Machine Learning (2019) 108:1395–1420
https://doi.org/10.1007/s10994-019-05801-6

Deep collective matrix factorization for augmented multi-view learning

Ragunathan Mariappan[1] · **Vaibhav Rajan**[1]

Received: 26 November 2018 / Accepted: 20 April 2019 / Published online: 17 May 2019

Abstract
Learning by integrating multiple heterogeneous data sources is a common requirement in many tasks. Collective Matrix Factorization (CMF) is a technique to learn shared latent representations from arbitrary collections of matrices. It can be used to simultaneously complete one or more matrices, for predicting the unknown entries. Classical CMF methods assume linearity in the interaction of latent factors which can be restrictive and fails to capture complex non-linear interactions. In this paper, we develop the first deep-learning based method, called dCMF, for unsupervised learning of multiple shared representations, that can model such non-linear interactions, from an arbitrary collection of matrices. We address optimization challenges that arise due to dependencies between shared representations through multi-task Bayesian optimization and design an acquisition function adapted for collective learning of hyperparameters. Our experiments show that dCMF significantly outperforms previous CMF algorithms in integrating heterogeneous data for predictive modeling. Further, on two tasks—recommendation and prediction of gene-disease association—dCMF outperforms state-of-the-art matrix completion algorithms that can utilize auxiliary sources of information.

Keywords Collective Matrix Factorization · Deep learning · Augmented multi-view learning · Bayesian optimization · Recommendation · Gene-disease prioritization

1 Introduction

Pairwise relational data, found in many domains, can be represented as matrices. Matrix completion, that predicts unknown entries in a matrix, is widely used in many applications, e.g. in recommender systems (Koren et al. 2009), computer vision (Hu et al. 2013) and bioinformatics (Natarajan and Dhillon 2014), to name a few. Often, the matrices are high-dimensional, sparse, and with inherent redundancies. Sufficient information may be present

Editors: Karsten Borgwardt, Po-Ling Loh, Evimaria Terzi, and Antti Ukkonen.

✉ Vaibhav Rajan
vaibhav.rajan@nus.edu.sg

[1] School of Computing, National University of Singapore, Singapore, Singapore

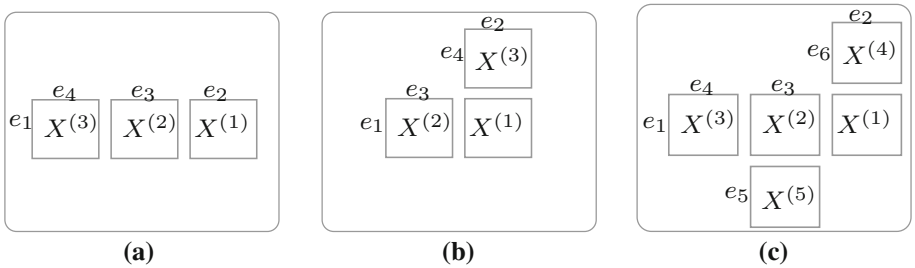

Fig. 1 Examples of **a** multi-view setting, **b** recommendation setting: 4 entities e_1, e_2, e_3, e_4 and 3 relations between the entities, matrices $X^{(1)}, X^{(2)}, X^{(3)}$; **c** augmented multi-view setting: 6 entities $e_1, e_2, e_3, e_4, e_5, e_6$ and 5 relations between the entities, matrices $X^{(1)}, X^{(2)}, X^{(3)}, X^{(4)}, X^{(5)}$

in latent substructures, that can be approximated through low-rank factorizations, and used in predictive models.

When information from multiple heterogeneous sources is available, predictive models benefit from latent representations that model correlated shared structure. In multi-view learning, *views* refer to measurements for the same subjects, that differ in source, datatype or modality. Each matrix, representing a view, has a relationship between two *entity types*, along each matrix dimension, and entity types may be involved in multiple views. For example, in Fig. 1a, entity e_1 could be patients and clinical data from three different sources (notes $X^{(1)}$, images $X^{(2)}$, and diagnoses $X^{(3)}$) may be used to obtain patient representations for modeling risk of diseases. When auxiliary information about multiple entity types are present, they could be effectively utilized to obtain latent representations. For example, in hybrid recommender systems, where side information matrices about users and movies are used in addition to the historical user-rating matrix to obtain user and movie representations (in Fig. 1b, $X^{(1)}$ is the user-rating matrix, $X^{(2)}$ has user-features and $X^{(3)}$ has movie-features). These latent representations are then used to recommend movies to users.

Collective Matrix Factorization (CMF) is a general technique to learn shared representations from arbitrary collections of heterogeneous data sources (Singh and Gordon 2008). CMF collectively factorizes the input set of matrices to learn a low-rank latent representation for each entity type from *all* the views in which the entity type is present. It can be used to simultaneously complete one or more matrices in the collection of matrices. Since CMF models arbitrary collections of matrices, this setting is also referred to as *augmented multi-view learning* (Klami et al. 2014). Figure 1c shows an example. Note that the augmented multi-view setting can generalize to any collection of matrices and subsumes the multi-view and recommendation settings.

Classical matrix factorization based approaches assume linearity in the interaction of latent factors which can be restrictive and fails to capture complex non-linear interactions. Modeling such non-linearities through neural models have significantly improved multi-view learning approaches with two views (Andrew et al. 2013; Wang et al. 2015) and multiple (but not augmented) views (Ngiam et al. 2011; Wang et al. 2017). A common approach is the use of deep autoencoders to obtain shared representations that form latent factors. However these methods cannot generalize to arbitrary collections of matrices. To use shared representations from autoencoders within CMF, learning would involve optimizing entity-specific autoencoder reconstruction losses as well as view-specific matrix reconstruction losses. The latter induces dependencies between the autoencoder networks that may result in simultaneous under-fitting in some networks and over-fitting in other networks (described

in Sect. 5.2). This makes collective learning of all latent representations challenging and, to scale to arbitrary collections of matrices, necessitates automatic hyperparameter selection.

In this paper, we develop dCMF, a deep learning architecture for collective factorization of arbitrary collection of matrices, that is, to our knowledge, the first deep augmented multi-view learning method. dCMF overcomes the limitation of previous CMF models that cannot capture complex non-linear interactions of latent factors. We address optimization challenges that arise due to dependencies between autoencoder representations within dCMF, through multi-task Bayesian optimization and an acquisition function that is adapted for collective learning of hyperparameters. Our experiments show that dCMF is better than previous CMF algorithms at integrating heterogeneous data for predictive modeling and significantly outperforms them on synthetic and real data. We demonstrate two applications of dCMF in matrix completion tasks: movie recommendations and prediction of gene-disease associations. In both tasks, dCMF significantly outperforms state-of-the-art algorithms on benchmark datasets.

2 Related work

2.1 Multi-view learning

Canonical Correlation Analysis (CCA) (Hotelling 1936) that learns maximally correlated features from two views has been the basis for many multi-view learning methods. Several variants have been studied that illustrate the benefits of multi-view learning over models that learn from concatenation of features in the views (Hardoon et al. 2004). *Augmented multi-view learning* generalizes the setting to arbitrary collections of matrices where latent factors are learnt through Collective Matrix Factorization (CMF) (Singh and Gordon 2008), thus enabling learning from auxiliary data sources. To model view-specific noise and allow a subset of matrices with shared structure independent of others, a group-wise sparse formulation was designed in gCMF (Klami et al. 2014).

2.2 Deep models for multi-view learning

Deep learning based extensions of some of these models have been developed. Multi-modal autoencoders were designed to learn shared representations from multiple views, called SplitAE (Ngiam et al. 2011). DCCA is a deep extension of CCA that maximizes correlation between non-linearly extracted features from each view (Andrew et al. 2013). DCCAE combines DCCA and SplitAE to get shared representations by maximizing a CCA-based objective (Wang et al. 2015). Thus DCCAE is designed for a multi-view setup with 2 views and 3 entities (e.g., only views $X^{(1)}$, $X^{(2)}$ in Fig. 1a) and obtains shared representations by maximizing the canonical correlation between the unshared entities (e_2, e_3), regularized by the autoencoder reconstruction error. All these deep models benefit from the non-linearities captured through the deep representations but are restricted to two input views.

There are supervised learning methods that model multi-view data. For example, in CDMF (Wang et al. 2017), a deep learning based solution is developed for the multi-view case (not augmented multi-view) where each view differs in its modality. They factorize each view into a modality-invariant factor and a modality-specific factor, where the latter is learnt using a neural network. Being supervised, entity representation learning in these methods is guided

by application-specific labels. None of these deep learning approaches can be used to model an arbitrary collection of matrices for unsupervised augmented multi-view learning.

2.3 Heterogeneous information networks

Another approach to representation of (augmented) multi-view data is through Heterogeneous Information Networks (HIN). A HIN contains multiple types of nodes (entities) and multiple types of edges (relations between entities). HIN embeddings obtain vectorial representations of nodes that preserve global structural properties of the network. Such embeddings can then be used for link prediction, node classification or clustering. They have also been used for recommendation (e.g., by Han et al. 2018; Shi et al. 2019). There are several approaches to learn HIN embeddings, including some that use deep neural networks (Chang et al. 2015). HIN models multiple relations between the same entities elegantly through multiple edge types while CMF-based methods would require specifying a matrix-specific link functions to model such relations. However, integrating side information such as node and edge attributes in HIN embeddings is challenging (Cui et al. 2019). In contrast, matrix-based approaches naturally model edge attributes (as matrix entries) of any type—binary, ordinal or real-valued. More details can be found in recent surveys (Shi and Philip 2017; Cui et al. 2019).

3 Background

3.1 Matrix factorization

For a matrix $X \in \mathbb{R}^{m \times n}$, a low-rank factorization aims to obtain latent factors $U^{(1)} \in \mathbb{R}^{m \times K}$, $U^{(2)} \in \mathbb{R}^{n \times K}$, such that $X \approx U^{(1)} \cdot U^{(2)^T}$, where the $K < \min(m, n)$ (see Fig. 2a). The factors are learnt by solving the optimization problem: $\text{argmin}_{U,V} L(X, U^{(1)} \cdot U^{(2)^T})$, where L denotes a loss function (e.g., $||X - U^{(1)} \cdot U^{(2)^T}||_F^2$, where $||.||_F^2$ is the Frobenius norm). Collaborative Filtering, for recommendations, uses such an approach where X is the rating matrix. A common approach to solving this is through a convex relaxation that minimizes the nuclear norm (the sum of singular values) of $U^{(1)} \cdot U^{(2)^T}$, which is equivalent to solving: $\min_{X=U^{(1)} \cdot U^{(2)^T}} ||U^{(1)}||_F^2 + ||U^{(2)}||_F^2$ (Srebro and Shraibman 2005).

