Bayesian Poisson Tensor Factorization For Dummies

Barış Kurt

May 24, 2018

Contents

1 Introduction

The aim of this document is to provide the derivations and a Python implementation for the Bayesian Poisson tensor factorization (BPTF). First we provide the derivations for the Bayesian Poisson tensor factorization using the same notation in [1]. The derivations are mainly generalizations of the equations in [2]. In the second part, we are going to give the details of our Python implementation, that uses Numpy, Scipy, and Tensorly [3] modules. This implementation is highly inspired from [4]. The main difference is that we use Tensorly library for tensor operations.

2 The Math

2.1 Notation

Let X be a $N \geq 2$ dimensional tensor of positive count data, i.e. each element $X(u_0)$ is a non-negative integer, where u_0 is called the *index set* of X. The PARAFAC factorization scheme

for X can be written as

$$X(u_0) \approx \hat{X}(u_0) = \sum_{\bar{u}_0} \prod_n Z_n(u_n) \tag{1}$$

where Z_n for $n \in [1, N]$ are the factors and u_n are their corresponding index sets. The index set $\bar{u}_n$ is defined as the set complement $(u - u_n)$, that is all indexes except the indexes of n.

Example

Let $X \in R^{I \times J \times K}$ be a 3 dimensional tensor, with individual elements $X(ijk)$. We can factorize X with 3 factors such that $Z_1 \in R^{I \times R}$, $Z_2 \in R^{J \times R}$ and $Z_3 \in R^{K \times R}$. The index sets would be

- $u = \{i, j, k, r\}$
- $u_0 = \{i, j, k\}$
- $\bar{u}_0 = \{r\}$
- $u_1 = \{i, r\}$
- $u_2 = \{j, r\}$
- $u_3 = \{k, r\}$

and the equation (1) would yield to

$$X(ijk) \approx \hat{X}(ijk) = \sum_{r=1}^{R} Z_1(ir) Z_2(jr) Z_3(kr)$$

2.2 Generative Model

As the name suggests, the Bayesian Poisson Tensor Factorization model is based on Poisson observations. Then, the natural choice for the factors would be to assume that they are generated by individual Gamma distributions. The generative model can be given as

$$Z_n(u_n) \sim \mathcal{G}\left(Z_n(u_n); A_n(u_n), \frac{B_n(u_n)}{A_n(u_n)}\right) \tag{2}$$

$$\lambda(u) = \prod_n Z_n(u_n) \tag{3}$$

$$W(u) \sim \mathcal{PO}(W(u); \lambda(u)) \tag{4}$$

$$X(u_0) = \sum_{\bar{u}_0} W(u) \tag{5}$$

$$\hat{X}(u_0) = \sum_{\bar{u}_0} \lambda(u) \tag{6}$$

where the $\mathcal{G}$ and $\mathcal{PO}$ are the Poisson and Gamma distributions.

2.3 Necessary Propability Distributions

The Poisson Distribution

The Poisson distribution is defined over nonnegative integers, with a single rate parameter λ, with the following pmf:

$$\mathcal{PO}(s;\lambda) = \exp(s\log\lambda - \lambda - \log\Gamma(s+1)) \tag{7}$$

The mean and expectation of a Poisson distributed random variable are both equal to λ.

$$\langle s\rangle = Var[s] = \lambda \tag{8}$$

The Gamma Distribution

Gamma distribution is defined over nonnegative real numbers by two parameters: the shape parameter k, and the scale parameter θ .

$$\mathcal{G}(z;k,\theta) = \exp((k-1)\log z - \frac{z}{\theta} - k\log\theta - \log\Gamma(k)) \tag{9}$$

The mean and variance of a Gamma distributed random variable are given as

$$\langle z\rangle = k\theta \tag{10}$$
$$Var[z] = k\theta^2 \tag{11}$$

Later on in our inference equations, we are going to use the expectation of $\log(z)$ and the entropy of the Gamma distribution. These are:

$$\langle\log(z)\rangle = \Psi(k) + \log(\theta) \tag{12}$$
$$\mathcal{H}_{\mathcal{G}}[k,\theta] = -\langle\mathcal{G}\rangle_{\mathcal{G}} = (1-k)\Psi(k) + \log\theta + k + \log\Gamma(k) \tag{13}$$

where $\Psi(k) = d\log\Gamma(k)/dk$ is the Digamma function.

