Deep Bayesian Nonparametric Factor Analysis

Arunesh Mittal
Columbia University
am4589@columbia.edu

Paul Sajda
Columbia University
psajda@columbia.edu

John Paisley
Columbia University
jpaisley@columbia.edu

Abstract

We propose a deep generative factor analysis model with beta process prior that can approximate complex non-factorial distributions over the latent codes. We outline a stochastic EM algorithm for scalable inference in a specific instantiation of this model and present some preliminary results.

1 Introduction

Latent factor models provide a means to discover shared latent structure in large datasets by uncovering relationships between the observed data. These models make the assumption that the observed data is a mixture of latent factors. The data generating process for such models can be viewed as matrix factorization model, where the data matrix $X \in \mathbb{R}^{D \times N}$ is modelled as the matrix product $X \approx \Phi\xi$, where $\Phi \in \mathbb{R}^{D \times M}$ is the factor loading matrix and $\xi \in \mathbb{R}^{M \times N}$ is a matrix of latent variables. The columns $x_{1:N}$ of the matrix X are the observed data, the columns $\Phi_{:,1:M}$ of Φ represent the M factors, and the M entries of each column ξ_n of the latent matrix ξ indicate the contribution of each of the factors that linearly combine to compose an observed data point x_n. That is, we allow each observation to possess combinations of up to M latent features. Then, given data the inference procedure entails learning the dictionary as well as the latent mixture components. For discrete ξ, computing the optimal values of ξ_n requires a search over 2^M possible binary vectors and is computationally intractable even for modestly sized M, hence, we must resort to greedy search over this space. In addition, since M is typically unknown, we would like to also infer the number of components in conjunction to the dictionary and components.

Building upon previous work [5, 6, 2, 10, 9], we propose a non-linear sparse coding factor analysis model based on Bayesian nonparametrics. The model employs non-linear "multiplexer" neural net that encodes latent binary vectors $z_n \in \{0, 1\}^K$ to sparse latent variables $\xi_n \in \mathbb{R}_+^M$. The network has the capacity to non-linearly explore the large parameter space over the latent encodings ξ_n. In addition, given a factorial distribution over z_n, the network can learn the correlation structure and approximate non-factorial distributions over the latent codes ξ_n at the deepest layer. This allows the model to generate better samples than traditional linear factor models with factorial prior over the latents. By defining a sparse beta-Bernoulli process prior on the z_n, the model learns the optimal size K. Despite the non-linearity of the model, the parameters of the model are still interpretable as the interaction between the factor loading matrix Φ and the outputs of the multiplexer network ξ_n is a linear operation. We propose a stochastic MAP-EM algorithm with a "selective" M step for efficient scalable inference in this model. [9].

2 Generative Model

We use the finite limit approximation to the beta process [2, 6]. For $\pi_k \sim \text{Beta}\left(\alpha\gamma/K, \alpha\left(1-\gamma\, K\right)\right)$ and $\Phi_k \sim g(\cdot)$, the random measure $H_K = \sum_{k=1}^{K} \pi_k \delta_{\Phi_k}$, $\lim_{K\to\infty} H_K$ converges in distribution to $H \sim BP(\alpha, \gamma)$. In addition, for $z_{nk} \sim \text{Bern}(\pi_k)$, where $\pi_k \sim \text{Beta}\left(\alpha\gamma/K, \alpha\left(1-\gamma\, K\right)\right)$, $k = 1, \ldots, K$, the random measure $G_n^K = \sum_{k=1}^{K} z_{nk}\delta_{\Phi_k}$ converges to a Bernoulli process [7].

Third workshop on Bayesian Deep Learning (NeurIPS 2018), Montréal, Canada.

