A Guide to Hierarchical Poisson Factorization

Mario Damiano Russo[1]

Bocconi University, Milan, Italy

Abstract. Hierarchical Poisson Factorization (HPF) is a Bayesian Model developed in 2015 by Gopalan, Hofman and Blei to improve over traditional Gaussian Matrix factorization recommender systems. This guide has the goal to provide a graduate-level explanation of the model and its functioning, and include a Python3 implementation of the model. Some preliminary knowledge in Variational Inference and Gaussian Matrix Factorization is required to understand this paper.

Keywords: Bayesian Modeling · Recommender Systems · Machine Learning · Variational Inference

1 Introduction

Hierarchical Poisson Factorization (HPF) [3] is a probabilistic recommender system that is fairly similar in functioning to traditional Gaussian Probabilistic Matrix Factorization (PMF) [4]. The main differences can be summed up in an added layer of prior distributions (the *activity/popularity* layer) and the use of Poisson and Gamma distribution to model ratings and attributes, respectively.

2 The Model

Hierarchical Poisson Factorization represents:

1. each item i with a vector of K latent attributes: β_i;
2. each user u with a vector of K latent preferences: θ_u.

The ratings are instead modeled as a *Poisson* distribution with parameter equal to the inner product of user preferences and item attributes:

$$y_{u,i} \sim Poisson(\theta_u^T \beta_i)$$

Additionally, both the vector of attributes β_i and the vector of preferences θ_u are distributed as *Gamma* distributions, with a rate parameter that is item/user-specific and is also distributed as a *Gamma*. The Gamma prior on the rate parameter that governs the Gamma distribution of attributes/preference is what allows HPF to capture the diversity of users and items, something that PMF is not able to do since it lacks this additional layer of priors.

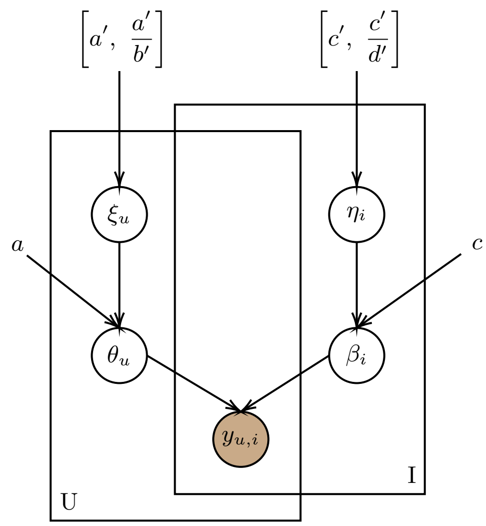

Fig. 1: Graphical Model for HPF

The full generative process of the data is reported below:

1. For each user u:
   (a) Sample an activity value $\xi_u \sim Gamma(a', \frac{a'}{b'})$
   (b) For each component $k = 1, ..., K$, sample a preference value:

$$\theta_{u,k} \sim Gamma(a, \xi_u)$$

2. For each item i:
   (a) Sample a popularity value $\eta_i \sim Gamma(c', \frac{c'}{d'})$
   (b) For each component $k = 1, ..., K$, sample a quality value:

$$\beta_{i,k} \sim Gamma(c, \eta_i)$$

3. For each (u, i) combination, sample a rating:

$$y_{u,i} \sim Poisson(Poisson(\theta_u^T \beta_i)$$

2.1 Advantages of HPF

The generative model of HPF presents a series of advantages over traditional Matrix Factorization techniques.

Sparse latent vector representations Imposing a low *shape* hyperparameter on the Gamma priors will put a higher mass on lower values (Figure 2). As a consequence, the user- and item-specific shape parameters of the Gamma distributions for the latent vector will also tend to put higher mass lower values for each element of the latent vector. This means that ultimately our model will tend to produce very sparse user- and item-specific latent vectors, leading to improved interpretability.