In our generative model, the parameters for Gamma distributions are $k = A_n(u_n)$ and $\theta = B_n(u_n)/A_n(u_n)$ respectively. This means that the mean of $Z_n(u_n)$ is $k\theta = B_n(u_n)$, which is independent of $A_n(u_n)$. The variance of $Z_n(u_n)$ becomes $k\theta^2 = B_n(u_n)^2/A_n(u_n)$, which means that as $A_n(u_n)$ gets smaller, the factors gets sparser.

The Multinomial Distribution

Let w be the outcome of x trials, such that w_i denote the number of times we sample i from a categorial distribution with probability p_i. Then we can write the Multinomial dsitribution as

$$\mathcal{M}(w;x,p) = \delta\left(x - \sum_i w_i\right)\exp\left(\log\Gamma(x+1) + \sum_i\left(w_i\log p_i - \log\Gamma(w_i+1)\right)\right) \tag{14}$$

The expectation of outcomes are $\langle w_i\rangle = xp_i$. The entropy of a Multinomial dsitribution is a horrible expression:

$$\mathcal{H}_M[w,x,p] = -\left\langle\log\delta\left(x - \sum_i w_i\right)\right\rangle - \log\Gamma(x+1) - \sum_i(\langle w_i\rangle\log p_i - \langle\log\Gamma(w_i+1)\rangle) \tag{15}$$

which does not have a closed form solution due to the expectation including Gamma function. Luckily, bloody terms will be cancelled out while we calculate the veriational bound.

2.4 Full Joint Likelihood

In our setup, the X tensor is our observations, and Z are our latent variables (factors). Now we introduce $\theta = \{A, B\}$ as the set of parameters for a simpler notation. Furthermore, we define a mask matrix M, to formulate the missing observations, such that $M(u_0) = 1$ if $X(u_0)$ is observed and $M(u_0) = 0$ otherwise. The full joint likelihood of the model is given as

$$p(X, W, Z|\theta) = p(X|W)p(W|Z)p(Z|\theta) \tag{16}$$

We can explicitly write the logarithm of the full joint likelihood as

$$\begin{aligned}
\log p(X, W, Z|\theta) &= \log p(X|W) + \log p(W|Z) + \log p(Z|\theta) && (17)\\
&= \sum_{u_0} M(u_0) \log \delta \left(X(u_0) - \sum_{\bar{u}_0} W(u) \right) \\
&\quad + \sum_{\bar{u}_0} \sum_{u_0} M(u_0) \left(W(u) \log \left(\prod_n Z_n(u_n) \right) - \prod_n Z_n(u_n) - \log \Gamma(W(u) + 1) \right) \\
&\quad + \sum_n \sum_{u_n} \Big((A_n(u_n) - 1) \log Z_n(u_n) - Z_n(u_n) \frac{A_n(u_n)}{B_n(u_n)} \\
&\qquad - A_n(u_n) \log \frac{B_n(u_n)}{A_n(u_n)} - \log \Gamma(A_n(u_n)) \Big) && (18)
\end{aligned}$$

where $\delta(x - y)$ is the degenerate distribution, such that $\delta(0) = 1$ and it is 0 for all other values.

