Community Detection Meets Neural Networks: An Experiment

Ziting Bai[b,1], **Zhiyuan Yang**[a,1], **and Qingyi Zhao**[a,1,2]

[a]Department of Computer Science, University of California, Los Angeles, CA, 90024; [b]Department of Mathematics, University of California, Los Angeles, CA, 90024

This manuscript was compiled on July 12, 2018

We examine several popular community detection method in the network science community, and draw inspiration from spectral clustering to come up with a neural-network based approach for detecting communities in complex networks. We then proceed to compare the performance of each algorithm on a set of established benchmark graphs.

Community Detection | Deep Learning | Complex Networks

Community detection in complex networks is an important topic that has drawn continuing interests from researchers in different domains–network science, biology, social sciences, etc. In the recent years, with the rise of deep learning outperforming almost all other machine learning methods for data classification/clustering tasks, interests have grown, especially in the Computer Science community, for applying neural-networks approaches to network community detection tasks, as they share many similarities in terms of problem formulation and goals.

In this paper, we will first introduce 'baseline' community detection algorithms we considered to compare results with. Notice that 'baseline' is not a implication of their performance– in fact, some of them are very competitive methods that performed community detection on our benchmark networks near-perfectly. We then introduce the neural-network based approach we consider for efficient encoding of network into lower dimensions. We compare it to other methods, and consider further improvements.

Baseline Community Detection Algorithms

Community detection has gained a huge amount of interests after the notion of modularity was introduced in (3), which is basically the number of edges falling within communities minus the number if placed at random in a network with similar structure. A lot of attention has been paid to maximize the modularity to achieve a optimal partitioning of graphs. However, it is proved in (1) that the modularity maximization problem is NP-complete in the strong sense. Besides this, there are many other categories of algorithms to detect communities in a complex network (4). One of the earliest and most traditional method for generating partitions of a graph is spectral clustering. It proposed to change the initial data points to a space whose coordinates are elements of eigenvectors of some form of similarity matrix of the original data, and it claimed that the representation induced from eigenvectors is more convenient and evident in terms of clustering (2). There are also methods that are divisive in the sense that it removes some edges or nodes in the network to divide the network into different components. For example, the algorithm of Girvan-Newman is based on iteratively removing the edge with the largest edge betweenness centrality(5). Some algorithms are based on similarity measures, such as Walktrap algorithm in which the distance metric is obtained from the probability of a random walker can walk from a node to another within a given number of steps(6). In this paper, we took three classic algorithms from three distinct categories as baselines: Louvain's Algorithm, Label Propagation Algorithm, and Non-negative Matrix Factorization.

Louvain's Algorithm. Louvain's algorithm is first proposed in (7) for community detection in weighted networks. It is a greedy modularity maximization algorithm based on the change in modularity given some condition, and modularity Q is defined in (3) as

$$Q = \frac{1}{2m}\sum_{i,j}[A_{ij} - \frac{k_i k_j}{2m}]\delta(c_i, c_j), \qquad [1]$$

where $m = \frac{1}{2}\sum_{ij} A_{ij}$ is the total number of edges, A_{ij} is the weight of the edge between node i and j, k_i represents the sum of weights of the edges attached to node i, and $\delta(\cdot)$ is the kronecker delta function. As shown in Eq. 1, the value of modularity in this case is always between -1 and 1, and it measures the fractions of edges connecting nodes within a community versus those placed at random between different communities.

This algorithm is divided into two major steps. In step 1, we try to find a structure of current network that achieves a local optimum, and in step 2, we modify the structure of the network based on the result of step 1. We call one combination of step 1 and 2 a *pass*.

Step 1: Consider a weighted network with N nodes. We first initialize all the nodes into different communities, so after the very first initialization process, there will be N communities in the network. Then we assign an order to the set of all

Significance Statement

Community detection in complex networks aims to cluster nodes into groups(communities) such that nodes have a "stronger" connection with nodes within the same group than that with nodes in different groups. In this work, we compare classic community detection algorithms on our benchmark datasets and introduce a new approach based on auto-encoder in the context of deep learning.

Z.B., Z.Y. Q.Z. designed and performed research, wrote the code, analyzed the results, and wrote the paper

The authors declare no conflict of interests.

[1]Z.B., Z.Y., and Q.Z. contributed equally to this work.