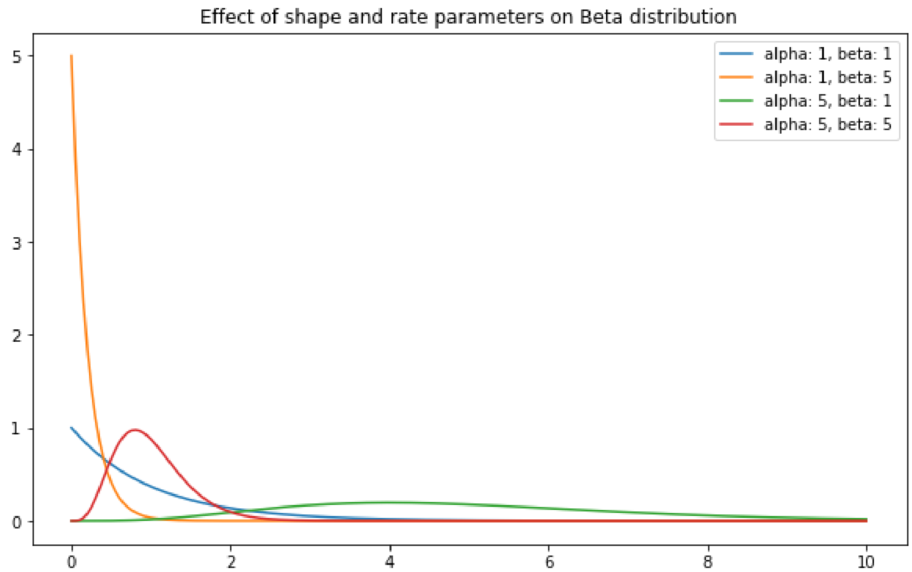

Fig. 2: Gamma distribution for different combinations of rate, shape

Long-tail modeling of users and items In empirical settings, we expect users and items to be distributed with long-tails. Some items, for example a cult movie, are likely to be consumed by a lot of people. Similarly, some users passionate about cinematography might have a very high number of consumed items. The Gamma priors of the model allow for this right-skewed distribution to be correctly represented, something that Gaussian Matrix Factorization is not able to do.

Improved performance on implicit feedback HPF is better at dealing with implicit feedback, i.e. when the non-consumption of an item can have a double meaning - user u did not consume item i either because he is not interested in it (negative feedback) or because he is not aware of its existence (uncertain feedback).

To solve this issue in Gaussian Factorization settings, it is a common choice to increase the variance prior hyperparameter for the ratings distribution to reflect the higher uncertainty about the true rating of the unobserved items [5].

The HPF model instead tends to give higher weight to the observed consumption data by construction, without the need of adjusting the prior hyperparameters.

Fast inference on sparse matrices Inference in HPF depends only on the observed ratings. In fact, we can write the likelihood of ratings as:

$$p(y) = \prod_u \prod_i \frac{(\theta_u^T \beta_i)^{y_{u,i}} \cdot e^{-\theta_u^T \beta_i}}{y_{u,i}!}$$

By noting that for all the unobserved ratings $y_{u,i} = 0$, we have $y_{u,i}! = 0! = 1$, the above equation becomes:

$$p(y) = \prod_{u,i:\ y_{u,i}>0} \frac{(\theta_u^T \beta_i)^{y_{u,i}}}{y_{u,i}!} \cdot \prod_u \prod_i e^{-\theta_u^T \beta_i}$$

This speeds up the training process.

3 Variational Inference

In this section we develop a Variational Inference algorithm for HPF, since the true posterior of the model is intractable.

To facilitate derivation, for each u, i pair we will make use of K additional latent variables: $z_{u,i;k} \sim Poisson(\theta_{u;k}\beta_{i;k})$. Note that $z_{u,i}$ is therefore a vector whose sum of the elements gives $y_{u,i}$. Since a sum of Poisson variables is distributed as a Poisson with parameter equal to the sum of the Poisson parameters, we can interchangeably use $y_{u,i}$ or $z_{u,i,k}$ during our algorithm derivation.

3.1 Deriving the full conditionals

The first step to build our variational optimization algorithm is to compute the full posterior for each one of the latent variables: $[\beta, \theta, \eta, \xi, z]$.

Preferences $\boldsymbol{\theta_{u,k}}$ and qualities $\boldsymbol{\beta_{i,k}}$ We start by deriving the full conditional for $\theta_{u,k}$. Since we can equivalently condition on both y or z, we will do so on the latter. Using Bayes' theorem:

$$p(\theta_{u,k} \mid \beta, \xi, z) = \frac{p(z \mid \theta_{u,k}, \beta, \xi) \cdot p(\theta_{u,k} \mid \beta, \xi)}{p(z \mid \beta, \xi)}$$

