Big Data Mining and Analytics

Volume 3 | Issue 1 Article 6

2020

# Sparse Deep Nonnegative Matrix Factorization

Zhenxing Guo
*the Academy of Mathematics and Systems Science, Chinese Academy of Sciences (CAS), Beijing 100190, China.*

Shihua Zhang
*NCMIS, CEMS, RCSDS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China the School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China.*

BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 02/06 pp 13–28
Volume 3, Number 1, March 2020
DOI: 10.26599/BDMA.2019.9020020

# Sparse Deep Nonnegative Matrix Factorization

Zhenxing Guo and Shihua Zhang*

**Abstract:** Nonnegative Matrix Factorization (NMF) is a powerful technique to perform dimension reduction and pattern recognition through single-layer data representation learning. However, deep learning networks, with their carefully designed hierarchical structure, can combine hidden features to form more representative features for pattern recognition. In this paper, we proposed sparse deep NMF models to analyze complex data for more accurate classification and better feature interpretation. Such models are designed to learn localized features or generate more discriminative representations for samples in distinct classes by imposing $L_1$-norm penalty on the columns of certain factors. By extending a one-layer model into a multilayer model with sparsity, we provided a hierarchical way to analyze big data and intuitively extract hidden features due to nonnegativity. We adopted the Nesterov's accelerated gradient algorithm to accelerate the computing process. We also analyzed the computing complexity of our frameworks to demonstrate their efficiency. To improve the performance of dealing with linearly inseparable data, we also considered to incorporate popular nonlinear functions into these frameworks and explored their performance. We applied our models using two benchmarking image datasets, and the results showed that our models can achieve competitive or better classification performance and produce intuitive interpretations compared with the typical NMF and competing multilayer models.

**Key words:** sparse Nonnegative Matrix Factorization (NMF); deep learning; Nesterov's accelerated gradient algorithm

## 1 Introduction

Nonnegative Matrix Factorization (NMF) is a powerful dimension reduction and pattern recognition technique in data analysis[1,2], and it has been widely used in diverse areas such as document clustering[3–5], face recognition[6,7], and microarray data analysis[8,9]. It is applied to decompose a nonnegative matrix $\boldsymbol{X}$ into the product of two low-rank nonnegative factor matrices $\boldsymbol{W}$ and $\boldsymbol{H}$ (i.e., $\boldsymbol{X} \approx \boldsymbol{WH}$ with $\boldsymbol{W} \geqslant 0$ and $\boldsymbol{H} \geqslant 0$), generating natural interpretations in many cases. Moreover, NMF has been extended into a number of variant forms, allowing for various sparse[10–12] or regularized models[13,14], most of which demonstrate distinct advantages in local feature extraction or data representation learning.

For sparse variants of NMF, Hoyer[10] found that the original NMF does not always obtain part-based features. Thus, he proposed new sparse models to explicitly control the sparseness degree of $\boldsymbol{W}$ or $\boldsymbol{H}$, and the new models can learn part-based features that cannot be revealed by the original NMF. However, Kim and Park[11] proposed that if strong sparsity constraints are imposed, some important information for gene selection in microarray analysis may be lost. They therefore established a varied version of sparse NMF model, in which the $L_1$-norm penalty was adopted to

• Zhenxing Guo is with the Academy of Mathematics and Systems Science, Chinese Academy of Sciences (CAS), Beijing 100190, China. E-mail: gzx911023@163.com.
• Shihua Zhang is with NCMIS, CEMS, RCSDS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China, and also with the School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: zsh@amss.ac.cn.
* To whom correspondence should be addressed.
Manuscript received: 2019-04-27; revised: 2019-07-15; accepted: 2019-10-10

constrain the sparseness of $\boldsymbol{W}$ or $\boldsymbol{H}$ columns, and this constriction was added as one term to the objective function. This model has been demonstrated capable of realizing biclustering for cancer subtype discovery and gene expression analysis. Peharz and Pernkopf[12] presented another sparse NMF version by adopting $L_0$-norm penalty to limit the exact number of zeros in $\boldsymbol{W}$ or $\boldsymbol{H}$. Additionally, graph-regularized NMF versions have also been explored. For example, Cai et al.[14] proposed a graph-regularized NMF by incorporating prior information of samples into the typical NMF. This helps to keep the original topological structure of the data after they have been projected into a subspace and usually leads to better clustering results.

Moreover, NMF has also been extended for data fusion and combinatorial patterns extraction when multiple data are analyzed. For example, Zhang et al.[15] developed a joint NMF (jNMF) technique to integrate multi-dimensional genomics data for the discovery of combinatorial patterns to reveal phenomena that would have been ignored with only a single type of data. This model provides a way to reveal the homogeneous relationships among different data. Furthermore, Zhang et al.[16] extended the jNMF to both sparse and graph-regularized version. Yang and Michailidis[17] proposed the integrative NMF (iNMF) model, which is unlike jNMF. In addition to the homogeneous effects, they also considered the heterogeneous effects of different types of data. Lastly, Žitnik and Zupan[18] proposed a matrix factorization-based data fusion method to integrate the relationships between heterogeneous datasets and describe the observed system from various views to reveal hidden associations.

Although there have been extensive variants of NMF models, most of them are single-layer models. Deep learning is becoming increasingly popular and has been demonstrated to be powerful in learning data representation[19–23]. However, typical deep learning is rather complicated, as it requires complex structures, needs empirical skills to tune several parameters, and lacks explicitly theoretical foundations. Recently, several new frameworks (e.g., deep Principal Component Analysis (PCA)[24], PCANet[25], and gcForest[26]) have been proposed to attempt to tackle these issues and provide alternative views into deep learning. Deep PCA[24], a model made of two layers and that learns a new data representation at each layer by applying zero-phase component analysis whitening filter followed by PCA, has been proposed to learn hierarchical features and obtain corresponding representations for face recognition. In another study, PCANet[25], which contains a two-layer architecture with PCA employed in each layer, has also been proposed to learn multistage filterbanks; the deep architecture is followed by binary hashing and block histograms for indexing and pooling to generate deep features for more accurate classification. The gcForest technique[26] assembles different types of random forests at each layer to learn deep features to realize comparable classification performance with a deep neural network or convolutional neural network, without establishing complex structures and tuning extensive parameters as found in traditional deep learning. Inspired by the success of the abovementioned frameworks, multi-layer factorizations are attractive to break down the complex problem hierarchically into multiple simple ones. Along these lines, Multi-layer NMF[27,28] and deep semi-NMF[29] have been proposed recently. The general idea of them is by stacking one-layer NMF or semi-NMF[30] into multiple layers to learn hierarchical relationships among features or hierarchical projections. However, these methods do not well reflect[28] or even ignore the sparse structure hidden in the complex data[27,29].

To this end, we developed sparse deep NMF models and explored their effectiveness in multiple aspects. In the new models, the first layer is responsible for breaking down the original data into multiple initial bases. Then the factorization in the second layer is to generate meaningful relationships among all the bases in the first layer to form relatively complex features. Again, the decomposition in the third or higher layer is to learn relationships among features from the former layer to combine them selectively. This process is repeated until the highest layer is reached. In the end, besides all relationship matrices, a data representation matrix will be automatically yielded. During the factorization in each layer, different sparsity constraints are added to localize partial features or generate more discriminative representations for samples in different classes. The key idea is that by thoroughly learning the hidden basis as well as the additional relationships across layers, we can obtain a high-level data representation matrix and yield intuitive interpretations for features generated in each layer.

In summary, we propose a series of generalized sparse deep NMF models by extending a single-layer algorithm into multiple layers, each of which

is designed to generate matrices, satisfying certain sparsity requirements. We also consider linearly inseparable data by incorporating nonlinear functions into the deep NMF in different ways. We adopted the Nesterov's accelerated gradient descent algorithm[31] to accelerate the optimization process with convergence rate $O\left(\frac{1}{k^2}\right)$, which is much faster than the traditional gradient descent algorithm with a convergence rate of $O\left(\frac{1}{k}\right)$ ($k$ is the number of iteration steps).

This paper is organized as follows: In Section 2, we introduce related works and explain their characteristics. In Section 3, we propose a series of diverse models as well as their corresponding optimization algorithms. In addition, we explore the effectiveness of incorporating different nonlinear functions into these models. In Section 4, we employ two benchmarking datasets, ORL and PIE-pose_27.0, to demonstrate the properties and performance of our models. We show that the ORL data (consisting of only 400 images) with a relatively small number of samples are not sufficient to train a deep model. With the PIE-pose_27.0 data, consisting of 2856 images, we conducted extensive experiments and compared them with other NMF variants to demonstrate the effectiveness of our models. We also tested how the structure of deep models, the number of layers, and the number of sub-bases in hidden layers affect the performance of our models. Finally, we conclude this paper in Section 5.

## 2 Related Work

In this section, we introduce three related studies and their contributions: two deep frameworks of matrix factorization models and the Nesterov's accelerated gradient descent algorithm for solving the typical NMF.

### 2.1 Multilayer NMF

The multilayer NMF model[28] extends the typical NMF explicitly to multiple layers as follows,

$$\begin{aligned} \min \quad & \frac{1}{2} \parallel \boldsymbol{X} - \boldsymbol{W}_1\boldsymbol{W}_2\boldsymbol{W}_3\cdots\boldsymbol{W}_L\boldsymbol{H}_L \parallel_{\mathrm{F}}^2, \\ \text{s.t.} \quad & \widetilde{\boldsymbol{H}}_{l-1} \approx \boldsymbol{W}_l\widetilde{\boldsymbol{H}}_l, \\ & \boldsymbol{W}_l \geqslant 0, \widetilde{\boldsymbol{H}}_l \geqslant 0,\ l = 1, 2, \ldots, L \end{aligned} \tag{1}$$

where $|\cdots|_{\mathrm{F}}$ is the Frobenious norm and $L$ is the number of layers. When $L = 1$, the model reduces to the typical NMF model. Ahn et al.[27] first proposed a multilayer NMF model containing three nonlinear layers. They developed a strategy called up-propagation algorithm to jointly update all weight matrices and demonstrated its role in extracting hierarchical features. Song et al.[28] proposed one multilayer NMF model and adopted non-smooth NMF (nsNMF)[32] to solve the typical NMF in each layer. The nsNMF utilizes one sparse matrix $\boldsymbol{S}$ to apply sparsity constraint to standard NMF: $\boldsymbol{S} = (1-\theta)\boldsymbol{I}(k) + \frac{\theta}{k}\boldsymbol{ones}(k)$, where $k$ is the number of bases in the corresponding layer; $\theta$ is the parameter for the smoothing effect, in the range of 0 to 1, $\boldsymbol{I}(k)$ is an identity matrix of size $k \times k$ with ones on the main diagonal and zeros elsewhere, and $\boldsymbol{ones}(k)$ indicates an all-ones matrix of size $k \times k$. The nsNMF smoothens a matrix by multiplying it with $\boldsymbol{S}$. In the multilayer NMF, the author smoothened the $\boldsymbol{H}$ matrix by multiplying $\boldsymbol{S}$ and $\boldsymbol{H}$ during iterations as $\boldsymbol{H} = \boldsymbol{S}\boldsymbol{H}$. Then $\boldsymbol{W}$ would become sparse to compensate for the loss of sparsity. They applied their model to the Reuters-21 578 collection dataset and showed its superiority in classification and feature hierarchies interpretations compared with the single-layer nsNMF.

### 2.2 Deep semi-NMF

Compared with the typical NMF, the semi-NMF does not require the original data and the basis matrix to be nonnegative. It extends the applicable range of NMF, but generates basis matrices that hardly show part-of-whole characteristics intuitively[30]. Deep semi-NMF[29] is an extension of semi-NMF by considering the matrix factorization in a multilayer manner. It is defined in the following format,

$$\begin{aligned} \min \quad & \frac{1}{2} \parallel \boldsymbol{X}^{\pm} - \boldsymbol{W}_1^{\pm}\boldsymbol{W}_2^{\pm}\cdots\boldsymbol{W}_L^{\pm}\boldsymbol{H}_L^{+} \parallel_{\mathrm{F}}^2, \\ \text{s.t.} \quad & \boldsymbol{H}_{l-1}^{+} \approx \boldsymbol{W}_l^{\pm}\boldsymbol{H}_l^{+}, \\ & \boldsymbol{H}_l \geqslant 0,\ l = 1, 2, \ldots, L \end{aligned} \tag{2}$$

where $\boldsymbol{X}^{\pm}$ means that the entries of the matrix $\boldsymbol{X}$ can be positive, negative, or zero values. This method first pre-trains each layer and then fine-tunes the whole system with the outcomes of the pre-training procedure. In general, without a nonnegative constraint, $\boldsymbol{W}_l^{\pm}$ can be directly updated by the least squares algorithm. Nonnegative $\boldsymbol{H}_l^{+}$ is updated in a multiplicative manner with the help of an auxiliary function as found in Ding et al.'s work[30]. Deep semi-NMF turns out to be useful in learning hierarchical projections, from the original data space to various subspaces spanned by hidden attributes (e.g., face expression and illumination angle for face image data). Thus, it can cluster samples

according to their hidden attributes in the corresponding layers.

