Scalable Recommender Systems through Recursive Evidence Chains

Anonymous Author(s)
Affiliation
Address
email

Abstract

Recommender systems can be formulated as a matrix completion problem, where the goal is to estimate missing ratings. A popular matrix completion algorithm is matrix factorization, where ratings are predicted from combining learned user and item parameter vectors. As dataset sizes increase many matrix factorization models suffer from slow rates of convergence, linear parameter scaling and a need to be retrained when new users or items are added. We develop a novel approach to generate user and item vectors on demand from the ratings matrix itself and a fixed pool of parameters. In our approach, each vector is generated using chains of evidence that link them to a small set of learned prototypical user and item latent vectors. We demonstrate that our approach has a number of desirable scaling properties, such as having a constant rate of convergence to a competitive RMSE, requiring the optimization of only a constant number of parameters.

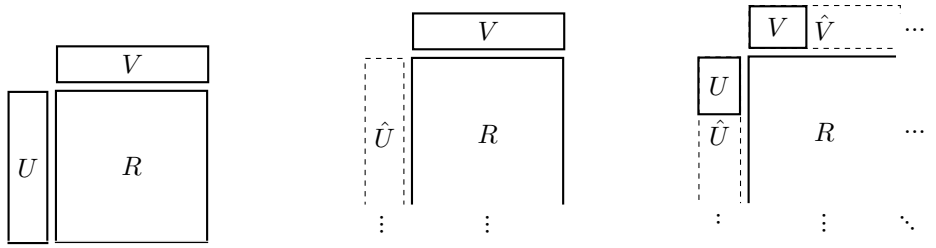
1 Introduction

The central aim of model-based collaborative-filtering methods is to predict a user's rating of an item using a small number of existing item preferences. An effective approach towards this problem is to formulate it as a matrix factorization problem. A ratings matrix $R \in \mathbb{R}^{M \times N}$ can be approximated as a low-rank factorization $R \approx UV^\top$, where $U \in \mathbb{R}^{M \times K}$, $V \in \mathbb{R}^{N \times K}$ and $K \ll \min(M, N)$ [6, 5]. The K-dimensional rows U_i and V_j of U and V are commonly referred to as the latent user and item vectors. Under this framework, each of the ratings entries R_{ij} is approximated by the inner product $U_i V_j^\top$.

Sources of inefficiency for matrix factorization models include linear parameter scaling, slow convergence rate, and an inability to handle online learning. Because ratings are estimated from user and item vectors, the number of parameters to be optimized grows at least linearly with the number of users and items. This in turn slows the rate of convergence for mini-batch based factorization algorithms, as there are more user-item pairs.

Previous work has addressed has addressed these issues, for example in [7], the authors considered a "row-less" architecture where user vectors are generated using a combination of item vectors. In the row-less formulation two things are learned: embeddings for each item, and a neural network to generate user embeddings as necessary as a function of the user's ratings as well as the rated item's embeddings. When generating a rating for a specific user and item, we must first generate the user embedding as a combination of the learned item embeddings. The rating can then be computed as a simple dot product of the generated user embedding and the known item embedding. Since all item embeddings associated with a user are used for every rating prediction, their optimization is tightly coupled and allows for a faster rate of convergence as shown 3. One can imagine a similar column-less formulation in which this dynamic is reversed; user latents are learned as well as a

Preprint. Work in Progress.



(a) Latent factor models (b) Rowless factor model (c) Proposed method

Figure 1.1: Dotted lines denote generated embeddings, as opposed to stored in memory. Ellipses show the direction in which the data can scale without having to add new parameters.

function to generate item latents from them, and item latents are only generated whenever they are needed to estimate a rating.

In this paper, we propose the Recursive Evidence Chains (REC) model. With REC, we do not store the entire latent matrices U and V but rather a small subset of it (the prototypes). A neural network is then learned to recursively generate the latent representations for non-prototypical users and items as necessary. This representation generation algorithm creates a highly coupled optimization problem, which allows REC to converge quickly regardless of data size. Since almost all embeddings are generated using a small pool of base-case embeddings, REC can achieve a good level of performance using a constant parameter size whilst also handling new users and new items well without retraining. Additionally, it is worth noting that the approach of recursively generating latents is not limited to the scope of recommendation systems, and can be applied to a broad range of problems.

2 Recursive Evidence Chains

We now propose a general framework for combining both rowless and columnless matrix factorization. Instead of allocating full embeddings for only columns or only rows, we pick a constant number of prototype users $\mathbf{P_u}$ and prototype items $\mathbf{P_v}$ such that $|\mathbf{P_u}| \ll M$ and $|\mathbf{P_v}| \ll N$. We select our prototypes to be the users and items with the most ratings. To aid with visualization, we sort our matrices such that the $\mathbf{P_u}$ users are the first $|\mathbf{P_u}|$ rows of the matrix, and similarly so for items.

Each prototype user and item receives an embedding, while all non-prototypical users u_i and non-prototypical items v_j are generated on-demand. Intuitively, because our method is row-less, we are able to predict each missing u_i embedding as a function of the ratings of that user and the embeddings of each item the user rated. Since our method is also column-less, we predict each missing v_j as a function of the ratings of that item and the embeddings of each user who rated that item. This introduces a recursion until we reach the prototypical users and items.

