IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 29, 2020, accepted October 27, 2020, date of publication October 30, 2020,
date of current version November 11, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3035120*

# Self-Supervised Feature Specific Neural Matrix Completion

**MEHMET AKTUKMAK, SAMUEL M. MERCIER, AND ISMAIL UYSAL, (Member, IEEE)**
Department of Electrical Engineering, University of South Florida, Tampa, FL 33620, USA
Corresponding author: Mehmet Aktukmak (maktukmak@usf.edu)

**ABSTRACT** Unsupervised matrix completion algorithms mostly model the data generation process by using linear latent variable models. Recently proposed algorithms introduce non-linearity via multi-layer perceptrons (MLP), and self-supervision by setting separate linear regression frameworks for each feature to estimate the missing values. In this article, we introduce an MLP based algorithm called feature-specific neural matrix completion (FSNMC), which combines self-supervised and non-linear methods. The model parameters are estimated by a rotational scheme which separates the parameter and missing value updates sequentially with additional heuristic steps to prevent over-fitting and speed up convergence. The proposed algorithm specifically targets small to medium sized datasets. Experimental results on real-world and synthetic datasets varying in size with a range of missing value percentages demonstrate the superior accuracy for FSNMC, especially at low sparsities when compared to popular methods in the literature. The proposed method has particular potential in estimating missing data collected via real experimentation in fundamental life sciences.

**INDEX TERMS** Matrix completion, non-linear regression, neural networks, self-supervised learning.

## I. INTRODUCTION

Missing value presence is a common problem which degrades the quality of the dataset and disturbs the data analysis. The data could be incomplete due to either the collection or the generation process. Since the missing values in the dataset make data processing and analysis more difficult, matrix completion becomes an important preprocessing step. Inference methods based on machine learning techniques have shown significant promise for the matrix completion task, which include wide-ranging applications from recommender systems [1] to operations research [2], from image processing [3] to product development [4] and high failure rate experiments [5].

A common approach is to assume that the data matrix is low rank that turns the problem into rank minimization. However, this problem is ill posed due to the non-convexity and discontinuity of the rank function [6]. Many approximate solutions have been proposed accordingly [7]. Nuclear norm minimization is one class of techniques that perform L1 norm minimization of the singular values to minimize the rank [6]. Many extensions have been introduced to this base model, i.e., performing singular value thresholding [8], penalization by Lq norm instead of L1 [9], and high rank estimation with subspace clustering by assuming multiple underlying low-rank sub-spaces [10]. In nuclear norm minimization, all the singular values are simultaneously minimized with a convex relaxation to the rank function. In the recent algorithms, the nuclear norm is modified by pruning out the largest singular values [11], and considering the partial sum of singular values [12]. However, all these methods still require singular value decomposition (SVD), which is computationally expensive, especially for large datasets. Recent studies try to overcome this drawback by using orthogonal matching pursuit [13] and multiple factor norms to make the optimization smooth and convex [14].

Unsupervised linear latent models are another family of models, widely used to estimate the missing entries of an incomplete matrix. Finding the latent factors and associated coefficients help imitating the data generation process for the missing entries to be inferred from the known entries of the observations. The subspace is restricted to be low-dimensional to capture the low-rank structure of the data. Principle component analysis (PCA) [15] and matrix factorization [16] are among the simple but effective linear latent variable models widely used for missing variable estimation. In the case of incomplete datasets, these methods estimate the model parameters iteratively, either with gradient descent or

The associate editor coordinating the review of this manuscript and approving it for publication was Tony Thomas.

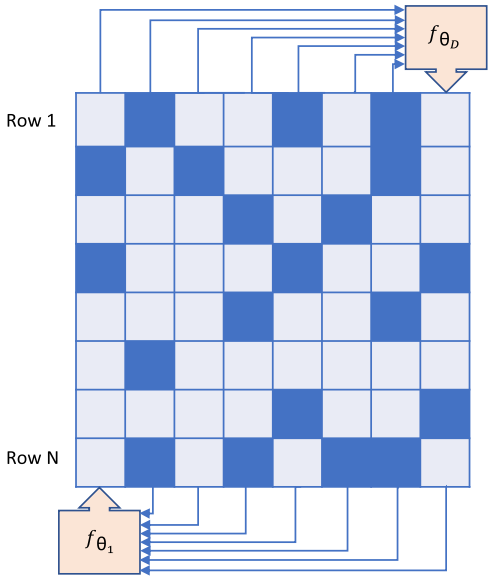



**FIGURE 1.** Graphical abstract of the proposed algorithm. Blue squares represent observed variables whereas light blue squares represent missing values. *D* neural networks are implemented to predict missing values by self-supervision.

alternating least squares methods. Observed and unobserved variables are interpreted probabilistically, where introducing priors for the latent variables is crucial to make the model less prone to over-fitting especially for the sparse datasets. Some other inference methods to optimize the model parameters include Variational inference [15] and stochastic gradient descent [17].

Another class of techniques use non-linear mapping between the latent and observed variables by inducing an unsupervised objective. In this context, Kernel PCA [18], [19], auto-encoder [20], [21] and deep auto-encoder [22] based methods have been proposed to increase the model capacity in order to capture more complex underlying structures. Application areas with promising results include DNA micro-arrays [18], traffic flow [19], metabolite data [20], recommender systems [21], and oil production [23]. The over-fitting is usually alleviated by shrinking the parameters with L2 norm constraints. The probabilistic interpretations of the standard auto-encoders, called Variational auto-encoders, use spherical Gaussian priors for the latent variables, which provides natural regularization for parameter shrinking [24].

Introducing self-supervision to the linear latent models has first been introduced in [25]. The model estimates latent factors by building a PCA model given the completed dataset from the previous iteration. Model parameters are then trimmed according to the known and unknown variables for each observation to form model for regression from the observed to the missing variables. Model over-fitting is avoided by using early stopping in [4] with promising results for several small scale datasets. Non-linear self-supervised methods are natural extensions of linear self-supervised methods. However, building separate complex regression models for each feature significantly increases the number of parameters to be estimated which may cause over-fitting if one can not regularize the model properly [26]. On the other hand, in particular cases, where the number of observations is several orders of magnitude larger than the number of features (such as scientific experimental datasets with relatively smaller fractions of missing values), they can provide higher accuracy [27]. Building independent models associated with each feature also allows parallel optimization of the models which can ultimately decrease the computational burden and/or processing times. In this article, we propose a non-linear self-supervised matrix completion algorithm based on multi-layer perceptions (MLP) specifically tailored for scientific experimental datasets. We assign an independent regression model for each feature and infer the parameters by using a rotational iterative method. At each iteration, the model parameters are first inferred by using the completed matrix from the previous iteration. Then, the missing variables are estimated given these newly inferred model parameters, where the cycle continues until convergence. We introduce additional heuristic steps to ensure a robust learning process with fast convergence and good generalization. Our results consistently show better accuracy especially for small to medium fractions of missing values for scientific experimental datasets with varying sizes. Note that, the proposed algorithm is a non-linear self-supervised method, which improves upon the linear self-supervised methods such as [4], [25] having already demonstrated superior performance on matrix completion tasks recently over numerous unsupervised learning methods. Specifically, we aim to exploit the representation power of neural networks to further improve the matrix completion performance of self-supervised learning methods. We only consider MLPs as the candidate neural network architecture, because scientific experiment datasets are in tabular format, i.e., there is no spatial or sequential relationship between the features, such as scale and position in-variance and temporal dependency.