Deep semi-NMF does not require the input data and the basis matrices to be nonnegative. Thus, it is hard to see the part-of-whole phenomenon because a mutual offset occurs when the basis matrices are combined. Sparsity constraints were not considered either. In addition, the learning process that automatically extracts the hierarchical projections from the raw data space to the multiple hidden attributes subspace is too abstract to understand. For face images, attributes such as face expression and photographing angle can be easily observed, but the hidden attributes of many other kinds of data are hard to know.

On the other hand, Ahn et al.[27] described an interesting multiple decomposition for a nonnegative matrix, but did not provide a detailed analysis of its structure. Moreover, the sparsity structure was not considered. As for the algorithm, they updated the feature matrices jointly without layer-wise initialization; thus, the results may vary greatly among experiments. Song et al.[28] chose the proper number of initial basis vectors through a single-layer nsNMF. They adopted nsNMF and attempted to control the sparsity level of one overall matrix by tuning parameter $\theta$, which cannot generate a column-wise sparse structure to localize features for each sample. Moreover, in their experiments, the results obtained by $\theta = 0$ are better than those obtained by $\theta \neq 0$, implying that the sparsity in their research does not work. For the optimization strategies, they adopted the traditional multiplicative update algorithm for the nsNMF of each layer and then stacked it into two layers. They did not fine-tune the whole system to reduce the total reconstruction error. This workflow is simple but may suffer from the problem of slow convergence. Moreover, it is sensitive to initial solutions, making it rather unstable.

However, by restricting the sparsity of each column of certain matrices, the sparse structure in the original data is well explored in our sparse deep NMF models. Specifically, we considered the sparsity problem of all basis matrices $\boldsymbol{W}_l$ to extract local features. We also tried to solve a sparse coding problem by constricting the sparsity of representation matrices $\boldsymbol{H}_l$, representing the original data with as few basis vectors as possible while maintaining the discriminative ability of representations for samples in distinct classes. As for the optimization strategy, initialization was first performed for each layer, at the beginning of which, NNSVD[33] was adopted to generate a good pair of initialization factors. Then we fine-tuned the whole system to reduce the total reconstruction error. Nesterov's accelerated gradient descent algorithm[31] was adopted to alleviate the problem of numerical instability and low convergence rate. We conducted extensive numerical tests to optimize our model's structure. Popular nonlinear functions as well as their incorporation were explored by extensive experiments.

### 2.3 NeNMF

Guan et al.[34] adopted Nesterov's accelerated gradient descent algorithm[31] to develop an efficient NMF solver named as NeNMF. It has been demonstrated to be capable of alleviating the frequently accounted problems of numerical instability and low convergence rate in other NMF algorithms. Here, we briefly introduce NeNMF. The idea of NeNMF was also adopted in our algorithms. It is known that NMF is a nonconvex optimization problem, and obtaining the optimal solution is impratical. Thus, the block coordinate descent method[35] is popular for seeking a local optimum. Given an initial pair of $\boldsymbol{W}^1$ and $\boldsymbol{H}^1$, the block coordinate descent method alternatively solves

$$\boldsymbol{H}^{t+1} = \arg\min_{H \geqslant 0} F(\boldsymbol{W}^t, \boldsymbol{H}) = \frac{1}{2} \parallel \boldsymbol{X} - \boldsymbol{W}^t \boldsymbol{H} \parallel_{\mathrm{F}}^2 \quad (3)$$

and

$$\boldsymbol{W}^{t+1} = \arg\min_{W \geqslant 0} F(\boldsymbol{W}, \boldsymbol{H}^t) = \frac{1}{2} \parallel \boldsymbol{X} - \boldsymbol{W}\boldsymbol{H}^t \parallel_{\mathrm{F}}^2 \quad (4)$$

until convergence, where $t$ is the iteration step. NeNMF employs Nesterov's accelerated gradient descent algorithm to solve Eqs. (3) and (4). Taking $\boldsymbol{H}$ for instance, at each iteration, $\boldsymbol{H}$ is updated by the projected gradient method, and the update is performed on a chosen search point. The step size is determined by the Lipschitz constant, not by the time consuming line search. Due to the convexity of Eq. (3) with respect to $\boldsymbol{H}$ and the continuity of the corresponding gradient, the convergence rate is $O\left(\frac{1}{k^2}\right)$ after $k$ steps in iteration, superior to that obtained for the conventional gradient descend algorithm, which is $O\left(\frac{1}{k}\right)$.

In particular, two sequences (i.e., $\boldsymbol{H}_k$ and $\boldsymbol{Y}_k$) are constructed and updated alternatively as follows,

$$\begin{aligned} \boldsymbol{H}^{t+1} = \arg\min_{H \geqslant 0} \{ & \Phi(\boldsymbol{Y}_k, \boldsymbol{H}) = F(\boldsymbol{W}^t, \boldsymbol{Y}_k) + \\ & \langle \nabla_{\boldsymbol{H}} F(\boldsymbol{W}^t, \boldsymbol{Y}_k), \boldsymbol{H} - \boldsymbol{Y}_k \rangle + \\ & \frac{\mathrm{LC}}{2} \parallel \boldsymbol{H} - \boldsymbol{Y}_k \parallel_{\mathrm{F}}^2 \} \end{aligned} \quad (5)$$

$$\boldsymbol{Y}_{k+1} = \boldsymbol{H}_k + \frac{\alpha_k - 1}{\alpha_{k+1}}(\boldsymbol{H}_k - \boldsymbol{H}_{k-1}) \tag{6}$$

where $\Phi(\boldsymbol{Y}_k, \boldsymbol{H})$ is the approximate function of $F(\boldsymbol{W}^t, \boldsymbol{H})$ on $\boldsymbol{Y}_k$, and LC is the corresponding Lipschitz constant. Moreover, $\boldsymbol{H}_k$ contains the approximate solution obtained by minimizing $\Phi(\boldsymbol{Y}_k, \boldsymbol{H})$ over $\boldsymbol{H}$, and $\boldsymbol{Y}_k$ stores the search point that is constructed by linearly combining the latest two approximate solutions, i.e., $\boldsymbol{H}_{k-1}$ and $\boldsymbol{H}_k$. According to Ref. [31], the combination coefficient is carefully updated in each iteration step as follows,

$$\alpha_{k+1} = \frac{1 + \sqrt{4\alpha_k^2 + 1}}{2} \tag{7}$$

Based on the Lagrange multiplier method, the Karush-Kuhn-Tucker (K.K.T.) conditions of Eq. (5) are as follows,

$$\nabla_H \phi(\boldsymbol{Y}_k, \boldsymbol{H}_k) \geqslant 0,$$
$$\boldsymbol{H}_k \geqslant 0,$$
$$\nabla_H \phi(\boldsymbol{Y}_k, \boldsymbol{H}_k) \otimes \boldsymbol{H}_k \geqslant 0 \tag{8}$$

where $\nabla_H \phi(\boldsymbol{Y}_k, \boldsymbol{H}_k) = \nabla_H \phi(\boldsymbol{W}^t, \boldsymbol{Y}_k) + \mathrm{LC}(\boldsymbol{H}_k - \boldsymbol{Y}_k)$ is the gradient of $\phi(\boldsymbol{Y}_k, \boldsymbol{H})$ with respect to $\boldsymbol{H}$ at $\boldsymbol{H}_k$, and $\otimes$ is the Hadamard product. By solving Eq. (8), we can obtain the update formula,

$$\boldsymbol{H}_k = P\left(\boldsymbol{Y}_k - \frac{1}{\mathrm{LC}}\nabla_H F(\boldsymbol{W}^t, \boldsymbol{Y}_k)\right) \tag{9}$$

where the operator $P(\boldsymbol{X})$ projects all the negative entries to zeros. By alternatively updating $\boldsymbol{H}_k$, $\boldsymbol{Y}_{k+1}$, and $\alpha_{k+1}$ with Eqs. (9), (6), and (7), respectively, until convergence, the optimal solution of Eq. (3) can be reached. As Eqs. (3) and (4) are symmetrical, the update for $\boldsymbol{W}^{t+1}$ will be obtained in a similar way. With the optimization strategy, NeNMF eventually converges fast.

## 3 Sparse Deep NMF

Similar to the general multilayer NMF framework, sparse deep NMF models factorize a nonnegative matrix into $L + 1$ nonnegative ones,

$$\boldsymbol{X} \approx \boldsymbol{W}_1 g^{-1}(\boldsymbol{W}_2 \cdots g^{-1}(\boldsymbol{W}_L \boldsymbol{H}_L)).$$

To make it more intuitive, we can split the equation into the following formula,

$$g(\boldsymbol{H}_{L-1}) \approx \boldsymbol{W}_L \boldsymbol{H}_L,$$
$$g(\boldsymbol{H}_{L-2}) \approx \boldsymbol{W}_{L-1} g^{-1}(\boldsymbol{W}_L \boldsymbol{H}_L),$$
$$\cdots,$$
$$g(\boldsymbol{H}_2) \approx \boldsymbol{W}_3 g^{-1}(\cdots g^{-1}(\boldsymbol{W}_L \boldsymbol{H}_L)),$$
$$g(\boldsymbol{H}_1) \approx \boldsymbol{W}_2 g^{-1}(\cdots g^{-1}(\boldsymbol{W}_L \boldsymbol{H}_L)),$$
$$\boldsymbol{X} \approx \boldsymbol{W}_1 \boldsymbol{H}_1,$$

where $g$ can be a linear or nonlinear function if necessary. All of the matrices above are required to be nonnegative. For the decomposition in the first layer, $\boldsymbol{W}_1$ is responsible for storing the initial sub-basis matrix, the columns of which should be sufficient enough to learn as much information as possible. Then the further factorization on $\boldsymbol{H}_1$ is to learn the relationships among various sub-basis vectors in $\boldsymbol{W}_1$ to combine them and form meaningful and decipherable features. Further decomposition on $\boldsymbol{H}_l$ ($l = 2, 3, \ldots, L-1$) serves as a similar function with the decomposition of $\boldsymbol{H}_1$. With this hierarchical learning structure, a sequence of sub-basis matrices $\boldsymbol{W}_l$ ($l = 1, 2, \ldots, L$) are generated. Complex features of raw data are extracted by additionally combining those sub-basis matrices. Meanwhile, once the complex features are learned, a high level data representation will be more representative for samples, leading to more accurate classification.

In our sparse deep NMF models, we thoroughly considered the sparse structure hidden in complex data. We first imposed $L_1$-norm penalty onto each column of $\boldsymbol{W}$ in each layer (denoted as SDNMF/L). We also considered to impose $L_1$-norm penalty onto each column of $\boldsymbol{H}$ (denoted as SDNMF/R), which helps to solve sparse coding problem by approximating the raw vector with as few bases as possible. SDNMF/L only imposes the $L_1$-norm penalty on each $\boldsymbol{W}_l$ ($l = 1, 2, \ldots, L$) while SDNMF/R only imposes the sparsity of $\boldsymbol{H}_l$. Afterward, we imposed sparse constraints on both $\boldsymbol{W}_l$ and $\boldsymbol{H}_l$. One model from this idea requires all factors to be sparse to generate both sparse sub-bases and representations (denoted as SDNMF/RL1). Intuitively, for a decomposition of $\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}$, if $\boldsymbol{W}$ is compelled to be sparse, $\boldsymbol{H}$ tends to be smooth. We strengthened this tendency by controlling the $L_1$-norm of each column of $\boldsymbol{W}_l$ while imposing the $L_2$-norm constraints onto final $\boldsymbol{H}_L$ (denoted as SDNMF/RL2). Moreover, we inspected the performance of Deep NMF model (DNMF). It is one case of our models where there is no sparsity constraint on any factors.