Two things to note: first, z and $\theta_{u,k}$ are constant with respect to $[\xi, \beta]$ and $[\beta]$ respectively, so we can omit these conditionings. Second, the denominator can be factored out since it is an integral over the entire domain of $\theta_{u,k}$. Thus, we get that:

$$
\begin{aligned}
p(\theta_{u,k} \mid \beta, \xi, z) &\propto p(z \mid \theta_{u,k}, \beta) \cdot p(\theta_{u,k} \mid \xi) \\
&= \prod_i p(z_{u,i;k} \mid \theta_{u,k}, \beta) \cdot p(\theta_{u,k} \mid \xi) \\
&= \prod_i \left\{ \frac{e^{-(\theta_{u,k}\beta_{i,k})} \cdot (\theta_{u,k}\beta_{i,k})^{z_{u,i;k}}}{z_{u,i;k}!} \right\} \cdot \left\{ \frac{\xi_u^a}{\Gamma(a)} \cdot e^{-\xi_u \theta_{u,k}} \cdot \theta_{u,k}^{a-1} \right\} \\
&\propto e^{-\theta_{u,k} \cdot \sum_i \beta_{i,k}} \cdot \theta_{u,k}^{\sum_i z_{u,i;k}} \cdot e^{-\xi_u \theta_{u,k}} \cdot \theta_{u,k}^{a-1} \\
&= e^{-\theta_{u,k} \cdot (\xi_u + \sum_i \beta_{i,k})} \cdot \theta_{u,k}^{a + \sum_i z_{u,i;k} - 1} \qquad (1)
\end{aligned}
$$

Where we have removed under the sign of proportionality all the values that do not depend on $\theta_{u,k}$. By leveraging conjugacy results between Poisson and Gamma distributions, we note that (1) is the kernel of a Gamma distribution with parameters $[a + \sum_i z_{u,i;k}, \xi_u + \sum_i \beta_{i,k}]$. Thus, the full conditional for the k-th element of the preferences vector of user u is:

$$
\theta_{u,k} \mid \beta, \xi, z \sim Gamma\left(a + \sum_i z_{u,i;k};\ \xi_u + \sum_i \beta_{i,k} \right) \qquad (2)
$$

By simple parameter substitution, the same exact steps used to obtain (1) can be used to determine the full conditional for the k-th element of the qualities vector for item i, $\beta_{i,k}$:

$$
\begin{aligned}
p(\beta_{i,k} \mid \theta, \eta, z) &= \frac{p(z \mid \beta_{i,k}, \theta, \eta) \cdot p(\beta_{i,k} \mid \theta, \eta)}{p(z \mid \theta, \eta)} \\
&\propto p(z \mid \beta_{i,k}, \theta) \cdot p(\beta_{i,k} \mid \eta) \\
&= \prod_u p(z_{u,i;k} \mid \beta_{i,k}, \theta) \cdot p(\beta_{i,k} \mid \eta) \\
&= \prod_u \left\{ \frac{e^{-(\theta_{u,k}\beta_{i,k})} \cdot (\theta_{u,k}\beta_{i,k})^{z_{u,i;k}}}{z_{u,i;k}!} \right\} \cdot \left\{ \frac{\eta_i^c}{\Gamma(c)} \cdot e^{-\eta_i \beta_{i,k}} \cdot \beta_{i,k}^{c-1} \right\} \\
&\propto e^{-\beta_{i,k} \cdot \sum_u \theta_{u,k}} \cdot \beta_{i,k}^{\sum_u z_{u,i;k}} \cdot e^{-\eta_i \beta_{i,k}} \cdot \beta_{i,k}^{c-1} \\
&= e^{-\beta_{i,k} \cdot (\eta_i + \sum_u \theta_{u,k})} \cdot \beta_{i,k}^{c + \sum_u z_{u,i;k} - 1}
\end{aligned}
$$

Thus our full conditional distribution for $\beta_{i,k}$ is:

$$
\beta_{i,k} \mid \theta, \eta, z \sim Gamma\left(c + \sum_u z_{u,i;k};\ \eta_i + \sum_i \theta_{i,k} \right) \qquad (3)
$$

Distributions (2) and (3) are the two full conditionals for the elements of the latent vectors.

Activity ξ_u and popularity η_i We now turn our head to deriving the full conditional distribution for the user activity value, ξ_u:

$$\begin{aligned}
p(\xi_u \mid \beta, \eta, \theta, z) &= \frac{p(\theta \mid \xi_u, \beta, \eta, z) \cdot p(\xi_u \mid \beta, \eta, z)}{p(\theta \mid \beta, \eta, z)} \\
&\propto p(\theta_u \mid \xi_u) \cdot p(\xi_u) \\
&= \prod_k \left\{ \frac{\xi_u^a}{\Gamma(a)} \cdot e^{-\xi_u \theta_{u,k}} \cdot \theta_{u,k}^{a-1} \right\} \cdot \left\{ \frac{[a'/b']^{a'}}{\Gamma(a')} \cdot e^{-[a'/b']\xi_u} \cdot \xi_u^{a'-1} \right\} \\
&\propto \xi_u^{Ka} \cdot e^{-\xi_u \sum_k \theta_{u,k}} \cdot e^{-[a'/b']\xi_u} \cdot \xi_u^{a'-1} \\
&= \xi_u^{Ka+a'-1} \cdot e^{-(\sum_k \theta_{u,k} + [a'/b'])\xi_u}
\end{aligned}$$

