ratings, can be seen as orthogonal to our approach and can be used in conjunction with our framework.

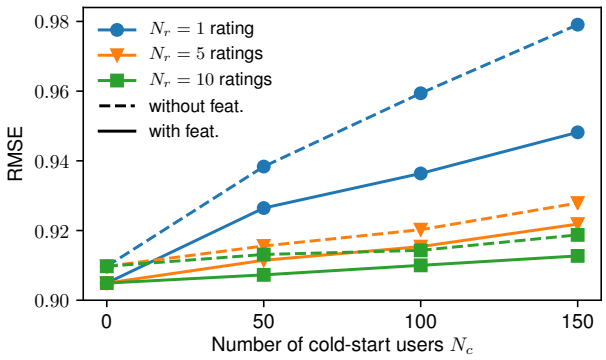

Figure 3: Cold-start analysis for ML-100K. Test set RMSE (average over 5 runs with random initialization) for various settings, where only a small number of ratings N_r is kept for a certain number of cold-start users N_c during training. Standard error is below 0.001 and therefore not shown. Dashed and solid lines denote experiments without and with side information, respectively.

[4] With a row-normalized adjacency matrix instead of the symmetric normalization from [13]. We found that both perform similarly in practice.

5 CONCLUSIONS

In this work, we have introduced graph convolutional matrix completion (GC-MC): a graph auto-encoder framework for the matrix completion task in recommender systems. The encoder contains a graph convolution layer that constructs user and item embeddings through message passing on the bipartite user-item interaction graph. Combined with a bilinear decoder, new ratings are predicted in the form of labeled edges.

The graph auto-encoder framework naturally generalizes to include side information for both users and items. In this setting, our proposed model outperforms recent related methods, as demonstrated on a number of benchmark datasets with feature- and graph-based side information. In settings without side-information, our model achieves results that are competitive with recent state-of-the-art collaborative filtering methods.

Our model can be extended to large-scale multi-modal data (comprised of text, images, and other graph-based information). In such a setting, the GC-MC model can be combined with recurrent or convolutional neural networks.

6 ACKNOWLEDGEMENTS

We would like to thank Jakub Tomczak, Christos Louizos, Karen Ullrich and Peter Bloem for helpful discussions and comments. This project is supported by SAP SE Berlin.

REFERENCES

[1] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2013. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

[2] Emmanuel Candes and Benjamin Recht. 2012. Exact matrix completion via convex optimization. *Commun. ACM* 55, 6 (2012), 111–119.

[3] Hanjun Dai, Bo Dai, and Le Song. 2016. Discriminative Embeddings of Latent Variable Models for Structured Data. In *International Conference on Machine Learning (ICML)*.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3837–3845.

[5] David K. Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems (NIPS)*. 2224–2232.

[6] Gintare Karolina Dziugaite and Daniel M Roy. 2015. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443* (2015).

[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *ICML*.

[8] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. *arXiv preprint arXiv:1706.02216* (2017).

[9] Prateek Jain and Inderjit S Dhillon. 2013. Provable inductive matrix completion. *arXiv preprint arXiv:1306.0626* (2013).

[10] Vassilis Kalofolias, Xavier Bresson, Michael Bronstein, and Pierre Vandergheynst. 2014. Matrix completion on graphs. *arXiv preprint arXiv:1408.1717* (2014).

[11] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Thomas N. Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *NIPS Bayesian Deep Learning Workshop* (2016).

[13] Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[14] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (Aug. 2009), 30–37. DOI: http://dx.doi.org/10.1109/MC.2009.263

[15] Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. 2013. Local Low-Rank Matrix Approximation. In *Proceedings of the 30th International Conference on Machine Learning (ICML) (Proceedings of Machine Learning Research)*, Sanjoy Dasgupta and David McAllester (Eds.), Vol. 28. PMLR, Atlanta, Georgia, USA, 82–90. http://proceedings.mlr.press/v28/lee13.html

[16] Xin Li and Hsinchun Chen. 2013. Recommendation as link prediction in bipartite graphs: A graph kernel-based machine learning approach. *Decision Support Systems* 54, 2 (2013), 880 – 890. DOI: http://dx.doi.org/https://doi.org/10.1016/j.dss.2012.09.019

[17] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence neural networks. *ICLR* (2016).

[18] Andriy Mnih and Ruslan R. Salakhutdinov. 2008. Probabilistic matrix factorization. In *Advances in neural information processing systems*. 1257–1264.

[19] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. *CVPR* (2017).

[20] Federico Monti, Michael Bronstein, and Xavier Bresson. 2017. Geometric matrix completion with recurrent multi-graph neural networks. In *Advances in Neural Information Processing Systems*. 3700–3710.

[21] Nikhil Rao, Hsiang-Fu Yu, Pradeep K. Ravikumar, and Inderjit S. Dhillon. 2015. Collaborative Filtering with Graph Information: Consistency and Scalable Methods. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 2107–2115.

[22] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*. ACM, 791–798.

[23] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th International Conference on World Wide Web*. ACM, 111–112.

[24] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[25] Florian Strub, Romaric Gaudel, and Jérémie Mary. 2016. Hybrid Recommender System Based on Autoencoders. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016)*. ACM, New York, NY, USA, 11–16. DOI: http://dx.doi.org/10.1145/2988450.2988456

[26] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning Deep Representations for Graph Clustering.. In *AAAI*. 1293–1299.

[27] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. (2018).

[28] Yin Zheng, Bangsheng Tang, Wenkui Ding, and Hanning Zhou. 2016. A neural autoregressive approach to collaborative filtering. In *Proceedings of the 33nd International Conference on Machine Learning*. 764–773.