We model the generative process for the data as follows: Given data $x = \{x_n\}_{n=1}^N$, the corresponding latent factors $z_n \in \{z_n\}_{n=1}^N$ are drawn from a Bernoulli process (BeP) parameterized by a beta process (BP), where, the Bernoulli process prior over each of the k factors $z_{nk} \in z_n$, is parameterized by π_k drawn from a beta process. Each latent variable ξ_n is drawn from the distribution $h(\cdot)$ parameterized by the latent factor z_n via a neural network $\varphi(\cdot)$ with parameters θ_φ, where, the L layered neural network $\varphi(\cdot)$, maps the binary latent code $z_n \in \{0,1\}^K$ to $\mathcal{S} \subset \mathbb{R}^M$ via a neural net: $\varphi(z_n) = \sigma_L W_L \sigma(W_{L-1}\sigma_{l-1}(\dots W_l \sigma_l(\dots \sigma_0(W_0 z_n))))$. The factor loading matrix $\Phi \in \{\mathcal{M} : \mathcal{M} \subset \mathbb{R}^{D\times M}\}$ is drawn from the distribution $g(\cdot)$ with parameters θ_Φ. The scaling factor λ_n for each data point x_n is drawn from a Gaussian distribution. Finally, each data point x_n is drawn from a isotropic Gaussian distribution where the mean is parameterized by the matrix-vector product of factor loading matrix Φ and the encoded latent factor ξ_n, scaled by λ_n.

$$\begin{aligned}
\pi_k &\sim \mathrm{Beta}\left(\alpha(\gamma/K), \alpha\,(1-\gamma\,K)\right)\\
z_{nk} &\sim \mathrm{Bern}(\pi_k)\\
\xi_n &\sim h(\varphi(z_n;\theta_\xi))\\
\Phi &\sim g(\Phi\,;\theta_\Phi)\\
\lambda_n &\sim \mathcal{N}(0,c)\\
x_n &\sim \mathcal{N}(\lambda_n \cdot \Phi\,\xi, \sigma^2 I)
\end{aligned}$$

In the subsequent discussion, we choose specific distributional forms for $h(\cdot)$ and $g(\cdot)$ to illustrate an instance of the general generative model outlined above. We illustrate how the generative model with sparsity inducing beta prior can be applied to non-parametric dictionary learning. Specifically, for dictionary learning, we choose $h(\cdot)$ to be a M-dimensional Dirichlet distribution $\mathrm{Dir}(\xi_n|\varphi(z_{n1}),\dots,\varphi(z_{nK}))$ and choose $g(\cdot)$ to be the Dirac measure $\delta_{\theta_\Phi}(\Phi)$, where $\theta_\Phi = D \in \mathbb{R}^{D\times M}$. Each data point x_n is then drawn from $\mathcal{N}(\lambda_n \cdot f_\theta(z_n), \sigma^2 I)$, where, $f_\theta(z_n) = \Phi\,\mathbb{E}[\xi_n]$ and $\theta = \{\theta_\xi, \theta_\Phi\}$. For our preliminary experiments we chose a dense neural network that maps z_n to $\mathbb{E}[\xi_n]$, where the last layer was a softmax layer that output values over the Δ_{M-1} simplex. More generally, one could use $\varphi(\cdot)$ to parameterize the natural parameters of a exponential family distribution such as a gamma or Poisson distribution [8].

3 MAP-EM Inference

We propose a MAP-EM algorithm to perform inference in this model. We compute point estimates for z and θ and posterior distributions over π and λ. Since, $\pi \perp\!\!\!\perp \lambda \mid z$, the conditional posterior distribution factorizes as: $p(\pi,\lambda \mid x,z,\theta) = p(\pi \mid z)p(\lambda \mid x,\theta)$. Hence, exploiting conjugacy in the model, we can analytically compute posterior distributions $q(\pi)$ and $q(\lambda)$. We compute point estimates for z and θ.

3.1 Stochastic E-Step

By conjugacy of the beta to the Bernoulli process we can compute $q(\pi) \triangleq p(\pi \mid z)$ analytically. In addition, to make inference scalable, we employ stochastic inference for π_k and use natural gradient of the posterior parameters (a_k, b_k) using random batch of data $S \subset \{x_n\}_{n=1}^N$to update the posterior parameters , where η is the stochastic gradient step [3]:

$$\begin{aligned}
q(\pi) = \textstyle\prod_k \mathrm{Beta}(\pi_k \mid a_k, b_k);\quad & a_k' = \alpha\tfrac{\gamma}{K} + \tfrac{N}{|S|}\textstyle\sum_{n\in S} z_{nk}\ ; b_k' = \alpha\left(1-\tfrac{\gamma}{K}\right) + \tfrac{N}{|S|}\textstyle\sum_{n\in S}(1-z_{nk})\\
& a_k \leftarrow (1-\eta)a_k + \eta\, a_k' \qquad ; b_k \leftarrow (1-\eta)b_k + \eta\, b_k' \qquad (1)
\end{aligned}$$