### 3.1 SDNMF/L

Specifically, SDNMF/L is formulated as follows,

$$\min \quad \frac{1}{2} \parallel \boldsymbol{X} - \boldsymbol{W}_1 g^{-1}(\boldsymbol{W}_2 \cdots g^{-1}(\boldsymbol{W}_L \boldsymbol{H}_L)) \parallel_{\mathrm{F}}^2 + \frac{1}{2} \sum_{l=1}^{L} \mu_l \sum_{j=1}^{n} \parallel \boldsymbol{W}_l(:, j) \parallel_1^2,$$

$$\text{s.t.}\quad g(\boldsymbol{H}_{l-1}) \approx \boldsymbol{W}_l\boldsymbol{H}_l,$$
$$\boldsymbol{W}_l \geqslant 0,\ \boldsymbol{H}_l \geqslant 0,\ l = 1, 2, \ldots, L \tag{10}$$

where $\| \boldsymbol{W}_l(:, j) \|_1$ denotes $\sum_i | \boldsymbol{W}_l(i, j) |$. Here, we adopted the square of $\| \boldsymbol{W}_l(:, j) \|$, in case that a severe penalty would cause the loss of useful information. Assume that $\boldsymbol{x}$ is one column vector of input data $\boldsymbol{X}$ and $\boldsymbol{h}$ is the corresponding coefficient vector in $\boldsymbol{H}_1$. As for $\boldsymbol{W}_1$, taking it as a linkage between the input vector $\boldsymbol{x}$ and the corresponding representation vector $\boldsymbol{h}$ in the first layer, the more sparse one of its column $w_k$ is, the fewer links there will be between $\boldsymbol{H}_k$ (the $k$-th element in $\boldsymbol{h}$) and $\boldsymbol{x}$. This means that each element in $\boldsymbol{h}$ will respond selectively to all the elements in $\boldsymbol{x}$, or each of them will only be activated by the certain members of $\boldsymbol{x}$, resulting in partial information of $\boldsymbol{x}$ being captured by $w_k$. This is consistent with the desire to extract localized features for each sample. The sparsity constraints for $\boldsymbol{W}_2$ aims at capturing local information in $\boldsymbol{H}_1$, which contains containing the rough relationships among all $w_k$ in $\boldsymbol{W}_1$, helping to selectively combine certain $w_k$ to form meaningful and distinguishable features. The function of sparsity of $\boldsymbol{W}$ in upper layer is similar to that of $\boldsymbol{W}_2$.

Here, we take $g(\boldsymbol{x}) = \boldsymbol{x}$ to simply illustrate the algorithm and nonlinear models later. The algorithm first pre-trains the model in a layer-wise way. For the $l$-th layer, it optimizes the following model,

$$\min\quad \frac{1}{2} \| \boldsymbol{H}_{l-1} - \boldsymbol{W}_l\boldsymbol{H}_l \|_{\mathrm{F}}^2 + \frac{1}{2}\mu_l \mathrm{tr}((\boldsymbol{\xi}_l\boldsymbol{W}_l)^{\mathrm{T}}(\boldsymbol{\xi}_l\boldsymbol{W}_l)),$$
$$\text{s.t.}\quad \boldsymbol{W}_l \geqslant 0,\ \boldsymbol{H}_l \geqslant 0 \tag{11}$$

where $\boldsymbol{H}_0 = \boldsymbol{X}$, $\boldsymbol{\xi}_l$ is a row vector with all components equal one. $\mathrm{Tr}((\boldsymbol{\xi}_l\boldsymbol{W}_l)^{\mathrm{T}}(\boldsymbol{\xi}_l\boldsymbol{W}_l))$ is a variant of $\| \boldsymbol{W}_l(:, j) \|_1^2$ due to the nonnegativity of $\boldsymbol{W}$. Different optimization strategies can be used to optimize the model. For example, the Projected Gradient (PG) method[36] takes advantage of the Armijo line search to estimate the optimal step size along the projection direction for solving each subproblem. However, the Armijo rule is a time-consuming search strategy, making PG inefficient. We can also consider the Projected NLS method (PNLS) proposed by Berry et al.[37] However, it is an approximate approach and cannot guarantee the convergence.

Previous studies have demonstrated that NeNMF can overcome the problem of low convergence rate and numerical instability[34]. Fortunately, SDNMF/L shares the common necessary properties required by Nesterov's optimal algorithm as NeNMF. In other words, the objective function $F(\boldsymbol{W}_l, \boldsymbol{H}_l)$ is convex with respect to $\boldsymbol{H}_l$ or $\boldsymbol{W}_l$, respectively, and the corresponding gradient is Lipschitz-continous. Thus, we adopted Nesterov's accelerated gradient descent algorithm with the following parameters related to $\boldsymbol{W}_l$ and $\boldsymbol{H}_l$ to solve Eq. (11),

$$\nabla_H F(\boldsymbol{W}_l^t, \boldsymbol{H}) = -\boldsymbol{W}_l^t\boldsymbol{H}_{l-1} + \boldsymbol{W}_l^t\boldsymbol{W}_l\boldsymbol{H} \tag{12}$$

and

$$\nabla_W F(\boldsymbol{W}, \boldsymbol{H}_l^t) = -\boldsymbol{H}_{l-1}\boldsymbol{H}_l^t + \boldsymbol{W}\boldsymbol{H}_l\boldsymbol{H}_l^t + \mu_l\boldsymbol{\xi}_l^t\boldsymbol{\xi}_l\boldsymbol{W} \tag{13}$$

We illustrated the pre-training procedure (**Algorithm 1** in Table 1), which alternatively minimizes one matrix factor with another one fixed until convergence. Particularly, in the $l$-th layer, suppose there have been $t$ pairs of $\boldsymbol{W}_l$ and $\boldsymbol{H}_l$, to get the next iteration point $\boldsymbol{H}_l^{t+1}$, **Subalgorithm 1** solves a subproblem by constructing two sub-sequences like that in problem Eq. (5) and updates them until convergence. $\boldsymbol{W}_l^{t+1}$ is obtained similarly with **Subalgorithm 2**. This process proceeds alternatively until convergence.

After pre-training each layer separately, the algorithm fine-tunes the system with initial points of all $\boldsymbol{W}_l$ and $\boldsymbol{H}_l$ to reduce the total reconstruction error as follows,

$$\min\quad C_{\mathrm{SDNMF/L}} = \frac{1}{2} \| \boldsymbol{X} - \boldsymbol{W}_1\boldsymbol{W}_2\cdots\boldsymbol{W}_L\boldsymbol{H}_L \|_{\mathrm{F}}^2 + \frac{1}{2}\sum_{l=1}^{L} \mu_l \mathrm{tr}((\boldsymbol{\xi}_l\boldsymbol{W}_l)^{\mathrm{T}}(\boldsymbol{\xi}_l\boldsymbol{W}_l)),$$
$$\text{s.t.}\quad \boldsymbol{H}_l \approx \boldsymbol{W}_{l+1}\boldsymbol{H}_{l+1},\ \forall l = 1, 2, \ldots, L-1,$$
$$\boldsymbol{W}_l \geqslant 0,\ \boldsymbol{H}_l \geqslant 0,\ \forall l = 1, 2, \ldots, L \tag{14}$$

**• Update rule for $\boldsymbol{W}$ during the fine-tuning process**

For $\boldsymbol{W}_l$, the subproblem in the fine-tune process is equivalent to the following one,

$$\min\quad \frac{1}{2} \| \boldsymbol{X} - \boldsymbol{W}_1\boldsymbol{W}_2\cdots\boldsymbol{W}_{l-1}\boldsymbol{W}\widetilde{\boldsymbol{H}}_l \|_{\mathrm{F}}^2 + \frac{1}{2}\lambda_l \mathrm{tr}((\boldsymbol{\xi}_l\widetilde{\boldsymbol{W}}_l)^{\mathrm{T}}(\boldsymbol{\xi}_l\widetilde{\boldsymbol{W}}_l)),$$
$$\text{s.t.}\quad \boldsymbol{W} \geqslant 0 \tag{15}$$

where $\widetilde{\boldsymbol{H}}_l$ is the reconstruction of $\boldsymbol{H}_l$ and $\widetilde{\boldsymbol{H}}_l \approx \widetilde{\boldsymbol{W}}_{l+1}\widetilde{\boldsymbol{H}}_{l+1}$. The objective function is convex with respect to $\boldsymbol{W}$, and the gradient

$$\nabla_W F(\boldsymbol{\Psi}, \boldsymbol{W}, \widetilde{\boldsymbol{H}}_l) = -\boldsymbol{\Psi}_{l-1}^{\mathrm{T}}\boldsymbol{X}\widetilde{\boldsymbol{H}}_l^{\mathrm{T}} + \boldsymbol{\Psi}_{l-1}^{\mathrm{T}}\boldsymbol{\Psi}_{l-1}\boldsymbol{W}\widetilde{\boldsymbol{H}}_l\widetilde{\boldsymbol{H}}_l^{\mathrm{T}} + \mu_l\boldsymbol{\xi}_l^{\mathrm{T}}\boldsymbol{\xi}_l\boldsymbol{W} \tag{16}$$

is Lipschitz-continuous with Lipschitz constant $LC_{W_l} = \| \boldsymbol{\Psi}_{l-1}^{\mathrm{T}}\boldsymbol{\Psi}_{l-1} \|_2 \cdot \| \widetilde{\boldsymbol{H}}_l\widetilde{\boldsymbol{H}}_l^{\mathrm{T}} \|_2 + \mu_l \| \boldsymbol{\xi}_l^{\mathrm{T}}\boldsymbol{\xi}_l \|_2$, where $\boldsymbol{\Psi}_{l-1} = \boldsymbol{W}_1\boldsymbol{W}_2\cdots\boldsymbol{W}_{l-1}$. Thus, this problem can be solved by **Subalgorithm 2**, where the $F(\boldsymbol{W}, \boldsymbol{H}_l^t)$ should be substituted with the one in Eq. (15), and $\boldsymbol{H}_l$ and $LC_{W_l}$ should be substituted with $\widetilde{\boldsymbol{H}}_l$ and

**Table 1 Algorithm 1.**

**Algorithm 1 Optimal gradient method for initialization**

**Input**: $\boldsymbol{H}_{l-1}$

**Output**: $\boldsymbol{W}_l$ and $\boldsymbol{H}_l$

1. Initialize $\boldsymbol{H}_l^1 \geqslant 0$ and $\boldsymbol{W}_l^1 \geqslant 0$ with NNSVD, $t = 1$

**Repeat**

2. Update $\boldsymbol{H}_l^{t+1}$ and $\boldsymbol{W}_l^{t+1}$ with

2.1 $\boldsymbol{H}_l^{t+1} =$ **Subalgorithm 1**$(\boldsymbol{W}^t, \boldsymbol{H}^t, \nabla_H F(\boldsymbol{W}^t, \boldsymbol{H}), LC_{H_l^t})$

2.2 $\boldsymbol{W}_l^{t+1} =$ **Subalgorithm 2**$(\boldsymbol{W}^t, \boldsymbol{H}^{t+1}, \nabla_W F(\boldsymbol{W}, \boldsymbol{H}^{t+1}), LC_{W_l^t})$

3. $t \leftarrow t + 1$

**Until** stopping criterion is satisfied

4. $\boldsymbol{W}_l = \boldsymbol{W}^t, \boldsymbol{H}_l = \boldsymbol{H}^t$

**Subalgorithm 1 Optimal gradient method for $\boldsymbol{H}_l$**

**Input**: $\boldsymbol{W}_l^t$ and $\boldsymbol{H}_l^t$

**Output**: $\boldsymbol{H}_l^{t+1}$

1. Initialize $\boldsymbol{H}_0 = \boldsymbol{H}_l^t, \boldsymbol{Y}_0 = \boldsymbol{H}_l^t, \alpha_0 = 1,$
$LC_{H_l^t} = \| (\boldsymbol{W}_l^t)^{\mathrm{T}} \boldsymbol{W}_l^t \|_2,\ k = 0$

**Repeat**

2. Update $\boldsymbol{H}_k$, $\alpha_{k+1}$, and $\boldsymbol{Y}_{k+1}$ with

2.1 $\boldsymbol{H}_k = P\left(\boldsymbol{Y}_k - \dfrac{1}{LC_{H_l^t}} \nabla_H F(\boldsymbol{W}_l^t, \boldsymbol{Y}_k)\right)$

2.2 $\alpha_{k+1} = \dfrac{1 + \sqrt{4\alpha_k^2 + 1}}{2}$

2.3 $\boldsymbol{Y}_{k+1} = \boldsymbol{H}_k + \dfrac{\alpha_k - 1}{\alpha_{k+1}} (\boldsymbol{H}_k - \boldsymbol{H}_{k-1})$

3. $k \leftarrow k + 1$

**Until** stopping criterion is satisfied

4. $\boldsymbol{H}_l^{t+1} = \boldsymbol{H}_k$

**Subalgorithm 2 Optimal gradient method for $\boldsymbol{W}_l$**

**Input**: $\boldsymbol{W}_l^t$ and $\boldsymbol{H}_l^t$ ($\boldsymbol{\Psi}_{l-1}^t$ need to be input when fine-tuning the system)

**Output**: $\boldsymbol{W}_l^{t+1}$

1. Initialize $\boldsymbol{W}_0 = \boldsymbol{W}_l^t, \boldsymbol{Z}_0 = \boldsymbol{W}_l^t, \beta_0 = 1,$
$L_{W_l^t} = \| (\boldsymbol{H}_l^t)(\boldsymbol{H}_l^t)^{\mathrm{T}} \|_2 + \mu_l \| \boldsymbol{\xi}_l^{\mathrm{T}} \boldsymbol{\xi}_l \|_2, k = 0$

**Repeat**

2. Update $\boldsymbol{W}_k$, $\beta_{k+1}$, and $\boldsymbol{Z}_{k+1}$ with

2.1 $\boldsymbol{W}_k = P\left(\boldsymbol{Z}_k - \dfrac{1}{LC_{W_l^t}} \nabla_W F(\boldsymbol{Z}_k, \boldsymbol{H}_l^t)\right)$

2.2 $\beta_{k+1} = \dfrac{1 + \sqrt{4\beta_k^2 + 1}}{2}$

2.3 $\boldsymbol{Z}_{k+1} = \boldsymbol{W}_k + \dfrac{\beta_k - 1}{\beta_{k+1}} (\boldsymbol{W}_k - \boldsymbol{W}_{k-1})$

3. $k \leftarrow k + 1$

**Until** stopping criterion is satisfied

4. $\boldsymbol{W}_l^{t+1} = \boldsymbol{W}_k$

$LC_{W_l} = \| \boldsymbol{\Psi}_{l-1}^{\mathrm{T}} \boldsymbol{\Psi}_{l-1} \|_2 \cdot \| \widetilde{\boldsymbol{H}}_l \widetilde{\boldsymbol{H}}_l^{\mathrm{T}} \|_2 + \mu_l \| \boldsymbol{\xi}_l^{\mathrm{T}} \boldsymbol{\xi}_l \|_2,$ respectively.