[2]To whom correspondence should be addressed. E-mail: zhaoqy1997@outlook.com

nodes (the ordering can affect the computational efficiency, but in this paper, we only consider the simplest case of random ordering). Following our pre-defined order, for node i with j neighbors, we assume it will be put into one of its neighbors' community C, and calculate the change in modularity ΔQ of the entire network as in Eq. 2:

$$\Delta Q = [\frac{\sum_{in}+k_{i,in}}{2m} - (\frac{\sum_{total}+k_i}{2m})^2] - [\frac{\sum_{in}}{2m} - (\frac{\sum_{total}}{2m})^2 - (\frac{k_i}{2m})^2] \quad [2]$$

where $\sum_{in}$ is the total weights of edges within community C, $\sum_{total}$ is the total weights of edges incident to nodes in C, k_i is the total weights of edges incident to node i, $k_{i,in}$ is the total weights of edges from node i to nodes in C, and m is the total weights of edges in the network. As a result, we get ΔQ_j for each neighboring community C_j that contains neighbor j of node i. Then we define ΔQ_{max} as

$$\Delta Q_{max} = \arg\max_j \Delta Q_j$$

. If ΔQ_{max} is positive, then we assign node i to the community of node j; otherwise, node i keeps unchanged. To finish one round of step 1, we repeat the above process until there is no positive modularity change in the network, which means that we reached a local maximum of modularity.

In step 2, we take the resulting communities from step 1 to be nodes, and define the edge weight between them to be the total weights of edges between nodes in the corresponding two communities. After step 2 is done, we reapply step 1 to the resulting network of the previous step, which means the beginning of a new pass. We keep repeating passes until the structure of the network does not change.

This is an extremely fast algorithm, since it is based on greedy optimization and finds local optimum at each step. The exact complexity is not known yet, but most of the running time is spent on the greedy optimization phase in step 1, and the complexity appears to be O(NlogN).

Label Propagation Algorithm. First proposed in (9), Label Propagation is a form of propagation that can detect communities in a network very efficiently. It is claimed in (9) that it takes a near-linear running time and the implementation is easy.

The initialization is the same with Louvain's algorithm introduced in the previous section: we assign each of N nodes in the network with a different community, so that we will have N distinct community at the very beginning of this algorithm. And we assign a random order to the set of nodes, which is claimed to be independent to the running time or the rate of convergence of the algorithm.

The propagation phase is defined as follows: following the order we initially defined, for each node i, we check all of its neighbors and record one communities that most of i's neighbors belong to, with ties broken randomly.

The propagation step might converge to a fixed network structure because it is possible to have nodes with the same maximum number of neighbors belonging to two or more communities. Then the network structure will never be stable given that we break the tie randomly between neighbors. Therefore, we define a stopping criterion as follows: we stop the propagation if for every node i, the number of its neighbors in the same communities as i's is greater than or equal to that of neighbors out of i's communities. It is worth to mention that this criterion is very similar to the definition of strong community, which is the same if we ignore the case of equality in the above stopping criterion.

This algorithm uses a synchronous update, where at time t, node x updates its label based on labels of its neighbors at time t-1. Therefore, the label of c at time t, $C_x(t) = f(C_{x_1}(t-1), C_{x_2}(t-1), ..., C_{x_m}(t-1))$ where m is the number of neighbors node x has. However, in this setting, the problem of oscillating labels will occur in bi-partite or near bi-partite structures in the network, such as a star graph. So we use asynchronous update instead, and it is defined as

$$C_x(t) = f(C_{x_{i1}}(t), ..., C_{x_{ik}}(t), C_{x_{i(k+1)}}(t-1), ..., C_{x_{im}}(t-1))$$

where $x_{i1}, ..., x_{im}$ are neighbors of node x in which $x_{i1}, ..., x_{ik}$ are neighbors already been updated in the current iteration and $x_{i(k+1)}, ..., x_{im}$ are those not yet updated in the current iteration.

Due to usage of random tie breaking strategy, there will be multiple possible partition of the same starting network, but they are found to be similar to each other (4). Therefore, it is always useful to consider the aggregation of all the resulting partitions, which can be used to detect the overlapping communities. This is method is famous for its extremely low computational cost, with each iteration running in O(m), and the number of iterations needed to converge depends on the network size. Another advantage of it is that it doesn't need any parameters, and the number and size of clusters can be random and are n

Non-negative Matrix Factorization. The Non-Negative Matrix Factorization method we used in this project is based on (8), where it utilize computationally efficient Bayesian nonnegative matrix factorization (NMF) to perform community detection. The advantage of this method is that it can detect overlapping community in cases where it is relevant. For cases that overlapping is not relevant, it can detect how strong each node is connected to each community, which could be valuable information. However, this method is $O(n^2)$ if using to its full potential, which can be extremely time costly. The run time can be reduced by setting a max rank before running the simulation, which can cause some problem for a network with unknown number of communities.