This article is organized as follows. An introduction to the previously proposed baseline models used for matrix completion is provided as background in Section II-A. Then, we explain the proposed approach and demonstrate its differences with the baseline methods in Section II-B. In Section III-A and Section III-B, the experimental setup including the datasets and competitive algorithms is introduced in detail. The optimization of the models and evaluation procedure are explained in Section III-C. The paper is then concluded with a discussion of the experimental results and possible directions for future work.

## II. METHODOLOGY

### A. BASE MODELS

#### 1) LATENT VARIABLE MODELS

Latent variable models have been widely used for matrix completion in recent years. Matrix factorization is a simple

but effective latent variable method used mostly in collaborative filtering applications to complete large sparse matrices [17]. In this method, a low dimensional latent factor is assigned for each feature and the instances are represented with factor coefficients. The factors are assumed to be lying in a lower dimensional latent space of dimension $K$, so that the observed matrix (of dimensions $N \times D$, where $N$ is the number of observations and $D$ is the number of features) is approximated with the multiplication of the latent factors with associated coefficients as follows:

$$\boldsymbol{x}_i = \boldsymbol{W}\boldsymbol{z}_i + \boldsymbol{m} + \boldsymbol{e}_i, \tag{1}$$

where row vector $\boldsymbol{x}_i \in \mathbb{R}^D$ represents the $i$th instance, $i = 1 : N$, of the incomplete matrix $\boldsymbol{X} \in \mathbb{R}^{N\times D}$, and $\boldsymbol{W} \in \mathbb{R}^{D\times K}$ denotes the factor loading matrix in which the latent factors of each feature are stacked together in order. $\boldsymbol{z}_i \in \mathbb{R}^K$ is the latent variable of instance $i$, i.e., the factor coefficient, so that the multiplication with the factor loading matrix gives the approximation of the variables belonging to the $i$th instance with the addition of column wise mean vector $\boldsymbol{m} \in \mathbb{R}^D$. $\boldsymbol{e}_i \in \mathbb{R}^D$ is the approximation error induced by the model emerging from the projection of the data onto the lower dimensional space, which is usually modeled as a zero mean Gaussian random variable.

The main goal of the optimization is to infer $\boldsymbol{Z}$, $\boldsymbol{W}$ and $\boldsymbol{m}$ given the observed variables of the incomplete matrix $\boldsymbol{X}$. Note that $\boldsymbol{Z} \in \mathbb{R}^{N\times K}$ matrix keeps all the latent variables in order, i.e., $\boldsymbol{Z} = \{z_i\}_{i=1:N}$. The objective of the model can be formulated as

$$min_{\boldsymbol{W},\boldsymbol{Z}} \sum_{(i,j)\in\Omega} (x_{ij} - \boldsymbol{w}_{j,:}\boldsymbol{z}_i - m_j)^2 + \lambda(\|\boldsymbol{w}_{j,:}\|^2 + \|\boldsymbol{z}_i\|^2), \tag{2}$$

in which the right hand-side of the equation corresponds to the regularization term to prevent the model from over-fitting with strength $\lambda$, that is usually optimized via cross-validation as a hyper-parameter. $\Omega$ is the set of tuples that keeps the index pairs of the observed variables, and $\boldsymbol{w}_{j,:} \in \mathbb{R}^{1xK}$ is the $j$th component of the factor loading matrix. Although Eq.2 uses mean squared error metric and L2 norm regularization for the reconstructions of known values in the matrix, many other metrics and regularization terms have also been proposed for different tasks [17], [28]. The researchers mostly prefer to use stochastic gradient or alternating least squares methods to optimize the parameters especially for big and sparse datasets [17].

Building latent variable models in the presence of missing values have also been studied over PCA algorithm [15], where the model building and exploiting steps are separated to enable the application of SVD to the incomplete matrix. The model building step is performed to infer the latent variables and factor loading matrix with SVD by using previous predictions of missing values. The resulting PCA model is given as

$$\boldsymbol{x}_{i,t} = \boldsymbol{W}_t\boldsymbol{z}_{i,t} + \boldsymbol{m}_t + \boldsymbol{e}_{i,t}, \tag{3}$$

where $t$ corresponds to the iteration number. Note that we add iteration index to the parameters to explicitly show the alternating update scheme. In the model building step, the parameters and the latent variables are optimized by using previous predictions of the missing values. In the model exploiting step, the predictions are calculated for the unobserved variables, whose indices are kept in the set $\bar{\Omega}$, by using previously optimized parameter values $\boldsymbol{z}_i$, $\boldsymbol{W}$ and $\boldsymbol{m}$ at iteration $t$. Then, for each missing value $x_{ij,(t+1)}$, the prediction is performed as follows:

$$x_{ij,(t+1)} = \boldsymbol{w}_{j,:,t}\boldsymbol{z}_{i,t} + m_{j,t} \quad (i,j) \in \bar{\Omega}. \tag{4}$$

To initialize the algorithm, the missing variables are filled with feature-wise means of the observed variables to enable initial SVD operation. Then, consecutive model building and exploiting steps are iterated until the convergence of either the predictions or the model parameters.

### 2) REGRESSION MODELS

Another recent approach to matrix completion is to combine a regression framework with a latent variable model to increase the accuracy of the predictions [25]. The missing variables of each observation are estimated within the regression model by using the predicted factor loading matrix and the known variables belonging to that observation. Once the column-wise mean vector is subtracted, i.e., $\boldsymbol{y}_{i,t} = \boldsymbol{x}_{i,t} - \boldsymbol{m}_t$, where $\boldsymbol{m}_t$ is calculated by using the known values of the matrix, PCA decomposition is applied to the model;

$$\boldsymbol{y}_{i,t} = \boldsymbol{W}_t\boldsymbol{z}_{i,t} + \boldsymbol{e}_{i,t}. \tag{5}$$

For the regression framework, as a preprocessing step, $\boldsymbol{W}_t$ is reordered and splitted separately for each observation that includes missing values, such as $\boldsymbol{W}_t = [\boldsymbol{W}_t^* \ \boldsymbol{W}_t^\#]$, in which $\boldsymbol{W}_t^*$ corresponds to the factor loadings of the known variables, and $\boldsymbol{W}_t^\#$ corresponds to the factor loadings of the unknown variables of the observation. The regression from the known variables $\boldsymbol{y}_{i,t}^*$ to the unknown variables $\boldsymbol{y}_{i,t}^\#$ is performed as

$$\boldsymbol{y}_{i,t}^\# = \boldsymbol{W}_t^\#(\boldsymbol{W}_t^{*T}\boldsymbol{W}_t^*)^{-1}\boldsymbol{W}_t^{*T}\boldsymbol{y}_{i,t}^*. \tag{6}$$

This process is repeated for all observations that include missing values, i.e., if all the observations contain at least one missing value, then $N$ regressions are performed. This regression framework is further improved in Trimmed Score Regression (TSR) algorithm [25] by introducing trimmed scores as

$$\boldsymbol{y}_{i,t}^\# = \boldsymbol{S}_t^{\#*}\boldsymbol{W}_t^*\left(\boldsymbol{W}_t^{*T}\boldsymbol{S}_t^{**}\boldsymbol{W}_t^*\right)^{-1}\boldsymbol{W}_t^{*T}\boldsymbol{y}_{i,t}^*, \tag{7}$$

where $(\boldsymbol{S}_t^{\#*} = \boldsymbol{Y}_t^{\#T}\boldsymbol{Y}_t^*/(N-1))$ and $(\boldsymbol{S}_t^{**} = \boldsymbol{Y}_t^{*T}\boldsymbol{Y}_t^*/(N-1))$ are the estimated covariance matrices, and $\boldsymbol{Y}_t = \{\boldsymbol{y}_{i,t}\}_{i=1:N}$ is the mean centered observation matrix. The algorithm can suffer from over-fitting thus early stopping is necessary to obtain better accuracy [4]. We call this variant as the TSRE algorithm.