2.5 Variational Bayes Derivations

In the Bayesian setting, we would like to calculate the posterior distribution over parameters θ conditioned on the observations X. By using the Bayes rule, we can re-write this posterior as

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)} \propto p(X|\theta)p(\theta) \tag{19}$$

since $p(X)$ is constant for all values of θ. Here, $p(X|\theta)$ is the marginal likelihood of parameters and written as

$$p(X|\theta) = \int_Z dZ \sum_W p(X, W, Z|\theta) \tag{20}$$

When this integral intractable, which is the case in our problem, one needs to approximate it. A common approximation scheme is to bound the log marginal likelihood as

$$\begin{aligned}
\log p(X|\theta) \geq \mathcal{L}_\theta &= \int_Z dZ \sum_W q(W, Z) \log \frac{p(X, W, Z|\theta)}{q(W, Z)} && (21)\\
&= \langle \log p(X, W, Z|\theta) \rangle_{q(W,Z)} + H_{q(W,Z)} && (22)
\end{aligned}$$

The bound has two parts, first it has the expectation of the log joint distribution under the auxillary distributon q and second the entropy of the q. In fact, the bound is exactly equal to the posterior when we choose the auxiliary distribution as the posterior of latent variables given observations [5].

$$q(W, Z) = p(W, Z|X, \theta) \tag{23}$$

However, $p(W, Z|X, \theta)$ can also be difficult to calculate (again, which is our case) and we approximate it with a mean field approximation as

$$q(W, Z) = q(W)q(Z) \tag{24}$$

After some bloody calculations, we come up with the fact that the distributions maximizing the lower bound are

$$q(W) \propto \exp\left(\langle \log p(X, W, Z|\theta)\rangle_{q(Z)}\right) \tag{25}$$

$$q(Z) \propto \exp\left(\langle \log p(X, W, Z|\theta)\rangle_{q(W)}\right) \tag{26}$$

Derivations for q(W)

Let's consider the terms in log likelihood that only includes $W(u)$:

$$q(W(u)) \propto \exp\left(\langle \log p(X, W, Z|\theta)\rangle_{q(Z)}\right) \tag{27}$$

$$\begin{aligned} q(W(u_0, :)) \propto \exp\Bigg(& M(u_0) \log \delta \left(X(u_0) - \sum_{\bar{u}_0} W(u)\right) \\ & + \sum_{\bar{u}_0} M(u_0) \left(W(u) \sum_n \langle \log Z_n(u_n)\rangle - \log \Gamma(W(u) + 1)\right)\Bigg) \end{aligned} \tag{28}$$

$$\propto \mathcal{M}(W(u_0, :), X(u_0), p(u_0, :))^{M(u_0)} \tag{29}$$

So, $W(u_0, :)$ becomes Multinomially distributed. For cases where $X(u_0)$ is observed, we have

$$p(u) = \frac{\exp\left(\sum_n \langle \log Z_n(u_n)\rangle\right)}{\sum_{\bar{u}_0} \exp\left(\sum_n \langle \log Z_n(u_n)\rangle\right)} = \frac{\prod_n \exp(\langle \log Z_n(u_n)\rangle)}{\sum_{\bar{u}_0} \prod_n \exp(\langle \log Z_n(u_n)\rangle)} \tag{30}$$

$$\langle W(u)\rangle = X(u_0)p(u) \tag{31}$$

Derivations for q(Z)

Now, let's consider the terms in log likelihood that only includes $Z_n(u_n)$:

$$\begin{aligned} q(Z_n(u_n)) \propto & \left(\sum_{\bar{u}_n} M(u_0)\langle W(u)\rangle + A_n(u_n) - 1\right) \log Z_n(u_n) \\ & - \left(\sum_{\bar{u}_n} M(u_0) \prod_{n' \neq n} \langle Z_{n'}(u_{n'})\rangle + \frac{A_n(u_n)}{B_n(u_n)}\right) Z_n(u_n) \end{aligned} \tag{32}$$

$$\propto \mathcal{G}(q(Z_n(u_n)); C_n(u_n), D_n(u_n)) \tag{33}$$

where the shape and scale parameters are

$$C_n(u_n) = A_n(u_n) + \sum_{\bar{u}_n} M(u_0)\langle W(u)\rangle \tag{34}$$

$$D_n(u_n) = \left(\frac{A_n(u_n)}{B_n(u_n)} + \sum_{\bar{u}_n} M(u_0) \prod_{n' \neq n} Z_{n'}(u_{n'})\right)^{-1} \tag{35}$$