2.1 Predicting latent factors in REC

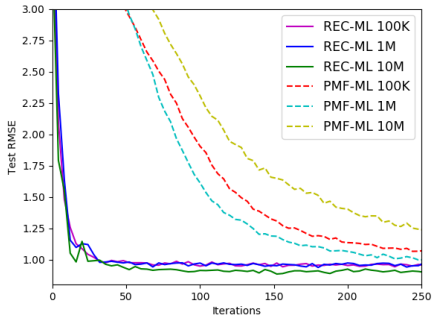
We denote by f_φ and f_ψ two feed-forward networks parametrized respectively by $\varphi \in \mathbb{R}^d$ and $\psi \in \mathbb{R}^p$. The former maps an item latent factor and rating pair to a user latent factor, while the latter maps a user latent factor and rating pair to an item latent factor. When collecting evidence to generate an embedding, we decompose the problem into whether or not our evidence stems from a prototypical user or item. For users, we define the latent factors as

$$U_i = \begin{cases} u_i & \text{if } i \in \mathbf{P_u} \\ \hat{u}_i = \frac{1}{|\Omega_i|} \sum_{\ell \in \Omega_i} f_\varphi\left(R_{i\ell}, V_\ell\right) & \text{otherwise.} \end{cases} \tag{2.1}$$

where we recall that Ω_i is the set of rated items by the i-th user. Similarly, for items, we have the function V_j:

$$V_j = \begin{cases} v_j & \text{if } j \in \mathbf{P_v} \\ \hat{v}_j = \frac{1}{|\overline{\Omega}_j|} \sum_{\ell \in \overline{\Omega}_j} f_\psi\left(R_{\ell j}, U_\ell\right) & \text{otherwise,} \end{cases} \tag{2.2}$$

where $\overline{\Omega}_j$ is the set of users that gave a rating for item j.



(a) Performance of REC and PMF on ML-100K, ML-1M and ML-10M for < 250 iterations.

Model	ML-100K	ML-1M
PMF [4]	0.952	0.883
NNMF [1]	0.903	0.843
Biased-MF [3]	0.911	0.852
CF-NADE [8]	-	0.829
REC	0.910	0.882

(b) Test RMSE results on ML-100K and ML-1M for various models. Scores reported for PMF, NNMF, Biased-MF (ML-100K/ML-1M) were taken from [1]. Scores reported for CF-NADE were taken from [8].

Figure 3.1: Performance of REC on RMSE.

We also implemented a depth limit parameter in order to prevent cycles.

Training REC In order to speed up training performance we introduce a pretraining phase where we perform PMF on only the prototypical block before training on all parameters. We found that this short constant-time (w.r.t dataset size, given fixed prototype size) procedure sped up the optimization process considerably.

3 Experiments

Implementation details. The standard configuration we took for REC, unless otherwise specified, is as follows: the number of prototypical users and items $|\mathbf{P_u}|$ and $|\mathbf{P_v}|$ are set to 50. We use a 3-layer feed-forward neural network with 200 neurons for each hidden layer. The activation functions of each net are ReLUs, with exception of the last layer which is taken to be linear. We set the default batch size to be 1000 and use the Adam optimizer [2] with a learning rate of 10^{-3} and regularization parameter $\lambda = 10^{-5}$. Our metric of evaluation across all experiments is test root-mean square error (RMSE).

Test RMSE comparisons. We begin by comparing the performance of REC to the following collaborative-filtering algorithms: PMF [4], NNMF [1], Biased-MF [3], and CF-NADE [8]. In all three experiments, we train for 2000 iterations. Table 3.1b gives the comparison scores. We find that for ML-100K, REC achieves a test RMSE performance comparable to standard collaborative-filtering algorithms. For ML-1M, while our method does not reach state-of-art performance, especially compared to CF-NADE, we believe that this can be substantially improved if we tune the number of prototype users $\mathbf{P_u}$ and items $\mathbf{P_v}$.

3.1 Coupling of parameters leads to faster convergence

We compare the performance of REC to PMF in the early stages of the training process. For PMF, we choose batch-sizes of 1000, 5000, and 10000 for ML-100K, ML-1M, and ML-10M respectively. The reason behind increasing batch-sizes is that for PMF, as opposed to REC, larger batch-sizes are required to achieve accurate training as the dataset size grows. In Figure 3.1a, we see that for all three datasets, REC converges to a RMSE < 1 in under 30 iterations. Furthermore, we find that REC learns very well in the early training process. On the other hand, PMF cannot reach a RMSE < 1 in 250 iterations; in fact, it only attains this at around iterations 400 and 360 for ML-100K and ML-1M, respectively.

As demonstrated in Figure 3.1a, REC has similar test RMSE convergence curves across increasingly large datasets while maintaining a constant number of parameters. This highlights the attractive scalability properties of REC. In contrast, we observe that the number of iterations it takes for PMF to converge depends on the size of the dataset.

References

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

[2] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, December 2014.

[3] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

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

[5] Jasson DM Rennie and Nathan Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, pages 713–719. ACM, 2005.

[6] Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 720–727, 2003.

[7] Patrick Verga, Arvind Neelakantan, and Andrew McCallum. Generalizing to unseen entities and entity pairs with row-less universal schema. *arXiv preprint arXiv:1606.05804*, 2016.

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