An analysis of the paper and its novelty

Written by: Itay Levinas
On the paper: Rianne van den Berg, Thomas N. Kipf, Max Welling, Graph Convolutional Matrix Completion (2017)

The method

Background

The writers consider a method for recommender systems. They view the problem of recommending as a problem of link prediction in a bipartite user-item graph. They describe their method of matrix completion, using a graph convolution-based auto-encoder.
Using their model, they can easily include side-information on the users or items.

Input

The dataset is represented as a bipartite graph $G = (W, E, R)$, where:
W is the set of nodes, which is a union of the users' nodes $\mathcal{U}$ and the items' nodes $\mathcal{V}$;
$E = \{(u_i, r, v_i) | u_i \in \mathcal{U}, v_i \in \mathcal{V}, r \in R\}$ is the set of edges;
R is the range of ratings that can be given to an item by a user.

Therefore, one input of the model is a matrix M of size $|\mathcal{U}| \times |\mathcal{V}|$, in which M_{ij} is the rating of the user i to the item j if exists, and a default value (in most cases, 0) otherwise. More accurately, the model receives a set of matrices $\{M_1, \dots, M_R\}$, binary matrices in which $(M_r)_{ij}$ is 1 if user i rated item j as r, and 0 otherwise.

Another input is a feature matrix X. In this model, this matrix is just an identity matrix, with unique one-hot vectors for each node.

Model

The model uses the input matrices in a graph convolutional encoder to get latent features for the user and item nodes, then decodes the features to predict a matrix $\widetilde{M}$, a new rating matrix where new predicted links appear. When training, $\widetilde{M}$ is taken to be the closest to M as possible, using RMSE metric.

The encoding process is made of two main layer types: the convolutional layer and the so-called dense layer.
The writers of the paper mention that their model worked best with one convolutional layer followed by a dense layer.

The convolutional layer(s) use every matrix M_r and the feature matrix (that may has been updated in a previous convolutional layer), as well as weight matrices W_r, such that the output for the user i will be:

$$h_i = \sigma\left[accum\left(\sum_{j\in\mathcal{N}_{i,1}} \frac{1}{c_{ij}} W_1 x_j, \dots, \sum_{j\in\mathcal{N}_{i,R}} \frac{1}{c_{ij}} W_R x_j\right)\right]$$

(The same is done for every item j)
Where:

- $\mathcal{N}_{i,r}$ is the set of neighbors of i by the rating r.
- c_{ij} is a normalization factor. In the paper, they took it to be either $\mathcal{N}_i$ (left) or $\sqrt{|\mathcal{N}_i||\mathcal{N}_j|}$ (symmetric).
- $accum(\cdot)$ is an accumulation function of all the sums. It can be either $sum(\cdot)$ (i.e. summarize the sums) or $stack(\cdot)$ (i.e. concatenate the sums to a vector).
- σ is an activation function, here taken to be $ReLU = \max\{0, x\}$.
- Here there is a use of a sum of messages: $\mu_{i\to j,r} = \frac{1}{c_{ij}} W_r x_j$

The dense layer receives h_i and h_j and returns the encoding $[U, V]$ of the users and items, respectively. It is done by the simple action:

$$u_i = \sigma(W h_i)$$

This time, W is the same weight matrix for all the nodes. σ is again an activation function that is taken her to be $ReLU$.

The decoder returns the probability distribution for an edge to be of each rating, using a bilinear operation and the softmax function:

$$\mathbb{P}(\widetilde{M}_{ij} = r) = \frac{e^{u_i^T Q_r v_j}}{\sum_{s\in R} e^{u_i^T Q_s v_j}}$$

The predicted rating is computed as:

$$\widetilde{M}_{ij} = \mathbb{E}[r] = \sum_{r\in R} r\, \mathbb{P}(\widetilde{M}_{ij} = r)$$

Training

The **loss** function used for training is a negative log likelihood function, summed over all pairs user-item for which there is a rating:

$$L = -\sum_{(i,j:M_{ij}\neq 0)} \sum_{r\in R} \mathbb{I}[M_{ij} = r] \log\left(\mathbb{P}(\widetilde{M}_{ij} = r)\right)$$

During training, **dropout** is used to make the model less sensitive to noise and able to predict new ratings. Dropping out is done randomly over the outgoing messages of every node, with a dropout probability of $p_{dropout}$. Also is applied a regular dropout to the hidden layer units.

In large datasets, the memory necessary for running the model is more than Berg et al.'s GPU memory (12 Gb). Therefore, they used **mini-batching** – for their loss function, they sample only a part of the user-item pairs, over which they sum.

Side-information

When using **side-information** given on the users and items, this model doesn't inject them in the input feature matrix X, but to the dense layer. In this case, that layer will be:

$$u_i = \sigma\left(W h_i + W_2^f f_i\right), with\ f_i = \sigma(W_1^f x_i^f + b)$$

The matrices W_n^f are trainable matrices, so basically the features are encoded and added to the dense layer. This way was chosen because sometimes those features alone are not informative enough to distinguish different nodes.

Weight-sharing

Optimizing the weight matrices is performed on every r, separate from the other ratings. However, all the ratings are not treated in the same frequency, so optimizing some matrices can be much more often than others.

To deal with this problem and a similar problem using Q_r, **weight-sharing** is used:

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

where T_s are learned. This way, the training is done using not only one rating value, but all the ratings below it as well.
Q_r is treated differently, using a weighted sum of basis matrices:

$$Q_r = \sum_{t=1}^{n_b} a_{rt} P_t$$

With a given number of basis matrices n_b and trainable coefficients a_{rt}.

The paper's novelty

This paper was the first to introduce **graph auto-encoder of user and item nodes together**. Previous papers had used auto-encoders on users or items separately.

The method involved a new and **efficient way to include side-information**.
Older methods that had included this option:

- Matrix factorization methods (i.e. approximate M by a multiplication of two low-rank matrices U, V), where the user and item features were input as vectors (e.g. $M_{ij} = x_i^T U V^T y_j$).
- A method where the side-information was used for regularizing the model.
- Another method which used convolutional neural network on graphs, combined with a recurrent neural network.

As usual for Welling's new ideas, he creates novel models which involve both a new, elegant and basic (in a meaning that it can easily be elaborated. For example, the messages don't have to be in a linear form) idea, with some ideas inspired by other papers (such as the weight-sharing).