In addition, since $q(\lambda) \triangleq p(\lambda \mid x,\theta,z)$ factorizes as $\prod_n q(\lambda_n) = \prod_n p(\lambda_n \mid x_n,\theta,z_n)$, and the posterior distribution over λ_n is also a Gaussian, we can analytically compute the posterior $q(\lambda_n)$:

$$q(\lambda_n) = \mathcal{N}(\lambda_n \mid \mu_{\lambda_n|x_n,z_n,\theta}, \sigma^2_{\lambda_n|x_n,z_n,\theta}) \qquad \begin{aligned}
\sigma^2_{\lambda_n|x_n,z_n,\theta} &= \left(c^{-1} + f_\theta(z_n)^\top f_\theta(z_n)/\sigma^2\right)^{-1} \qquad (2)\\
\mu_{\lambda_n|x_n,z_n,\theta} &= (\sigma^2_{\lambda_n|x_n,z_n,\theta})(f_\theta(z_n)^\top x_n)/\sigma^2 \qquad (3)
\end{aligned}$$

Algorithm 1: Stochastic MAP-EM for Sparse Coding

```
function SPARSE_CODE($x \in \mathbb{R}^{N \times D}, z \in \{0,1\}^{N \times K}$)
    Initialize $\rho,\ a_k,\ b_k,\ \kappa \in (0.5, 1],\ \tau_0 \geq 0$
    while not converged do
        $S_t \subset \{1, \ldots, N\}$                                          ▷ Take mini-batch
        $\eta \leftarrow (\tau_0 + t)^{-\kappa}$                                   ▷ Learning rate schedule
        for $n \in S_t$ do                                         ▷ Update $q(\lambda)$ using Eq. (2) and (3)
            $q(\lambda_n) := \mathcal{N}(\mu_{\lambda_n|x_n,z_n,\theta}, \sigma_{\lambda_n|x_n,z_n,\theta})$
        for $n \in S_t$ do                                             ▷ Update $q(\pi)$ using Eq. (1)
            for $k \in \{1 \ldots K\}$ do
                $q(\pi_k) := \mathrm{Beta}(\pi_k \mid a_k, b_k)$
        Initialize: $\forall\, (n \in S_t)\ \Omega_n = \emptyset$
        for $n \in S_t$ do                        ▷ Update z to maximize $\mathcal{L}_\pi(z, \theta)$ using Eq. (4) and (6)
            Initialize: $\forall\, k\ z_{nk} = 0,\ \zeta^- = 0$
            while $\|\Omega_n\|_0 < L$ do
                $j^* \leftarrow \mathrm{argmax}_{j \setminus \Omega_n}\ \ln p(x_n \mid z_{\Omega_n} = 1, z_{nj} = 1) + \mathbb{E}_{q(\pi)}\left[\ln p(z_{\Omega_n} = 1, z_{nj} = 1 \mid \pi_k)\right]$
                $\zeta^+ = \ln p(x_n \mid z_{\Omega_n} = 1, z_{nj*} = 1) + \mathbb{E}_{q(\pi)}\left[\ln p(z_{\Omega_n} = 1, z_{nj^*} = 1 \mid \pi_k)\right]$
                if $\zeta^+ > \zeta^-$ then
                    $\Omega_n \leftarrow \Omega_n \cup \{j\}$
                    $\zeta^- \leftarrow \zeta^+$
                else
                    break
        $\theta \leftarrow$ ADAM($\mathbb{E}_{q(\lambda)}[\ln p(x \mid \theta, z, \lambda)]$, stepsize $= \rho$)   ▷ Update $\theta$ to max. $\mathcal{L}_{\pi,\lambda}(z, \theta)$ using Eq. (5)
```