Again, thanks to conjugacy properties between the distribution of a Gamma variable and a Gamma that has that variable as shape parameter, we can see that the full conditional of ξ_u is proportional to a Gamma kernel. Thus, the full conditional distribution of the activity parameter for user u, ξ_u is:

$$p(\xi_u \mid \theta_u) = Gamma\left(Ka + a';\ \sum_k \theta_{u,k} + [a'/b'] \right) \tag{4}$$

The same exact steps used to obtain (4) can be used to derive the full conditional of item popularity η_i:

$$\begin{aligned}
p(\eta_i \mid \beta, \xi, \theta, z) &= \frac{p(\beta \mid \eta_i, \xi, \theta, z) \cdot p(\eta_i \mid \xi, \theta, z)}{p(\beta \mid \xi, \theta, z)} \\
&\propto p(\beta \mid \eta_i) \cdot p(\eta_i) \\
&= \prod_k \left\{ \frac{\eta_i^c}{\Gamma(c)} \cdot e^{-\eta_i \beta_{i,k}} \cdot \beta_{i,k}^{c-1} \right\} \cdot \left\{ \frac{[c'/d']^{c'}}{\Gamma(c')} \cdot e^{-[c'/d']\eta_i} \cdot \eta_i^{c'-1} \right\} \\
&\propto \eta_i^{Kc} \cdot e^{-\eta_i \sum_k \beta_{i,k}} \cdot e^{-[c'/d']\eta_i} \cdot \eta_i^{c'-1} \\
&= \eta_i^{Kc+c'-1} \cdot e^{-(\sum_k \beta_{i,k} + [c'/d'])\eta_i}
\end{aligned}$$

Thus, the full conditional for the item popularity value η_i is:

$$\eta_i \mid \beta_u \sim Gamma\left(Kc + c';\ \sum_k \beta_{i,k} + [c'/d'] \right) \tag{5}$$

Distributions (4) and (5) are the two full conditionals for user activity and item popularity, respectively.

Rating vector elements $z_{u,i;k}$ The last full conditional we need to derive is for $z_{u,i;k}$. Thanks to the results illustrated by Bol'shev in [2], we can prove that the joint distribution of n Poisson random variables $x_i \sim Poisson(\lambda_i)$, conditional on their sum $K = \sum_{i=1}^{n} x_i$, is distributed according to a multinomial with K trials and probability vector:

$$\left[\frac{\lambda_i}{\sum_{i=1}^{n} x_i}\right]$$

Below we will rewrite Bol'shev's proof with notation adapted to our use case. First, note that the distribution of $z_{u,i}$ is:

$$\begin{aligned} p(z_{u,i} \mid \theta_u \beta_i) &= \prod_{k=1}^{K} Poisson(\theta_{u,k}\beta_{i,k}) \\ &= \prod_{k=1}^{K} \frac{e^{-\theta_{u,k}\beta_{i,k}} \cdot (\theta_{u,k}\beta_{i,k})^{z_{u,i;k}}}{z_{u,i;k}!} \\ &= e^{-\sum_{k=1}^{K}\theta_{u,k}\beta_{i,k}} \cdot \prod_{k=1}^{K} \frac{(\theta_{u,k}\beta_{i,k})^{z_{u,i;k}}}{z_{u,i;k}!} \end{aligned} \tag{6}$$

Second, remember that the distribution of the sum of Poisson-distributed random variables is distributed according to a Poisson with rate equal to the sum of all the Poisson rates.

$$p\left(y_{u,i} \mid \sum_{k=1}^{K} \theta_{u,k}\beta_{i,k}\right) = e^{-\sum_{k=1}^{K}\theta_{u,k}\beta_{i,k}} \cdot \frac{(\sum_{k=1}^{K}\theta_{u,k}\beta_{i,k})^{y_{u,i}}}{y_{u,i}!} \tag{7}$$

Thus the conditional distribution of $z_{u,i}$ given $y_{u,i}$ is equal to:

$$\begin{aligned} p(z_{u,i} \mid y_{u,i}) &= p(z_{u,i})/p(y_{u,i}) \\ &= \left\{\prod_{k=1}^{K} \frac{(\theta_{u,k}\beta_{i,k})^{z_{u,i;k}}}{z_{u,i;k}!}\right\} \Bigg/ \left\{\frac{(\sum_{k=1}^{K}\theta_{u,k}\beta_{i,k})^{y_{u,i}}}{y_{u,i}!}\right\} \end{aligned}$$