Then, from equations 12 and 13, we get

$$\langle Z_n(u_n)\rangle = C_n(u_n)D_n(u_n) \tag{36}$$

$$\langle \log Z_n(u_n)\rangle = \Psi(C_n(u_n)) + \log D_n(u_n) \tag{37}$$

Derivations for Lower Bound

Now, let's write the bound explicitly.

$$\mathcal{L}_\theta = \langle \log p(X, W, Z|\theta) \rangle_{q(W,Z)} + H_{q(W,Z)} \tag{38}$$

$$\begin{aligned}
&= \sum_{u_0} M(u_0) \left\langle \log \delta \left(X(u_0) - \sum_{\bar{u}_0} W(u) \right) \right\rangle \\
&+ \sum_{\bar{u}_0} \sum_{u_0} M(u_0) \left(\langle W(u) \rangle \sum_n \langle \log Z_n(u_n) \rangle - \prod_n \langle Z_n(u_n) \rangle - \langle \log \Gamma(W(u)+1) \rangle \right) \\
&+ \sum_n \sum_{u_n} \left((A_n(u_n)-1) \langle \log Z_n(u_n) \rangle - \langle Z_n(u_n) \rangle \frac{A_n(u_n)}{B_n(u_n)} - A_n(u_n) \log \frac{B_n(u_n)}{A_n(u_n)} - \log \Gamma(A_n(u_n)) \right) \\
&- \sum_{u_0} M(u_0) \log \Gamma(X(u_0)+1) - \sum_{\bar{u}_0} \sum_{u_0} M(u_0) \langle W(u) \rangle \log p(u) \\
&+ \sum_{\bar{u}_0} \sum_{u_0} M(u_0) \langle \log \Gamma(W(u)+1) \rangle - \sum_{u_0} M(u_0) \left\langle \log \delta \left(X(u_0) - \sum_{\bar{u}_0} W(u) \right) \right\rangle \\
&+ \sum_n \sum_{u_n} \left((1 - C_n(u_n)) \Psi(C_n(u_n)) + \log D_n(u_n) + C_n(u_n) + \log \Gamma(C_n(u_n)) \right)
\end{aligned} \tag{39}$$

Luckily, some terms cancel out each other and we can re-arrange the summation indexes to get

$$\begin{aligned}
\mathcal{L}_\theta &= - \sum_u M(u_0) \left(\prod_n \langle Z_n(u_n) \rangle \right) \\
&+ \sum_n \sum_{u_n} \langle \log Z_n(u_n) \rangle \left(\sum_{\bar{u}_n} M(u_0) \langle W(u) \rangle + A_n(u_n) - 1 \right) \\
&+ \sum_n \sum_{u_n} \left(- \langle Z_n(u_n) \rangle \frac{A_n(u_n)}{B_n(u_n)} - A_n(u_n) \log \frac{B_n(u_n)}{A_n(u_n)} - \log \Gamma(A_n(u_n)) \right) \\
&- \sum_{u_0} M(u_0) \log \Gamma(X(u_0)+1) - \sum_{\bar{u}_0} \sum_{u_0} M(u_0) \langle W(u) \rangle \log p(u) \\
&+ \sum_n \sum_{u_n} \left((1 - C_n(u_n)) \Psi(C_n(u_n)) + \log D_n(u_n) + C_n(u_n) + \log \Gamma(C_n(u_n)) \right)
\end{aligned} \tag{40}$$