3.2 Collective Matrix Factorization (CMF)

CMF aims to jointly obtain low-rank factorizations of M matrices (indexed by m), $X^{(m)} = [x_{ij}^{(m)}]$, that describe relationships between E entities $(e_1, \ldots e_E)$, each with dimension d_e. The entities corresponding to the rows and columns of the m^{th} matrix are denoted by r_m and c_m respectively. Figure 1 shows three examples. Each matrix is approximated by product of low rank-K factors that form the representations of the associated row and column entities: $X^{(m)} \approx U^{(r_m)} U^{(c_m)^T}$ where $U^{(e)} = [u_{ik}^{(e)}] \in \mathbb{R}^{d_e \times K}$ is the low-rank matrix for entity type e. Any two matrices sharing the same entity type use the same low-rank representations as part of the approximation, which enables sharing information. For example, in Fig. 1b, the same latent factor $U^{(e_1)}$ is used to reconstruct the three matrices $X^{(1)} \approx U^{(e_1)} U^{(e_2)^T}$, $X^{(2)} \approx U^{(e_1)} U^{(e_3)^T}$, $X^{(3)} \approx U^{(e_1)} U^{(e_4)^T}$. The latent factors are learnt by solving the optimization problem:

$$\text{argmin}_{\{U^{(e)} \in \mathbb{R}^{d_e \times K}\}_{e \in E}} \sum_{m=1}^{M} L\left(X^{(m)}, U^{(r_m)} U^{(c_m)^T}\right) + \sum_{e=1}^{E} \mathcal{R}\left(U^{(e)}\right) \quad (1)$$

where M is the total number of input matrices, E is the total number of entities and $\mathcal{R}$ is a regularizer. For $\mathcal{R}(U^{(e)}) = \lambda ||U^{(e)}||_F^2$, Bouchard et al. (2013) show that this formulation generalizes the nuclear norm for a single matrix to a *collective nuclear norm* defined on an arbitrary set of matrices (with the reasonable assumption that a pair of entity types do not share more than one view). Although this is a non-convex problem, in practice, solutions obtained through Stochastic Gradient Descent yield good performance (Bouchard et al. 2013).

4 Problem statement

Given M matrices (indexed by m), $X^{(m)} = [x_{ij}^{(m)}]$, that describe relationships between E entities $(e_1, \ldots e_E)$, each with dimension d_e, we aim to jointly obtain latent representations of each entity $U^{(e_i)}$ and low-rank factorizations of each matrix $X^{(m)} \approx U^{(r_m)} \cdot U^{(c_m)^T}$, such that $U^{(e_i)} = f_\theta^{e_i}([X]_{e_i})$ where $f^{(.)}$ is an entity-specific non-linear transformation parameterized by θ and $[X]_{e_i}$ denotes all matrices in the collection that contains a relationship of entity e_i. The entities corresponding to the rows and columns of the m^{th} matrix are denoted by r_m and c_m respectively.

We assume that the relationship between these matrices and the constituent E entities is provided as a bipartite entity-matrix relationship graph $G(V_E, V_M, D)$, where vertices V_E, V_M represent entities and matrices respectively. Edges $(e_i, X^{(m)}), (e_j, X^{(m)}) \in D$ are present if there exists an input matrix $X^{(m)} \in V_M$ capturing the relationship between the entities $e_i, e_j \in V_E$ (see Fig. 2a, b).

5 Deep collective matrix factorization (dCMF)

Similar to the formulation in (1), we aim to learn entity-specific latent representations by solving the following optimization problem:

$$\text{argmin}_{\theta, \{U^{(e)} \in \mathbb{R}^{d_e \times K}\}_{e \in E}} \sum_{m=1}^{M} L\left(X^{(m)}, f_\theta^{r_m}([X]_{r_m}) \cdot \left(f_\theta^{c_m}([X]_{c_m})\right)^T\right) + \sum_{e=1}^{E} \mathcal{R}\left(U^{(e)}\right) \quad (2)$$

We now describe how we model f to induce non-linearity and $\mathcal{R}$ for regularization, to develop the dCMF model.

There are several ways to model non-linearity (f). Common choices in unsupervised learning include kernels, Restricted Boltzmann machine (RBM) (Hinton and Salakhutdinov 2006) and Autoencoders (Ngiam et al. 2011). Kernel machines are not recommended for representation learning due to its over-reliance on the smoothness assumption, i.e., the value of the learned function at a data point depends mostly on the training examples that are closest to it (Bengio et al. 2013). Instead Bengio et al. (2013) advocate nonparametric models such as neural networks whose model complexity can be controlled through hyperparameters. RBM is a stochastic generative model with intractable maximum likelihood function and complex training procedures. In contrast, autoencoders can be trained efficiently and have been used effectively for multi-view representation learning (Wang et al. 2015). The use of autoencoders to model f also allows us to use the autoencoder reconstruction loss as the

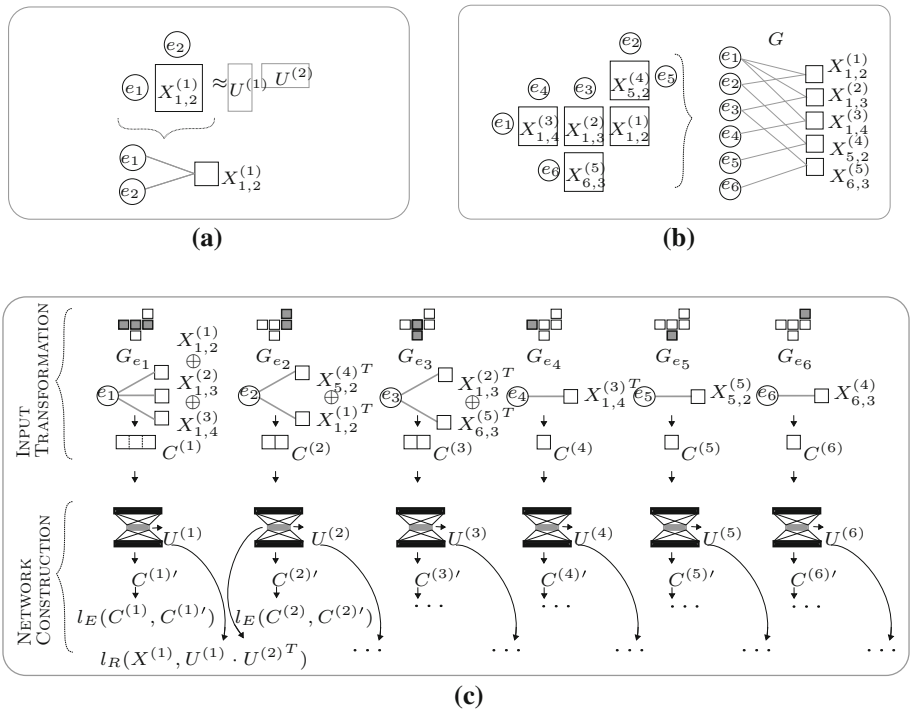

Fig. 2 **a** Entity-matrix relationship graph for a single view. **b** A collection of views and its entity-matrix relationship graph [square nodes: matrices, circular nodes: entities]. **c** dCMF model construction for the example in (**b**)

regularizer $\mathcal{R}$. Such autoencoder-based regularization has been used in DCCAE (Wang et al. 2015) for multi-view learning from two views. There are several autoencoder architectures (Goodfellow et al. 2016) that can be used; we leave that investigation for future work. Here we choose the simplest architecture with multiple fully-connected hidden layers.

Since entities can be shared across matrices, we have to obtain autoencoder-based shared representations from multiple matrices simultaneously to enable sharing of information across matrices. Further, this must be done in a way that can be generalized to an arbitrary collection of input matrices. To accomplish this we design a neural architecture as described in the following section. Generalizing to augmented multi-view learning, compared to relatively simpler settings like multi-view learning (Fig. 1a), leads to non-trivial optimization challenges that we discuss and address in later sections.

5.1 Model construction and training

There are two steps in dCMF model construction:

1. Input Transformation: For each entity e_i, we create a new matrix $C^{(i)}$, that we call *concatenated matrix*, by concatenating all the matrices containing entity e_i, i.e., all the matrices that are neighbors of e_i in $G(V_E, V_M, D)$. Note that we transform M input matrices to E concatenated matrices, and a single input matrix ($X^{(m)}$) may be used in

multiple concatenated matrices ($C^{(i)}$). The concatenation ensures that for each entity, we use the information from all available input matrices to learn its representation.

2. Network Construction: We then use E (dependent) autoencoders to obtain the latent factors $U^{(i)}$ from the concatenated matrices $C^{(i)}$. For each entity e_i our network has an autoencoder whose input is $C^{(i)}$, and the decoding is represented by $C^{(i)'}$. The bottleneck or encoding of each autoencoder, after training, forms the latent factor $U^{(i)}$. The latent factors are learnt by training all the autoencoders together by solving:

$$\text{argmin}_{\{U^{(e)} \in \mathbb{R}^{d_e \times K}\}_E} \sum_{m=1}^{M} l_R \left(X^{(m)}, X^{(m)\prime} \right) + \sum_{e=1}^{E} l_E (C^{(e)}, C^{(e)\prime}) \tag{3}$$

where l_E is the reconstruction loss between the autoencoder's input $C^{(i)}$ and the decoding $C^{(i)\prime}$; l_R is the matrix reconstruction loss, where the reconstructed matrix $X^{(m)\prime} = U^{(r_m)} \cdot U^{(c_m)^T}$ of the view $X^{(m)}$ is obtained by multiplying the associated row and column entity representations $U^{(r_m)}$ and $U^{(c_m)}$. We call the summations in Eq. (3) the matrix reconstruction loss ($\mathcal{L}_R$) and autoencoder reconstruction loss ($\mathcal{L}_E$) respectively.

Thus, while CMF factorizes each matrix as $X^{(m)} \approx U^{(r_m)} \cdot U^{(c_m)^T}$, dCMF performs non-linear factorization using autoencoders as $X^{(m)} \approx g_\theta^{(r_m)}(C^{(r_m)}) \cdot g_\theta^{(c_m)}(C^{(c_m)^T})$, where g_θ is the encoder corresponding to the entity, with parameter set θ, obtained by collectively minimizing the sum of all the matrix reconstruction and autoencoder reconstruction losses as described above.

Illustration In Fig. 2a we show a single matrix $X^{(1)}_{1,2}$ and its two entities e_1 and e_2. The corresponding entity-matrix graph below has 2 circular nodes for two entities and 1 square node for the matrix. In Fig. 2b, we show the graph for the collection of 5 matrices and 6 entities (e_1 to e_6) (from Fig. 1c). Consider, for instance, the entity e_1. There exists 3 matrices with relationships of entity e_1 with three other entities e_2, e_3 & e_4. Hence there are 3 edges from the node representing $e_1 \in V_E$ to the nodes $X^{(1)}, X^{(2)}, X^{(3)} \in V_M$.