### 3) BAYESIAN METHODS

Continuous latent variable models [29] are the generalization of both PCA and matrix factorization models, in which the latent variables are treated as random variables. For real valued datasets, it is convenient to use Gaussian distributions for the representation of both latent and observed variables, which results in Linear Gaussian framework. This framework is regarded as the probabilistic version of PCA. The main advantage of this approach is to allow EM to be used for parameter estimation, that has many benefits for big datasets such as computational efficiency and avoiding the sample covariance matrix computation. Since the graphical model is directed from the latent variables to the observed variables, the sample generation and the missing value inference is easily performed [29]. The continuous latent variable model in the case of numerical observations is defined as

$$p(\boldsymbol{x}_i|\boldsymbol{z}_i, \Theta) = \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{W}\boldsymbol{z}_i + \boldsymbol{m}, \boldsymbol{\Sigma}_x), \tag{8}$$

where $\boldsymbol{\Sigma}_x$ is the noise covariance matrix of the observed variables, and $\Theta = \{\boldsymbol{W}, \boldsymbol{m}\}$ is the set of model parameters. Note that $\boldsymbol{\Sigma}_x = \boldsymbol{0}_K$ corresponds to the classical PCA, and $\boldsymbol{\Sigma}_x = \sigma^2 \boldsymbol{I}_K$ to the Probabilistic Principal Component Analysis (PPCA) model [29]. The prior distribution over the latent variables is defined as Gaussian, i.e., $p(\boldsymbol{z}_i) = \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{0}_K, \boldsymbol{I}_K)$. In this setting, the posterior distribution of the latent variables is also Gaussian,

$$p(\boldsymbol{z}_i|\boldsymbol{x}_i, \Theta) = \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i). \tag{9}$$

EM algorithm can efficiently be used to estimate the parameters of the posterior distribution $\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}_{i=1:N}$ as well as the model parameters $\Theta = \{\boldsymbol{W}, \boldsymbol{m}\}$ by using the known variables of the matrix.

Variational Bayesian Principal Component Analysis (VBPCA) is a fully bayesian treatment of the linear latent variable model defined above, specifically proposed for missing value estimation [15]. Bayesian treatment is especially beneficial for application domains containing high fraction of missing values such as Recommender systems [30], [31]. Unlike PPCA models [29], VBPCA algorithm treats all the model parameters as random variables in addition to the factors. By using proper prior distributions, the model itself penalizes more complex distributions to naturally avoid overfitting. Similarly, the Gaussian priors for the parameters can conveniently be used for the real valued datasets,

$$p(\boldsymbol{m}) = \mathcal{N}(\boldsymbol{m}|0, v_m \boldsymbol{I}_K), \tag{10}$$

$$p(\boldsymbol{W}) = \prod_{j=1}^{K} \mathcal{N}(\boldsymbol{w}_{j,:}|0, v_{m,j} \boldsymbol{I}_K). \tag{11}$$

The model assumes factorization over all the factor loadings packed in the matrix $\boldsymbol{W}$. The parameter set for this model becomes $\Theta = \{\boldsymbol{Z}, \boldsymbol{W}, \boldsymbol{m}\}$, and hyper-parameter set is given as $\zeta = (v_y, v_m, v_{m,j})$. In this setting, the posterior of parameters is not tractable. Hence, instead of inferring the exact posterior $p(\Theta|\boldsymbol{X}, \zeta)$, it is approximated through a simple distribution $q(\Theta)$, which is a form of Variational inference based on mean field approximation [29]. The cost function of the model that is to be minimized is given as

$$\begin{aligned} C(q(\Theta), \zeta) &= \int q(\Theta) log \frac{q(\Theta)}{p(\boldsymbol{X}, \Theta|\zeta)} d\Theta \\ &= \int q(\Theta) log \frac{q(\Theta)}{p(\Theta|\boldsymbol{X}, \zeta)} d\Theta - log p(\boldsymbol{X}|\Theta), \end{aligned} \tag{12}$$

where the first term on the right hand side of the equation is KL divergence between the exact and approximate posterior while the second term is the marginal likelihood. Since the KL divergence is always positive, it forms a lower bound for the model likelihood. VBPCA algorithm introduces factorized form for the approximate posterior as follows:

$$q(\Theta) = \prod_{j=1}^{D} q(m_j) \prod_{j=1}^{D} q(\boldsymbol{w}_{j,:} \prod_{i=1}^{N} q(\boldsymbol{z}_i). \tag{13}$$

Each factor in the parameter set is assumed to be Gaussian, in which the parameters of these distributions are inferred from the observed variables by using EM algorithm given the observed values collected in the set $\Omega$. EM steps are performed to compute the means and co-variances/variances of the distribution parameters as well as the point estimates of the hyper-parameters.

### 4) DEEP LEARNING MODELS

Deep Auto-encoders are popular deep neural network models that seek lower dimensional representations of the observations, which can be interpreted as deep neural network versions of PCA models [32]. Traditional auto-encoders does not have probabilistic interpretation since the latent variables are assumed to be deterministic thus estimated point wise. Let neural network $f$ that performs the mapping from the observed space to the latent space be defined over the parameter set $\Theta_e$, called encoding parameters. Similarly, let neural network $g$ that performs mapping from the latent space to the observed space be defined over $\Theta_d$, called decoding parameters. The mappings are given as

$$\boldsymbol{x}_i = f(\boldsymbol{z}_i, \Theta_d), \tag{14}$$

$$\boldsymbol{z}_i = g(\boldsymbol{x}_i, \Theta_e), \tag{15}$$

such that the reconstruction of an observation with missing values under the trained model parameters is performed by passing through the encoder and the decoder respectively, i.e.,

$$r(\boldsymbol{x}_i|\Theta_d, \Theta_e) = f(g(\boldsymbol{x}_i, \Theta_e), \Theta_d), \tag{16}$$

where $r(\boldsymbol{x}_i|\Theta_d, \Theta_e)$ corresponds to the reconstructed vector through encoding and decoding of $\boldsymbol{x}_i$. AutoREC is an auto-encoder based algorithm specifically proposed to deal with missing values [21]. The parameters of the model $\Theta = \{\Theta_d, \Theta_e\}$ are optimized by using the observed variables of the input matrix hence the objective is defined with the addition of L2 norm regularization on the parameters as follows

$$min_\Theta \sum_{(i,j)\in\Omega} (x_{ij} - r(x_{ij}|\Theta))^2 + \lambda h(\Theta), \tag{17}$$

where $r(x_{ij}|\Theta)$ is the reconstructed $j$th feature of instance $i$ under model current model parameters.