By using the identities in equations 34 and 37, we can further simplify the bound equation as

$$\begin{aligned}
\mathcal{L}_\theta &= - \sum_u M(u_0) \left(\prod_n \langle Z_n(u_n) \rangle \right) \\
&+ \sum_n \sum_{u_n} \left(- \langle Z_n(u_n) \rangle \frac{A_n(u_n)}{B_n(u_n)} - A_n(u_n) \log \frac{B_n(u_n)}{A_n(u_n)} - \log \Gamma(A_n(u_n)) \right) \\
&- \sum_{u_0} M(u_0) \log \Gamma(X(u_0)+1) - \sum_{\bar{u}_0} \sum_{u_0} M(u_0) \langle W(u) \rangle \log p(u) \\
&+ \sum_n \sum_{u_n} \left(C_n(u_n) (\log D_n(u_n) + 1) + \log \Gamma(C_n(u_n)) \right)
\end{aligned} \tag{41}$$

3 The Code

We implemented the BPTF equations in Python using the Numpy, Scipy and Tensorly [3] libraries. The basic operations we need to understand before implementing the above equations

are the tensor unfolding and the Khatri-Rao product. We also define the *uttkrp* function, which is the product of an unfolded tensor with the Khatri-Rao product of matrices. In this section, we will first define the basic operations and map the update equations with the Python code snippets to better explain the implementation.

3.1 Tensor Unfolding

Let X be a tensor of size $I \times J \times K$. The unfolding along a mode of a tensor yields a matrix whose first dimension is equal to the length of the tensor along the given mode. For example, unfold(X, 0) gives a matrix of size $I \times (J \times K)$, and unfold(X, 1) gives a matrix of size $J \times (I \times K)$. As a concrete example, let X be a $3 \times 4 \times 2$ tensor. We unfold X in 3 possible ways as follows:

$$X = \left[\begin{bmatrix} x_{000} & x_{001} \\ x_{010} & x_{011} \\ x_{020} & x_{021} \\ x_{030} & x_{031} \end{bmatrix}, \begin{bmatrix} x_{100} & x_{101} \\ x_{110} & x_{111} \\ x_{120} & x_{121} \\ x_{130} & x_{131} \end{bmatrix}, \begin{bmatrix} x_{200} & x_{201} \\ x_{210} & x_{211} \\ x_{220} & x_{221} \\ x_{230} & x_{231} \end{bmatrix} \right]$$

$$\text{unfold}(X, 0) = \begin{bmatrix} x_{000} & x_{001} & x_{010} & x_{011} & x_{020} & x_{021} & x_{030} & x_{031} \\ x_{100} & x_{101} & x_{110} & x_{111} & x_{120} & x_{121} & x_{130} & x_{131} \\ x_{200} & x_{201} & x_{210} & x_{211} & x_{220} & x_{221} & x_{230} & x_{231} \end{bmatrix}$$

$$\text{unfold}(X, 1) = \begin{bmatrix} x_{000} & x_{001} & x_{100} & x_{101} & x_{200} & x_{201} \\ x_{010} & x_{011} & x_{110} & x_{111} & x_{210} & x_{211} \\ x_{020} & x_{021} & x_{120} & x_{121} & x_{220} & x_{221} \\ x_{030} & x_{031} & x_{130} & x_{131} & x_{230} & x_{231} \end{bmatrix}$$

$$\text{unfold}(X, 2) = \begin{bmatrix} x_{000} & x_{010} & x_{020} & x_{030} & x_{100} & x_{110} & x_{120} & x_{130} & x_{200} & x_{210} & x_{220} & x_{230} \\ x_{001} & x_{011} & x_{021} & x_{031} & x_{101} & x_{111} & x_{121} & x_{131} & x_{201} & x_{211} & x_{221} & x_{231} \end{bmatrix}$$

3.2 Khatri-Rao Product

Khatri-Rao product is the column-wise Kronecker product of two matrices. As an example, let $A \in I \times R$, $B \in J \times R$ and $C \in K \times R$ be the latent factors with $R = 2$.