We illustrate dCMF model construction in Fig. 2c for the example from Fig. 2b. We construct $E = 6$ autoencoders, one per entity. The autoencoder construction for entity e_1 is illustrated in the first column of Fig. 2c. We show the subgraph G_{e_1} consisting of 3 edges corresponding to the 3 views $X^{(1)}_{1,2}$, $X^{(2)}_{1,3}$, $X^{(3)}_{1,4}$. Hence $C^{(1)} = X^{(1)}_{1,2} \oplus X^{(2)}_{1,3} \oplus X^{(3)}_{1,4}$, where $\oplus$ denotes row or columnwise concatenation of the matrices. To pictorially illustrate this we show a miniature of the setup in Fig. 2b on top of each column in Fig. 2c, greying out the boxes corresponding to the matrices involved in $C^{(i)}$ construction. We also show $C^{(i)}$ as a block containing concatenated boxes (equal to the number of matrices $C^{(i)}$ is composed of) with a label $C^{(i)}$ below each of the subgraphs which is also the input to the corresponding autoencoder. Similarly we construct the autoencoder for e_2 and the input $C^{(2)}$ by concatenating matrices corresponding to the edges of G_{e_2} as illustrated in the second column of Fig. 2c. Thus we have 6 columns in Fig. 2c for each of the 6 entities in setup of Fig. 2b. To avoid clutter in Fig. 2c, we show only two examples of the autoencoder reconstruction loss l_E for entities e_1 and e_2 and one example matrix reconstruction loss l_R for the matrix $X^{(1)}$. In total there are $E = 6$ autoencoder reconstruction loss terms and $M = 5$ matrix reconstruction loss terms. Note that this construction can be generalized to any number of entities and matrices.

Notice that the input dimension, which depends on $C^{(e)}$, is different for each autoencoder and the bottleneck layer dimension (the chosen low rank K) is common across all autoencoders. So, the number of layers for each autoencoder is a hyperparameter that is chosen adaptively for each autoencoder as follows: We start with the autoencoder's input dimension

obtained from $C^{(i)}$ and multiply it with a fraction f_k (a hyperparameter) to get the size of the first encoding layer. We then multiply the first encoding layer's size again with f_k to get the second encoding layer's size. We repeat this and continue to add layers until we cross K which is the common encoding/bottleneck size for all the autoencoders. We add a decoding layer corresponding to each of the encoding layer. This approach helps to adaptively decide the number of layers and their size for each autoencoder based on their input size i.e. more layers are added for inputs of higher dimension and vice-versa.

5.2 Optimization

dCMF learns the representation of all the input entities collectively by training all E autoencoders simultaneously. The objective function $\mathcal{L}_R + \mathcal{L}_E$ is non-convex. Note that although the autoencoder reconstruction losses in $\mathcal{L}_E$ are independent to each of the autoencoders, the matrix reconstruction losses in $\mathcal{L}_R$ are dependent on multiple autoencoders, through the latent factors used in matrix reconstruction. Below we describe the problems that arise specifically due to the augmented multi-view setup and how we address them.

Entity size, shape and interactions The *entity-size* $d_{e_i} = |e_i|$ is the number of instances of an entity e_i i.e. count of rows/columns depending on whether it is the row/column entity of the matrix. Recall that the *concatenated matrix* $C^{(i)}$ is constructed by concatenation of all the matrices $X^{(m)}$ associated with an entity e_i. The *entity-shape* of an entity e_i is (p, q), where p and q are the row and column dimensions of the corresponding matrix $C^{(i)}$ respectively. Thus by virtue of dCMF's model construction p and q becomes the size and feature dimension of the input to the autoencoder for learning the entity representation. Let *entity-interactions* N_{e_i} be the number of matrices sharing entity e_i. E.g., consider the entity e_1 in Fig. 3a. The matrix $C^{(1)}$ is constructed by row-wise concatenation of $X^{(1)}_{1,2}$, $X^{(3)}_{1,3}$. Here $q = |e_2| + |e_3|$, $p = |e_1|$ and $N_{e_1} = 2$.

If $q \gg p$ for an entity e then there may be under-fitting in the learning of $U^{(e)}$, since the dimensionality of the autoencoder input $C^{(i)}$ is high and the number of samples are few. The situation worsens if N_e is large i.e. the entity is related to many other entities, leading to increasing dimensions in $C^{(i)}$. Pre-training or increasing the number of hidden layers of the corresponding autoencoder may help in such conditions. On the other hand, if $p \gg q$ then it may lead to over-fitting, since the dimensionality of $C^{(i)}$ is low and the number of samples are high. This may happen when the *entity-size* of each of the associated N_e entities are all small. This can be addressed by adding suitable regularizers or through early stopping during the training of the corresponding autoencoder. A schematic is shown in Fig. 3a. Thus, $|e|$, p, q and N_e can all influence dCMF performance and require careful hyperparameter selection, separately for each autoencoder, noting that in dCMF, the autoencoder losses are not independent since $\mathcal{L}_R$ depends on multiple autoencoders collectively. Since manually tuning these hyperparameters is infeasible for arbitrary collection of matrices, we address this problem through Bayesian optimization.

Mixed sparsity levels An augmented multi-view setup may contain both sparse and dense matrices. E.g., in recommendation, the rating matrix is sparse but side information matrices may be dense. If an autoencoder's input is a concatenation of both sparse and dense matrices, then the learnt representation will be dense with potentially many small values in order to bring down the autoencoder reconstruction loss l_E. This results in a higher matrix reconstruction error $\mathcal{L}_R$ for the sparse matrices as the corresponding reconstructed matrices using dense row and column entity representations are not sparse. To handle this one can attempt to learn

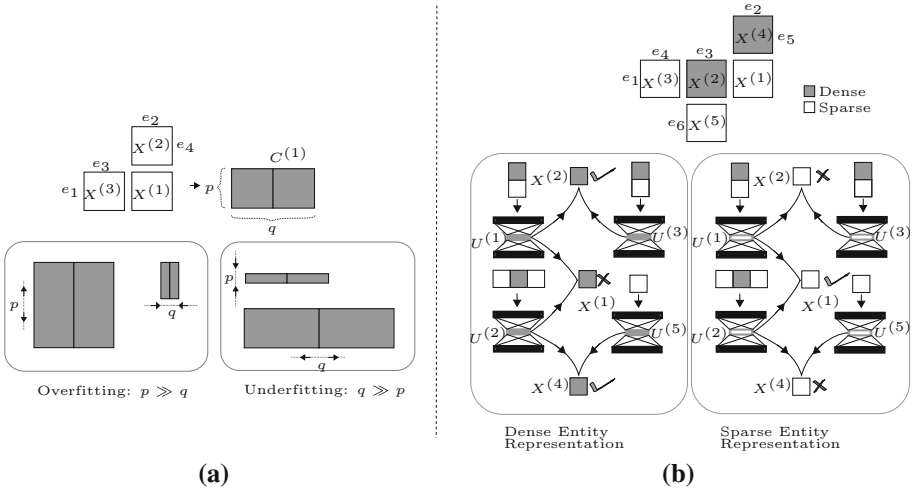

Fig. 3 **a** Entity size, shape and interactions in dCMF. **b** Effect of mixed sparsity levels in dCMF

sparse representations using sparse autoencoders. But any dense matrix associated with the entity with sparse representation will suffer from higher $\mathcal{L}_R$. We illustrate this in Fig. 3b for the example in Fig. 2b with sparse $X^{(1)}$, $X^{(3)}$, $X^{(5)}$ and dense $X^{(2)}$, $X^{(4)}$. We can see that if the learnt representations for $U^{(.)}$ are dense then it favours reconstruction of $X^{(2)}$ and $X^{(4)}$ but not $X^{(1)}$. On the other-hand if the representations learnt are sparse then it favours the reconstruction of $X^{(1)}$ but not $X^{(2)}$ and $X^{(4)}$.

Other autoencoder-based unsupervised deep learning algorithms do not face these challenges. For instance, the Improved Deep Embedded Clustering (IDEC) algorithm (Guo et al. 2017) aims to learn entity representations that favour clustering, using autoencoders, by optimizing a clustering oriented loss regularized by the autoencoder reconstruction loss. Although the objective function is similar, since they have a single autoencoder, problems due to multiple dependencies (through matrix reconstruction loss in dCMF) do not arise.

Hyperparameter tuning

Manual tuning is infeasible for complex models with large number of hyperparameters and complex models; and random search or grid search based approaches (Bergstra and Bengio 2012) are either too time-consuming or not effective. In the case of dCMF, there are many hyperparameters related to (a) Model construction and (b) Optimization. Hence dCMF training devoid of hyperparameter tuning usually results in poor performance due to the dependencies that may result in simultaneous over/under-fitting in different autoencoders as described earlier. See "Appendix A" for a list of hyperparameters.

Hyperparameter tuning can be formulated as an optimization problem:

$$\mathbf{p}^* = \underset{\mathbf{p} \in \mathcal{P}}{\operatorname{argmin}}\ \mathbf{L}(\mathbf{p}) \tag{4}$$

where $\mathbf{p}^*$, from the B-dimensional hyperparameter space $\mathcal{P}$ denotes the optimal values for all B hyperparameters and the objective function is the collective dCMF loss $\mathbf{L} = \mathcal{L}_E + \mathcal{L}_R$. Note that model training optimizes $\mathbf{L}$ with respect to the model parameters, and not the hyperparameters. The functional form of $\mathbf{L}(\mathbf{p})$ is not known but its value, for any given input

$\mathbf{p}$, can be computed. Thus, this is a black-box function optimization problem that has been successfully addressed through Bayesian optimization (Snoek et al. 2012).

Multi-task Bayesian optimization (MTBO) We briefly explain multi-task Bayesian optimization (MTBO) (for more details see Bergstra et al. 2011; Snoek et al. 2012; Swersky et al. 2013) before describing how dCMF adopts MTBO.

Bayesian optimization (BO) is a sequential model-based approach for solving the black-box function optimization problem. The key idea is to learn a *surrogate model* $\mathcal{M}$ that captures our beliefs about the unknown objective function ($\mathbf{L}(\mathbf{p})$). This model is learnt from *data*, $\mathcal{D}_n = (\mathbf{p}_1, \mathbf{L}_1), \ldots, (\mathbf{p}_n, \mathbf{L}_n)$, that consists of sequential evaluations of $\mathbf{L}(\mathbf{p})$ for different values of $\mathbf{p}$. Generating this data sequence requires making the decision of which $\mathbf{p}$ to evaluate next, at each step. This decision is made through an *acquisition function* α. These functions are designed to have optima at points with high uncertainty in the surrogate model (thus facilitating exploration) and/or at points with high predictive values in the surrogate model (thus facilitating exploitation). Acquisition functions have known functional forms and are usually easier to optimize than the original objective function. The surrogate model is updated sequentially with each observed data point. Over multiple steps, the landscape of the black-box function ($\mathbf{L}(\mathbf{p})$) is learnt by the surrogate model and can be exploited by the acquisition function to yield values of $\mathbf{p}$ that are, on average, closer to the optimal $\mathbf{p}^*$.