Variational auto-encoders are probabilistic extensions of traditional auto-encoder, which assumes a similar graphical model with the linear latent variable models except the mappings are non-linear through MLPs. The main objective is still to maximize the likelihood of data given the model parameters, i.e., $p(\boldsymbol{X}|\Theta)$. The distributions of the observed variables are modeled as Gaussian,

$$p(\boldsymbol{x}_i|\boldsymbol{z}_i, \Theta_d) = \mathcal{N}(\boldsymbol{x}_i|f(\boldsymbol{z}_i, \Theta_d), \boldsymbol{\Psi}), \tag{18}$$

where $\boldsymbol{\Psi} = \sigma^2 \boldsymbol{I}_D$ is noise covariance matrix. Mean of the distribution of each observation is linked with the corresponding latent variables through a non-linear mapping by the decoder implemented by using MLP. The prior distributions of the latent variables are same as the continuous latent linear models, i.e., $p(\boldsymbol{z}_i) = \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{0}, \boldsymbol{I}_K)$. The cost function that should be minimized to maximize the data likelihood is the same as VBPCA algorithm given by Eq.12. The main difference is on the definition of the approximate posterior over the latent variables. More specifically, VBPCA uses factorized Gaussian distributions, whereas VAE uses MLP, called the encoder network defined as

$$q(\boldsymbol{z}_i|\boldsymbol{x}_i) = \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{\mu}(\boldsymbol{x}_i, \Theta_\mu), \boldsymbol{\Sigma}(\boldsymbol{x}_i, \Theta_\Sigma)). \tag{19}$$

Here there are two encoding networks because the posterior of $\boldsymbol{z}_i$ is Gaussian, thus requiring estimation of both mean and diagonal covariance matrix. $\Theta_\mu$ is the parameter set for encoding mean vector, and $\Theta_\Sigma$ is the parameter set for encoding diagonal covariance. The overall parameters of the networks, $\Theta_\mu$, $\Theta_\Sigma$ and $\Theta_d$, are optimized using stochastic gradient descent [24].

### B. PROPOSED MODEL

In this section, we define our proposed model Feature Specific Neural Matrix Completion (FSNMC) and its training scheme, then we state the differences and potential advantages over the aforementioned models. FSNMC is an MLP based approach developed specifically for the matrix completion task. We introduce $D$ MLPs, where $D$ is the number of features, with independent parameters to increase model capacity to obtain better imputation performance. Each of the MLPs is associated with a single feature, in which the parameters are optimized within a regression framework. It uses a rotational scheme similar to the PCA models described in Section II-A1, where the parameters are optimized given the values of the observed variables as the first step, and then exploited to estimate the associated missing variables as the second step.

We first initialize $D$ independent MLPs associated with each feature denoted as $f_j$ with parameter set $\Theta_j$, where $j = 1 : D$. The associated variable of each network is removed from the input vector, i.e., is excluded from the regressors, so that the number of input nodes for each network becomes $D - 1$. The removed variable then becomes a single output node to form a regular regression model. The corresponding model is given as

$$x_{ij} = f_j(\boldsymbol{x}_{i\tilde{j}}, \Theta_j), \tag{20}$$

where $j = 1, \ldots, D$ corresponds to the network index. The model assumes variable $x_{ij}$ is linked with the other variables $x_{i\tilde{j}}$ of the $i$th observation through the global parameter set $\Theta_j$. The optimization goal is to estimate all the global parameter sets, i.e., $\{\Theta_j\}_{j=1:D}$, that minimize the reconstruction errors of the given observed variables of the matrix $\boldsymbol{X}$.

The training algorithm has a rotational scheme, where the model parameters are optimized at each iteration given the completed matrix and the estimated parameters from the previous iteration. Hence, we denote $\Theta_{j,t}$ as the parameters of the $j$th network at iteration $t$. The objective is to find the optimal $\Theta_t = \{\Theta_{j,t}\}_{j=1,\ldots,D}$ that minimizes the cost function given below for each iteration $t$:

$$min_{\Theta_{j,t}} \sum_{(i,j)\in\Omega} (x_{ij,t} - f_j(x_{i\tilde{j},t}, \Theta_{j,t}))^2 + \lambda h(\Theta_{j,t}). \tag{21}$$

For instance, for a two layer MLP network structure, the parameter set that is to be optimized will be $\Theta_{j,t} = \{\boldsymbol{W}_{1j,t} \in \mathbb{R}^{D\times M}, \boldsymbol{W}_{2j,t} \in \mathbb{R}^M, \boldsymbol{m}_{1j,t} \in \mathbb{R}^M, m_{2j,t} \in \mathbb{R}\}$. A graphical representation for this structure is given in Figure 2. The cost function is similar to AutoREC. However, unlike AutoREC, we optimize the network parameters iteratively after updating the missing values. The cost function additionally includes a regularization term to control the over-fitting with a strength parameter $\lambda$. A convenient choice is to use L2 norm on the weight parameters to shrink them towards small numbers as follows

$$h(\Theta_{j,t}) = \|W_{1j,t}\|^2 + \|W_{2j,t}\|^2. \tag{22}$$

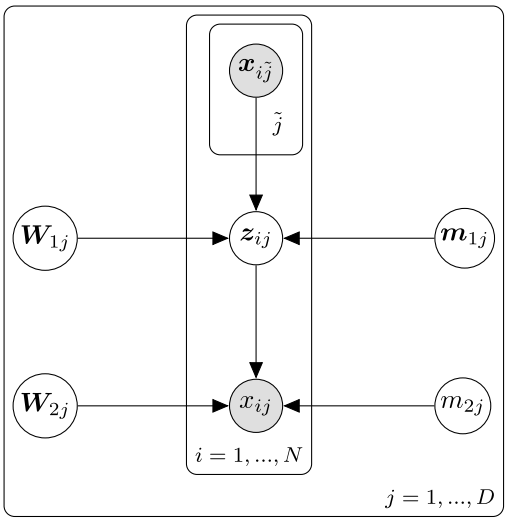



**FIGURE 2.** Graphical model of FSNMC for a two layer MLP configuration.

After optimizing the parameters of the networks at iteration $t$, the model parameters are then exploited to infer

**Algorithm 1** FSNMC Algorithm

**procedure** FSNMC($\boldsymbol{X}$, $\Omega$)
  Initialize parameters $\Theta_0 = \{\Theta_{j,0}\}_{j=1,\dots,D}$ randomly;
  Normalize $\boldsymbol{X}$;
  $x_{ij,(0)} = 0 \ (i,j) \in \bar{\Omega}$; ▷ Initialize missing variables
  $t = 1$;
  **while** $conv(\boldsymbol{X}_t - \boldsymbol{X}_{t-1})$ **do** ▷ Check for convergence
    **for** $j = 1 : D$ **do** ▷ Parameter inferring loop
      $\Theta^0_{j,t} = \Theta^F_{j,t-1}$; ▷ Transfer parameters
      $\Theta^F_{j,t} = min_{\Theta_{j,t}} \sum_{(i,j)\in\Omega}(x_{ij,t} - f_j(\boldsymbol{x}_{\tilde{ij},t}, \Theta_{j,t}))^2 + \lambda h(\Theta_{j,t})$ ▷ Train $j$th network
    **for** $j = 1 : D$ **do** ▷ Model exploiting loop
      $r(x_{ij,t}|\Theta^F_{j,t}) = f_j(x_{\tilde{ij}}, \Theta^F_{j,t})$; ▷ Reconstruct missing variables
      $x_{ij,(t+1)} := x_{ij,t} + \rho(r(x_{ij,t}|\Theta_{j,t}) - x_{ij,t}) \ (i,j) \in \bar{\Omega}$; ▷ Update missing variables
    $t = t + 1$;
  **return** $\boldsymbol{X}_t$