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \\ a_{20} & a_{21} \end{bmatrix}, \quad B = \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \\ b_{20} & b_{21} \\ b_{30} & b_{31} \end{bmatrix}, \quad C = \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix}$$

The Tensorly function khatri_rao (kr) gives the following products:

$$\text{kr}([\text{A, B}]) = \begin{bmatrix} a_{00}b_{00} & a_{01}b_{01} \\ a_{00}b_{10} & a_{01}b_{11} \\ a_{00}b_{20} & a_{01}b_{21} \\ a_{00}b_{30} & a_{01}b_{31} \\ a_{10}b_{00} & a_{11}b_{01} \\ a_{10}b_{10} & a_{11}b_{11} \\ a_{10}b_{20} & a_{11}b_{21} \\ a_{10}b_{30} & a_{11}b_{31} \\ a_{20}b_{00} & a_{21}b_{01} \\ a_{20}b_{10} & a_{21}b_{11} \\ a_{20}b_{20} & a_{21}b_{21} \\ a_{20}b_{30} & a_{21}b_{31} \end{bmatrix}, \text{kr}([\text{A, C}]) = \begin{bmatrix} a_{00}c_{00} & a_{01}c_{01} \\ a_{00}c_{10} & a_{01}c_{11} \\ a_{10}c_{00} & a_{11}c_{01} \\ a_{10}c_{10} & a_{11}c_{11} \\ a_{20}c_{00} & a_{21}c_{01} \\ a_{20}c_{10} & a_{21}c_{11} \end{bmatrix}, \text{kr}([\text{B, C}]) = \begin{bmatrix} b_{00}c_{00} & b_{01}c_{01} \\ b_{00}c_{10} & b_{01}c_{11} \\ b_{10}c_{00} & b_{11}c_{01} \\ b_{10}c_{10} & b_{11}c_{11} \\ b_{20}c_{00} & b_{21}c_{01} \\ b_{20}c_{10} & b_{21}c_{11} \\ b_{30}c_{00} & b_{31}c_{01} \\ b_{30}c_{10} & b_{31}c_{11} \end{bmatrix}$$

But, where exactly does this function fit in our equations? Let's look at the following objects:

$$\prod_{n} Z_n(u_n) \equiv \text{khatri\_rao}(Z) \tag{42}$$

$$\prod_{n' \neq n} Z_{n'}(u_{n'}) \equiv \text{khatri\_rao}(Z, n) \tag{43}$$

In the second form, the n^{th} factor is excluded. We can also take the Khatri-Rao product of all factors, sum the result in latent dimension and reshape to get the full tensor $\hat{X}$, which is the approximation of X in standard PARAFAC setting. This also equals to the kruskal_to_tensor function in Tensorly:

$$\hat{X}(u_0) = \sum_{\bar{u}_0} \prod_{n} Z_n(u_n) \tag{44}$$

$$\hat{X} = \text{khatri\_rao}(Z).\text{sum(axis=-1).reshape(X.shape)} \tag{45}$$

$$\equiv \text{kruskal\_to\_tensor}(Z) \tag{46}$$

3.3 UTTKRP: Unfolded Tensor Times Khatri Rao Product of Factors

This is a very useful function that is defined in [4], which multiplies the unfolded tensor with a Khatri-Rao product of factors. The tensor corresponding to mode n will be excluded out in the product. The resulting matrix has the shape of factor Z_n:

```
uttrkrp(X, Z, n) = np.dot(unfold(X, n), khatri_rao(Z, n))
```

This function corresponds to the following equation:

$$\sum_{\bar{u}_n} X(u_0) \prod_{n' \neq n} Z_{n'}(u_{n'}) \equiv \text{uttkrp}(X, Z, n) \tag{47}$$