Many different choices of surrogate models and acquisition functions have been explored. Gaussian processes (GP) can be used to model priors over functions and are closed under sampling which makes them an elegant choice for a surrogate model in BO—after each data point is generated (using the acquisition function), the updated model is also a GP with updated mean and covariance functions. They have been successfully used in BO for hyperparameter optimization (Bergstra et al. 2011; Snoek et al. 2012).

BO $(\mathcal{N}, \boldsymbol{L})$
- **inputs** : dCMF Network, $\mathcal{N}$ and Loss function, $\mathbf{L}$
- **outputs**: Best hyperparameter set $\mathbf{p}^{\#}$ and trained network ($\mathcal{N}$) parameters $\Theta^{\#}$
- Generate Data $\mathcal{D}_m = (\mathbf{p}_i, \mathbf{L}_i, V_i, \Theta_i),\ i = 1, \ldots, m$ from m random samples of $\mathbf{p}$ with corresponding validation set performance: $V_1, \ldots V_m$ and network parameters: $\Theta_1, \ldots \Theta_m$.
- Train MTGP surrogate model on $\mathcal{D}_m$: $\mathcal{M}(\boldsymbol{\mu}, \boldsymbol{\sigma}; \mathcal{D}_m(\mathbf{p}, \mathbf{L}))$
- **for** $n = m+1, m+2, \ldots$ **do**
  - $||\mathbf{L}^*||^1 = \min_{i \le n}\{||\mathbf{L}(\mathbf{p}_i)||^1\}, \quad \mathbf{p}^* = \text{argmin}_{\mathbf{p}_{i \le n}} ||\mathbf{L}(\mathbf{p}_i)||^1$
  - Select new hyperparameters $\mathbf{p}_{n+1} = \text{argmax}\, EI_n(\mathbf{p}_{n+1})$ using $\mu = \boldsymbol{\mu}_{sum}(\mathbf{p}_{n+1}), \sigma = \boldsymbol{\sigma}_{sum}(\mathbf{p}_{n+1}), \mathbf{L}(\mathbf{p}_n^*) = ||\mathbf{L}^*||^1$ in Eq. 5
  - Train network $\mathcal{N}$ with hyperparameters $\mathbf{p}_{n+1}$ (using SGD) to obtain its parameters Θ_{n+1}, loss $\mathbf{L}_{n+1}$ and validation set performance V_{n+1}
  - Augment data $\mathcal{D}_{n+1} = \mathcal{D}_n \cup (\mathbf{p}_{n+1}, \mathbf{L}_{n+1}, V_{n+1}, \Theta_{n+1})$
  - Update surrogate model $\mathcal{M}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ using $\mathcal{D}_{n+1}(\mathbf{p}, \mathbf{L})$
- **end**
- Choose best $\mathbf{p}^{\#}$ and $\Theta^{\#}$ corresponding to best V_i or minimum $\mathbf{L}_i$
- **return** $\boldsymbol{p}^{\#}, \Theta^{\#}$

Algorithm 1: Bayesian optimization for dCMF

A common choice for the acquisition function is *Expected Improvement (EI)* (Jones 2001), that has a closed form for GP, does not require its own tuning parameter and has been shown to perform well in minimization settings (Snoek et al. 2012). EI is the expectation that $\mathbf{p}_{n+1}$ will improve $\mathbf{L}$ (negatively, as we would like to minimize a loss) over $\mathbf{p}_n^*$ which is the best observation from n steps of BO so far, i.e. $\mathbf{p}_n^* = \text{argmin}_{\mathbf{p}_{i \le n}} \mathbf{L}(\mathbf{p}_i)$,

and $EI_n(\mathbf{p}_{n+1}) = E_n[\max\{(\mathbf{L}(\mathbf{p}_n^*) - \mathbf{L}(\mathbf{p}_{n+1})), 0\}]$, where the expectation E_n is under the posterior distribution given evaluations of $\mathbf{L}$ at $\mathbf{p}_1, \ldots, \mathbf{p}_n$. The next value is chosen by $\mathbf{p}_{n+1} = \text{argmax}\, EI_n(\mathbf{p}_{n+1})$. For a GP as $\mathcal{M}$, with predictive variance $\sigma(\mathbf{p}_{n+1}; \mathcal{D}_n, \mathcal{M})$ and predictive mean $\mu(\mathbf{p}_{n+1}; \mathcal{D}_n, \mathcal{M})$:

$$EI_n(\mathbf{p}_{n+1}) = \sigma[\gamma(\mathbf{p}_{n+1})\Phi(\gamma(\mathbf{p}_{n+1})) + \phi(\gamma(\mathbf{p}_{n+1}))] \tag{5}$$

where $\gamma(\mathbf{p}_{n+1}) = (\mathbf{L}(\mathbf{p}_n^*) - \mu)/\sigma$, and Φ and ϕ denote the CDF and PDF of the standard normal distribution respectively.

The extension of GP to vector-valued functions is through Multi-Task Gaussian Processes (MTGP), that can model outputs of multiple correlated tasks (Bonilla et al. 2007). Swersky et al. (2013) demonstrate the advantages of MTGP as a surrogate model in several tasks with multiple dependent loss functions.

MTBO for dCMF A straightforward approach to solve the hyperparameter tuning problem for dCMF (problem (4)) is to use BO with GP as a surrogate model. However, we find that using MTGP within dCMF shows better performance. In dCMF, autoencoder training and matrix reconstruction tasks entail minimizing the sum of losses: $\mathbf{L} = \mathcal{L}_E + \mathcal{L}_R = \sum_{e=1}^{E} l_E^e + \sum_{m=1}^{M} l_R^m$. Considering each of these as separate tasks, we have $E + M$ correlated tasks. We use MTGP as a surrogate model for BO, with the kernel specified through the intrinsic corregionalization model (Coburn 2000): $\mathcal{K}((p,t),(p',t')) = \mathcal{K}_t(t,t') \otimes \mathcal{K}_p(p,p')$ where $\otimes$ denotes the Kronecker product, $\mathcal{K}_p$ is a kernel measuring the similarity between the hyperparameters $\mathbf{p}$ and $\mathcal{K}_t$ is the kernel measuring the similarities between the tasks. To ensure positive semidefiniteness of $\mathcal{K}_t$, it is parameterized through a Cholesky decomposition (Bonilla et al. 2007): $\mathcal{K}_t = GG^T$, where G is lower triangular. To model the dependencies between all the tasks, we initialize $\mathcal{K}_t$ as a unit matrix. Note that for each step in BO, MTGP yields an $(E + M)$-dimensional output.

The EI acquisition function does not directly generalize to the multi-task case. So, Swersky et al. (2013) use a heuristic approach, where a GP prior is used for the average output of the tasks, and the average predictive mean and predictive variance of multiple tasks are used to select the next candidate. Instead, we use the sum of the predictive mean and variance (denoted by $\boldsymbol{\mu}_{sum}$ and $\boldsymbol{\sigma}_{sum}$ respectively) of each task since our final objective is to optimize the sum of losses. The output of the MTGP surrogate model $\mathcal{M}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ is scalarized (a common approach for multi-objective functions, e.g., in Knowles 2006), by using the 1-norm. Denote the best value of the scalarized output by $||\mathbf{L}^*||^1$. Then our EI-based criterion can be computed using $\mu = \boldsymbol{\mu}_{sum}, \sigma = \boldsymbol{\sigma}_{sum}, \mathbf{L}(\mathbf{p}_n^*) = ||\mathbf{L}^*||^1$ in Eq. 5. Essentially, this heuristic chooses the next hyperparameter from regions where the tasks show high total predictive variance (exploration) or high total predictive mean (exploitation). We empirically evaluate this heuristic as the acquisition function with MTGP as the surrogate model and found it to be more effective than GP–based BO and random search for dCMF (see "Appendix A"). Algorithm 1 shows the complete Bayesian optimization strategy using MTGP as the surrogate model and our acquisition function heuristic. The final hyperparameter set may be chosen based on the loss function or validation set performance.

Complete dCMF algorithm

Algorithm 2 shows the complete dCMF algorithm. Unless mentioned otherwise, we use MTGP as the surrogate model with the acquisition function described above for hyperparameter tuning. We use stochastic gradient descent (SGD) for training.

dCMF $(G, \mathcal{X})$
inputs : Entity-matrix relationship graph $G(V_E, V_M, D)$,
Input matrices $\mathcal{X} = X^{(1)}, \ldots, X^{(M)}$
outputs: Entity representations $\mathcal{U} = U^{(1)}, \ldots, U^{(E)}$,
Matrix reconstructions $\mathcal{X}' = X^{(1)'}, \ldots, X^{(M)'}$
// Input Transformation
foreach *entity* $e_i \in V_E$ **do**
  $X_{list} = [X^{(m)}$ **if** $(e_i, X^{(m)}) \in D]$
  Construct *concatenated-matrix* $C^{(i)}$ = **concat**(X_{list})
end
// Network Construction
foreach *entity* $e_i \in V_E$ **do**
  Construct *Autoencoder* $\mathcal{A}^{(i)}$
end
Construct network $\mathcal{N}$ with $\mathcal{A}^{(i)}$ and collective loss $\mathbf{L} = \mathcal{L}_E + \mathcal{L}_R$.
// Training and Hyperparameter Tuning
// Run Algorithm 1 using network $\mathcal{N}$ and loss $\mathbf{L}$
// Obtain best performing parameters $\Theta^{\#}$ and hyperparameters $\mathbf{p}^{\#}$
$\mathbf{p}^{\#}, \Theta^{\#}$ = **BO**$(\mathcal{N},\mathbf{L})$
// Entity representation generation
foreach *entity* e_i *in* V_E **do**
  $U^{(i)} = g^{(i)}_{\mathbf{p}^{\#},\theta^{\#}}(C^{(i)})$
end
// Matrix reconstruction
foreach *matrix* $X^{(m)}$ *in* V_M **do**
  $X^{(m)\prime} = U^{(r_m)} \cdot U^{(c_m)^T}$
end
return $\mathcal{U}, \mathcal{X}'$

Algorithm 2: Deep Collective Matrix Factorization

Loss functions The loss functions $\mathcal{L}_R$ and $\mathcal{L}_E$ measure the model's average performance in reconstructing *all* the entries of the input X and concatenated C matrices respectively. The error metric for $\mathcal{L}_R^{(m)}$ depends on the data type of $X^{(m)}$, e.g., root mean squared error for real values and cross entropy for binary or categorical values. The choice of error metric for $\mathcal{L}_E$ is not straightforward if the concatenated matrix contains multiple data types. One way to address this is to transform the matrices $[X]_e$ associated with entity e, such that $C^{(e)}$ is of single data type (e.g. by scaling or PCA). We could also use multi-modal autoencoder architectures (Ngiam et al. 2011) designed to learn shared representations from multiple views of potentially different data types. In our experiments, we use root mean squared loss (for both $\mathcal{L}_R$ and $\mathcal{L}_E$). The root mean squared loss is more sensitive to larger errors and outliers as desired in the applications we present.