• **Update rule for $\boldsymbol{H}$ during the fine-tuning process**

For $\boldsymbol{H}_l$, the subproblem is formulated as follows,

$$\begin{aligned} \min \quad & \frac{1}{2} \| \boldsymbol{X} - \boldsymbol{\Psi}_l \boldsymbol{H} \|_{\mathrm{F}}^2, \\ \text{s.t.} \quad & \boldsymbol{H} \geqslant 0. \end{aligned}$$

The objective function above is convex with respect to $\boldsymbol{H}$ and its gradient,

$$\nabla_H F(\boldsymbol{\Psi}_l, \boldsymbol{H}) = -\boldsymbol{\Psi}_l^{\mathrm{T}} \boldsymbol{X} + \boldsymbol{\Psi}_l^{\mathrm{T}} \boldsymbol{\Psi}_l \boldsymbol{H} \tag{17}$$

is Lipschitz-continuous with the constant $LC_{H_l} = \| \boldsymbol{\Psi}_l^{\mathrm{T}} \boldsymbol{\Psi}_l \|_2$, where $\boldsymbol{\Psi}_l = \boldsymbol{\Psi}_{l-1} \boldsymbol{W}_l$. We utilized **Subalgorithm 1** to update $\boldsymbol{H}_l$, where the objective function $F(\boldsymbol{W}_l^t, \boldsymbol{H})$ should be substituted with the one in Eq. (17), and $\boldsymbol{W}_l^t$ and the Lipschitz constant should be substituted with $\boldsymbol{\Psi}_l$ and $LC_{H_l} = \| \boldsymbol{\Psi}_l^{\mathrm{T}} \boldsymbol{\Psi}_l \|_2$, respectively. Finally, we proposed **Algorithm 2** for SDNMF/L based on the above analysis.

• **Algorithm complexity analysis**

During the initialization process, to update $\boldsymbol{H}_l$, we need to compute $LC_{H_l^t}$ in **Subalgorithm 1**. Without loss of generality, we let $l = 1$. The complexity of computing $LC_{H_1^t}$ is $O(mk_1^2 + k_1^3)$, where $m$ is the number of rows of data $\boldsymbol{X}$; $k_1$ denotes the number of sub-bases in the first layer. The complexity of computing $\boldsymbol{H}_k$ in **Subalgorithm 1** is $O(mnk_1) + I_{\text{inner}} \cdot (nk_1^2)$, where $I_{\text{inner}}$ denotes the number of iterations in **Subalgorithm 1**. Thus, the complexity to get $\boldsymbol{H}_1^{t+1}$ through **Subalgorithm 1** is $O(mk_1^2 + k_1^3 + mnk_1) + I_{\text{inner}} \cdot (nk_1^2)$. Similarly, the complexity to get $\boldsymbol{W}_1^{t+1}$ is $O(nk_1^2 + k_1^3 + mnk_1) + I_{\text{inner}} \cdot (mk_1^2)$ in **Subalgorithm 2**, where the complexity of computing $LC_{W_1^t}$ is dominated by computing $\| (\boldsymbol{H}_1^t)(\boldsymbol{H}_1^t)^{\mathrm{T}} \|_2$ because $\boldsymbol{\xi}_1^{\mathrm{T}} \boldsymbol{\xi}_1$ is one matrix with all unit elements whose $\| \cdot \|_2$ is the length of $\boldsymbol{\xi}_1$. Thus, the complexity

**Algorithm 2 Optimal algorithm for SDNMF/L**

**Input**: $\boldsymbol{X} \in \mathbf{R}^{m \times n}$, the size of each layer

**Output**: Weight matrices $\boldsymbol{W}_l$ and representation matrices $\boldsymbol{H}_l (\forall l)$

**Initialize all layers**:

for all layers do

$(\boldsymbol{W}_l, \boldsymbol{H}_l) =$ **Algorithm 1**$(\boldsymbol{H}_{l-1}, \text{layers}(l))$

end for

**end initialization**

**Repeat**

for all layers do

$$\widetilde{\boldsymbol{H}}_l = \begin{cases} \boldsymbol{H}_l, & \text{for } l = L; \\ \boldsymbol{W}_{l+1} \widetilde{\boldsymbol{H}}_{l+1}, & \text{for } l < L \end{cases}$$

$\boldsymbol{\Psi}_{l-1} = \boldsymbol{W}_1 \boldsymbol{W}_2 \cdots \boldsymbol{W}_{l-1}$

$\boldsymbol{W}_l \leftarrow$ **Subalgorithm 2** $(\boldsymbol{W}_l, \widetilde{\boldsymbol{H}}_l, \boldsymbol{\Psi}_{l-1}, \nabla_W F(\boldsymbol{W}, \widetilde{\boldsymbol{H}}_l), LC_{W_l})$

$\boldsymbol{\Psi}_l \leftarrow \boldsymbol{\Psi}_{l-1} \boldsymbol{W}_l$

$\boldsymbol{H}_l \leftarrow$ **Subalgorithm 1** $(\boldsymbol{H}_l, \boldsymbol{\Psi}_l, \nabla_H F(\boldsymbol{\Psi}_l, \boldsymbol{H}), LC_{H_l})$

end for

**Until** stopping criterion is satisfied

of initialization in the first layer is $I_{\text{out}} \cdot (O(mk_1^2 + nk_1^2 + mnk_1) + I_{\text{inner}} \cdot O(mk_1^2 + nk_1^2))$, where $I_{\text{out}}$ is the number of iterations in **Algorithm 1**. Let $R = \max\{k_1, k_2, \ldots, k_L\}$, the complexity of initialization for each layers is $I_{\text{out}}(O(mR^2 + nR^2 + mnR) + I_{\text{inner}} \cdot O(mR^2 + nR^2))$.

During the fine-tuning process in **Algorithm 2**, we updated each factor while keeping the others fixed and the process began from $\boldsymbol{W}_1$. Similar to the initialization, the complexity of tuning the two factors in each layer is $O(mR^2 + nR^2 + mnR) + f_{\text{inner}} \cdot (mR^2 + nR^2)$, where $f_{\text{inner}}$ is the iteration number in the corresponding subalgorithm to yield each factor.

In a word, the computing complexity of SDNMF/L is $L \cdot (I_{\text{out}} \cdot (O(mR^2 + nR^2 + mnR) + I_{\text{inner}} \cdot O(mR^2 + nR^2))) + f_{\text{out}} \cdot (L \cdot (O(mR^2 + nR^2 + mnR) + f_{\text{inner}} \cdot (mR^2 + nR^2)))$. Empirically, both $I_{\text{inner}}$ and $f_{\text{inner}}$ are very small. If we ignore the inner iteration in the both processes, the computing complexity is the same as that of the deep semi-NMF[29].

• **Convergence analysis**

According to Ref. [38], nonlinear Gauss-Seidel method under convex constraints will eventually converge to the critical point (where the gradient is zero) if two conditions are satisfied. For two-block coordinate problem, if the objective function of each subproblem is convex and the sequence generated by the algorithm has limit points, then every limit point is a critical point of the objective function. However, in an $m$-block ($m \geqslant 3$) coordinate problem, the conditions must be that the objective function of $f(x_1, x_2, \ldots, x_m)$ is strictly quasi-convex with respect to $m-2$ variables (or blocks), and the sequence generated by the algorithm has limit points.

In our framework, when $L \geqslant 3$, we have $m \geqslant 4$. To analyze the convergence of SDNMF/L model, we first consider the existence of the limit points of the sequence generated by SDNMF/L. The fact that the objective is decreasing under the sequence supports that they are in a level set of $C_{\text{SDNMF/L}}$. $C_{\text{SDNMF/L}}$ is continuous, so this level set is closed. If this level set is unbounded, then $C_{\text{SDNMF/L}}$ is unbounded on this level set, contradicting with the definition of level set. Thus, the level set is a bounded and closed set (compact set). Correspondingly, the generated sequence within a compact set has limit points. Although we cannot demonstrate the final convergence since the strict quasi-convexity of $C_{\text{SDNMF/L}}$ is hard to prove, it does decrease after each iteration.

### 3.2 SDNMF/R

Here, we consider to control the $L_1$-norm of columns of each $\boldsymbol{H}_l$ to deal with a sparse coding problem. It helps to represent the raw data with as few bases as possible while maintaining the ability to discriminate samples of different classes. Specifically, SDNMF/R is formulated as follows,

$$\begin{aligned} \min \quad & C_{\text{SDNMF/R}} = \frac{1}{2} \| \boldsymbol{X} - \boldsymbol{W}_1 \boldsymbol{W}_2 \cdots \boldsymbol{W}_L \boldsymbol{H}_L \|_{\text{F}}^2 + \\ & \frac{1}{2} \sum_{l=1}^{L} \lambda_l \text{tr}((\boldsymbol{e}_l \boldsymbol{H}_l)^{\text{T}} (\boldsymbol{e}_l \boldsymbol{H}_l)), \\ \text{s.t.} \quad & \boldsymbol{H}_l \approx \boldsymbol{W}_{l+1} \boldsymbol{H}_{l+1}, \\ & \boldsymbol{W}_l \geqslant 0, \ \boldsymbol{H}_l \geqslant 0, \forall l = 1, 2, \ldots, L-1 \end{aligned} \tag{18}$$

where $\boldsymbol{e}_l$ is a row vector with all components equal one. Similarly, we adopted an initialization procedure like SDNMF/L using **Algorithm 1** to accelerate the optimization, and then we fine-tuned all $\boldsymbol{W}_l$ and $\boldsymbol{H}_l$ to reduce the total reconstruction error. During the fine-tuning process, for a specific $l$, the following two subproblems were considered to fine-tune $\boldsymbol{W}_l$ and $\boldsymbol{H}_l$,

$$\boldsymbol{W}_l = \arg\min_{W \geqslant 0} \frac{1}{2} \| \boldsymbol{X} - \boldsymbol{\Psi}_{l-1} \boldsymbol{W} \widetilde{\boldsymbol{H}}_l \|_{\text{F}}^2 \tag{19}$$

$$\boldsymbol{H}_l = \arg\min_{H \geqslant 0} \frac{1}{2} \| \boldsymbol{X} - \boldsymbol{\Psi}_l \boldsymbol{H} \|_{\text{F}}^2 + \lambda_l \text{tr}(\boldsymbol{e}_l \boldsymbol{H}_l)^{\text{T}} (\boldsymbol{e}_l \boldsymbol{H}_l) \tag{20}$$

where $\boldsymbol{\Psi}_{l-1} = \boldsymbol{W}_1 \boldsymbol{W}_2 \cdots \boldsymbol{W}_{l-1}$ and $\boldsymbol{\Psi}_l = \boldsymbol{\Psi}_{l-1} \boldsymbol{W}_l$.

Subproblem Eq. (19) can be solved by **Subalgorithm 2**. The objective function, $\boldsymbol{H}$ and Lipschitz constant should be substituted with the one in Eq. (19), $\widetilde{\boldsymbol{H}}_l$ and $LC_{W_l} = \| \boldsymbol{\Psi}_{l-1}^{\text{T}} \boldsymbol{\Psi}_{l-1} \|_2 \cdot \| \widetilde{\boldsymbol{H}}_l \widetilde{\boldsymbol{H}}_l^{\text{T}} \|_2$, respectively. Subproblem Eq. (20) can be solved by **Subalgorithm 1** by substituting the objective function, $\boldsymbol{W}$ and Lipschitz constant with the one in Eq. (20), $\boldsymbol{\Psi}_l = \boldsymbol{\Psi}_{l-1} \boldsymbol{W}_l$ and $L_{H_l} = \| \boldsymbol{\Psi}_l^{\text{T}} \boldsymbol{\Psi}_l \|_2 + \lambda_l \| \boldsymbol{e}_l^{\text{T}} \boldsymbol{e}_l \|_2$, respectively. Thus, similar to SDNMF/L, with corresponding parameters in **Algorithm 2** replaced, SDNMF/R will find its solution.