3.4 Classical PARAFAC Update Equations

The classical solution to the PARAFAC model is the alternating least squares [6]. For the above example, if we fix B and C factors, and unfold X along the first dimension (we call it X_A), we would get the following equation

$$X_A = AZ'_{BC} \tag{48}$$

where Z_{BC} is the Khatri-Rao product of B and C. The solution for A is the solution least squares:

$$A = X_A Z_{BC} (Z'_{BC} Z_{BC})^{-1} \tag{49}$$

Similarly, we would get

$$B = X_B Z_{AC} (Z'_{AC} Z_{AC})^{-1} \tag{50}$$

$$C = X_C Z_{AB} (Z'_{AB} Z_{AB})^{-1} \tag{51}$$

We can give a general update function for all parafac factors as follows:

```
def update_classical_parafac(X, Z):
    for n in range(X.ndim):
        Z[n] = np.dot(unfold(X, n), np.linalg.pinv(khatri_rao(Z, n).T))
```

3.5 BPTF Update Equations

In order to implement BPTF, we need to code the equations 30, 31, 34, 35, 36, and 37. First, let's define the objects that we need in our implementation. Let X be the tensor to factorize, and $Z[1:N]$ be an array of factors. We define W, C and D as in update equations. We define $E_n(u_n) = \langle Z_n(u_n) \rangle$ and $L_n(u_n) = \exp(\langle \log Z_n(u_n) \rangle)$. Implementing equations 36 and 37 are straightforward, however, updating C and D needsome tricks.

Update C[n]

For an efficient implementation [2], let's combine the equations 30, 31, and 34:

$$C_n(u_n) = A_n(u_n) + \sum_{\bar{u}_n} M(u_0) X(u_0) \frac{\prod_{n'} L_{n'}(u_{n'})}{\sum_{\bar{u}_0} \prod_{n'} L_{n'}(u_{n'})} \tag{52}$$

The denominator can be defined as

$$\hat{X}_L(u_0) = \sum_{\bar{u}_0} \prod_{n'} L_{n'}(u_{n'}) \tag{53}$$

$$\hat{X}_L = \text{kruskal\_to\_tensor(L)} \tag{54}$$

and we can write

$$C_n(u_n) = A_n(u_n) + \sum_{\bar{u}_n} M(u_0) \frac{X(u_0)}{\hat{X}_L(u_0)} \prod_{n'} L_{n'}(u_{n'}) \tag{55}$$

$$= A_n(u_n) + L_n(u_n) \sum_{\bar{u}_n} M(u_0) \frac{X(u_0)}{\hat{X}_L(u_0)} \prod_{n' \neq n} L_{n'}(u_{n'}) \tag{56}$$

The part with the summation is actually matrix multiplication, where the first matrix is the $M(u_0)X(u_0)/\hat{X}_L(u_0)$ unfolded along mode n, and the second matrix is the Khatri-Rao product of all L matrices except L_n (see Eq 47). The whole equation can be written in one line:

```
C[n] = A[n] + L[n] * uttkrp(M * (X / kruskal_to_tensor(L), L, n)
```

Update D[n]

Calculating $\sum_{\bar{u}_n} M(u_0) \prod_{n' \neq n} \langle Z_{n'}(u_{n'}) \rangle$ is exactly what uttkrp does. Hence the equation 35 can be coded as:

```
D[n] = 1 / ( A[n] / B[n] +  uttkrp(M, E, n) )
```

Everything in one place

We can combine the update equations in one function as:

```
def update_bptf(X, A, B, C, D, E, L):
    for n in range(X.ndim):
        C[n] = A[n] + L[n] * uttkrp(M * (X / kruskal_to_tensor(L)), L, n)
        D[n] = 1 / ( A[n] / B[n] +  uttkrp(M, E, n) )
        E[n] =  C[n] * D[n]
    elbo = calc_bound(X, M, A, B, C, D, E, L)
    for n in range(X.ndim):
        L[n] =  np.exp( sp.psi(C[n]) + np.log(D[n]) )
    return elbo
```

The reason we call the bound calculation function before updating L matrices is for an efficiency trick, which will be explained in the next session.