3.2 M-Step

Given the joint density, we can compute the MAP objective $\mathcal{L}_{\lambda,\pi}(z, \theta)$, alternatively, we marginalize out λ from $p(x, \lambda \mid \theta, z)$ to compute the objective $\mathcal{L}_\pi(z, \theta)$:

$$\mathcal{L}_{\lambda,\pi}(z, \theta) = \mathbb{E}_{q(\lambda)}\left[\ln p(x \mid \theta, z, \lambda)\right] + \mathbb{E}_{q(\pi)}\left[\ln p(z \mid \pi)\right]$$
$$\mathcal{L}_\pi(z, \theta) = \ln p(x \mid \theta, z) + \mathbb{E}_{q(\pi)}\left[\ln p(z \mid \pi)\right]$$

The marginal over x_n and the conditional expectations can be computed analytically:

$$\ln p(x_n \mid \theta, z_n) = -.5\big[\ln\Big(1 + c\,\sigma^{-2} f_\theta(z_n)^\top f_\theta(z_n)\Big) + \ln\Big|\sigma^2 I\Big| + x_n^\top\Big(\sigma^{-2} I - \sigma^{-2} f_\theta(z_n) f_\theta(z_n)^\top / c^{-1}\sigma^2 + f_\theta(z_n)^\top f_\theta(z_n)\Big)\, x_n\big] \quad (4)$$

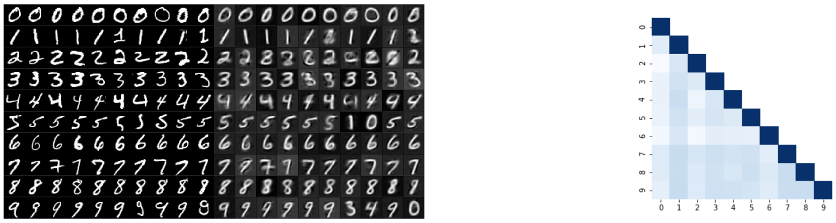
$$\mathbb{E}_{q(\lambda)}\left[\ln p(x \mid \theta, z, \lambda)\right] = \|x_n - \mu_{\lambda_n|x_n,z_n,\theta} \cdot f_\theta(z_n)\|^2 + \sigma^2_{\lambda_n|x_n,z_n,\theta} \cdot f_\theta(z_n)^\top f_\theta(z_n) \quad (5)$$

$$\mathbb{E}_{q(\pi)}\left[\ln p(z_n \mid \pi)\right] = \textstyle\sum_k z_{nk}[\psi(a_k - \psi(a_k + b_k)] + (1 - z_{nk})[\psi(b_k) - \psi(a_k + b_k)] \quad (6)$$

where $\psi(\cdot)$ is the digamma function.

To optimize $\mathcal{L}_{\lambda,\pi}(z, \theta)$, we employ a greedy algorithm, which is similar to the matching pursuit used by K-SVD [1]. We use z_{Ω_n} to denote a k-vector, corresponding to the latent vector for the nth data point, where, $\forall j \in \Omega_n, z_{nj} = 1$ and $\forall j \notin \Omega_n, z_{nj} = 0$. To compute the sparse code given a data point x_n, we start with an empty active set Ω_n, then $\forall j \in \{1, \ldots, K\}$, we individually set each $z_{nj} = 1$ to find $j^* \in \{1, \ldots, K\} \setminus \Omega_n$ that maximizes $\mathcal{L}_{\lambda,\pi}(z_{\Omega_n \cap \{j^*\}}, \theta)$. We compute the scores $\zeta^+ \triangleq \mathcal{L}_{\lambda,\pi}(z_{\Omega_n \cap \{j^*\}}, \theta)$ and $\zeta^- \triangleq \mathcal{L}_{\lambda,\pi}(z_{\Omega_n}, \theta)$. We add j^* to Ω_n only if $\zeta^+ > \zeta^-$, this step is necessary because unlike matching pursuit, the neural net $\varphi(\cdot)$ introduces a non-linearity from z_{Ω_n} to ξ_n, hence, adding j^* to Ω_n can decrease $\mathcal{L}_{\lambda,\pi}(z_{\Omega_n}, \theta)$. For each x_n, we repeat the preceding greedy steps to sequentially add factors to Ω_n till $\mathcal{L}_{\lambda,\pi}(z_{\Omega_n}, \theta)$ ceases to monotonically increase.