This algorithm is evolved around the equation $\widehat{V}_{ij} = \sum_{k=1}^{K} w_{ik} h_{kj}$, where V is the expected adjacency matrix and w and h are the non-negative matrices. The goal is to adjust W and H to get the expected adjacency matrix $\widehat{V}$ as close as to the original adjacency matrix, and W and H would represent our community detection results, where w_{ik} represents the degree of participation node i is in the community k.

The NMF method of community detection is to some extent similar to the autoencoder-based algorithm that we want to discuss, in a way that the NMF improves its non-negative matrices through a for loop until the result is close enough. The training equations of $W \in \mathbb{R}^{N\times K}$ and $H \in \mathbb{R}^{K\times N}$, where N is the total node number and K is the number of latent modules, is given by

$$H \leftarrow (\frac{H}{W^T 1 + BH}) \cdot [W^T (\frac{V}{WH})]$$

.

and

$$W \leftarrow (\frac{W}{1H^T + WB}) \cdot [(\frac{V}{WH})H^T]$$

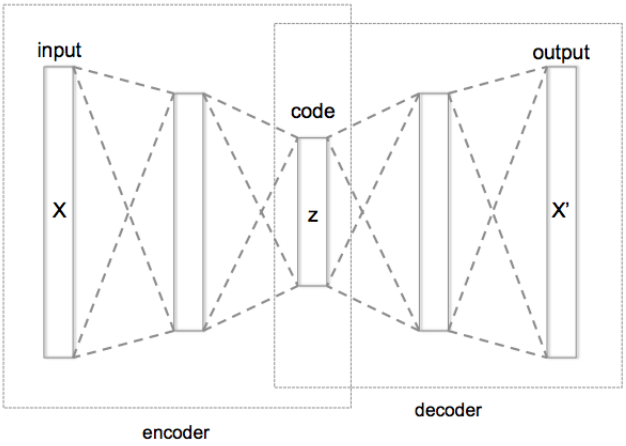
.

where V $\in \mathbb{R}^{N \times N}$ is the original adjacency matrix, a and b are fixed parameters, B $\in \mathbb{R}^{K \times K}$ is a matrix with β_k on its diagonal and zero everywhere else. For further information about the origin of Posterior Cost function, refer to the Supplementary below.

The original author mentioned cutting the zero columns generated throughout the for loop, but did not implement this mechanic in the code provided. Therefore, we implemented this mechanic. After each loop comparing $\widehat{V}$ and V, we go through B matrix and find the diagonal entry that is greater than a threshold, and erase the column corresponding to that entry from W and H, since a high β_k means that column is consist of most zeros and is irrelavent. This threshold is dependent on the network size and is not set for each instance.

Autoencoder-based algorithm

The particular formation of artificial neural networks we consider is called 'Autoencoders'. In short, they are structured so that the input and output layer shares the same amount of neurons, while in the hidden layers, the number of neurons get smaller.



Furthermore, constraints are set so that it tries to learn an approximation of the identity function as the target function from output layer. By doing so, the final values of the 'code' layer, as we can see from the graph above, can be viewed as an efficient encoding of the original input in lower dimensions–the 'encoder' learns a function

$$E(\mathbf{X}) = \hat{\mathbf{X}},$$

where $\mathbf{X} \in \mathbb{R}^{n \times n}$ is the original input, while $\hat{X} \in \mathbb{R}^{n \times k}$ is the transformed representation of $\mathbf{x}$ in K dimension, k $\ll$ n. The decoder, on the other hand, performs the opposite task: given $\hat{\mathbf{X}}$, it tries to reconstruct $\mathbf{Y} \in \mathbb{R}^{n \times n}$ such that $\mathbf{Y} \approx \mathbf{X}$:

$$D(\hat{\mathbf{X}}) = \mathbf{Y} \approx \mathbf{X}$$

In the context of network detection, the choice of original input to the autoencoder(and other neural-network based approaches) is not well-studied, different inputs have been used (but not explained): the graph Laplacian $\mathbf{L}$, the modularity matrix (3) $\mathbf{B}$, and other matrices based on an pair-wise similarity measure of nodes in the network. For our analysis, we used each and all of them as input to the autoencoder, and chose the one that gives us the best results in the end.