3.6 BPTF Bound Calculation

Now, we need to write down the bound equation 41 in matrix algebra. The only difficult part is the term containing $\langle W(u) \rangle \log p(u)$. Let's expand this term with identities 30 and 31:

$$\mathcal{B}_{part} = \sum_{\bar{u}_0} \sum_{u_0} M(u_0) \langle W(u) \rangle \log p(u) \tag{57}$$

$$= \sum_{\bar{u}_0} \sum_{u_0} M(u_0) X(u_0) p(u) \log p(u) \tag{58}$$

$$= \sum_{u_0} M(u_0) X(u_0) \sum_{\bar{u}_0} \frac{\prod_n \exp(\langle \log Z_n(u_n) \rangle)}{\sum_{\bar{u}_0} \prod_n \exp(\langle \log Z_n(u_n) \rangle)} \log \left(\frac{\prod_n \exp(\langle \log Z_n(u_n) \rangle)}{\sum_{\bar{u}_0} \prod_n \exp(\langle \log Z_n(u_n) \rangle)} \right) \tag{59}$$

we can write this as matrix multiplication as

$$\mathcal{B}_{part} = \sum_{u_0} M(u_0) X(u_0) \sum_{\bar{u}_0} \frac{\prod_n L_n(u_n)}{\hat{X}_L(u_0)} \left(\log \left(\prod_n L_n(u_n) \right) - \log(\hat{X}_L(u_0)) \right) \tag{60}$$

$$= \sum_{u_0} M(u_0) X(u_0) \left(\frac{1}{\hat{X}_L(u_0)} \sum_{\bar{u}_0} \left(\prod_n L_n(u_n) \right) \log \left(\prod_n L_n(u_n) \right) - \log(\hat{X}_L(u_0)) \right) \tag{61}$$

One important thing to note here is that the L matrices are the ones used in calculating the sufficient statistics $p(u)$, which means that they are actually the *old* values, while other variables such as C, D and E are the updated ones. Therefore, we either need to cache the old L matrices to calcualte the bound after updating all factors, or, we calculate bound during the factor updates, just before updating L's. The second option is more efficient and we choose to do so.

```
def calc_bound(X, M, A, B, C, D, E, L):
    kr_L = khatri_rao(L)
    kt_L = kruskal_to_tensor(L)
    kt_E = kruskal_to_tensor(E)
    sum_kr_L_log_kr_L = (kr_L * log(kr_L)).sum(axis=-1).reshape(X.shape)
    B_part = np.sum(M * X * (sum_kr_L_log_kr_L / kt_L - np.log(kt_L))
    bound = -np.sum(M * kt_E) - B_part - np.sum(M * gammaln(X+1))
    for n in range(X.ndim):
        bound -= np.sum(E[n] * A[n] / B[n])
        bound -= np.sum(A[n] * np.log(B[n] / A[n]))
        bound -= np.sum(np.gammaln(A[n]))
        bound += np.sum(C[n] * (np.log(D[n]) + 1) + np.gammaln(C[n]))
    return bound
```

References

[1] B. Ermis and A. T. Cemgil, "A Bayesian Tensor Factorization Model via Variational Inference for Link Prediction." *ArXiv*, 2014.

[2] A. T. Cemgil, "Bayesian inference for nonnegative matrix factorisation models." *Computational intelligence and neuroscience*, vol. 2009, 2009.

[3] Tensorly. [Online]. Available: https://www.tensorly.org

[4] Bayesian poisson tensor factorization. [Online]. Available: https://github.com/aschein/bptf

[5] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.

[6] R. Bro, "Parafac. tutorial and applications," *Chemometrics and Intelligent Laboratory Systems*, vol. 38, no. 2, pp. 149 – 171, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0169743997000324