the missing values. Given the parameter set $\Theta_{j,t}$, the reconstruction of a missing variable $x_{ij,t}$ is obtained with a single forward pass through each network,

$$r(x_{ij,t}|\Theta_{j,t}) = f_j(x_{\tilde{ij}}, \Theta_{j,t}). \tag{23}$$

Instead of updating each missing variable with the reconstruction, we introduce a damping parameter to smooth out the learning, which is empirically observed to provide better convergence stability. Consequently, the final update equation for iteration $t$ is given as

$$x_{ij,(t+1)} = x_{ij,t} + \rho(r(x_{ij,t}|\Theta_{j,t}) - x_{ij,t}) \ (i,j) \in \bar{\Omega}, \tag{24}$$

where $\rho$ is the damping parameter.

We perform the optimization of the cost function, and exploitation of the trained model to update missing values at each iteration. However, re-training each network with the updated missing variables with random parameter initialization at each iteration would result in a substantial increase in training time. Instead we store the network parameters between iterations where each network's final weights are carried through. For example, at iteration $t = 0$, the model parameters are initialized with small random numbers as is customary in neural network training. After the first iteration, the optimized model parameters lead to a minima point over the surface of the cost function. Since the difference between two consecutive iterations is the slowly changing missing values, beginning from a random point on the cost function surface for the next optimization would be time consuming. Assuming that the point found in the previous optimization is close to the minima of the cost function of the previous iteration, the number of epochs needed to reach the optimal minima shall reduce dramatically with the application of this method. The weights of the $j$th network are carried through as $\Theta^0_{j,t+1} = \Theta^F_{j,t}$, where $\Theta^0_{j,t+1}$ corresponds to the initial parameters of network $j$ at iteration $t+1$ and $\Theta^F_{j,t}$ corresponds to the final optimized parameter at iteration $t$. Algorithm 1 provides the detailed pseudo-algorithm of the proposed rotational approach.

If we assume the neural network used for each feature is equivalent in structure, and each of them has $l$ hidden layers ($l > 2$) and $K$ hidden nodes per layer, the time complexity of a single forward pass of $N$ examples will be $O(N * (D * K + l * K^2 + l))$, where $D$ is the dimension of the input. Similarly, a back-propagation invokes the same computational complexity due to its inverse traversal of the same computational graph. If we assume that the algorithm requires $F$ epochs to converge in its inner loops, then we have $O(F * N * (D * K + l * K^2 + l))$ for parameter inferring loop of each feature. Furthermore, we perform this step for the network of each feature resulting in $O(F * N * D * (D * K + l * K^2 + l))$. Note that the model exploiting loop requires a single forward pass which does not affect asymptotic complexity. Also, the parameter storage procedure decreases the number of epochs required for the inner loop convergence for the latter outer iterations. That results in an amortized computational complexity of $O(N*D*(D*K+l*K^2+l))$ for a single outer iteration. The main burden can be recognized as a quadratic dependency on the number of features-$D$. However, first, we target scientific experimental datasets, which are generally classified as tall datasets, having a significantly less number of features than the number of instances, i.e., $N \gg D$. Second, the parameter inferring and model exploiting loops are performed sequentially, thus allowing the former loop to be parallelized. To this end, with efficient parallelization, we can conclude that the time complexity of the model is asymptotically equivalent to the standard neural network training complexity.

The proposed algorithm differs from the methods introduced in Section II-A with the following aspects:

- Linear latent variable models assume common latent space for all features. However, FSNMC defines a separate latent space, i.e, a separate set of latent variables for each feature, which increases both the model capacity and the flexibility.
- PCA, Matrix Factorization and Auto-encoder methods minimize a common reconstruction error defined over

all of the observed variables of the incomplete matrix. However, FSNMC defines an independent optimization goal for each feature.

- TSRE algorithm assumes a linear relationship between the observed and unobserved variables of each instance and the regression framework is based on this linear relationship over the factor loading matrix. However, FSNMC uses MLPs to extract the non-linear relationships between the associated features and the regressors.
- Although our proposed algorithm is a predictive model unlike the generative models such as VBPCA and VAE, for matrix completion tasks the accuracy is generally more important than expressing the domain knowledge. FSNMC specifically focuses on the former by modeling a non-linear strong regression framework.

## III. COMPARATIVE STUDY

### A. EXPERIMENTAL SETUP

We perform experiments to compare the performance of the proposed model with four baseline algorithms, TSRE, VBPCA, AutoREC, and VAE. We select four small-sized, i.e., the number of observations is smaller than 1000, and two medium-sized, i.e., the number of observations is larger than 3000, real world datasets to observe the performances with respect to varying matrix sizes. The datasets are labeled as concrete, nutrient, wine, olive oil, protein and abalone. The concrete dataset ($N = 103$, $D = 10$) describes three mechanical features of the concrete according to the concentration of seven ingredients for different samples of concrete, resulting in a total number of 10 features. The dataset is complete with no inherent missing values, and is available on the University of California Irvine's Machine Learning Repository (UCI-MLR) [33]. The nutrient dataset ($N = 104$, $D = 20$) describes the nutritional values of different vegetables. For some vegetables, the content in some nutrients is denoted as ''traces''. In this case, the content of the nutrient is assumed to be negligible and zero. The dataset is maintained by Health Canada, and already contains 1.9% missing values [34]. The wine dataset ($N = 173, D = 13$) describes chemical attributes of wine samples. The dataset is complete, and is available on the UCI-MLR [35]. Olive oil dataset ($N = 572, D = 8$) describes the concentration of the fatty acids in the wine produced in different regions. The dataset is also complete and compiled by Forina *et al.* [36]. The protein dataset ($N = 45730$, $D = 9$) includes a significantly higher number of samples that describe the physicochemical properties of the protein tertiary structure [37]. The final dataset is the Abalone ($N = 4177, D = 8$), which consists of the attributes of abalones, measured for each individual animal [38]. The dataset has gender information as a categorical variable which is removed from the dataset to obtain a homogeneous real valued matrix.