For optimization of $\mathcal{L}_{\lambda,\pi}(z, \theta)$ w.r.t. z, the scoring procedure to add factors to Ω_n is similar to the correlation score used by matching pursuit step used K-SVD [1]. The expected log prior on z imposes an approximate beta process penalty. Low probability factors learned through $q(\pi)$ will lead negative scores ζ^+, and hence eliminate latent factors to encourage sparsity of z_n. During optimization once $q(\pi_k)$ falls below a certain threshold we no longer need to consider the z_{nk} when optimizing $\mathcal{L}_{\lambda,\pi}(z, \theta)$. This allows for speed up of the sparse coding routine over iterations.



(a) Example reconstructions of MNIST digits from inferred sparse codes. Left 10 columns true data, right 10 columns approximations.

(b) Expected factor sharing between pairs of digits.

Figure 1

We can maximize $\mathcal{L}_\pi(z, \theta)$ w.r.t. θ, which includes both the neural net parameters θ_ξ and the dictionary θ_Φ by stochastic optimization using ADAM [4]. First order gradient methods with moment estimates such as ADAM, can implicitly take into account the rate of change of natural parameters (a_k, b_k) for $q(\pi)$ when optimizing the neural net parameters θ_ξ and the dictionary θ_Φ. The full sparse coding algorithm is outlined in Alg. (1).

4 Preliminary Results

We trained a 3 layered neural network $\varphi(\cdot)$ with 100 hidden units with a softmax output layer. We chose a factor loading matrix D of size 78×256 and set the number of factors $K = 75$. For our prior parameters we set $\alpha = 1$,$\gamma = 1$, $\sigma = 10$ and $c = 1e15$. We set a constant learning rate for the neural network to 0.001 and the learning rate schedule parameters $\tau = 100$ and $\kappa = 0.6$ for $q(\pi)$. We trained our model on the MNIST data with batch size of 200 for $10,000$ iterations. The deep nonparametric dictionary learning model was able to reconstruct digits well from the inferred sparse codes z_n Fig. (1a). Over the course of training, the beta-process sparsity prior encouraged only a small subset of the K factors to be ultimately used Fig. (3) while optimizing the factor loading matrix as well as the neural net parameters. In addition, the model learned shared factors across the digits. We show the factor sharing across the digits by calculating the expected number of factors shared between all pairs of two digits (normalized by the largest value) Fig. (1b) .

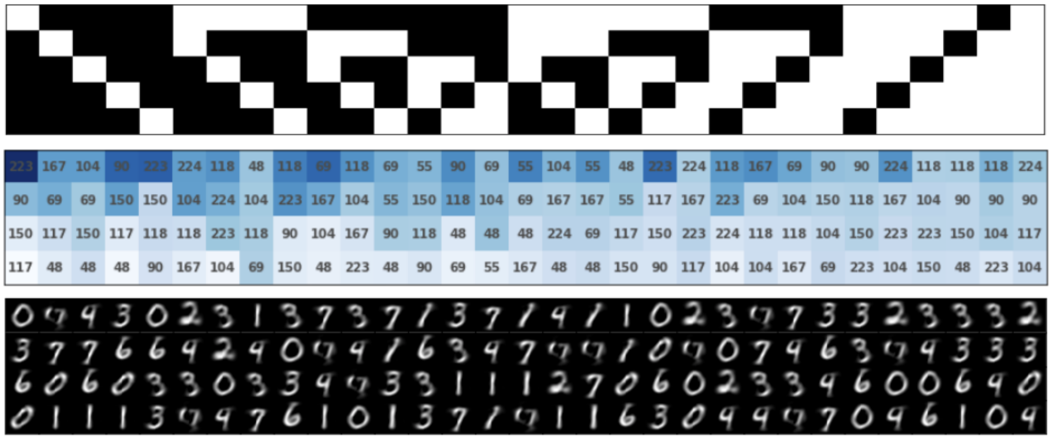


Figure 2: Combinatorial outputs for top five input bits with highest $\mathbb{E}[\pi]$. (Top) Each column corresponds to input sequence z_n and each row entry indicates whether the top kth element of sequence was selected or not (white = 1/black = 0). (Middle) The corresponding output $\varphi(z_n; \theta_\xi)$, where each column corresponds to the top four output elements with highest activation probability for z_nth input. The numbers corresponds to the output element index, and the color indicates the probability of a particular output element being turned on given an input sequence z_n. (Bottom) Inferred factor loadings corresponding to the top four output elements with highest activation probability.