Since $y_{u,i} = \sum_{k=1}^{K} z_{u,i;k}$, we can rewrite:

$$\begin{aligned} p(z_{u,i} \mid y_{u,i}) &= \prod_{k=1}^{K} \frac{(\theta_{u,k}\beta_{i,k})^{z_{u,i;k}}}{z_{u,i;k}!} \cdot \frac{y_{u,i}!}{(\sum_{k=1}^{K}\theta_{u,k}\beta_{i,k})^{z_{u,i;1}}(\sum_{k=1}^{K}\theta_{u,k}\beta_{i,k})^{z_{u,i;2}}\ldots} \\ &= y_{u,i}! \cdot \prod_{k=1}^{K} \left(\frac{\theta_{u,k}\beta_{i,k}}{\sum_{k=1}^{K}\theta_{u,k}\beta_{i,k}}\right)^{z_{u,i;k}} \cdot \frac{1}{z_{u,i;k}!} \end{aligned} \tag{8}$$

Notice that equation (8) is a multinomial distribution. Thus the full conditional for $z_{u,i}$ will be:

$$p(z_{u,i} \mid y_{u,i}, \theta_u, \beta_i) = Multinomial\left(y_{u,i}, \frac{\theta_u \beta_i}{\sum_{k=1}^{K}\theta_{u,k}\beta_{i,k}}\right) \tag{9}$$

3.2 Mean-field family assumption

Once we have derived the full conditional distributions, we set our full variational distribution to belong to the mean-field family:

$$\begin{aligned} q(\beta, \theta, \eta, \xi, z) = & \prod_{i,k} q(\beta_{i,k} \mid \lambda_{i,k}) \prod_{u,k} q(\theta_{u,k} \mid \gamma_{u,k}) \\ & \prod_{i} q(\eta_i \mid \tau_i) \prod_{u} q(\xi_u \mid \kappa_u) \prod_{u,i} q(z_{u,i} \mid \phi_{u,i}) \end{aligned} \tag{10}$$

Where each variational distribution is distributed according to the same distribution of the respective full conditional. For example, $q(\theta_{u,k} \mid \gamma_{u,k})$ will be a Gamma distribution just like in (2), with $\gamma_{u,k}$ being a 2-element vector with rate and shape variational parameters.

3.3 CAVI Algorithm

With the full conditionals and the mean-field variational distribution in place, we can derive the **Coordinate Ascent Variational Inference (CAVI)** Algorithm. In CAVI, the goal is to find the set of variational parameters $[\lambda, \gamma, \tau, \kappa, \phi]$ that make (10) as close as possible to the true posterior $p(\beta, \theta, \eta, \xi, z \mid y)$.

The update rules for the variational parameters derive from an optimization problem in which we try to minimize the Kullback-Leibler Divergence between the variational distribution and the posterior. As shown in [1], it can be proven that when the full posterior $q_j(z_j)$ belongs to the exponential family, the update rule for its variational parameter becomes:

$$v_j = E_{q_{-j}}[\eta_j(z_{-j}, x)^T] \tag{11}$$

Where v_j is the natural parameter of the exponential form of the variational distribution $q_j(z_j)$, and $\eta_j(z_{-j}, x)^T$ is the natural parameter of the relative full conditional $p(z_j \mid z_{-j}, x)$. Equation (11) will be used to derive all 5 update rules for the variational parameters.

While going through the derivation, we must keep in mind two important facts:

1. The natural parameter for a $Gamma(\alpha, \beta)$ distribution is:

$$\eta = [\alpha - 1; \;\; -\beta]$$

2. The natural parameter for a $Multinomial(n, p)$ with probability vector $p = [p_1, ..., p_k]$ is:

$$\eta = [log\ p_1, ..., log\ p_k]$$

Updating $\boldsymbol{\gamma_{u,k}}$ in $\boldsymbol{q(\theta_{u,k} \mid \gamma_{u,k})}$ From equation (2) we know that the full conditional of $\theta_{u,k}$ is a *Gamma* $(a + \sum_i z_{u,i;k};\ \xi_u + \sum_i \beta_{i,k})$. Let $\gamma_{u,k} = [\gamma^s_{u,k}, \gamma^r_{u,k}]$ be a vector variational parameter with shape and rate values of the variational Gamma distribution, then the update rule follows:

$$\gamma^s_{u,k} - 1 = E_q\left[a + \sum_i z_{u,i;k} - 1\right]$$

Since $z_{u,i;k}$ is the k-th element of a multinomial random variable, we can write its expected value as the product between the number of trials $(y_{u,i})$ and the relative probability $(\phi_{u,i;k})$. Thus the CAVI update rule for $\gamma^s_{u,k}$ is:

$$\gamma^s_{u,k} = a + \sum_i y_{u,i}\phi_{u,i;k} \tag{12}$$

The CAVI update rule for $\gamma^r_{u,k}$ is

$$\gamma^r_{u,k} = E_q\left[\xi_u + \sum_i \beta_{i,k}\right]$$

Now remember that in the variational distribution both ξ_u and $\beta_{i,k}$ are Gamma random variables. Thus, their expected value is given by the ratio of their shape and rate variational parameters.

$$\gamma^r_{u,k} = \frac{\kappa^s_u}{\kappa^r_u} + \sum_i \frac{\lambda^s_{i,k}}{\lambda^r_{i,k}} \tag{13}$$

Updating $\boldsymbol{\lambda_{i,k}}$ in $\boldsymbol{q(\beta_{i,k} \mid \lambda_{i,k})}$ From equation (3) we know that the full conditional of $\lambda_{i,k}$ is a *Gamma* $(c + \sum_i z_{u,i;k};\ \eta_i + \sum_u \theta_{u,k})$. The derivation steps are the same as in $\gamma_{u,k}$ and yield the following update rules:

$$\lambda^s_{i,k} = E_q\left[c + \sum_i z_{u,i;k}\right] = c + \sum_i y_{u,i}\phi_{u,i;k} \tag{14}$$

$$\lambda^r_{i,k} = E_q\left[\eta_i + \sum_u \theta_{u,k}\right] = \frac{\tau^s_i}{\tau^r_i} + \sum_u \frac{\gamma^r_{u,k}}{\gamma^s_{u,k}} \tag{15}$$

Updating $\boldsymbol{\kappa_u}$ in $\boldsymbol{q(\xi_u \mid \kappa_u)}$ From equation (4) we know that the full conditional of ξ_u is a *Gamma* $(Ka + a';\ \ \sum_k \theta_{u,k} + [a'/b'])$. Thus, the update rules are:

$$\kappa^s_u = E_q[Ka + a'] = Ka + a' \tag{16}$$

Notice that the update rule (16) does not depend on any other variational parameter, thus once set it does not need to be updated at each iteration of the CAVI algorithm.

The update rule for the rate variational parameter is:

$$\kappa^r_u = E_q\left[a'/b' + \sum_k \theta_{u,k}\right] = a'/b' + \sum_k \frac{\gamma^r_{u,k}}{\gamma^s_{u,k}} \tag{17}$$

Updating τ_i in $q(\eta_i \mid \tau_i)$ From equation (5) we know that the full conditional of η_i is $Gamma\,(Kc + c';\, \sum_k \beta_{i,k} + [c'/d'])$. Thus, the update rules are:

$$\tau_i^s = E_q[Kc + c'] = Kc + c' \tag{18}$$

Also here, notice that the update rule (18) does not depend on any other variational parameter, thus once set it does not need to be updated at each iteration of the CAVI algorithm.

The update rule for the rate variational parameter is:

$$\tau_i^r = E_q\left[c'/d' + \sum_k \beta_{i,k} +\right] = c'/d' + \sum_k \frac{\lambda_{i,k}^s}{\lambda_{i,k}^r} \tag{19}$$

Updating $\phi_{u,i}$ in $q(z_{u,i} \mid \phi_{u,i})$ Compared to the other four variational parameters (which characterize Gamma distributions), $\phi_{u,i}$ is a vector of K probabilities that acts as a parameter for a multinomial distribution. Thus, our update rule will be slightly different:

$$log\ \phi_{u,i} = E_q\left[log\ \frac{\theta_u \beta_i}{\sum_k \theta_{u,k}\beta_{i,k}}\right]$$

Since $\sum_k \theta_{u,k}\beta_{i,k}$ is a normalizing constant, we can drop it under the sign of proportionality. Moreover, since θ_u and β_i are independent in the variational distribution, we can rewrite:

$$\phi_{u,i} \propto exp\left\{E_q\left[log\ \theta_u + log\ \beta_i\right]\right\}$$

Knowing that the expectation of the log of a $Gamma(\alpha, \beta)$ random variable is $\Psi(\alpha) - log\beta$, where $\Psi(\cdot)$ is the digamma function, we can write the update rule for $\phi_{u,i}$ as:

$$\phi_{u,i;k} \propto \Psi(\gamma_{u,k}^s) - log(\gamma_{u,k}^r) + \Psi(\lambda_{i,k}^s) - log(\lambda_{i,k}^r) \tag{20}$$

Equations (12) - (20) are the core results of this section and will be implemented in the CAVI algorithm, shown in Algorithm 1.