Matrix completion Reconstruction of the matrices is obtained by multiplying the latent representations learnt for the corresponding row and column entities. Note that such a reconstruction yields real numbers that can be ordered and can be interpreted as scores for prediction or ranking tasks.

Time complexity The training time complexity of dCMF is dominated by the autoencoder with largest input C of dimension, say, $m \times d$ and is $\mathcal{O}(mdr)$ where r is the number of neurons in the first layer. For BO, the time complexity is dominated by the matrix inversion step at each step for updating the MTGP model. For $t = E + M$ tasks and n steps in BO, the time complexity is $\mathcal{O}(n(t^3n^3 + mdr))$. For matrix completion, for a given matrix $X =$

$U^{(r)} \cdot U^{(c)^T}$, where $U^{(r)}$ and $U^{(c)}$ are inferred latent factors of dimensions $l \times K$ and $j \times K$, the time complexity is $\mathcal{O}(lKj)$, where K is the assumed low rank.

6 Experiments

We first evaluate the performance of dCMF on various settings of sparsity level, size and shape of matrices, using synthetic data, to validate that dCMF addresses the optimization-related challenges discussed earlier. We then evaluate the performance of dCMF on real-world benchmark datasets for two matrix completion tasks: movie recommendation and prediction of gene-disease association. The source code for dCMF and data for all our experiments are available on our public repository.[1]

6.1 Effects of sparsity, size and shape

We simulated datasets with 4 entities and 3 views based on the recommendation setup (Fig. 1b). We generated $U^{(e_1)}$, $U^{(e_2)}$, $U^{(e_3)}$, $U^{(e_4)}$ with $K = 100$ and the desired dimensions (mentioned below), with values sampled from a uniform distribution ranging between 0 and 1. We constructed the views $X^{(1)}_{m\times n}$, $X^{(2)}_{m\times u}$ and $X^{(3)}_{v\times n}$ using the corresponding factors, where subscripts indicate dimensions. To impart sparsity in a matrix $X^{(m)}$, random entries of the corresponding row/column entity factors $U^{(r_m)}$ or $U^{(c_m)}$ were set to zero until the desired level of sparsity was obtained. We use the Root Mean-Squared Error (RMSE) in predicting the central matrix $R = X^{(1)}$ as the performance measure, $RMSE = \sqrt{\frac{1}{|T|}\sum_{R_{ij}\in T}(R_{ij} - R'_{ij})}$, where R_{ij} is the ground truth and R'_{ij} the corresponding prediction. T denotes the test set. In all experiments we perform 5-fold cross validation over the non-zero entries of the central matrix R.

Sparsity Consider sparsity level as the proportion of zero entries in the central matrix $X^{(1)}$. To illustrate how sparsity impacts the performance of dCMF, we simulated 3 artificial datasets with same dimensions ($m = 1000,\ n = 2000,\ u = 200,\ v = 400$) and increasing sparsity levels 0.3, 0.5 and 0.7. 25,000 non-zero entries randomly chosen from the test fold was used as the test set. This is to ensure that we measure RMSE over the same test set (since varying sparsity levels varies the number of non-zero entries and thereby the test fold size). CMF, gCMF and dCMF were used (with K set to 100) to predict the entries in the test set. No input transformation was performed for these experiments. It can be seen from Fig. 4a that increase in sparsity results in increased RMSE in CMF, gCMF and dCMF, with dCMF consistently outperforming CMF and gCMF.

Size and shape We simulated the first dataset with dimensions $m = 400,\ n = 800,\ u = 80,\ v = 160$. Then we created two other datasets that are 3 and 5 times the size of the first one. Fig. 4b shows the performance of CMF, gCMF and dCMF. We define *imbalance-ratio* of a view with shape $m \times n$ as $\left(1 - \frac{\min(m,n)}{\max(m,n)}\right)$. Thus the *imbalance-ratio* is 0 if $m = n$ and increases otherwise. We created 3 datasets with $n = 2000$, $u = 200$, $v = 400$ and increasing *imbalance-ratios* by varying m: 0.5 ($m = 1000$), 0.75($m = 500$) and 0.875 ($m = 250$). Fig. 4c shows the performance of all three methods (with K set to 100) on these datasets. We find that BO is able to effectively select hyperparameters for different sizes and imbalance ratios and dCMF consistently outperforms CMF and gCMF in all the settings.

[1] https://bitbucket.org/cdal/dcmf.

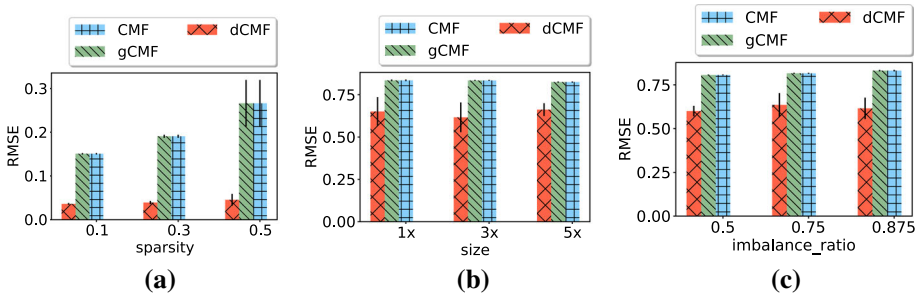

Fig. 4 Impact of **a** sparsity, **b** entity size and **c** view shape on performance of dCMF and CMF using synthetic datasets

6.2 Case study: hybrid recommender systems

Among the large number of recommendation algorithms developed, arguably, the most well known are Collaborative Filtering (CF) methods that factorize the historical user-item rating matrix to obtain latent user representations. Content-based methods use item descriptions or user profiles to recommend items that are similar to items found in a user's history (e.g. Pazzani and Billsus 1997). CF has been more successful than content-based methods but suffers from two problems: (1) real world rating matrices are large and sparse which impacts the latent factors learnt and deteriorates recommendation performance, (2) they cannot be used to recommend items to a user with no previous ratings, known as the *cold-start* problem. Hybrid methods combine the strengths of both these methods by incorporating user and item information as *side information* within CF.

Deep learning models have been successful in obtaining good representations in recommender systems. Among the earliest models, is Collaborative Deep Learning (CDL) (Wang et al. 2015), that jointly performs deep representation learning for the content side information and collaborative filtering for the rating matrix. To obtain these representations stacked denoising autoencoders (Vincent et al. 2010) are used in a Bayesian formulation. A more scalable and efficient architecture that combined CF with marginalized denoising autoencoders (Chen et al. 2012) was used in Deep Collaborative Filtering (DCF) (Li et al. 2015). CDL has also been recently extended to model multimedia side information in a more robust manner in Collaborative Variational AutoEncoder (CVAE) (Li and She 2017). Note that CDL and CVAE model only the rating and content matrices and not user side information. The use of additional side information was leveraged in the additional stacked denoising autoencoder (aSDAE), that was designed to integrate side information into the latent factors efficiently. Using a combination of aSDAE and matrix factorization, was shown to outperform CDL and DCF (Dong et al. 2017). Another variant of the stacked denoising autoencoder was used with Convolutional Neural Networks to generate user and item latent features respectively and combined in probabilistic model called Probabilistic HybriD model (PHD) (Liu et al. 2017) that was shown to outperform aSDAE.

Recommendation with dCMF A typical recommendation setting with side information contains 4 entities: users, items, user-features and item-features (of dimensions m, n, u, v respectively) and 3 matrices as shown in Fig. 1b and described in Table 1. Matrix $X^{(1)}$ is usually very sparse due to unknown ratings and the recommendation task is to complete this matrix to obtain future movie recommendations for users.

Table 1 Recommendation dataset statistics (Col: column, Dim: dimension)

Datasets			MovieLens-100K		MovieLens-1M	
Matrix	Row entity	Col entity	Row Dim	Col Dim	Row Dim	Col Dim
$X^{(2)}$	User	User features	943	823	6040	3467
$X^{(1)}$	User	Movies	943	1682	6040	3706
$X^{(3)}$	Movie Features	Movies	2374	1682	4296	3706

Prediction Prediction is directly obtained through matrix completion by multiplying the latent representations learnt for row and column entities of the rating matrix as $X^{(1)'} = U^{(e_1)} \cdot U^{(e_2)^T}$. Similar to other CMF-based methods, dCMF can be used to address the cold-start problem. Note that $U^{(e_1)}$ will not contain an entry for a first time user and so, $X^{(1)'}$ will also not contain recommendations for this user. To overcome this cold-start problem, we use $X^{(2)}{}_{m\times u} = U^{(e_1)}_{m\times K} \cdot U^{(e_3)^T}_{u\times K} \implies U^{(e_1)} = X^{(2)}{}_{m\times u} \cdot (U^{(e_3)^T}_{u\times K})^{-1}$, which can be used to estimate an unknown user's latent factor. For a single user's feature vector $h_{1\times u}$, this yields the recommendation: $X^{(1)'}_{1\times n} = (h_{1\times u} \cdot (U^{(e_3)^T}_{u\times K})^{-1}) U^{(e_2)^T}_{n\times K}$.

Data We use two large benchmark datasets: (1) MovieLens-100K and (2) MovieLens-1M. The ratings are between 1 and 5 (star ratings). Both the datasets contain user demographic information (age, gender, occupation, zip) and movie metadata (title, genre). We constructed the rating matrix $X^{(1)}$ by binarizing the ratings (to predict user-movie associations) and bag-of-words feature matrices $X^{(2)}$ and $X^{(3)}$, as described in Dong et al. (2017). Matrix statistics after feature processing are shown in Table 1.

Baselines We compare our performance with state-of-the-art hybrid recommendation algorithms that use both the row (user) and column (movie) features of the central matrix (rating). Our main baseline is aSDAE (Dong et al. 2017), a hybrid recommender model that was shown to outperform DCF, CDL, CMF and PMF. Note that aSDAE can be used for recommendation with side-information but cannot be used for augmented multi-view learning. In addition we also compare with PHD (Liu et al. 2017), DCF (Li et al. 2015), and IMC (Natarajan and Dhillon 2014), that can use side information for recommendations. We also compare our performance with that of CMF and gCMF that use collective matrix factorization.

Evaluation metric Since these ratings are implicit and we do not evaluate ranking, we use Recall@N (averaged over all users) as our evaluation metric. Dong et al. (2017) use Recall@N for the same reason. Let r_i be the row corresponding to the predictions for user i in the predicted rating matrix $X^{(1)'}$, S_i^N be the set of top N predictions from the sorted r_i and S_i^T be the test set for the user i. Recall@N $= \frac{|S_i^N \cap S_i^T|}{|S_i^T|}$. Following Dong et al. (2017), for each dataset, we measure the average performance over 5 runs. In each run, 95% of the ratings are randomly selected for training and the remaining 5% for the test set. p values are computed using the Friedman test (Demšar 2006).