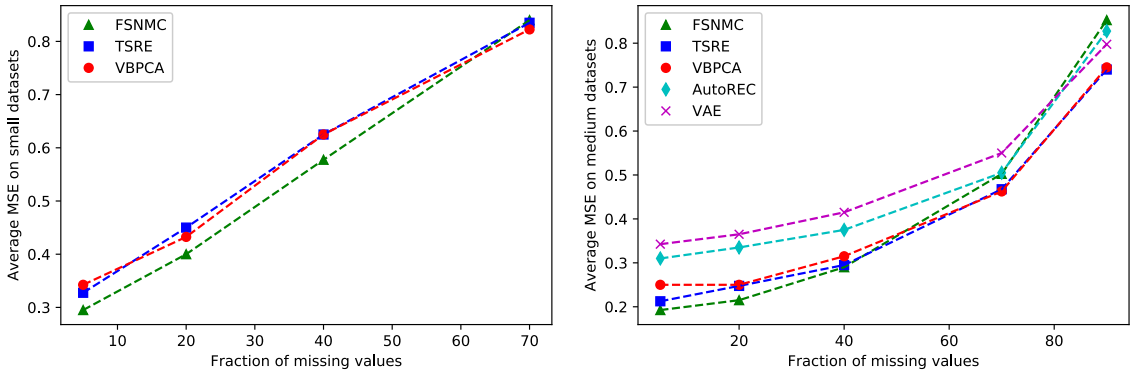
### B. IMPLEMENTATION DETAILS

TSRE algorithm is an iterative method, where SVD is performed at each iteration to estimate the factor loading matrix $\boldsymbol{W}_t$, followed by the regression of the missing variables given the observed variables through $\boldsymbol{W}_t$. After the regression is performed to update the missing values for each instance by using Eq. 7, the model parameter $\boldsymbol{m}_t$ is evaluated by column-wise mean operation on the matrix updated with the new predictions. The model parameter set to be estimated is given as $\Theta_t = \{\boldsymbol{W}_t, \boldsymbol{m}_t\}$ and the hyper-parameter set to be optimized is given as $\zeta = \{\gamma, K\}$, in which $\gamma$ defines the fraction of observed variables to be used for early stopping, and $K$ corresponds to the number of latent variables.

VBPCA algorithm performs Variational Bayesian Expectation Maximization (VB-EM) [29] to update both the parameter and hyper-parameter sets given the observed variables of the instances. The parameter set to be estimated is given as $\Theta = \{\boldsymbol{W}, \boldsymbol{Z}, \boldsymbol{m}\}$ and the hyper-parameter set is given as $\zeta = \{v_y, v_m, v_{m,j}, K\}$. Note that VBPCA optimizes both the parameter and hyper-parameter sets except the dimension of the latent space, $K$. This hyper-parameter is optimized with cross-validation.

For the MLP based approaches including AutoREC, VAE and FSNMC, the network structure is defined over the hyper-parameter $l$, which corresponds to the number of hidden layers, and $K$, which is the number of latent nodes in each single hidden layer. These values are optimized with cross-validation. The search space is defined over a grid from a predefined set $l = \{1, 2\}$ and $K = \{2, 3, 4, 5\}$. The trainable parameters of these models are optimized by using the stochastic gradient descent method. AutoREC has encoder and decoder MLP parameters, $\Theta = \{\Theta_d, \Theta_e\}$. Cross-validation is performed to optimize the network structure and the regularization strength $\lambda$, so that the complete hyper-parameter set can be denoted as $\zeta = (K, l, \lambda)$. VAE, on the other hand, does not have a regularization parameter thus the hyper-parameter set is defined as $\zeta = (K, l)$, and the parameter set is given as $\Theta = \{\Theta_\mu, \Theta_\Sigma, \Theta_d\}$.

Unlike VAE and AutoREC, FSNMC is an iterative algorithm such that parameter optimization is performed at each iteration $t$ given the updated missing values from the previous step. Another major difference is the number of networks associated with the number of features. In this case, the parameter set of the algorithm to be estimated by using the stochastic gradient descent method at iteration $t$ is given as $\Theta_t = \{\Theta_{j,t}\}_{j=1,\ldots,D}$. On the other hand, the hyper-parameter set of the model is $\zeta = (K, l, \lambda, \gamma, \rho)$. Two of these parameters, $\gamma$, which is the fraction of the observed variables to be used for early stopping, and damping parameter $\rho$, are fixed empirically as %5 and 0.1 respectively. FSNMC algorithm is implemented by using TensorFlow, and the complete script is accessible on GitHub repository at https://github.com/maktukmak/NNMC.




(a) On Small Datasets

(b) On Medium Datasets

**FIGURE 3.** **Averaged MSE values obtained over all the datasets.**

### C. CROSS-VALIDATION AND EVALUATION PROCEDURE

The hyper-parameter optimization starts with separating some of the observed variables in the training set and keeping them in a separate validation set. The fraction of splitting is fixed at %5 such that %95 of the observed variables is used to infer the parameter set $\Theta$ while the rest is used for optimizing the hyper-parameter set $\zeta$. Since the experiments are repeated with varying fractions of missing values, the hyper-parameter optimization is repeated for each algorithm, each dataset, and each fraction of missing value combination. Each optimization exploit grid search with combination of the elements of $\zeta$, i.e., $|\zeta|!$ experiments are performed for each scenario. Mean squared error on the validation set is used as performance metric, which is given as

$$MSE_{val} = \frac{1}{|\bar{\Omega}_{val}|} \sum_{i,j \in \bar{\Omega}_{val}} (x_{ij} - r(x_{ij}|\Theta, \zeta))^2. \tag{25}$$

We repeat each experiment ten times to obtain a statistical average error, and then we select the hyper-parameter combination corresponding to the lowest error.

Given the optimized hyper-parameter set of each test scenario, we run the algorithms ten times by merging the validation and training sets. The experimental setup has slight differences for small and medium datasets. We use Missing Completely at Random (MCAR) mechanism to remove %5, %20, %40 and %70 of the variables for small datasets and keep them in the test set. However, for medium datasets, we setup an additional experiment with %90 fraction of missing values to observe the performances at a relatively more sparse setting. Removing %90 of the variables in small datasets results in artificially low performance for each algorithm likely due to having very few variables to infer the model parameters and are not reported here. The reported performance metric is the mean squared error evaluated on the test set, which is given as

$$MSE_{test} = \frac{1}{|\bar{\Omega}|} \sum_{i,j \in \bar{\Omega}} (x_{ij} - r(x_{ij}|\Theta, \zeta))^2. \tag{26}$$

To perform the experiments, we exploit high-performance computing cluster of the University of South Florida to employ parallelization among several processors.

**TABLE 1.** **MSE comparison on small datasets.**

| | FSNMC | TSRE | VBPCA |
|---|---|---|---|
| **Concrete** | | | |
| %5 | **0.14 (± 0.02)** | 0.15 (± 0.03) | 0.23 (± 0.03) |
| %20 | **0.33 (± 0.03)** | 0.39 (± 0.05) | 0.37 (± 0.04) |
| %40 | **0.59 (± 0.07)** | 0.70 (± 0.08) | 0.74 (± 0.07) |
| %70 | **0.94 (± 0.06)** | 0.95 (± 0.06) | 0.96 (± 0.05) |
| **Nutrient** | | | |
| %5 | **0.36 (± 0.03)** | 0.44 (± 0.04) | 0.43 (± 0.03) |
| %20 | **0.48 (± 0.03)** | 0.52 (± 0.05) | 0.52 (± 0.03) |
| %40 | **0.63 (± 0.05)** | 0.66 (± 0.07) | 0.67 (± 0.06) |
| %70 | **0.81 (± 0.07)** | 0.83 (± 0.07) | 0.83 (± 0.06) |
| **Wine** | | | |
| %5 | **0.47 (± 0.03)** | 0.51 (± 0.03) | 0.49 (± 0.02) |
| %20 | **0.51 (± 0.04)** | 0.56 (± 0.03) | 0.53 (± 0.04) |
| %40 | 0.62 (± 0.06) | 0.64 (± 0.04) | **0.61 (± 0.05)** |
| %70 | 0.87 (± 0.05) | 0.82 (± 0.06) | **0.79 (± 0.04)** |
| **Olive Oil** | | | |
| %5 | **0.21 (± 0.02)** | 0.21 (± 0.04) | 0.22 (± 0.03) |
| %20 | **0.28 (± 0.03)** | 0.33 (± 0.03) | 0.31 (± 0.04) |
| %40 | **0.47 (± 0.07)** | 0.50 (± 0.06) | 0.48 (± 0.06) |
| %70 | 0.74 (± 0.07) | 0.74 (± 0.07) | **0.71 (± 0.06)** |