### 3.3 SDNMF/RL1

Next, we consider to control the sparsity of both $\boldsymbol{W}$ and $\boldsymbol{H}$. It generates sparse basis matrices as well as a sparse representation.

$$\begin{aligned} \min \quad C_{\text{SDNMF/RL1}} = & \frac{1}{2} \| \boldsymbol{X} - \boldsymbol{W}_1 \boldsymbol{W}_2 \cdots \boldsymbol{W}_L \boldsymbol{H}_L \|_{\text{F}}^2 + \\ & \frac{1}{2} \sum_{l=1}^{L} \mu_l \text{tr}((\boldsymbol{\xi}_l \boldsymbol{W}_l)^{\text{T}} (\boldsymbol{\xi}_l \boldsymbol{W}_l)) + \\ & \frac{1}{2} \lambda_L \text{tr}((e_L \boldsymbol{H}_L)^{\text{T}} (e_L \boldsymbol{H}_L)), \end{aligned}$$

$$\begin{aligned} &\text{s.t.} \quad \boldsymbol{H}_l \approx \boldsymbol{W}_{l+1}\boldsymbol{H}_{l+1}, \\ &\boldsymbol{W}_l \geqslant 0, \boldsymbol{H}_l \geqslant 0, \forall l = 1, 2, \ldots, L-1 \end{aligned} \tag{21}$$

To solve Eq. (21), initialization for each layer is also necessary. For a specific $l$, the problem is

$$\begin{aligned} \min \quad & \frac{1}{2}\|\boldsymbol{H}_{l-1} - \boldsymbol{W}_l\boldsymbol{H}_l\|_{\mathrm{F}}^2 + \frac{1}{2}\mu_l \mathrm{tr}((\boldsymbol{\xi}_l\boldsymbol{W}_l)^{\mathrm{T}}(\boldsymbol{\xi}_l\boldsymbol{W}_l)) + \\ & \frac{1}{2}\lambda_l \mathrm{tr}((\boldsymbol{e}_l\boldsymbol{H}_l)^{\mathrm{T}}(\boldsymbol{e}_l\boldsymbol{H}_l)), \\ \text{s.t.} \quad & \boldsymbol{W}_l \geqslant 0,\ \boldsymbol{H}_l \geqslant 0 \end{aligned} \tag{22}$$

where $\frac{1}{2}\lambda_l \mathrm{tr}((\boldsymbol{e}_l\boldsymbol{H}_l)^{\mathrm{T}}(\boldsymbol{e}_l\boldsymbol{H}_l))$ is considered only when $l = L$. The objective function of Eq. (22) is convex with respect to $\boldsymbol{W}$ or $\boldsymbol{H}$ separately, and the respective Lipschitz constant is easy to calculate. Therefore, back to **Algorithm 1**, the solution for Eq. (22) can now be obtained with parameters related to $\boldsymbol{W}$ and $\boldsymbol{H}$, substituted with the following ones,

$$\boldsymbol{W}: \begin{cases} \nabla_W F(\boldsymbol{W}, \boldsymbol{H}_l) = -\boldsymbol{H}_{l-1}\boldsymbol{H}_l^{\mathrm{T}} + \boldsymbol{W}\boldsymbol{H}_l\boldsymbol{H}_l^{\mathrm{T}} + \\ \qquad\qquad \mu_l\boldsymbol{\xi}_l^{\mathrm{T}}\boldsymbol{\xi}_l\boldsymbol{W}; \\ LC_W = \|\ \boldsymbol{H}_l\boldsymbol{H}_l^{\mathrm{T}}\ \|_2 + \mu_l\ \|\ \boldsymbol{\xi}_l^{\mathrm{T}}\boldsymbol{\xi}_l\ \|_2 \end{cases} \tag{23}$$

$$\boldsymbol{H}: \begin{cases} \nabla_H F(\boldsymbol{W}_l, \boldsymbol{H}) = -\boldsymbol{W}_l^{\mathrm{T}}\boldsymbol{H}_{l-1} + \boldsymbol{W}_l^{\mathrm{T}}\boldsymbol{W}_l\boldsymbol{H} + \\ \qquad\qquad \lambda_l\boldsymbol{e}_l^{\mathrm{T}}\boldsymbol{e}_l\boldsymbol{H}; \\ LC_H = \|\ \boldsymbol{W}_l^{\mathrm{T}}\boldsymbol{W}_l\ \|_2 + \lambda_l\ \|\ \boldsymbol{e}_l^{\mathrm{T}}\boldsymbol{e}_l\ \|_2 \end{cases} \tag{24}$$

where $\lambda_l\boldsymbol{e}_l^{\mathrm{T}}\boldsymbol{e}_l\boldsymbol{H}$ and $\lambda_l \cdot \|\ \boldsymbol{e}_l^{\mathrm{T}}\boldsymbol{e}_l\ \|_2$ in Eq. (24) are removed if $l \neq L$, meaning that we control the sparsity of $\boldsymbol{H}_L$ to obtain a high-level final presentation. After pre-training each layer separately, we fine-tuned the weights of all layers as well as the final representation with the initial approximation points of $\boldsymbol{W}_l$ and $\boldsymbol{H}_L$ to reduce the total reconstruction error of Eq. (21). In particular, for a specific $l$ and a factor matrix $\boldsymbol{W}_l$, consider the model in Eq. (15), where

$$\boldsymbol{W}: \begin{cases} \nabla_W F(\boldsymbol{W}, \widetilde{\boldsymbol{H}}_l) = -\boldsymbol{\Psi}_{l-1}^{\mathrm{T}}\boldsymbol{X}\widetilde{\boldsymbol{H}}_l^{\mathrm{T}} + \\ \qquad\qquad \boldsymbol{\Psi}_{l-1}^{\mathrm{T}}\boldsymbol{\Psi}_{l-1}\boldsymbol{W}\widetilde{\boldsymbol{H}}_l\widetilde{\boldsymbol{H}}_l^{\mathrm{T}} + \mu_l\boldsymbol{\xi}_l^{\mathrm{T}}\boldsymbol{\xi}_l\boldsymbol{W}; \\ LC_W = \|\ \widetilde{\boldsymbol{H}}_l\widetilde{\boldsymbol{H}}_l^{\mathrm{T}}\ \|_2 + \mu_l\ \|\ \boldsymbol{\xi}_l^{\mathrm{T}}\boldsymbol{\xi}_l\ \|_2 \end{cases} \tag{25}$$

By replacing the corresponding parameters in **Subalgorithm 2** with Eq. (25), a new point for $\boldsymbol{W}_l$ in the fine-tuning process will be obtained. For $\boldsymbol{H}_L$, consider the model Eq. (20), where

$$\boldsymbol{H}: \begin{cases} \nabla_H F(\boldsymbol{\Psi}_l, \boldsymbol{H}) = -\boldsymbol{\Psi}_L^{\mathrm{T}}\boldsymbol{X} + \boldsymbol{\Psi}_L^{\mathrm{T}}\boldsymbol{\Psi}_L\boldsymbol{H} + \lambda_L\boldsymbol{e}_L^{\mathrm{T}}\boldsymbol{e}_L\boldsymbol{H}; \\ LC_H = \|\ \boldsymbol{\Psi}_L^{\mathrm{T}}\boldsymbol{\Psi}_L\ \|_2 + \lambda_L\ \|\ \boldsymbol{e}_L^{\mathrm{T}}\boldsymbol{e}_L\ \|_2 \end{cases} \tag{26}$$

Similarly, with the above given parameters, a new point for $\boldsymbol{H}_L$ can also be obtained using **Subalgorithm 1**, indicating that SDNMF/RL1 can also be optimized with **Algorithm 2**.

### 3.4 SDNMF/RL2

The SDNMF/RL1 above is devoted to find sparse bases as well as a sparse representation of raw data. Intuitively, for a decomposition of $\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}$, if $\boldsymbol{W}$ is compelled to be sparse, $\boldsymbol{H}$ will tend to be smooth. Here, we tried to strengthen this trend by controlling the $L_1$-norm of each column of all weights while exerting $L_2$-norm constraints onto final $\boldsymbol{H}$ (denoted as SDNMF/RL2).

$$\begin{aligned} \min \quad & C_{\text{SDNMF/RL2}} = \frac{1}{2}\|\ \boldsymbol{X} - \boldsymbol{W}_1\boldsymbol{W}_2\cdots\boldsymbol{W}_L\boldsymbol{H}_L\ \|_{\mathrm{F}}^2 + \\ & \frac{1}{2}\sum_{l=1}^{L}\mu_l \mathrm{tr}((\boldsymbol{\xi}_l\boldsymbol{W}_l)^{\mathrm{T}}(\boldsymbol{\xi}_l\boldsymbol{W}_l)) + \\ & \frac{1}{2}\lambda_L\ \|\ \boldsymbol{H}_L\ \|_{\mathrm{F}}^2, \\ \text{s.t.} \quad & \boldsymbol{H}_l \approx \boldsymbol{W}_{l+1}\boldsymbol{H}_{l+1},\ \ \boldsymbol{W}_l \geqslant 0, \\ & \boldsymbol{H}_L \geqslant 0,\ \ \forall l = 1, 2, \ldots, L-1 \end{aligned} \tag{27}$$

The optimization procedure, either initialization or fine-tuning, is similar to that of SDNMF/RL1 as long as the $\boldsymbol{e}_L^{\mathrm{T}}\boldsymbol{e}_L$ in Eqs. (24) and (26) are replaced by identity matrix $\boldsymbol{I}$ in the optimization of Eq. (27).

### 3.5 Nonlinear function

In this part, we took $g(\boldsymbol{x})$ to be a nonlinear function and considered four choices: tanh, root, sigmoid, and softplus. Due to the incorporation of nonlinear functions, solving nonlinear systems is different from solving linear systems, but the process is still divided into initialization and the fine-tuning steps. In the initialization, at first, we decomposed $\boldsymbol{X} = \boldsymbol{W}_1\boldsymbol{H}_1$. Prior to using $\boldsymbol{H}_1$ in the next layer, we projected it in an element-wise way with the nonlinear function. Similarly, all $\boldsymbol{H}_l$ ($l = 2, 3, \ldots, L-1$) will be nonlinearly projected before making it as the input of the next layer. As for $\boldsymbol{H}_L$, we can project it or just leave it unchanged, resulting in two ways of incorporation. After projection, the strategy to solve $\boldsymbol{H}_{l-1} = \boldsymbol{W}_l\boldsymbol{H}_l$ is the same as that in Table 1.

However, with a nonlinear structure, Lipschitz constant for the gradient of nonlinear functions is hard to obtain, which means that Nesterov's optimal method may be not suitable to fine-tune the system. Thus, we fine-tuned the system with the traditional project gradient descent algorithm. The gradient for each factor was computed as follows,

$$\frac{\partial f}{\partial \boldsymbol{H}_L} = \boldsymbol{W}_L^{\mathrm{T}} \frac{\partial f}{\boldsymbol{W}_L \boldsymbol{H}_L} = \boldsymbol{W}_L^{\mathrm{T}} \left( \frac{\partial f}{g^{-1}(\boldsymbol{W}_L \boldsymbol{H}_L)} \odot \nabla g^{-1}(\boldsymbol{W}_L \boldsymbol{H}_L) \right) = \boldsymbol{W}_L^{\mathrm{T}} \left( \frac{\partial f}{\boldsymbol{H}_{L-1}} \odot \nabla g^{-1}(\boldsymbol{W}_L \boldsymbol{H}_L) \right) \tag{28}$$

$$\frac{\partial f}{\partial \boldsymbol{W}_l} = \frac{\partial f}{\boldsymbol{W}_l \boldsymbol{H}_l} \boldsymbol{H}_l^{\mathrm{T}} = \left( \frac{\partial f}{g^{-1}(\boldsymbol{W}_l \boldsymbol{H}_l)} \odot \nabla g^{-1}(\boldsymbol{W}_l \boldsymbol{H}_l) \right) \boldsymbol{H}_l^{\mathrm{T}} = \left( \frac{\partial f}{\boldsymbol{H}_{l-1}} \odot \nabla g^{-1}(\boldsymbol{W}_l \boldsymbol{H}_l) \right) \boldsymbol{H}_l^{\mathrm{T}} \tag{29}$$

We first updated $\boldsymbol{H}_L$ with Eq. (29), which can be computed according to the chain rule. Then, we updated $\boldsymbol{W}_l$, $l \in \{2, \ldots, L\}$ with Eq. (30). As for $\boldsymbol{H}_l$, $l \in \{1, 2, \ldots, L-1\}$, it is approximated directly by the nonlinear projection of the product of $\boldsymbol{W}_{l+1}$ and $\boldsymbol{H}_{l+1}$, i.e., $\boldsymbol{H}_l \approx g^{-1}(\boldsymbol{W}_{l+1}\boldsymbol{H}_{l+1})$. For $\boldsymbol{W}_1$, we updated it with **Subalgorithm 2** in Table 1.