Results Figure 5a, b shows the performance of all the methods on the MovieLens-100K dataset. Figure 5c, d shows the performance results on MovieLens-1M dataset (recommendation and CMF baselines shown separately). In both the datasets, we observe that dCMF significantly outperforms all the baseline hybrid recommendation methods aSDAE, DCF, IMC and PHD ($p = 0.003$), as well as previous CMF methods, CMF and gCMF ($p = 0.018$).

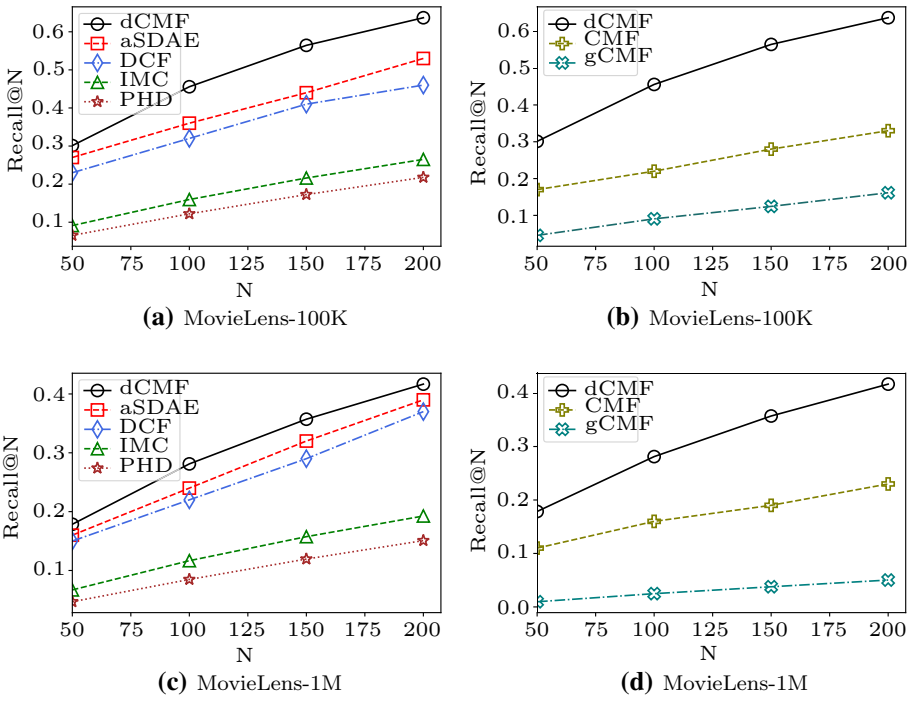

Fig. 5 Performance of dCMF and baseline methods [(left) recommendation algorithms, (right) CMF-based algorithms] on two benchmark datasets

6.3 Case study: gene-disease association prediction

Identifying the genes associated with diseases is an important problem in biomedical sciences. Knowledge of such associations not only improve our understanding genomic interactions but also facilitate the design of treatment strategies. As a result, there has been active research in this area with many experimental methods to determine such associations such as genome-wide association studies (GWAS) (Frayling 2007) and RNA interference screens (Boutros and Ahringer 2008). However experimental methods are expensive, time-consuming and may be specific to certain classes of diseases (Piro and Di Cunto 2012). As a result various computational approaches have been developed to aid the discovery of such associations, such as knowledge-based methods (e.g., Zhou and Skolnick 2016) and methods based on text mining (e.g., Kolker et al. 2015), crowdsourcing (e.g., Loguercio et al. 2013) and networks (e.g., Singh-Blom et al. 2013; Zeng et al. 2017). Comprehensive surveys of these methods can be found in Piro and Di Cunto (2012); Opap and Mulder (2017); Seyyedrazzagi and Navimipour (2017).

Tremendous heterogeneity can be found in biological data—comprising measurements from diverse aspects of our complex biological systems—that are used to infer gene-disease associations. When the evidence for an association can be found through multiple independent sources, it is more likely to be true; indeed, methods that can leverage the heterogeneity have been reported to have superior performance (Pers et al. 2011). Hence, many heterogeneous network based methods have been developed for predicting gene-disease association. For example, HSSVM (Zeng et al. 2017) and CATAPULT (Singh-Blom et al. 2013), both

can integrate different biological networks (like protein-protein interactions, disease-disease similarities) and also relevant data from other species. The main limitation of such methods is that they cannot be used for genes or diseases with no known associations (similar to the cold-start problem in recommendation).

A matrix completion based approach, Inductive Matrix Completion (IMC), was proposed by Natarajan and Dhillon (2014) where the problem is modeled as a recommendation problem. Genes and diseases are analogous to users and movies respectively, the rating matrix is analogous to the gene-disease association matrix which is also partially observed and sparse. Similar to hybrid recommender systems, side information as features for genes and diseases can be used from various data sources, to improve the predictive accuracy of the model. Their method is found to significantly outperform previous best methods that cannot integrate multiple data sources. Further, their method can also predict associations for genes or diseases with no previously known associations.

However IMC is limited to using features of genes or diseases, i.e., only data that can be transformed into the matrices described in Fig. 1b: gene features ($X^{(2)}$), disease features ($X^{(3)}$) and gene-disease associations ($X^{(1)}$). Any other auxiliary source of information that may be pertinent to discovering gene-disease association cannot be incorporated. Since CMF-based methods can obtain latent representations from arbitrary collection of matrices, such auxiliary information can be modeled. We show that for gene-disease prediction, such sources, indeed improve the performance of predicting gene-disease association, in an augmented multi-view setting (Fig. 2b).

Data We use four publicly available biomedical data sources:

1. *DisGeNET* (Piñero et al. 2016) is a database of known gene-disease associations, collected from expert curated repositories, GWAS catalogues, animal models and scientific literature.
2. *The Cancer Genome Atlas (TCGA)* (Weinstein et al. 2013) contains genomic and clinical data of 33 different cancers and over 10,000 patients.
3. *Humannet* (Lee et al. 2011) is a functional gene network of human genes obtained by integration of 21 types of 'omics' data sources. Each edge in HumanNet is associated with the probability of a true functional linkage between two genes.
4. *UMLS Metathesaurus* (Schuyler et al. 1993) is a large database of biomedical concepts and their relationships.

We only consider the expert curated gene-disease associations from DisGeNET for our dataset construction, since these are the most reliable. We also restrict our data to a single cancer (Breast Cancer) in TCGA. With these restrictions, there were 11939 genes that were present in all three databases: DisGeNET, TCGA and HumanNet, with 1093 and 11809 associated patients and diseases respectively. We chose a random subset of 2000 genes and associated diseases (968) and all the patients (1093). For these genes and diseases, the gene-disease association matrix $X^{(1)}$ was constructed by using all known associations from DisGeNET: there were 69850 associations, resulting in sparsity level of 96.5%. To construct $X^{(2)}$ we used RNA-Seq Expresssion data from TCGA, where a single sample per patient was chosen. TCGA also contains 115 demographic and clinical features for these patients. We chose a subset of 8 numeric and 21 categorical features as listed in Table 4 in the "Appendix B", with the less than 50% missing values. We then transformed the categorical features to their one-hot encodings, normalized the numeric features and obtained a total of 86 patient features. Gene-gene and disease-disease graphs for the selected genes and diseases were obtained from HumanNet and UMLS respectively. Similar to preprocessing done by Natarajan and Dhillon (2014), we use principal components of the adjacency matrices of these

Table 2 Gene-disease association: dataset statistics (Col: column, Dim: dimension)

Matrix	Row entity	Col entity	Row Dim	Col Dim
$X^{(1)}$	Gene	Disease	2000	968
$X^{(2)}$	Gene	Patient	2000	1093
$X^{(3)}$	Gene	Gene features	2000	1000
$X^{(4)}$	Disease features	Disease	500	968
$X^{(5)}$	Patient features	Patient	86	1093

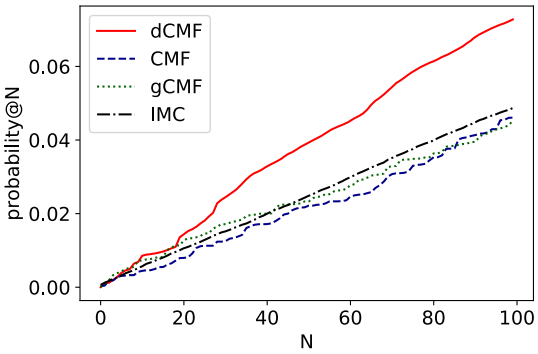

Fig. 6 Gene-disease association prediction: performance of dCMF and baselines

graphs as features to obtain matrices $X^{(3)}$, $X^{(4)}$. Note that this dataset forms the augmented multi-view setup shown in Fig. 2b. Table 2 shows the entity type for each matrix and the matrix dimensions.

Baselines IMC has been found to outperform heterogeneous network based methods like CATAPULT (Singh-Blom et al. 2013). In a recent work, another heterogeneous network based method HSSVM (Zeng et al. 2017) was proposed but it could not outperform CATAPULT. So, we use IMC as the main baseline for predicting gene-disease associations. Note that IMC cannot utilize information from $X^{(5)}$ (patient-patient_features). The other baselines are CMF and gCMF that can model all the views.

Evaluation metric As discussed in Natarajan and Dhillon (2014), an appropriate metric for this task is *probability@N*. For each disease in the test set, the genes are ranked by the score predicted by each method. The cumulative distribution of the ranks, *probability@N*, is the probability that the rank at which a hidden gene-disease pair is retrieved is less than a threshold N. We created 5 folds of the gene-disease association matrix entries (using only known associations) for cross validation. We report the *probability@N* averaged over the 5 folds for N ranging from 1 to 100. p values are computed using the Friedman test (Demšar 2006).

Results Figure 6 shows the performance of dCMF, gCMF, CMF and IMC on our dataset for different values of N. The performance of IMC, CMF and gCMF are comparable with IMC doing marginally better than CMF and gCMF. While dCMF is comparable to IMC below $N = 10$, dCMF significantly outperforms all three baselines at all values of N above 10 ($p < 0.0001$).

Although all three CMF-based methods can utilize the information in the matrix $X^{(5)}$, which IMC cannot, only dCMF can outperform IMC. This suggests that by modeling non-

linear interactions, dCMF is better than CMF and gCMF, at integrating heterogeneous data for predictive modeling.

7 Conclusion

We present dCMF, a neural architecture for CMF, that, to our knowledge, is the first deep augmented multi-view learning technique. dCMF effectively learns latent entity representations, shared across multiple matrices and models their non-linear interactions, that previous CMF methods cannot. Our empirical results show that by modeling non-linear interactions, dCMF effectively integrates heterogeneous data sources and obtains shared representations for predictive modeling that are better than those of several state-of-the-art methods.

Learning dCMF model parameters involves optimizing both entity-specific autoencoder losses as well as matrix-specific reconstruction losses. The latter induces a dependency between the latent representations, which necessitates principled hyperparameter tuning to scale our neural architecture to an arbitrary collection of matrices. Through multi-task Bayesian optimization and an acquisition function that is adapted for dCMF, we effectively address these challenges. Our experiments demonstrate that dCMF significantly outperforms previous CMF methods in both simulated and real datasets. We demonstrate two applications of dCMF: movie recommendations and prediction of gene-disease associations. In both tasks, dCMF significantly outperforms state-of-the-art algorithms on three benchmark datasets.