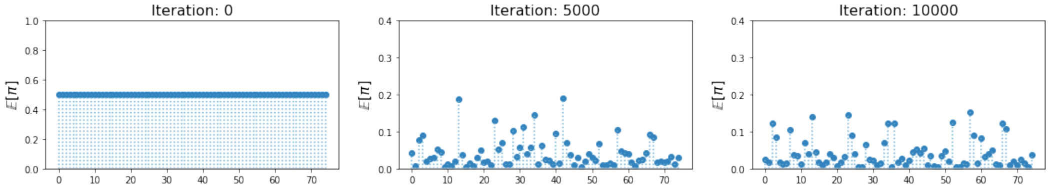


Figure 3: Inferred $\mathbb{E}[\pi]$ indicating increasing sparsity over iterations.

The non-linear factor analysis model is a more expressive model compared to a linear factor analysis model due to the fact that the neural net $\varphi(z_n; \theta_\xi)$ can can index the factor loading matrix Φ in a non-linear fashion. That is, indices selected by a factor sequence z_n such that $z_{nj} = 1$ can be unselected by a sequence z_m with $z_{mj} = 1, z_{ml} = 1$. To illustrate this we trained the same model as above with the same training parameters, however, constrained the factor loading matrix entries to be non-negative. This model can be viewed as non-linear non-negative matrix factorization. We then illustrate the non-linearity of factor selection in Fig. (2), where the difference between factor sequences between column one and six is just on additional factor, however, the top five factor loadings selected by the network, given the two sequences, are entirely different.

5 Conclusion and Future Work

Our non-linear factor analysis model can approximate complex non-factorial distributions over the latent codes. Our MAP-EM algorithm, allows for the exploration of the large combinatorial space over the latent encodings. In addition, the beta-process sparsity prior encourages only a small subset of the K factors to be utilized. In implementations of our our algorithm, one could choose to start ignoring indices of factor sequences whose expected selection falls below certain threshold. This allows our inference procedure to speed up over time as we have to check a smaller number of indices during the M step.

Our specific algorithm for deep sparse coding leverages conjugacy in the model to marginalize out the scaling factor λ during the M step. This makes the algorithm resilient to scaling of the data since we account for scaling of each data point x_n by inferring the scaling factor λ_n. As discussed, deep sparse coding for dictionary learning is a specific instantiation of the generative model outlined in section 2. More generally, for appropriate choice of priors $g(\cdot)$ on Φ, and $h(\cdot)$ of ξ, the generative model encompasses a broad class of models such as PCA, sparse coding, sparse PCA and sparse matrix factorization/non-negative sparse coding. In future work, we plan to extend the above inference algorithm to this broader class of models.

References

[1] M. Aharon, M. Elad, A. Bruckstein, et al. K-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311, 2006.

[2] T. L. Griffiths and Z. Ghahramani. The indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12(Apr):1185–1224, 2011.

[3] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] D. Knowles and Z. Ghahramani. Nonparametric bayesian sparse factor models with application to gene expression modeling. *The Annals of Applied Statistics*, pages 1534–1552, 2011.

[6] J. Paisley and L. Carin. Nonparametric factor analysis with beta process priors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 777–784. ACM, 2009.

[7] J. Paisley and M. I. Jordan. A constructive definition of the beta process. *arXiv preprint arXiv:1604.00685*, 2016.

[8] R. Ranganath, L. Tang, L. Charlin, and D. Blei. Deep exponential families. In *Artificial Intelligence and Statistics*, pages 762–771, 2015.

[9] S. Sertoglu and J. Paisley. Scalable bayesian nonparametric dictionary learning. In *EUSIPCO*, pages 2771–2775, 2015.

[10] M. Zhou, H. Chen, J. Paisley, L. Ren, L. Li, Z. Xing, D. Dunson, G. Sapiro, and L. Carin. Nonparametric bayesian dictionary learning for analysis of noisy and incomplete images. *IEEE Transactions on Image Processing*, 21(1):130–144, 2012.