## 4 Experiment

### 4.1 Data

We applied our models (DNMF, SDNMF/R, SDNMF/L, SDNMF/RL1, and SDNMF/RL2) on two benchmarking image datasets: ORL and PIE-pose_27.0, and compared them with other NMF-related models. Specifically, we compared our models with single-layer models: the Projected gradient NMF (PgNMF), NeNMF, and deep matrix factorization models: deep semi-NMF and multilayer models. Generally, a large number of samples are required to train a deep model to learn the complex features. Otherwise, the model may not be able to outperform single layer models. We tested our models first on the ORL dataset. It contains 40 subjects and each subject has 10 images of equal size $112 \times 92$ under different conditions. Thus, the data size of ORL is $10\,304 \times 400$. It was randomly split into a training set with a size of $10\,304 \times 320$ and a test set with size of $10\,304 \times 80$. The second dataset is PIE-pose_27.0. It contains 68 subjects and each subject has 42 images of equal size of $32 \times 32$ under each condition. Thus, the size of this data is $1024 \times 2856$. In our experiments, PIE-pose 27.0 was randomly split into seven independent pairs of training and test sets for cross validation, where the size of each training set was $1024 \times 2448$ and the size for test set was $1024 \times 408$. With these two datasets, we first investigated the structure of our models to figure out the proper number of layers as well as the number of sub-bases in the hidden layers.

### 4.2 Evaluation criteria

In this article, we adopted three measures, including Normalized Mutual Information (NMI)[39], Error Rate (ER)[40] and Naive Precision (NP). Given the standard class partition $C^*$ and the obtained class partition $C$, we first constructed a confusion matrix $N$, whose rows correspond to the classes in $C^*$ and columns correspond to the classes in $C$. Let $N_{ij}$ denote the number of samples overlapped by the $i$-th real and the $j$-th obtained classes. The NMI is defined as

$$\mathrm{NMI}(C, C^*) = \frac{-2 \sum_{i=1}^{|C|} \sum_{j=1}^{|C^*|} N_{ij} \log\left( \frac{N_{ij} N}{N_{i\cdot} N_{\cdot j}} \right)}{\sum_{i=1}^{|C|} N_{i\cdot} \log\left( \frac{N_{i\cdot}}{N} \right) + \sum_{j=1}^{|C^*|} N_{\cdot j} \log\left( \frac{N_{\cdot j}}{N} \right)},$$

where $|C|$ is the number of classes in $C$; $N_{i\cdot}$ is the sum of the $i$-th row of $N$; $N_{\cdot j}$ is the sum of the $j$-th column of $N$. The ER is defined as follows,

$$\mathrm{ER}(\boldsymbol{Z}, \boldsymbol{Z}^*) = \sqrt{\| \boldsymbol{Z}^* (\boldsymbol{Z}^*)' - \boldsymbol{Z}\boldsymbol{Z}' \|},$$

where $\boldsymbol{Z}$ and $\boldsymbol{Z}^*$ are the indicator matrices for $C$ and $C^*$, respectively. The NP is computed as follows,

$$\mathrm{NP}(C, C^*) = \frac{1}{|C^*|} \sum_{i=1}^{|C^*|} \frac{\max\limits_{j \in \{1,2,\ldots,|C^*|\}} N_{ij}}{|C_i^*|},$$

where $C_i^*$ denotes the number of objects in $i$-th classes of $C^*$.

### 4.3 Results of test with ORL data

The ORL dataset has a relatively small number of samples, which is insufficient to train a deep model. Thus, we initially set $L = 2$. We chose $k_2 = 80$, 120, and 160 and $k_1 = 100$, 200, 300, 400, 500, 600, 700, and 800. To investigate the role of an extra layer, we fixed $k_2$ while varying $k_1$; that is, for each $k_2$ we ran our models with $k_1$ varying from 100 to 800. Figure 1 shows the NMI and ER of all models under $k_2 = 160$, from which we selected the appropriate value of $k_1$ for all models. For example, we chose $k_1 = 300$ for DNMF and SDNMF/R, $k_1 = 700$ for SDNMF/L, and $k_1 = 500$ for SDNMF/RL2. Similarly, we chose appropriate $k_1$ for $k_2 = 80$ and 120 for each model including deep semi-NMF. For deep semi-NMF, we fixed $k_1 = 700$ according to its performance across all tested values. After selection, we compared our sparse deep models with deep semi-NMF and single layer models (Fig. 2). Each of box in Figs. 1 and 2 represents the distribution of 200 precision scores. They were generated by 20

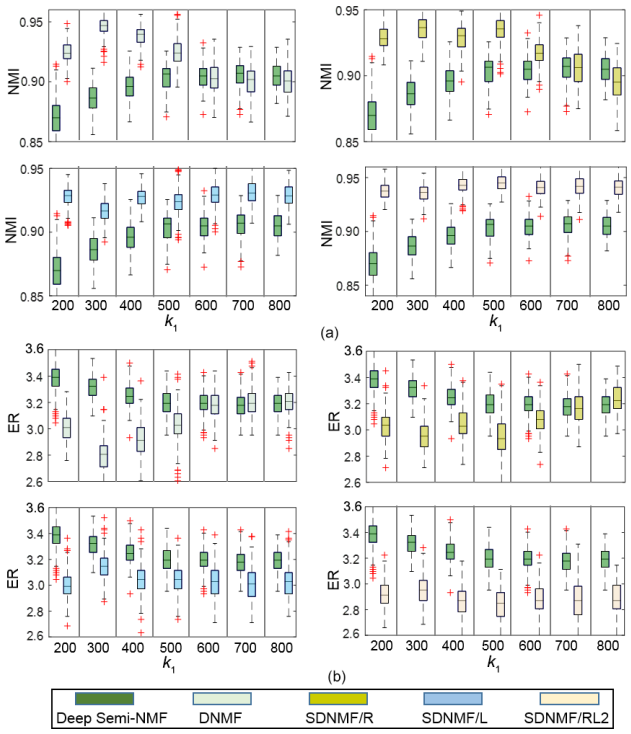


**Fig. 1 Comparison of two-layer models with layer size = [$k_1$, 160] in terms of (a) NMI and (b) ER. Each sub-figure shows the results of deep semi-NMF and our models for comparison.**

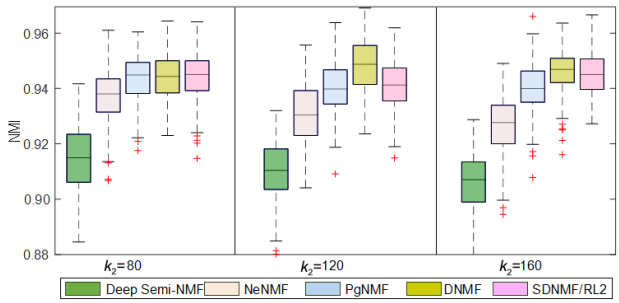


**Fig. 2 Comparison of PgNMF, NeNMF, deep semi-NMF, DNMF, and SDNMF/RL2 in terms of NMI under different $k_2$. Under each $k_2$, we chose the best $k_1$ among 100, 200, 300, 400, 500, 600, 700, and 800 for each deep model.**

computing experiments on ORL, and $K$-means was applied 10 times on the representation matrix of each experiment for robust classification. Results show that the classification ability of our models outperforms that of deep semi-NMF, but does not always outperform PgNMF and NeNMF. These results indeed confirm our consideration that ORL data does not contain enough samples to sufficiently train a deep model.

## 4.4 Results of test with PIE data

### 4.4.1 Structure optimization

In the previous work, the number of layers is specified without a thorough consideration. Different dataset and algorithms may require different structures. Similar to the test for deep semi-NMF[29], we first conducted several experiments on the whole PIE-pose_27.0 dataset (without splitting the images into training and testing sets) to compare the classification performance of all models under different numbers of layers ($L$ = 2, 3, and 4). For each model, each value of $L$ was tested by 50 experiments. The number of sub-bases in the hidden layer in each experiment differs. They were randomly chosen from exponential distributions and then ranked to ensure that the value in lower layer was larger than that in the higher layer. To make a roughly fair comparison, the number of sub-bases in the last layer was fixed to be 40.

As can be seen from Fig. 3, $L = 2$ tends to be the best choice for PIE-pose_27.0 data classification. To figure out the specific structure for our models, we made a series of detailed experiments on the randomly split seven independent pairs of training and testing PIE datasets. We compared the results of each model with the layer size [300, 40], [300, 80], [600, 40], and [600, 80] to determine the value of $k_1$ (Fig. 4). All of the boxes related to the PIE-pose_27.0 in the following

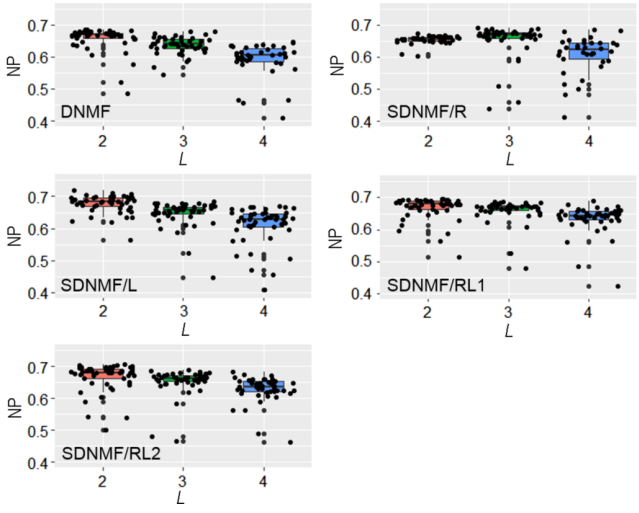


**Fig. 3 Classification precision of our linear models on PIE-pose_27.0 dataset in terms of NP with different layer numbers ($L$ = 2, 3, and 4). The number of sub-bases in hidden layers was randomly extracted from certain exponential distributions. Then they were ranked to satisfy the relationship $k_{\text{lower layer}} > k_{\text{higher layer}}$. For example, for SDNMF/L with $L$ = 3 and layer size = [$k_1$, $k_2$, $k_3$], the values of $k_1$ and $k_2$ were drawn from exponential distributions, with $k_1 > k_2$, and $k_3$ is fixed to be 40. The experiments for SDNMF/L under each layer size was repeated 50 times, generating 50 NMI values. The rest experiments for the other models were performed in the same way.**

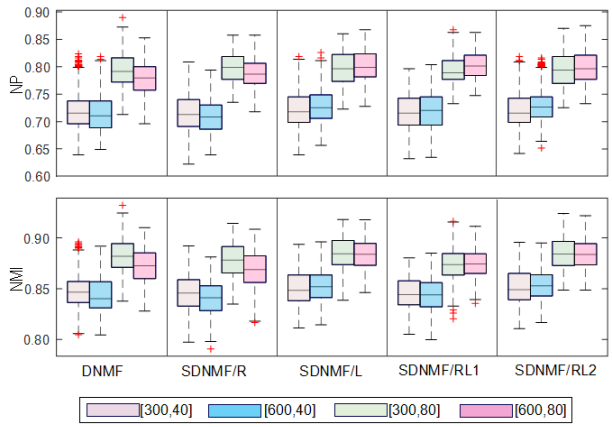


**Fig. 4** $k_1$ **selection of our proposed linear deep models on PIE-pose_27.0 dataset. Each linear model was tested under four conditions: layer size = [300, 40], [300, 80], [600, 40], and [600, 80].**

experiments (Figs. 4–7) represent the distribution of 1400 precision scores: Under each condition, for each pair of training and testing datasets, each model ran 20 times and $K$-means was adopted 10 times for each running to seek robust classification performance.