This work can be extended in several ways. To address the problem of mixed sparsity levels in the input matrices, we could explore other architectural variants. E.g., architectures similar to that in Ngiam et al. (2011) could be used, that can also model view-specific noise and naturally handle different data types. The effect of other types of autoencoders, such as variational autoencoders (Kingma and Welling 2014), could also be studied further. Techniques to improve the scalability of training and hyperparameter tuning can be explored. Finally negative transfer, that is known to affect CMF (Lan et al. 2016), requires further investigation within the dCMF architecture.

Appendix A: Hyperparameters

We briefly describe the hyperparameters that can be tuned in dCMF. The model hyperparameters include the entity representation dimension K and the fraction f_k, that decide the number of neurons/units in each layer and the number of layers adaptively based on the input dimension. The optimization related hyperparameters to be searched include learning rate, weight-decay, batch size, maximum epochs and convergence threshold. Table 3 lists all the hyperparameters.

A.1 Hyperparameter optimization

A.1.1 Evaluation of acquisition function heuristic

We illustrate the effect of our acquisition function heuristic through an example. We consider a setup with two outputs and a single input. The two tasks are defined by the functions $f_1(x)$ and $f_2(x)$ below, where x is the single one dimensional input.

Table 3 List of dCMF hyperparameters

Learning algorithm parameters	Model parameters
Learning rate	Fraction f_k (Number of layers; Number of neurons per layer)
Convergence threshold	Encoding and decoding activation function choice
Weight decay	Entity representation size K
Batch size	
Maximum epochs	
Pre-training requirement (convergence threshold)	

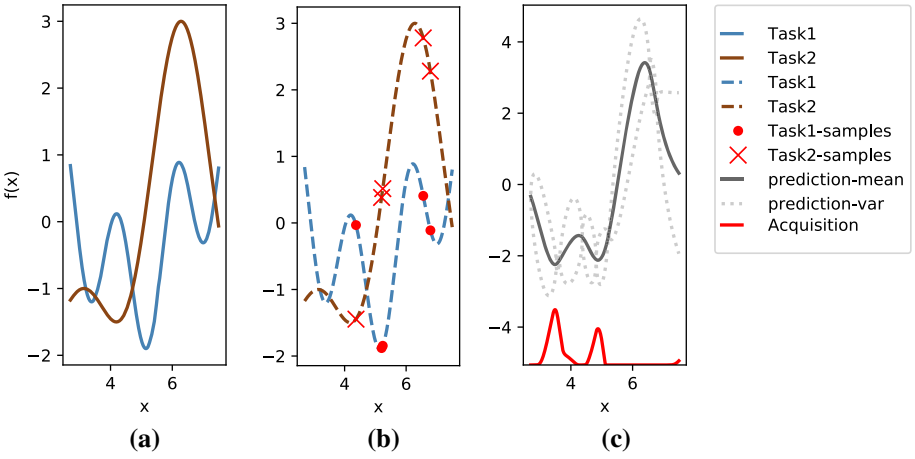

Fig. 7 Illustration of expected improvement based heuristic

$$f_1(x) = sin(x) + sin((10/3) * x)$$
$$f_2(x) = 2cos(x) + cos(2x)$$

The two functions are shown in Fig. 7a for the domain of $x \in [2.5, 7.5]$. As an initial design for BO we used 5 randomly sampled x values and the corresponding $f_1(x)$ and $f_2(x)$ to train the surrogate model (MTGP). We then performed 5 BO steps and the corresponding samples selected based on our acquisition function are shown in Fig. 7b. We illustrate the acquisition function values against the MTGP predictive mean and variance's 1-norm in Fig. 7c. It can be seen that the acquisition function peaks correspond to high variance or high (negative) mean.

A.1.2 Evaluation of surrogate model

To evaluate our approach to hyperparameter selection, we constructed a synthetic dataset in the augmented multi-view setting shown in Fig. 2b. The dataset consists of 6 entities $e_1, \ldots, e_6$ of dimensions 1000, 2000, 20, 150, 300 and 250 respectively. dCMF (Algorithm 2) was run with three different choices of algorithm 1: (1) BO with GP (Snoek et al. 2012) (denoted by dCMF-GP), (2) Random search (Bergstra and Bengio 2012) (denoted

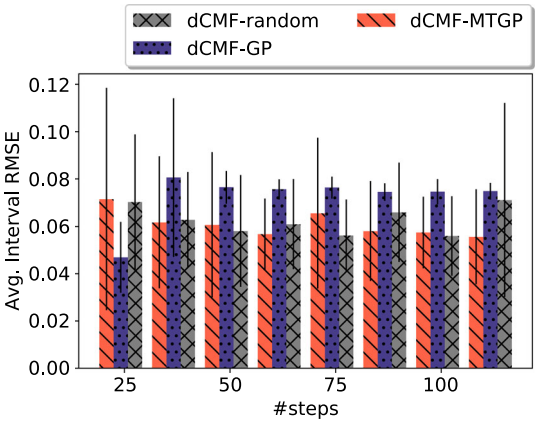

Fig. 8 Surrogate model selection for hyperparameter tuning

by dCMF-random) (3) BO with MTGP, as described above using our acquisition function heuristic (denoted by dCMF-MTGP). We set $n = 200$ steps in Algorithm 1. At every step we reconstruct the matrix $X^{(1)}$ and compute $l_R(X^{(1)}, X^{(1)\prime})$ on a held-out test set using RMSE. We use average cumulative RMSE computed in intervals of 25 steps as our evaluation criterion. Figure 8 shows that dCMF-MTGP has the lowest RMSE after 100 steps, while dCMF-GP initially has the lowest RMSE but is consistently higher after 50 steps. dCMF-random does not have a consistent performance.

A.2 dCMF hyperparameter settings for case studies

In this section we list the hyperparameter settings, found using MTBO (Algorithm 1), that was used in our experiments.

A.2.1 Hybrid recommender system

We set tanh as the activation function in all the encoding and decoding layers. Setting the activation function as tanh allows the range of the learnt factors U to be between -1 and $+1$. This provides flexibility in reconstructing input matrices X thereby lowering the matrix reconstruction loss $\mathcal{L}_R$. For the recommendation datasets we empirically found tanh to do better than ReLu. During training, data was used in 2 batches. We did pretraining for MovieLens-100K and not for MovieLens-1M dataset as the pretraining did not improve the performance. In the side matrices, we perform Maximum Absolute Scaling in which we do not shift/center the data but translate each feature such that their maximal absolute value is 1.0. With the manual settings described so far following are the best hyperparameters $\mathbf{p}^*$ as found by BO in 200 steps: $f_k = 0.01, k = 200$, learning rate 10^{-4} and convergence threshold 10^{-5}(MovieLens-100K) & 10^{-4}(MovieLens-1M).

A.2.2 Gene disease association prediction

We set tanh as the activation function in all the encoding and decoding layers. We did not do pretraining and data was used as a single batch during training. With these manual settings following are the best hyperparameters $\mathbf{p}^*$ as found by BO in 200 steps: $f_k = 0.6, k = 100$, learning rate 0.0002, and convergence threshold 0.0006.

Appendix B: Dataset details for gene disease association prediction

The list of patient features selected from the TCGA dataset for our case study on gene-disease prediction is shown in Table 4.

Table 4 List of patient features

Categorical	Numeric
American Joint Committee on Cancer Tumor Stage Code	Diagnosis Age
Neoplasm Disease Lymph Node Stage American Joint Committee on Cancer Code	Death from Initial Pathologic Diagnosis Date
American Joint Committee on Cancer Metastasis Stage Code	Positive Finding Lymph Node Hematoxylin and Eosin Staining Microscopy Count
Neoplasm Disease Stage American Joint Committee on Cancer Code	Disease Free (Months)
New Neoplasm Event Post Initial Therapy Indicator	Lymph Node(s) Examined Number
Metastatic tumor indicator	Last Alive Less Initial Pathologic Diagnosis Date Calculated Day Value
Overall Survival Status	HER2 ihc score
Disease Free Status	Overall Survival (Months)
Patient's Vital Status	
ER Status By IHC	
Prior Cancer Diagnosis Occurence	
Micromet detection by ihc	
PR status by ihc	
Person Neoplasm Status	
Ethnicity Category	
Tissue Retrospective Collection Indicator	
Disease Surgical Margin Status	
Sex	
Primary Lymph Node Presentation Assessment Ind-3	
Neoadjuvant Therapy Type Administered Prior To Resection Text	
Tissue Prospective Collection Indicator	

Appendix C: Model complexity

In this section we empirically investigate the following: Is the performance improvement due to dCMF solely because of larger number of model parameters? In other words, if CMF or gCMF were to use larger number of free parameters, would their performance improve and be similar to that of dCMF?

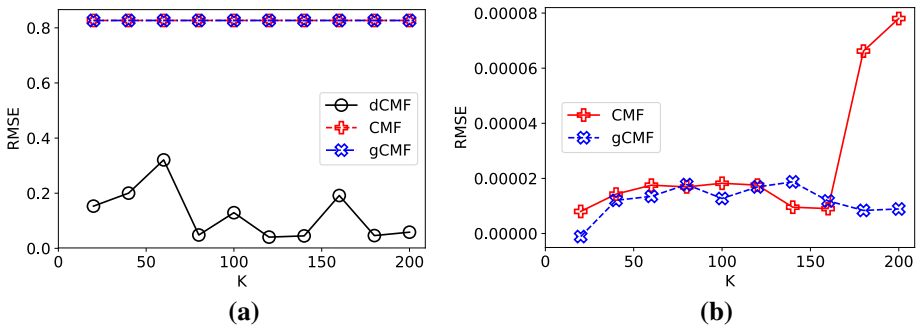

Fig. 9 **a** Performance of dCMF, gCMF and CMF at different values of K. **b** Zoomed-in version of (**a**) where y-axis is y + 0.8267, to show performance of CMF and gCMF

We first analyze the number of free parameters in CMF, gCMF and dCMF:

CMF The number of parameters in CMF $p_{cmf} = p_u$, where $p_u = \sum_{e \in E}(|e| * K)$. CMF extends the alternating projection method and uses a Newton-Raphson step in a gradient-descent based algorithm to estimate all the latent factors (Singh and Gordon 2008).

gCMF In gCMF a variational Bayesian solution is developed wherein additional parameters are present for the distributions assumed. So, the number of parameters, $p_{gcmf} = p_u + p_g$,

$$p_g = |\{\tau^{(m)}\}_{m \in M}| + |\{\alpha_{e,k}\}_{e \in E, k \in (1 \ldots K)}| + |\{\mu_{(e,m)}, \sigma^2_{(e,m)}\}_{e \in E, m \in M}| + |\{p_0, q_0, a_0, b_0\}|$$

where, $\alpha_{e,k}$: Gaussian likelihood precision for latent factors, $\tau^{(m)}$: precision for error terms, $\{p_0, q_0, a_0, b_0\}$: gamma prior parameters and $\mu_{(e,m)}$: mean and $\sigma^2_{(e,m)}$: variance of the bias terms. $|.|$ denotes set cardinality.

dCMF The number of parameters in dCMF, $p_{dcmf} = p_u + p_d$, where $p_d = \sum_{e \in E} |\{par(\mathcal{A}^{(e)})\}|$ and $par(\mathcal{A}^{(e)})$ are the parameters associated with the autoencoder corresponding to entity e. Note that $p_{cmf} = p_u$, $p_{gcmf} \approx \mathcal{O}(p_u)$ and $p_{dcmf} \approx \mathcal{O}(p_u + p_d)$ and by varying K we can control the model complexity.