Algorithm 1: CAVI Algorithm for HPF

Initialize $\gamma_u, \kappa_u^r, \lambda_i, \tau_i^r$ to the prior's with a small randomic offset. Set:

$$\kappa_u^s = Ka + a' \qquad \tau_i^s = Kc + c'$$

while *The model has not converged* **do**
  for $u, i : y_{u,i} > 0$ **do**
    for $k = 1, ..., K$ **do**
      $\phi_{u,i;k} \propto \Psi(\gamma_{u,k}^s) - log(\gamma_{u,k}^r) + \Psi(\lambda_{i,k}^s) - log(\lambda_{i,k}^r)$
    end
  end
  for $u = 1, ..., U$ **do**
    $\gamma_{u,k}^s = a + \sum_i y_{u,i}\phi_{u,i;k}$
    $\gamma_{u,k}^r = \frac{\kappa_u^s}{\kappa_u^r} + \sum_i \frac{\lambda_{i,k}^s}{\lambda_{i,k}^r}$
    $\kappa_u^r = a'/b' + \sum_k \frac{\gamma_{u,k}^r}{\gamma_{u,k}^s}$
  end
  for $i = 1, ..., I$ **do**
    $\lambda_{i,k}^s = c + \sum_i y_{u,i}\phi_{u,i;k}$
    $\lambda_{i,k}^r = \frac{\tau_i^s}{\tau_i^r} + \sum_u \frac{\gamma_{u,k}^r}{\gamma_{u,k}^s}$
    $\tau_i^r = c'/d' + \sum_k \frac{\lambda_{i,k}^s}{\lambda_{i,k}^r}$
  end
end

A Python3 implementation of the CAVI algorithm for HPF can be found in Appendix 1.

References

1. Blei, D.M., Kucukelbir, A., McAuliffe, J.D.: Variational inference: A review for statisticians. Journal of the American statistical Association **112**(518), 859–877 (2017)
2. Bol'shev, L.N.: On a characterization of the poisson distribution and its statistical applications. Theory of Probability & Its Applications **10**(3), 446–456 (1965)
3. Gopalan, P., Hofman, J.M., Blei, D.M.: Scalable recommendation with hierarchical poisson factorization. In: UAI. pp. 326–335 (2015)
4. Mnih, A., Salakhutdinov, R.R.: Probabilistic matrix factorization. In: Advances in neural information processing systems. pp. 1257–1264 (2008)
5. Wang, C., Blei, D.M.: Collaborative topic modeling for recommending scientific articles. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 448–456 (2011)

