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 $\Phi$ represent the $M$ factors, and the $M$ entries of each column $\xi_n$ of the latent matrix $\xi$ 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 $\xi$, computing the optimal values of $\xi_n$ 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 $\xi_n$. 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 $\xi_n$ 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 $\Phi$ and the outputs of the multiplexer network $\xi_n$ 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 $\pi_k \sim \text{Beta}\left(\alpha \gamma / K, \alpha \left(1 - \gamma \right) K\right)$ 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 $\pi_k \sim \text{Beta}\left(\alpha \gamma / K, \alpha \left(1 - \gamma \right) K\right)$,
We model the generative process for the data as follows: Given data \( q \), the corresponding latent factors \( z_n \in \{ z_n \} \) are drawn from a Bernoulli process (BP) parameterized by a beta process (BP), where, the Bernoulli process prior over each of the \( k \) factors \( z_{nk} \), is parameterized by \( \pi_k \) drawn from a beta process. Each latent variable \( \xi_n \) is drawn from the distribution \( h(\cdot) \) parameterized by the latent factor \( z_n \) via a neural network \( \varphi(\cdot) \) with parameters \( \theta_k \), where the \( L \) layer neural network \( \varphi(\cdot) \) maps the binary latent code \( z_n \) to \( S \subset \mathbb{R}^M \) via a neural net: \( \varphi(z_n) = \sigma(W_L \sigma(W_{L-1} \cdots \sigma(W_0 z_n))) \). 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 \( \theta_\Phi \). The scaling factor \( \lambda_n \) for each data point \( x_n \) is drawn from a Gaussian distribution. Finally, each data point \( x_n \) is drawn from an isotropic Gaussian distribution where the mean is parameterized by the matrix-vector product of factor loading matrix \( \Phi \) and the encoded latent factor \( \xi_n \), scaled by \( \lambda_n \).

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

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 \( \text{Dir}(\xi_n \varphi(z_n); \ldots, \varphi(z_n K)) \) and choose \( g(\cdot) \) to be the Dirac measure \( \delta_\Phi(\Phi) \), 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, \( f_\theta(z_n) = \Phi E[\xi_n] \) and \( \theta = \{ \xi, \theta_\Phi \} \). For our preliminary experiments we chose a dense neural network that maps \( z_n \) to \( E[\xi_n] \), where the last layer was a softmax layer that output values over the \( \Delta_{M-1} \) simplex. More generally, one could use \( \varphi(\cdot) \) to parameterize the natural parameters of an exponential family distribution such as a gamma or Poisson distribution.

3 MAP-EM Inference

We propose a MAP-EM algorithm to perform inference in this model. We compute point estimates for \( z \) and \( \theta \) and posterior distributions over \( \pi \) and \( \lambda \). Since, \( \pi \perp \lambda \mid z \), 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 \( \theta \).

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 \( \pi_k \) and use natural gradient of the posterior parameters \( (a_k, b_k) \) using random batch of data \( S \subset \{ x_n \} \) to update the posterior parameters, where \( \eta \) is the stochastic gradient step:

\[
q(\pi) = \prod_k \text{Beta}(\pi_k \mid a_k, b_k): \quad a_k' = a_k + \frac{1}{|S|} \sum_{x_n \in S} z_{nk} ; \\
b_k' = b_k + \frac{1}{|S|} \sum_{x_n \in S} (1 - z_{nk}) \\
a_k \leftarrow (1 - \eta) a_k + \eta a_k' \\
b_k \leftarrow (1 - \eta) b_k + \eta b_k'
\]

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 \( \lambda_n \) is also a Gaussian, we can analytically compute the posterior \( q(\lambda_n) \):