As can be seen, the classification results with $k_1 = 300$ and $k_1 = 600$ are consistent (Fig. 4). To strengthen the ability of learning necessary information in the first layer, we chose $k_1 = 600$. For choice of value of $k_2$, we selected from 40, 80, 120, 160, and 200. Figure 5 shows that, in addition to DNMF (it is one case of our models where there is no sparsity constraint on any $\boldsymbol{W}_l$ or $\boldsymbol{H}_l$), the precisions of the four models rose as $k_2$ grew at first and reached the climax under $k_2 = 120$ or $k_2 = 160$ and then fell. We then fixed $k_2 = 160$, and added an extra layer to the existing two-layer models. We compared three-layer models with two-layer ones as presented in Fig. 6. The comparison shows that, most of our models,

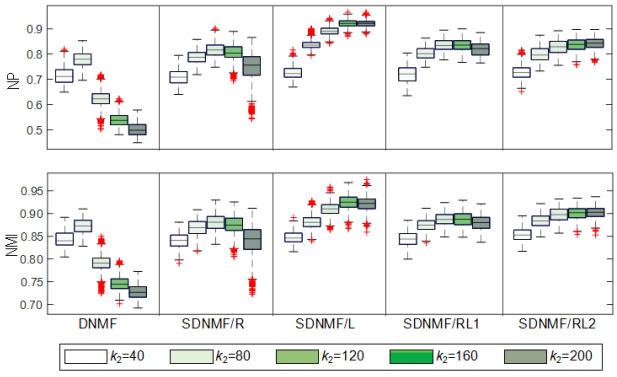


**Fig. 5** $k_2$ **selection of our proposed linear deep models on PIE-pose_27.0 dataset. Each linear model was tested under five conditions: layer size = [600, 40], [600, 80], [600, 120], [600, 160], and [600, 200].**

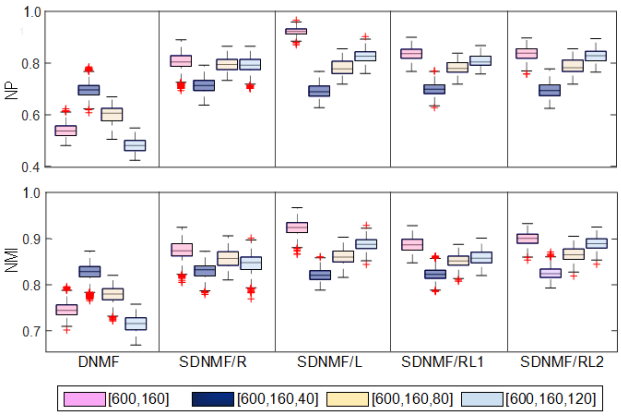


**Fig. 6 Correspondent comparison between two-layer and three layer linear deep models on PIE-pose_27.0 dataset. Specifically, the layer size for three-layer model contains [600,160, 40], [600,160, 80], and [600,160,120]. The layer size for our two-layer models is [600,160].**

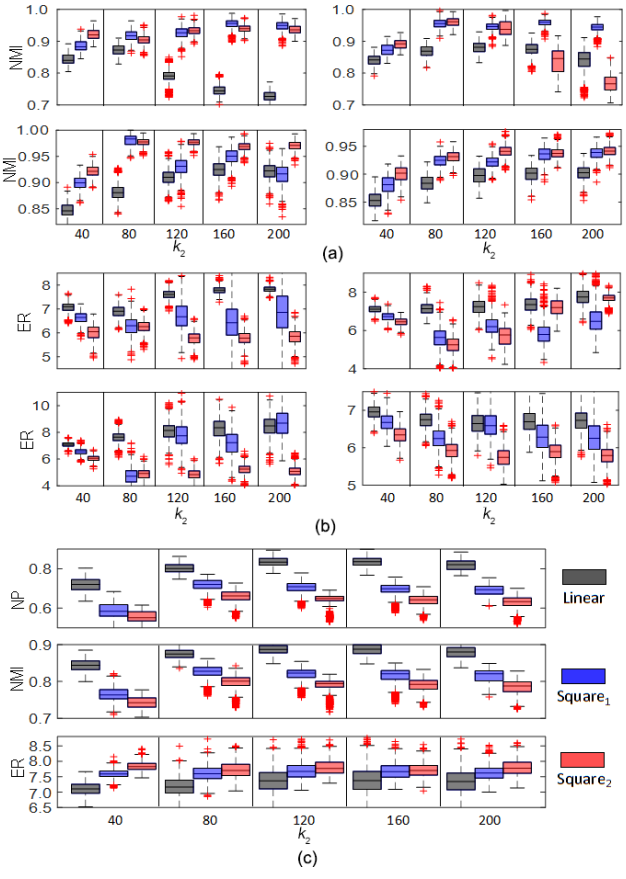


**Fig. 7 Comparison among linear and nonlinear models. Square$_1$ means that we only added nonlinear function onto $H_1$, whereas Square$_2$ means that both $H_1$ and $H_2$ are projected by the nonlinear function. The classification precisions in terms of the NMI of DNMF, SDNMF/R, SDNMF/L, and SDNMF/RL2 are shown in (a) and (b), respectively. The results of SDNMF/RL1 in terms of NP, NMI, and ER are shown in (c).**

except DNMF, reached their best with layer size= [600, 160]. Based on these comparisons in Fig. 6, we kept all models as two-layer models for the PIE-pose_27.0 data.

### 4.4.2 Role of nonlinear function

In this section, we incorporated nonlinear functions (tanh, root, sigmoid, and softplus) into our models in case that the dataset was linearly inseparable. After inspection, only the root function was left. We took $g(\boldsymbol{x}) = \boldsymbol{x}^{1/2}$ and incorporated it into the original linear models with $L = 2$ and layer size = [600,160]. Adding nonlinear functions means that, before using $\boldsymbol{H}$ as the input of the next layer or as the final usage for classification, we first projected it nonlinearly. Since $L = 2$, we tried two ways to incorporate the root function into our models. One was that we only projected $\boldsymbol{H}_1$, leaving $\boldsymbol{H}_2$ unchanged. The other way was that we projected both $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$. The steps to solve the nonlinear models are similar to those for linear models. We first initialized each layer and then fine-tuned the whole system. We compared the performance of each nonlinear model with corresponding linear models (Fig. 7).

Figure 7 shows that there are significant improvements of DNMF, SDNMF/L, and SDNMF/RL2 in NMI and ER after introducing nonlinear transformation of $g(\boldsymbol{x}) = \boldsymbol{x}^{1/2}$. Specifically, for DNMF, the changing of NMI and ER of the linear model are even reversed by nonlinear function. As for SDNMF/R, its learning ability is strengthened by projecting $\boldsymbol{H}_1$ but damaged by projecting both $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$. One possible reason is that it is inappropriate to project $\boldsymbol{H}_2$ with $g(\boldsymbol{x}) = \boldsymbol{x}^{1/2}$ because, according to the design and corresponding results of our experiments, the elements of $\boldsymbol{H}_2$ are smaller than 1 so that the projection of $g(\boldsymbol{x}) = \boldsymbol{x}^{1/2}$ will enlarge all the elements and make $\boldsymbol{H}_2$ smoother, undermining the original sparsity generated by the our initial sparse constriction. For SDNMF/RL1, we required sparsity on both $\boldsymbol{W}$ and $\boldsymbol{H}$. This may cause the loss of important information, and thus a good approximation cannot be achieved. This situation might be worsen by projecting $\boldsymbol{H}_1$ or $\boldsymbol{H}_1$ and $\boldsymbol{H}_2$ with the root function.

We compared our models (with and without nonlinear transformations) with PgNMF, NeNMF, deep semi-NMF, and multilayer NMF ($\theta = 0.001$) in terms of NMI, NP, and ER (Table 2). As can be seen, among all of our proposed models, SDNMF/L performed best. Furthermore, the model SDNMF/L outperforms PgNMF, NeNMF, and multilayer NMF in NP and NMI, even without the nonlinear transformation with the root function. The results that SDNMF/L generated better classification results than the single-layer NMF indicates that both the sparse strategy and the optimization strategies of SDNMF/L play important roles in its good performance. In particular, the sparsity constraints for all the columns of $\boldsymbol{W}$ help to extract localized features and to learn class-specific deep features contributing to more distinctive representations for classification. The fine-tuning process in the optimization approximates the original data more closely. As for the comparison with deep semi-NMF, a gap exists between linear SDNMF/L and deep semi-NMF. This is reasonable since the learning ability is limited by the nonnegativity of our models, whereas semi-NMF is more likely to succeed without the constriction on basis matrices. Nevertheless, by adopting a root function, the learning ability of SDNMF/L rose to a new level, generating more representative coefficient matrices than the deep semi-NMF to distinguish samples of different classes. We noticed that the ER of linear SDNMF/L was higher than those of the compared models. It is because that sometimes multiple (up to 5) classes were clustered into the same group by linear SDNMF/L.

**Table 2 Comparison of NMF-related models on PIE data.**

| Method | NMI | NP | ER |
|---|---|---|---|
| PgNMF | 0.884 | 0.786 | 6.691 |
| NeNMF | 0.914 | 0.868 | 6.729 |
| Deep Semi-NMF | 0.945 | 0.910 | 5.772 |
| Multi-layer NMF (linear) | 0.905 | 0.845 | 6.616 |
| Multi-layer NMF (square$_1$) | 0.909 | 0.851 | 6.332 |
| Multi-layer NMF (square$_2$) | 0.888 | 0.813 | 6.510 |
| SDNMF/L (linear) | 0.925 | 0.922 | 8.271 |
| SDNMF/L (square$_1$) | 0.950 | 0.946 | 7.195 |
| SDNMF/L (square$_2$) | **0.968** | **0.951** | **5.234** |
| SDNMF/RL2 (linear) | 0.901 | 0.837 | 6.706 |
| SDNMF/RL2 (square$_1$) | 0.936 | 0.910 | 6.320 |
| SDNMF/RL2 (square$_2$) | 0.937 | 0.902 | 5.882 |
| SDNMF/R (linear) | 0.875 | 0.805 | 7.353 |
| SDNMF/R (square$_1$) | 0.958 | 0.942 | 5.857 |
| SDNMF/R (square$_2$) | 0.837 | 0.722 | 7.20 |
| DNMF (linear) | 0.750 | 0.543 | 7.786 |
| DNMF (square$_1$) | 0.954 | 0.943 | 6.485 |
| DNMF (square$_2$) | 0.940 | 0.903 | 5.779 |
| SDNMF/RL1 (linear) | 0.890 | 0.836 | 7.390 |
| SDNMF/RL1 (square$_1$) | 0.814 | 0.690 | 7.674 |
| SDNMF/RL1 (square$_2$) | 0.789 | 0.636 | 7.717 |

Note: All results from deep models were generated under layer size = [600, 160], while the results of PgNMF and NeNMF were obtained with $k$ = 160.

However, this situation can be avoided when the root function is incorporated (especially in the second way). In addition, the learning abilities of the nonlinear SDNMF/RL2, SDNMF/R, as well as DNMF were also greatly strengthened by such a nonlinear function.

### 4.4.3 Feature hierarchies

The SDNMF/L model can not only yield a good classification on data PIE-pose_27.0, but also learn feature hierarchies to intuitively show part-of-whole characteristics. This benefits from our sparsity constraints on $\boldsymbol{W}_s$. Sparse $\boldsymbol{W}_1$ helps to learn part-based information for each sample, and sparse $\boldsymbol{W}_2$ selectively combines the initial bases in the first layer to generate composite bases and form relatively complex features. Again, $\boldsymbol{W}_3$ selectively combines the composite bases in the second layer to show discriminative features for samples in distinct classes. As all the features are extracted, a high-level and more meaningful representation for each class will be automatically obtained to achieve accurate classification and feature interpretation. Figure 8 shows how a three-layer linear SDNMF/L model learns feature hierarchies. In Fig. 8a, the left part contains 10 samples of class 36, the right part contains coefficients (some columns of $\boldsymbol{H}_3$) for all samples in class 36, from which we know that samples of class 36 are mainly reconstructed by the seventh (the seventh row mostly in red) composite basis in $\boldsymbol{W}_1\boldsymbol{W}_2\boldsymbol{W}_3$ (listed as the first face image in Fig. 8d). Since the combination result of the composite bases is present in $\boldsymbol{W}_1\boldsymbol{W}_2$ in the second layer, we obtained the top five elements of the seventh column of $\boldsymbol{W}_3$ (coefficient in Fig. 8c) and located the corresponding composite bases in $\boldsymbol{W}_1\boldsymbol{W}_2$ (the five face images to the left in Fig. 8c). To find the related initial sub-bases in the first layer, we chose the first image in Figure 8c, which has the largest coefficient and located its column number in $\boldsymbol{W}_1\boldsymbol{W}_2$ (column 13). Then we ranked the 13th column of $\boldsymbol{W}_2$, and obtained the top five elements (coefficient in Fig. 8b), and located the