A Appendice 1

```python
import numpy as np
from scipy.special import digamma
import sys
from tqdm import trange
from sklearn.metrics import mean_squared_error

def mse(prediction, ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return mean_squared_error(prediction, ground_truth)


class HPF():
    """
    Initialize a Hierarchical Poisson Factorization Recommender System.
    Model by Gopalan et al. (2013)

    Parameters:
    ----------
        - K : int
          dimensionality of the latent preferences and qualities vectors.
        - a_1, b_1 : floats
          prior hyperparameters on the Gamma(a_1, a_1/b_1) prior for
          user activity value.
        - a : float
          shape hyperparameter for the Gamma(a, user_activity) prior
          for the elements of user u's preference vector.
        - c_1, d_1 : floats
          prior hyperparameters on the Gamma(c_1, c_1/d_1) prior for
          item popularity value.
        - c : float
          shape hyperparameter for the Gamma(a, item_popularity) prior
          for the elements of item i's qualities vector.
    """

    def __init__(self, K, a_1, b_1, a, c_1, d_1, c):
        self.K = K
        self.a_1 = a_1
```

```
        self.b_1 = b_1
        self.a = a
        self.c_1 = c_1
        self.d_1 = d_1
        self.c = c


    def fit(self, epochs, train, val=None):
        """
        Fit a Hierarchical Poisson Factorization Model to
        training data.

        Parameters:
        ----------
            - epochs : int
              number of training epochs.
            - train : numpy.array
              (U X I) array where each row is a user, each column is
              an item.
        """
        # initialize error lists
        self.train_error = []
        self.train = train
        self.val = val
        self.U, self.I = self.train.shape

        if self.val.any():
            self.val_error = []

        # intialize variational parameters to the prior
        self.__initialize_variational_params()

        self.resume_training(epochs)


    def resume_training(self, epochs):
        """
        Resume HPF training for additional epochs.

        Parameters:
        ----------
            - epochs : int
              number of additional training epochs.
        """
```

```
        pbar = trange(epochs, file=sys.stdout, desc = "HPF")
        for iteration in pbar:
            # for each each u,i for which the rating is > 0:
            for u, i in zip(self.train.nonzero()[0],
            ↪  self.train.nonzero()[1]):

                # update the variational multinomial parameter
                self.phi[u,i] = [np.exp(digamma(self.gamma_shp[u,
                ↪  k]) - np.log(self.gamma_rte[u, k])
                        + digamma(self.lambda_shp[i, k]) -
                        ↪  np.log(self.lambda_rte[i, k])) for k
                        ↪  in range(self.K)]
                # normalize the multinomial probability vector
                self.phi[u,i] =
                ↪  self.phi[u,i]/np.sum(self.phi[u,i])

            #for each user, update the user weight and activity
            ↪  parameters
            for u in range(self.U):
                self.gamma_shp[u] = [(self.a +
                ↪  np.sum(self.train[u, :] * self.phi[u,:,k]))
                ↪  for k in range(self.K)]
                self.gamma_rte[u] =
                ↪  [(self.kappa_shp/self.kappa_rte[u] +
                ↪  np.sum(self.lambda_shp[:,
                ↪  k]/self.lambda_rte[:,k])) for k in
                ↪  range(self.K)]
                self.kappa_rte[u] = (self.a_1/self.b_1) +
                ↪  np.sum(self.gamma_shp[u, :]/self.gamma_rte[u,
                ↪  :])

            #for each item, update the item weight and popularity
            ↪  parameters
            for i in range(self.I):
                self.lambda_shp[i] = [(self.c +
                ↪  np.sum(self.train[:, i] * self.phi[:,i,k]))
                ↪  for k in range(self.K)]
                self.lambda_rte[i] =
                ↪  [(self.tau_shp/self.tau_rte[i] +
                ↪  np.sum(self.gamma_shp[:,
                ↪  k]/self.gamma_rte[:,k])) for k in
                ↪  range(self.K)]
                self.tau_rte[i] = (self.c_1/self.d_1) +
                ↪  np.sum(self.lambda_shp[i,
                ↪  :]/self.lambda_rte[i, :])
```

```
            # obtain the latent vectors:
            self.theta = self.gamma_shp/self.gamma_rte
            self.beta = self.lambda_shp/self.lambda_rte
            self.prediction = np.dot(self.theta, self.beta.T)

            self.train_error.append(mse(self.prediction, self.train))
            if self.val.any():
                # Note to self: Very misleading measures! Explicit ratings in train here are zero. Useful only for convergence diagnostic purposes!!
                self.val_error.append(mse(self.prediction, self.val))
                pbar.set_description(f"HPF Val MSE: {np.round(self.val_error[-1], 4)} - Progress")
            else:
                pbar.set_description(f"HPF Train MSE: {np.round(self.train_error[-1], 4)} - Progress")



    def __initialize_variational_params(self):
        # phi: (U X I X K) matrix of variational parameters for the multinomial
        self.phi = np.zeros(shape=[self.U, self.I, self.K])

        #variational parameter random initialization
        # k_rte: (U X 1) array - a_1's with small offset
        self.kappa_rte = (np.random.uniform(-0.3, 0.3, size=self.U) + 1) * self.a_1
        # tau_rte: (I X 1) array - c_1's with small offset
        self.tau_rte = (np.random.uniform(-0.3, 0.3, size=self.I) + 1) * self.c_1

        # gamma_shp, gamma_rte: (U X K) numpy arrays
        self.gamma_shp = np.random.gamma(shape=self.a_1, scale=(self.b_1/self.a_1), size=(self.U, self.K))
        self.gamma_rte = (np.random.uniform(-0.3, 0.3, size=(self.U, self.K)) + 1) * self.a
        # lambda_shp, lambda_rte: (I X K) numpy arrays
        self.lambda_shp = np.random.gamma(shape=self.c_1, scale=(self.c_1/self.d_1), size=(self.I, self.K))
        self.lambda_rte = (np.random.uniform(-0.3, 0.3, size=(self.I, self.K)) + 1) * self.c
```

```

        # k_shp, tau_shp intialization rules come from the CAVI algorithm and do not need to be updated at each iteration.
        # these values are constant for each u, i respectively, so they are a scalar.
        self.kappa_shp = self.a_1 + (self.K * self.a)
        self.tau_shp = self.c_1 + (self.K * self.c)
```