\[
q(\lambda_n) = \mathcal{N}(\lambda_n \mid \mu_{\lambda_n} \mid x_n, z_n, \theta, \sigma_{\lambda_n}^2) ; \quad \\
\sigma_{\lambda_n|x_n,z_n,\theta}^2 = \left( c^{-1} + f_\theta(z_n)^\top f_\theta(z_n) / \sigma^2 \right)^{-1} \\
\mu_{\lambda_n|x_n,z_n,\theta} = (\sigma_{\lambda_n|x_n,z_n,\theta}^2)(f_\theta(z_n)^\top x_n) / \sigma^2
\]
Algorithm 1: Stochastic MAP-EM for Sparse Coding

```
function SPARSE_CODE(x ∈ ℝ^{N×D}, z ∈ \{0, 1\}^{N×K})
    Initialize ρ, a_k, b_k, κ ∈ (0.5, 1], τ_0 ≥ 0
    while not converged do
        S_l ∈ \{1, ..., N\}
        η ← (τ_0 + t)^−Ω
        for n ∈ S_l do
            q(λ_n) := N(\mu_{λ_n} | x_n, z_n, θ, σ_{λ_n} | x_n, θ)
        for n ∈ S_l do
            for k ∈ \{1,...,K\} do
                q(π_k) := Beta(π_k | a_k, b_k)
        Initialize: ∀(n ∈ S_l) Ω_n = ∅
        for n ∈ S_l do
            Initialize: ∀k z_{nk} = 0, ζ̇ = 0
            while ||Ω_n||_0 < L do
                j∗ ← argmax_j∈Ω_n ln p(x_n | z_{n,j} = 1, ζ̇ = 0) + ln p(z_{n,j} = 1 | π_k)
                ζ̇ = ln p(x_n | z_{n,j} = 1, ζ̇ = 0) + 2 ln p(z_{n,j} = 1 | π_k)
                if ζ̇ > ζ̇ then
                    Ω_n ← Ω_n ∪ {j}
                else
                    ζ̇ ← ζ̇
                break
            θ ← ADAM(\E_q(λ) [ln p(x | θ, z, λ)], steps=ρ) → Update θ to max. L_{θ,λ}(z, θ) using Eq. (5)
```

3.2 M-Step

Given the joint density, we can compute the MAP objective L_{θ,λ}(z, θ), alternatively, we marginalize out λ from p(x, λ | θ, z) to compute the objective L_θ(z, θ):

\[
L_θ(z, θ) = \E_q(λ) [ln p(x | θ, z, λ)] + \E_q(π) [ln p(z | π)]
\]

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

\[
ln p(x_n | θ, z_n) = -0.5 [ln (1 + c σ^2 f_θ(z_n)^T f_θ(z_n))] + ln |σ^2 I| + \sum_k (σ^2 f_θ(z_n)^T f_θ(z_n))/c^σ^2 + f_θ(z_n)^T f_θ(z_n)
\]

(4)

\[
\E_q(λ) [ln p(x | θ, z, λ)] = ||x_n - \mu_{λ_n} | x_n, z_n, θ, f_θ(z_n)||^2 + σ_{λ_n}^2 | x_n, z_n, θ, f_θ(z_n)^T f_θ(z_n)
\]

(5)

\[
\E_q(π) [ln p(z | π)] = \sum_k z_{nk}[ψ(a_k - ψ(a_k + b_k)) + (1 - z_{nk})(ψ(b_k) - ψ(a_k + b_k))]
\]

(6)

where ψ(·) is the digamma function.

To optimize L_{θ,λ}(z, θ), we employ a greedy algorithm, which is similar to the matching pursuit used by K-SVD [11]. We use z_{Ω_n} to denote a k-vector, corresponding to the latent vector for the nth data point, where, ∀j ∈ Ω_n, z_{nj} = 1 and ∀j /∈ Ω_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 ∀j ∈ {1, ..., K}, we individually set each z_{nj} = 1 to find j∗ ∈ \{1, ..., K\} \ Ω_n that maximizes L_{θ,λ}(z_{Ω_n∪{j}}, θ). We compute the scores ζ̇ ≜ L_{θ,λ}(z_{Ω_n∪{j}}, θ) and ζ̇ ≜ L_{θ,λ}(z_{Ω_n}, θ). We add j∗ to Ω_n only if ζ̇ > ζ̇, this step is necessary because unlike matching pursuit, the neural net ψ(·) introduces a non-linearity from z_{Ω_n} to ζ_n, hence, adding j∗ to Ω_n can decrease L_{θ,λ}(z_{Ω_n}, θ). For each x_n, we repeat the preceding greedy steps to sequentially add factors to Ω_n till L_{θ,λ}(z_{Ω_n}, θ) ceases to monotonically increase.

For optimization of L_{θ,λ}(z, θ) 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 [11]. The expected log prior on z imposes an approximate beta process penalty. Low probability factors learned through q(π) will lead negative scores ζ̇, and hence eliminate latent factors to encourage sparsity of z_n. During optimization once q(π_k) falls below a certain threshold we no longer need to consider the z_{nk} when optimizing L_{θ,λ}(z, θ). This allows for speed up of the sparse coding routine over iterations.
We can maximize $L_\pi(z, \theta)$ w.r.t. $\theta$, which includes both the neural net parameters $\theta_\xi$ and the dictionary $\theta_\Phi$ 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 $\theta_\xi$ and the dictionary $\theta_\Phi$. 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 $78 \times 256$ 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 $E[\pi]$. (Top) Each column corresponds to input sequence $z_n$ and each row entry indicates whether the top $k$th 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 $z_n$th 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.
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 index the factor loading matrix $\Phi$ 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 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 $\lambda$ 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 $\lambda_n$. 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 $\Phi$, and $h(\cdot)$ of $\xi$, 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.