Note that this idea of using an autoencoder to perform dimensionality reduction on the network is analogous to what spectral clustering tries to do–project a representation of the network to its leading k eigenvectors to reduce dimensionality for later clustering step. However here, we perform this task in an unsupervised way, in the sense that no basis vectors are chosen for the newer dimensions; we also allow non-linear mapping by the use of nonlinear activation functions in the neurons.

Once we have the new representation of the network, we can utilize standard clustering algorithms (such as K-Means) to obtain clusters (communities) of the network. K-Means is chosen for its simplicity in implementation and efficiency in computation (typically O(K * n * d)).

The standard K-Means algorithm repeats itself in two steps:

1. **Assuption**: for a given number of clusters k, randomly choose k data points as the 'centroids' of each cluster, and assign cluster membership to all other points according to the smallest value of a distance measure–L2 norm being the most frequently used.

2. **Update**: Once each node is assigned a cluster membership, we recompute the cluster centroids based on currently assignment.

Repeat the two steps until the assignment becomes stable. The final clusters become the output. Since K-Means depends on random assignment, multiple runs might be needed for optimal output. We can also use the Kmean++ initialization steps(10) to overcome this issue. Also, since the number of clusters is often unknown beforehand, we can perform K-Means algorithm on a range of values of k and choose the one that maximize a target measure–in our case, the network modularity.

Benchmark Datasets

To test the above methods, we use different realizations of the Lancichinetti–Fortunato–Radicchi benchmark(11) (LFR). LFR is an algorithm for generating benchmark networks, specifically for the purpose of testing community detection algorithms. It is developed to account for heterogeneity of both community sizes and degrees of nodes in many real-life networks. Specifically, it generates graphs with power-law degree distribution and community size distribution.

Various parameters about the network can be specified for LFR, so that networks of different size and properties can be generated and compared with. One parameter with particular importance is what is called the *mixing parameter*, denoted by μ. It is the fraction of edges for every nodes in a given community that connects to nodes outside the community. In different words, for any given node in a community, 1 - μ is the

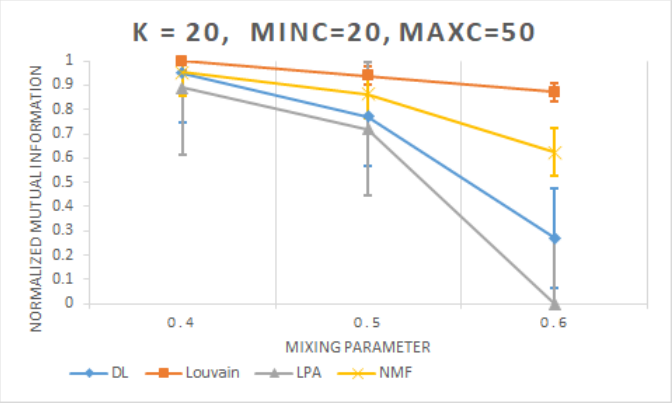
fraction of edges that connects to nodes within the community. This parameter is interesting because a value smaller than 0.5 gives us communities in a 'strong' sense(12).

Normalized Mutual Information. Mutual Information is a measurement of the mutual dependency of two variables. In other words, it gives a numerical measurement on how much information one random variable X can gain through the other random variable Y. The calculation of Mutual Information I(X;Y) is $I(X;Y) = H(X) + H(Y) - H(X,Y)$, where $H(X)$ and $H(Y)$ are marginal entropies and $H(X,Y)$ is the joint entropy. We used the normalized version, the Normalized Mutual Information (NMI) so that we can measure and compare the NMI between different clusterings having different number of clusters.(13)

$$NMI(X,Y) = \frac{2 \cdot I(X,Y)}{H(X) + H(Y)}$$

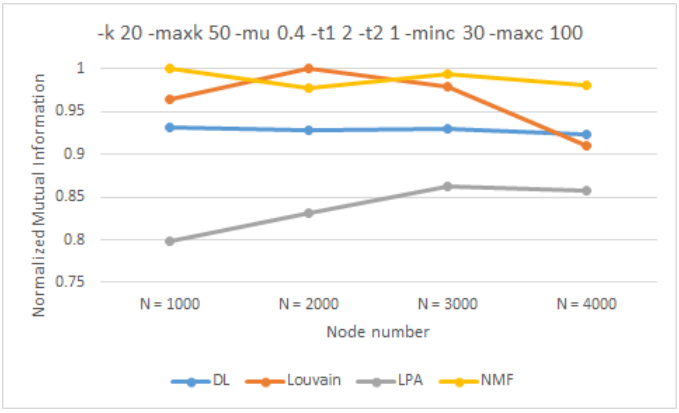
Results

Using the NMI and the dataset, and the four method we discussed above. We performed a series of simulation. The following results are obtained after averaging 10 runs of a certain parameter. We first want to see the result when varying the mixing parameter ɥ. We fixed mean degree k at 20, max k at 50, minimum community at 20 and maximum community at 50, with 1000 nodes and see the result of varying ɥ from 0.4 to 0.6.