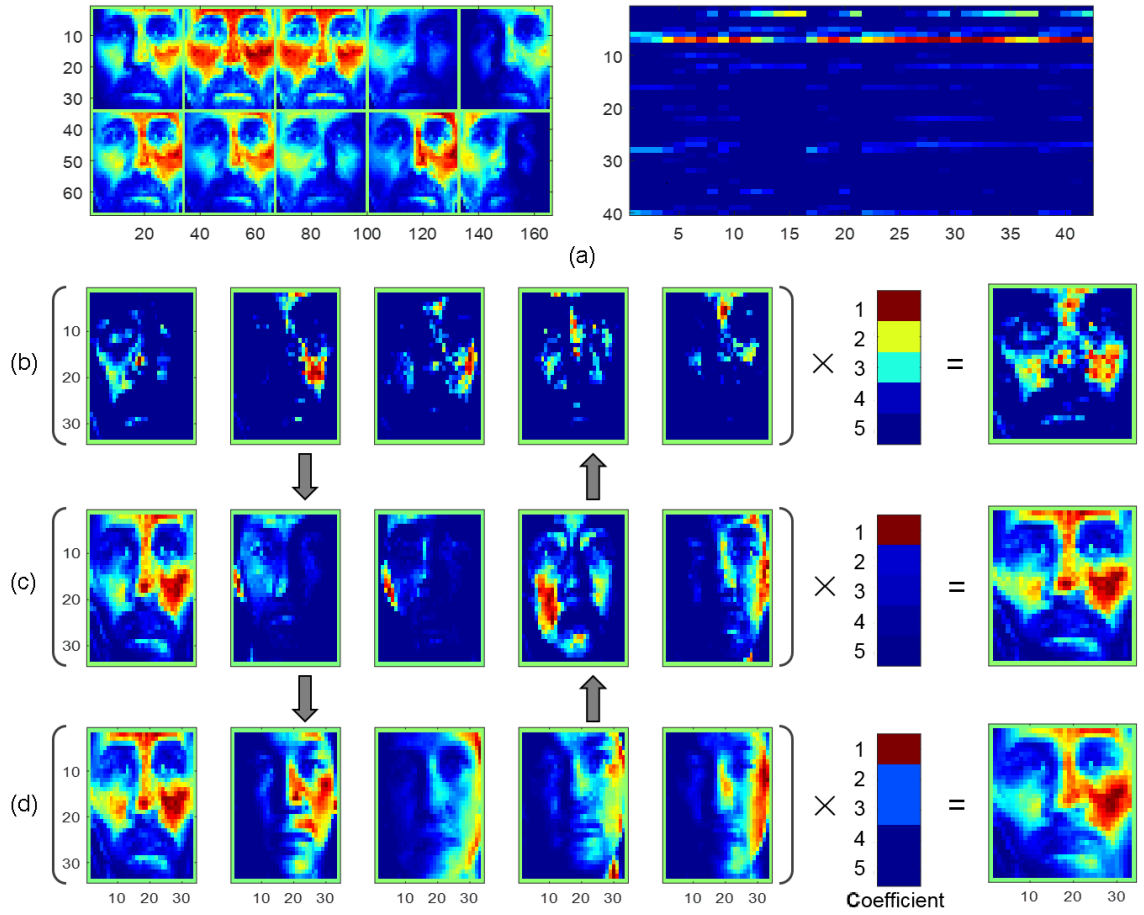


**Fig. 8 Hierarchical feature interpretation by a linear SDNMF/L model with layer size = [193, 141, 40]. (a) 10 samples of class 36 and the representation matrix of class 36. Certain bases as well as corresponding coefficients in the first, second and third layers are shown in (b), (c), and (d), respectively. The horizontal and vertical coordinates indicate the indexs of each matrix (or image).**

corresponding columns of the $\boldsymbol{W}_1$ (the five images to the left of Fig. 8b).

## 5 Conclusion

In this paper, we propose sparse deep nonnegative matrix factorization models that satisfy different sparsity requirements. By extending the original NMF into multilayer models, our models can learn data in a hierarchical way. Model structure optimization was implemented for different datasets. We explored the incorporation of nonlinear functions into these models. We adopted the Nesterov's accelerated algorithm to accelerate the computing process during optimization. We evaluated the time complexity of our algorithm framework and showed that it is comparable to deep semi-NMF. We demonstrated that our models can learn more discriminative representations to obtain competitive classification accuracy compared with other NMF models. In addition, they can also generate intuitive interpretations for the features extracted across all layers, but single layer NMF or deep semi-NMF.

Note that our models performed differently even on the same dataset, there must be some underlying reasons for this, which should be explored further. Future studies should investigate why some nonlinear functions performed well and others did not. Lastly, although the complexity of our model is comparable to other NMF variants, we need to search for more efficient optimization strategies to deal with the increasing big data.

### Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 11661141019 and 61621003), the National Ten Thousand Talent Program for Young Top-notch Talents, Chinese Academy Science (CAS) Frontier Science Research Key Project for Top Young Scientist (No. QYZDB-SSW-SYS008), and the Key Laboratory of Random Complex Structures and Data Science, CAS (No. 2008DP173182).

### References

[1] D. D. Lee and H. S. Seung, Learning the parts of objects by non-negative matrix factorization, *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[2] D. D. Lee and and H. S. Seung, Algorithms for non-negative matrix factorization, in *Proceedings of 14th Neural Information Processing Systems*, Denver, CO, USA, 2000, pp. 556–562.

[3] W. Xu, X. Liu, and Y. Gong, Document clustering based on non-negative matrix factorization, in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Toronto, Canada, 2003, pp. 267–273.

[4] V. P. Pauca, F. Shahnaz, M. W. Berry, and R. J. Plemmons, Text mining using non-negative matrix factorizations, in *Proceedings of the 4th SIAM International Conference on Data Mining, Society for Industrial and Applied Mathematics*, Lake Buena Vista, FL, USA, 2004, pp. 452–456.

[5] F. Shahnaz, M. W. Berry, V. P. Pauca, and R. J. Plemmons, Document clustering using nonnegative matrix factorization, *Information Processing and Management*, vol. 42, no. 2, pp. 373–386, 2006.

[6] Y. Wang, Y. Jia, C. Hu, and M. Turk, Non-negative matrix factorization framework for face recognition, *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, no. 4, pp. 495–511, 2005.

[7] D. Guillamet and J. Vitria, Classifying faces with nonnegative matrix factorization, in *Proceedings of 5th Catalan Conference for Artificial Intelligence*, Castelln, Spain, 2002, pp. 24–31.

[8] J. P. Brunet, P. Tamayo, T. R. Golub, and J. P. Mesirov, Metagenes and molecular pattern discovery using matrix factorization, *Proceedings of the National Academy of Sciences USA*, vol. 101, no. 12, pp. 4164–4169, 2004.

[9] Q. Qi, Y. Zhao, M. Li, and R. Simon, Non-negative matrix factorization of gene expression profiles: A plug-in for BRB-ArrayTools, *Bioinformatics*, vol. 25, no. 4, pp. 545–547, 2009.

[10] P. O. Hoyer, Non-negative matrix factorization with sparseness constraints, *Journal of Machine Learning Research*, vol. 5, pp. 1457–1469, 2004.

[11] H. Kim and H. Park, Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis, *Bioinformatics*, vol. 23, no. 12, pp. 1495–1502, 2007.

[12] R. Peharz and F. Pernkopf, Sparse nonnegative matrix factorization with $L_0$-constraints, *Neurocomputing*, vol. 80, pp. 38-46, 2012

[13] D. Cai, X. He, X. Wu, and J. Han, Non-negative matrix factorization on manifold, in *Proceedings of the 8th IEEE International Conference on Data Ming*, Pisa, Italy, 2008, pp. 63–72.

[14] D. Cai, X. He, J. Han, and T. S. Huang, Graph regularized nonnegative matrix factorization for data representation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1548–1560, 2011.

[15] S. Zhang, C. C. Liu, W. Li, H. Shen, P. Laird, and X. J. Zhou, Discovery of multi-dimensional modules by integrative analysis of cancer genomic data, *Nucleic Acids Research*, vol. 40, no. 19, pp. 9379–9391, 2012.

[16] S. Zhang, Q. Li, J. Liu, and X. J. Zhou, Integrating multiple functional genomic data to define microRNA-genes regulatory modules by a sparse network-regularized multiple matrix factorization method, *Bioinformatics*, vol. 27, no. 13, pp. i401-i409, 2011.

[17] Z. Yang and G. Michailidis, A non-negative matrix factorization method for detecting modules in heterogeneous omics multi-modal data, *Bioinformatics*, vol. 32, no. 1, pp. 1–8, 2016.

[18] M. Žitnik and B. Zupan, Data fusion by matrix factorization, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 1, pp. 41–53, 2015.

[19] G. E. Hinton and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
[20] G. E. Hinton, S. Osindero, and Y. W. Teh, A fast learning algorithm for deep belief nets, *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
[21] A. R. Mohamed, G. Dahl, and G. E. Hinton, Deep belief networks for phone recognition, in *Proceedings of the NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, Vancouver, Canada, 2009, vol. 1, no. 9, p. 39.
[22] A. R. Mohamed, G. Dahl, and G. E. Hinton, Acoustic modeling using deep belief networks, *IEEE Transactions on Audio Speech Language Processing*, vol. 20, no. 1, pp. 14–22, 2012.
[23] G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.
[24] V. E. Liong, J. Lu, and G. Wang, Face recognition using deep PCA, in *Proc. of 2013 9th IEEE International Conference on Information, Communications and Signal Processing (ICICS)*, Tainan, China, 2013, pp. 1–5.
[25] T. H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, PCANet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, 2015.
[26] Z. H. Zhou and J. Feng, Deep forest: Towards an alternative to deep neural networks, arXiv preprint arXiv:1702.08835v1, 2017.
[27] J. H. Ahn, S. Choi, and J. Oh, A multiplicative up-propagation algorithm, in *Proceedings of the 21st International Conference on Machine Learning (ACM)*, Banff, Canada, 2004, p. 3.
[28] H. A. Song, B. K. Kim, T. L. Xuan, and S. Y. Lee, Hierarchical feature extraction by multi-layer non-negative matrix factorization network for classification task, *Neurocomputing*, vol. 165, pp. 63–74, 2015.
[29] G. Trigeorgis, K. Bousmalis, S. Zafeiriou, and B. W. Schuller, A deep matrix factorization method for learning attribute representations, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 3, pp. 417–429, 2017.
[30] C. H. Ding, T. Li, and M. T. Jordan, Convex and semi-nonnegative matrix factorizations, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 45–55, 2010.
[31] Y. Nesterov, Smooth minimization of non-smooth functions, *Mathematical Programming*, vol. 103, no. 1, pp. 127–152, 2005.
[32] A. Pascual-Montano, J. M. Carazo, K. Kochi, D. Lehmann, and R. D. Pascual-Marqui, Nonsmooth nonnegative matrix factorization (nsNMF), *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 3, pp. 403–415, 2006.
[33] C. Boutsidis and E. Gallopoulos, SVD-based initialization: A head start for nonnegative matrix factorization, *Pattern Recognition*, vol. 41, no. 4, pp. 1350–1362, 2008.
[34] N. Guan, D. Tao, Z. Luo, and B. Yuan, NeNMF: An optimal gradient method for nonnegative matrix factorization, *IEEE Transactions on Signal Processing*, vol. 60, no. 6, pp. 2882–2898, 2012.
[35] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA, USA: Athena Scientific, 1999.
[36] C. J. Lin, Projected gradient methods for nonnegative matrix factorization, *Neural Computation*, vol. 19, no. 10, pp. 2756–2779, 2007.
[37] M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca, and R. J. Plemmons, Algorithms and applications for approximate nonnegative matrix factorization, *Computational Statistics and Data Analysis*, vol. 52, no. 1, pp.155–173, 2007.
[38] L. Grippo and M. Sciandrone, On the convergence of the block nonlinear Gauss-Seidel method under convex constraints, *Operations Research Letters*, vol. 26, no. 3, pp. 127–136, 2000.
[39] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, Comparing community structure identification, *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 9, p. P09008, 2005.
[40] Y. R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, Analyzing communities and their evolutions in dynamic social networks, *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 3, no. 2, p. 8, 2009.

**Shihua Zhang** received the PhD degree from the Academy of Mathematics and Systems Science, Chinese Academy of Sciences, in 2008 with the highest honor. He has worked in the same institute as an assistant professor (2008), an associate professor (2013), and a full professor (2018). His research interests are mainly in bioinformatics and computational biology, data mining, pattern recognition, and machine learning. He has won various awards and honors including Ten Thousand Talent Program—Young Top-notch Talent (2017), the National Natural Science Foundation of China for Excellent Young Scholars (2014), Outstanding Young Scientist Program of Chinese Academy Science (CAS) (2014), and Youth Science and Technology Award of China (2013). Now he serves as an editorial board member of *BMC Genomics*, *Frontiers in Genetics*, *Scientific Reports*, and so on. He is a member of the ACM, IEEE, ISCB, and SIAM.

**Zhenxing Guo** is a master student in the Academy of Mathematics and Systems Science, Chinese Academy of Sciences. Her research interests include bioinformatics, data mining, and pattern recognition.