### D. RESULTS AND DISCUSSION

The performance comparison for small scale datasets is given in Table 1. This table is compiled with the results for FSNMC, TSRE and VBPCA algorithms. AutoREC and VAE algorithms are excluded due to their significantly lower accuracy, most likely due to the small size of the dataset. Bold values correspond to the minimum errors associated with each specific test case. FSNMC clearly outperforms the other algorithms in majority of the test scenarios, especially for low fractions of missing values. In order to demonstrate a holistic view of the performance, Figure 3.a shows the average MSE values over all datasets for each of the varying fractions of

**TABLE 2.** MSE comparison on medium datasets.

| | FSNMC | TSRE | VBPCA | AutoREC | VAE |
|---|---|---|---|---|---|
| **Protein** | | | | | |
| %5 | **0.21 (± 0.02)** | 0.24 (± 0.02) | 0.24 (± 0.03) | 0.32 (± 0.03) | 0.32 (± 0.02) |
| %20 | **0.22 (± 0.02)** | 0.26 (± 0.03) | 0.25 (± 0.02) | 0.34 (± 0.04) | 0.34 (± 0.05) |
| %40 | **0.30 (± 0.04)** | **0.30 (± 0.04)** | **0.30 (± 0.03)** | 0.37 (± 0.03) | 0.39 (± 0.04) |
| %70 | **0.44 (± 0.04)** | 0.47 (± 0.03) | **0.44 (± 0.03)** | 0.50 (± 0.05) | 0.50 (± 0.07) |
| %90 | 0.81 (± 0.06) | **0.72 (± 0.05)** | 0.73 (± 0.05) | 0.75 (± 0.07) | 0.76 (± 0.06) |
| **Abalone** | | | | | |
| %5 | **0.15 (± 0.03)** | **0.15 (± 0.03)** | 0.30 (± 0.08) | 0.30 (± 0.04) | 0.36 (± 0.05) |
| %20 | **0.16 (± 0.03)** | 0.21 (± 0.04) | 0.23 (± 0.05) | 0.34 (± 0.06) | 0.34 (± 0.06) |
| %40 | **0.24 (± 0.04)** | 0.25 (± 0.04) | 0.27 (± 0.05) | 0.36 (± 0.05) | 0.38 (± 0.06) |
| %70 | 0.48 (± 0.06) | **0.39 (± 0.05)** | 0.40 (± 0.04) | 0.51 (± 0.06) | 0.49 (± 0.04) |
| %90 | 0.84 (± 0.05) | **0.71 (± 0.05)** | **0.71 (± 0.04)** | 0.82 (± 0.07) | 0.76 (± 0.07) |

missing values. Except the fraction of %70, we can conclude that FSNMC is the superior choice to infer missing values.

The MSE performances for medium scale datasets is compiled in Table 2. In this case, we include AutoREC and VAE algorithms since they provide competitive performance. However, especially for low fractions of missing values, FSNMC continuous to consistently outperform all other approaches regardless of the dataset, whereas TSRE and especially VBPCA provide better accuracy at higher fractions. Synthetic dataset results are compatible with the general observations made for real-world datasets. Figure 3.b shows the average MSE values over all the datasets for each of the varying fractions of missing values where the proposed FSNMC method and VBPCA outperform the competing algorithms at low and high fractions of missing values respectively.

## IV. CONCLUSION AND FUTURE WORK

This article presents an MLP based self-supervised matrix completion algorithm to deal with missing values, specifically in the scientific experimental datasets characterized as non-sparse and small to medium sized. The performance of the proposed FSNMC algorithm is compared with the state-of-the-art methods. The experiments with a wide range of combinations on dataset sizes and fractions of missing values demonstrate the superior performance of the proposed algorithm on both small and medium sized datasets.

The first limitation of the proposed model is the computational complexity that scales quadratically with respect to the number of features. This can be problematic if the implementation is not efficiently optimized for parallel processing during the optimization of the networks associated with each feature. Hence, we mainly suggest the proposed algorithm for tall datasets as most of the scientific experiment datasets are. The second limitation comes from the number of additional hyper-parameters due to the structure of the algorithm itself as discussed in Section III-B. This might increase the tuning time of the algorithm significantly if one uses grid search to find the optimal hyper-parameter set. Fortunately, there exists some continuous optimization methods based on Gaussian processes to reduce the grid search time drastically, which can be adapted to the proposed algorithm. Third limitation is the decreasing performance of the model when the dataset sparsity exceeds approximately 70%. In this setting, we would suggest using linear statistical models such as the VBPCA. Because, in a non-linear neural network model, a significantly low ratio between the number of observed variables and the number of trainable parameters might easily result in over-fitting.

Future work will include ensemble methods to combine the predictions of FSNMC with the other promising matrix completion algorithms, such as VBPCA, for generally superior performance regardless of the dataset size and sparsity. Furthermore, the heterogeneous datasets including both categorical and numerical features will be considered. Specifically, instead of regression for each feature, classification frameworks can be utilized to impute the mixed-data type observations.

## REFERENCES

[1] A. Gogna and A. Majumdar, "Matrix completion incorporating auxiliary information for recommender system design," *Expert Syst. Appl.*, vol. 42, no. 14, pp. 5789–5799, Aug. 2015.
[2] Y. Liu, Y. Pan, Z. Sun, and D. Huang, "Statistical monitoring of wastewater treatment plants using variational Bayesian PCA," *Ind. Eng. Chem. Res.*, vol. 53, no. 8, pp. 3272–3282, Feb. 2014.
[3] H. Wang, R. Zhao, and Y. Cen, "Rank adaptive atomic decomposition for low-rank matrix completion and its application on image recovery," *Neurocomputing*, vol. 145, pp. 374–380, Dec. 2014.
[4] S. Mercier, M. Mondor, B. Marcos, C. Moresoli, and S. Villeneuve, "Estimation of missing values in a food property database by matrix completion using PCA-based approaches," *Chemometric Intell. Lab. Syst.*, vol. 166, pp. 37–48, Jul. 2017.
[5] X.-Y. Pan, Y. Tian, Y. Huang, and H.-B. Shen, "Towards better accuracy for missing value estimation of epistatic miniarray profiling data by a novel ensemble approach," *Genomics*, vol. 97, no. 5, pp. 257–264, May 2011.
[6] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *J. ACM*, vol. 58, no. 3, pp. 1–37, 2011.
[7] X. Peng Li, L. Huang, H. Cheung So, and B. Zhao, "A survey on matrix completion: Perspective of signal processing," 2019, *arXiv:1901.10885*. [Online]. Available: http://arxiv.org/abs/1901.10885
[8] J.-F. Cai, E. J. Candès, and Z. Shen, "A singular value thresholding algorithm for matrix completion," *SIAM J. Optim.*, vol. 20, no. 4, pp. 1956–1982, Jan. 2010.
[9] G. Marjanovic and V. Solo, "On $l_q$ optimization and matrix completion," *IEEE Trans. Signal Process.*, vol. 60, no. 11, pp. 5714–5724, Nov. 2012.
[10] B. Eriksson, L. Balzano, and R. Nowak, "High-rank matrix completion," in *Proc. Artif. Intell. Statist.*, 2012, pp. 373–381.