We now experimentally study the impact of K (ranging between 20 and 200) on CMF, gCMF and dCMF's performance. We generate a synthetic dataset with 4 entities and 3 views based on the recommendation setup (Fig. 1b). We generated $U^{(e_1)}$, $U^{(e_2)}$, $U^{(e_3)}$, $U^{(e_4)}$ with $K = 100$ and the desired dimensions $|e_1| = 400$, $|e_2| = 800$, $|e_3| = 80$ and $|e_4| = 160$, with values sampled from a uniform distribution ranging between 0 and 1. We constructed the views $X^{(1)}_{|e_1| \times |e_2|}$, $X^{(2)}_{|e_1| \times |e_3|}$ and $X^{(3)}_{|e_4| \times |e_2|}$ using the corresponding factors, where subscripts indicate dimensions. For this synthetic dataset, for $K = 20$, $p_{dcmf} \approx 134.4$K and both $p_{cmf}, p_{gcmf} \approx 28.8$K. For $K = 94$, both $p_{cmf}, p_{gcmf} \approx 135$K. Thus, the model complexity of CMF and gCMF with $K \approx 94$ can be considered as roughly equivalent to dCMF with $K = 20$. Similarly model complexity of CMF and gCMF with $K \approx 187$ can be considered as roughly equivalent to that of dCMF with $K = 40$.

For each K varying between 20 and 200 in steps of 20, we obtained the matrix $X^{(1)'}$ using the factors $U^{(1)}$ and $U^{(2)}$ obtained using dCMF, CMF and gCMF. The RMSE between the predicted $X^{(1)'}$ and original $X^{(1)}$ is shown in Fig. 9a. The RMSE values of CMF and gCMF are nearly the same and hence indistinguishable in the figure; so, a zoomed-in version is shown in Fig. 9b.

It can be seen that dCMF consistently outperforms both CMF and gCMF at all values of K. In particular, we can compare the performance of dCMF at $K = 20$ (40) and CMF/gCMF at

$K = 100\,(200)$ that are of roughly equal model complexity and observe that dCMF performs better. In fact, the performance of CMF or gCMF does not improve with increase in K.

References

Andrew, G., Arora, R., Bilmes, J., & Livescu, K. (2013). Deep canonical correlation analysis. In *Proceedings of the 30th international conference on machine learning*, pp. 1247–1255.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Bergstra, J. S., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Proceedings of the 24th international conference on neural information processing systems*, pp. 2546–2554.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*, 281–305.

Bonilla, E. V., Chai, K. M., & Williams, C. (2007). Multi-task Gaussian process prediction. In *Proceedings of the 20th international conference on neural information processing systems*, pp. 153–160.

Bouchard, G., Yin, D., & Guo, S. (2013). Convex collective matrix factorization. In *Proceedings of the sixteenth international conference on artificial intelligence and statistics*, pp. 144–152.

Boutros, M., & Ahringer, J. (2008). The art and design of genetic screens: RNA interference. *Nature Reviews Genetics*, *9*(7), 554.

Chang, S., Han, W., Tang, J., Qi, G.-J., Aggarwal, C. C., & Huang, T. S. (2015). Heterogeneous network embedding via deep architectures. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 119–128. ACM.

Chen, M., Xu, Z., Weinberger, K., & Sha, F. (2012). Marginalized denoising autoencoders for domain adaptation. In *Proceedings of the 29th international conference on machine learning*, pp. 1627–1634.

Coburn, T. C. (2000). Geostatistics for natural resources evaluation. *Technometrics*, *42*(4), 437–438.

Cui, P., Wang, X., Pei, J., & Zhu, W. (2019). A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*, *31*(5), 833–852.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Dong, X., Yu, L., Wu, Z., Sun, Y., Yuan, L., & Zhang, F. (2017). A hybrid collaborative filtering model with deep structure for recommender systems. In *Proceedings of the thirty-first AAAI conference on artificial intelligence*, pp. 1309–1315.

Frayling, T. M. (2007). Genome-wide association studies provide new insights into type 2 diabetes aetiology. *Nature Reviews Genetics*, *8*(9), 657.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning* (Vol. 1). Cambridge: MIT Press.

Guo, X., Gao, L., Liu, X., & Yin, J. (2017). Improved deep embedded clustering with local structure preservation. In *Proceedings of the twenty-sixth international joint conference on artificial intelligence*, pp. 1753–1759.

Han, X., Shi, C., Wang, S., Philip, S. Y., & Song, L. (2018). Aspect-level deep collaborative filtering via heterogeneous information networks. In *Proceedings of the twenty-seventh international joint conference on artificial intelligence*, pp. 3393–3399.

Hardoon, D. R., Szedmak, S., & Shawe-Taylor, J. (2004). Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, *16*(12), 2639–2664.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507.

Hotelling, H. (1936). Relations between two sets of variates. *Biometrika*, *28*(3/4), 321–377.

Hu, Y., Zhang, D., Ye, J., Li, X., & He, X. (2013). Fast and accurate matrix completion via truncated nuclear norm regularization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(9), 2117–2130.

Jones, D. R. (2001). A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, *21*(4), 345–383.

Kingma, D. P., & Welling, M. (2014). Auto-encoding variational Bayes. In *International conference on learning representations*.

Klami, A., Bouchard, G., & Tripathi, A. (2014). Group-sparse embeddings in collective matrix factorization. In *International conference on learning representations*.

Knowles, J. (2006). ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, *10*(1), 50–66.

Kolker, E., et al. (2015). Finding text-supported gene-to-disease co-appearances with MOPED-Digger. *Omics: A Journal of Integrative Biology*, *19*(12), 754–756.
Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30–37.
Lan, C., Wang, J., & Huan, J. (2016). Towards a theoretical understanding of negative transfer in collective matrix factorization. In *Proceedings of the thirty-second conference on uncertainty in artificial intelligence*, pp. 367–376.
Lee, I., Blom, U. M., Wang, P. I., Shim, J. E., & Marcotte, E. M. (2011). Prioritizing candidate disease genes by network-based boosting of genome-wide association data. *Genome Research*, *21*(7), 1109–1121.
Li, X., & She, J. (2017). Collaborative variational autoencoder for recommender systems. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 305–314.
Li, S., Kawale, J., & Fu, Y. (2015). Deep collaborative filtering via marginalized denoising auto-encoder. In *Proceedings of the 24th ACM international conference on information and knowledge management*, pp. 811–820.
Liu, J., Wang, D., & Ding, Y. (2017). PHD: A probabilistic model of hybrid deep collaborative filtering for recommender systems. In *Proceedings of the ninth Asian conference on machine learning*, pp. 224–239.
Loguercio, S., Good, B. M., & Su, A. I. (2013). Dizeez: An online game for human gene-disease annotation. *PLoS ONE*, *8*(8), 71171.
Natarajan, N., & Dhillon, I. S. (2014). Inductive matrix completion for predicting gene-disease associations. *Bioinformatics*, *30*(12), 60–68.
Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., & Ng, A. Y. (2011). Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning*, pp. 689–696.
Opap, K., & Mulder, N. (2017). Recent advances in predicting gene-disease associations. *F1000Research* 6.
Pazzani, M., & Billsus, D. (1997). Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, *27*(3), 313–331.
Pers, T. H., et al. (2011). Meta-analysis of heterogeneous data sources for genome-scale identification of risk genes in complex phenotypes. *Genetic Epidemiology*, *35*(5), 318–332.
Piñero, J., et al. (2016). DisGeNET: A comprehensive platform integrating information on human disease-associated genes and variants. *Nucleic Acids Research*, *45*(D1), 833–839.
Piro, R. M., & Di Cunto, F. (2012). Computational approaches to disease-gene prediction: Rationale, classification and successes. *The FEBS Journal*, *279*(5), 678–696.
Schuyler, P. L., Hole, W. T., Tuttle, M. S., & Sherertz, D. D. (1993). The UMLS metathesaurus: Representing different views of biomedical concepts. *Bulletin of the Medical Library Association*, *81*(2), 217.
Seyyedrazzagi, E., & Navimipour, N. J. (2017). Disease genes prioritizing mechanisms: A comprehensive and systematic literature review. *Network Modeling Analysis in Health Informatics and Bioinformatics*, *6*(1), 13.
Shi, C., Hu, B., Zhao, W. X., & Philip, S. Y. (2019). Heterogeneous information network embedding for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, *31*(2), 357–370.
Shi, C., & Philip, S. Y. (2017). *Heterogeneous information network analysis and applications*. Berlin: Springer.
Singh, A. P., & Gordon, G. J. (2008). Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 650–658.
Singh-Blom, U. M., et al. (2013). Prediction and validation of gene-disease associations using methods inspired by social network analyses. *PLoS ONE*, *8*(5), 58977.
Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 25th international conference on neural information processing systems*, pp. 2951–2959.
Srebro, N., & Shraibman, A. (2005). Rank, trace-norm and max-norm. In *International conference on computational learning theory*, pp. 545–560.
Swersky, K., Snoek, J., & Adams, R. P. (2013). Multi-task Bayesian optimization. In *Proceedings of the 26th international conference on neural information processing systems*, pp. 2004–2012.
Vincent, P., et al. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, *11*, 3371–3408.
Wang, W., Arora, R., Livescu, K., & Bilmes, J. (2015). On Deep Multi-view Representation Learning. In *Proceedings of the 32nd international conference on machine learning*, pp. 1083–1092.
Wang, Q., Sun, M., Zhan, L., Thompson, P., Ji, S., & Zhou, J. (2017). Multi-modality disease modeling via collective deep matrix factorization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1155–1164.
Wang, H., Wang, N., & Yeung, D.-Y. (2015). Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1235–1244.

Weinstein, J. N., et al. (2013). The Cancer Genome Atlas Pan-Cancer analysis project. *Nature Genetics*, *45*(10), 1113–1120.

Zeng, X., Liao, Y., Liu, Y., & Zou, Q. (2017). Prediction and validation of disease genes using HeteSim scores. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *14*(3), 687–695.

Zhou, H., & Skolnick, J. (2016). A knowledge-based approach for predicting gene-disease associations. *Bioinformatics*, *32*(18), 2831–2838.