[11] Y. Hu, D. Zhang, J. Ye, X. Li, and X. He, "Fast and accurate matrix completion via truncated nuclear norm regularization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 9, pp. 2117–2130, Sep. 2013.
[12] T.-H. Oh, Y.-W. Tai, J.-C. Bazin, H. Kim, and I. S. Kweon, "Partial sum minimization of singular values in robust PCA: Algorithm and applications," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 4, pp. 744–758, Apr. 2016.
[13] Z. Wang, M.-J. Lai, Z. Lu, W. Fan, H. Davulcu, and J. Ye, "Rank-one matrix pursuit for matrix completion," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 91–99.
[14] C. Xu, Z. Lin, and H. Zha, "A unified convex surrogate for the schatten-p norm," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 1–8.
[15] A. Ilin and T. Raiko, "Practical approaches to principal component analysis in the presence of missing values," *J. Mach. Learn. Res.*, vol. 11, no. 7, pp. 1957–2000, 2010.
[16] B. Walczak and D. L. Massart, "Dealing with missing data: Part I," *Chemometrics Intell. Lab. Syst.*, vol. 58, no. 1, pp. 15–27, 2001.
[17] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, Aug. 2009.
[18] Y. Shan and G. Deng, "Kernel PCA regression for missing data estimation in DNA microarray analysis," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2009, pp. 1477–1480.
[19] L. Li, Y. Li, and Z. Li, "Efficient missing data imputing for traffic flow by considering temporal and spatial dependence," *Transp. Res. C, Emerg. Technol.*, vol. 34, pp. 108–120, Sep. 2013.
[20] M. Scholz, F. Kaplan, C. L. Guy, J. Kopka, and J. Selbig, "Non-linear PCA: A missing data approach," *Bioinformatics*, vol. 21, no. 20, pp. 3887–3895, Oct. 2005.
[21] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "AutoRec: Autoencoders meet collaborative filtering," in *Proc. 24th Int. Conf. World Wide Web*, 2015, pp. 111–112.
[22] J. Fan and T. Chow, "Deep learning based matrix completion," *Neurocomputing*, vol. 266, pp. 540–549, Nov. 2017.
[23] M. H. Nguyen and F. Torre, "Robust kernel principal component analysis," in *Proc. Adv. Neural Inf. Process. Syst.*, 2009, pp. 1185–1192.
[24] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013, *arXiv:1312.6114*. [Online]. Available: http://arxiv.org/abs/1312.6114
[25] A. Folch-Fortuny, F. Arteaga, and A. Ferrer, "PCA model building with missing data: New proposals and a comparative study," *Chemometric Intell. Lab. Syst.*, vol. 146, pp. 77–88, Aug. 2015.
[26] S. Mercier and I. Uysal, "Noisy matrix completion on a novel neural network framework," *Chemometric Intell. Lab. Syst.*, vol. 177, pp. 1–7, Jun. 2018.
[27] M. Aktukmak, S. Mercier, and I. Uysal, "A neural net framework for accumulative feature-based matrix completion," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–6.
[28] J. Lee, S. Kim, G. Lebanon, Y. Singer, and S. Bengio, "LLORMA: Local low-rank matrix approximation," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 442–465, 2016.
[29] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
[30] M. Aktukmak, Y. Yilmaz, and I. Uysal, "A probabilistic framework to incorporate mixed-data type features: Matrix factorization with multimodal side information," *Neurocomputing*, vol. 367, pp. 164–175, Nov. 2019.
[31] M. Aktukmak, Y. Yilmaz, and I. Uysal, "Quick and accurate attack detection in recommender systems through user attributes," in *Proc. 13th ACM Conf. Recommender Syst.*, Sep. 2019, p. 348.
[32] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proc. ICML Workshop Unsupervised Transf. Learn.*, 2012, pp. 37–49.
[33] I.-C. Yeh, "Modeling slump flow of concrete using second-order regressions and artificial neural networks," *Cement Concrete Composites*, vol. 29, no. 6, pp. 474–480, Jul. 2007.
[34] Health Canada. (2008). *Nutrient Value of Some Common Foods—Booklet*. Accessed: Oct. 30, 2020. [Online]. Available: https://www.canada.ca/content/dam/hc-sc/migration/hc-sc/fn-an/alt_formats/pdf/nutrition/fiche-nutri-data/nvscf-vnqau-eng.pdf
[35] M. Lichman, *UCI Machine Learning Repository, Wine*. Irvine, CA, USA: Univ. California, 2008.
[36] M. Forina, C. Armanino, S. Lanteri, and E. Tiscornia, "Classification of olive oils from their fatty acid composition," in *Food Research and Data Analysis*, H. Martens and H. Russwurm, Jr., Eds. London, U.K.: Applied Science, 1983.
[37] M. Lichman, *UCI Machine Learning Repository, Protein Tertiary Structure*. Irvine, CA, USA: Univ. California, 2008.
[38] M. Lichman, *UCI Machine Learning Repository, Abalone*. Irvine, CA, USA: Univ. California, 2008.

**MEHMET AKTUKMAK** received the B.S. degree in electrical and electronics engineering from Hacettepe University, Ankara, Turkey, in 2009, and the M.S. degree in electrical and electronics engineering from Middle East Technical University, Ankara, in 2012. He is currently pursuing the Ph.D. degree in electrical engineering with the University of South Florida, Tampa, FL, USA. From 2009 to 2017, he worked in a military technology company in Turkey as a Senior Digital Design Engineer. His research interests include real-time image/video processing, matrix completion, data fusion, variational inference, meta learning, and recommender systems.

**SAMUEL M. MERCIER** received the B.Eng. degree in biotechnological engineering from the University of Sherbrooke, QC, Canada, in 2012, and the Ph.D. degree in chemical engineering, on the topic of modeling and optimization of pasta processes, in 2016. From 2016 to 2018, he joined the University of South Florida, Tampa, FL, USA, to conduct research in the field of artificial intelligence applied to food processes. Since 2018, he has been the Chief Data Scientist at Decathlon, Canada. His research interests include computer vision, multivariable analysis, and recommendation systems.

**ISMAIL UYSAL** (Member, IEEE) received the B.S. degree in electrical and electronics engineering from Middle East Technical University, Ankara, Turkey, in 1998, and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Florida (UF), Gainesville, FL, USA, in 2006 and 2008, respectively. From 2008 to 2010, he was a Postdoctoral Research Fellow with the Research Center for Food Distribution and Retailing, UF. Since 2010, he has been with the University of South Florida, where he is currently an Assistant Professor of Electrical Engineering and the Director of the Radio Frequency Identification (RFID) Laboratory for Applied Research, College of Engineering. His research interests include deep machine learning theory and applications in semi-supervised and unsupervised settings, data-oriented applications of RFID systems in healthcare and food supply chains, and signal processing algorithms for brain–computer interfaces.