First thing to note is that LPA method drops to 0 when ɥ is 0.6, which is caused by the LPA algorithm, since LPA's simulation terminates only when the number of its neighbors in the same communities as i's is greater than or equal to that of neighbors out of i's communities in the produced community. This condition cannot be satisfied when ɥ=0.6 because there is an average of 60% edge connected to other communities. It is further discussed in the stopping condition in LPA section.

We then want to see the result when varying the node number n. We fixed mean degree k at 20, max edge at 50, minimum community at 30 and maximum community at 100, with ɥ at 0.4. We vary n from 1000 to 4000. Ideally, we want to perform a exponential node increment analysis, but since 10000 node simulation is extremely complex and time-costly for the NMF method, we performed the simulation on a linear increment.



As we can see, the deep learning algorithm performed fairly stable as the node number increases.

The neural network model is not the best performing community detection algorithms, but it has a lot of potential to improve. The neural network deep learning algorithm we use in this paper is still very primitive. The structure can still be improved. With the computational ability of the modern GPU, we can add more layers to the network or stack multiple auto encoders together to possibly obtain better results.

ACKNOWLEDGMENTS. The authors of this paper would like to thank Professor Mason Porter for his guidance through out this research project, and also TA Yacoub Kureh for his explanation on some of the key concepts.

1. Brandes U, Delling D, Gaertler M, Goerke R, Hoefer M, Nikoloski Z and Wagner(2006) *Maximizing modularity is hard* Preprint physics/0608255
2. W.E. Donath and A.J. Hoffman(1973) *Lower Bounds for the Partitioning of Graphs* IBM J. Research and Development, pp. 420-425
3. Newman, M.E.J.(2006) *Modularity and community structure in networks.* In: Proc. National Academy of Sciences, USA
4. Fortunato, S. (2010) *Community Detection in Graphs.* Physics Reports (486:3-5), pp. 75-17
5. M. Girvan and M. E. J. Newman(2002) *Community structure in social and biological networks* Proc. Natl. Acad. Sci. USA, 99, pp. 8271–8276
6. Pons P, Latapy M (2005) *Computing communities in large networks using random walks.* Computer and Information Sciences—ISCIS
7. Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.(2008) *Fast unfolding of communities in large networks.* J. Stat. Mech. P10008.
8. Psorakis, I., Roberts, S., Ebden, M., Sheldon, B.(2011)*Overlapping Community Detection Using Bayesian Non-Negative Matrix Factorization.* Physical Review E, vol. 83, no. 6
9. Raghavan UN, Albert R, Kumara S (2007) *Near linear time algorithm to detect community structures in large-scale networks.* Phys Rev E 76: 036106.
10. Arthur, David, and Sergei Vassilvitskii. *k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics, 2007.
11. Lancichinetti, Andrea, Santo Fortunato, and Filippo Radicchi. *Benchmark graphs for testing community detection algorithms.* Physical review E 78.4 (2008): 046110.
12. Radicchi, Filippo, et al. *Defining and identifying communities in networks.* Proceedings of the National Academy of Sciences of the United States of America 101.9 (2004): 2658-2663.
13. L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, J. Stat. Mech., 2005, P09008 (2005)

Supplementary

This Supplementary is a summary of Posterior-based cost function discussed in section II.B. of (8), to aid the understanding of section NMF above. None of the work below is original.

The joint distribution over all variables is:

$$p(\mathbf{V},\mathbf{W},\mathbf{H},\beta) = P(\mathbf{V}|\mathbf{W},\mathbf{H})p(\mathbf{W}|\beta)P(\mathbf{H}|\beta)p(\beta). \quad [3]$$

hence the posterior over model parameters given the observations is:

$$p(\mathbf{W},\mathbf{H},\beta|\mathbf{V}) = \frac{P(\mathbf{V}|\mathbf{W},\mathbf{H})p(\mathbf{W}|\beta)P(\mathbf{H}|\beta)p(\beta)}{p(\mathbf{V})}. \quad [4]$$

Noting that p(V) is a constant in 4, we take logarithm on both sides of 4:

$$\begin{aligned}\mathcal{U} &= -\log p(\mathbf{W},\mathbf{H},\beta|\mathbf{V})p(\mathbf{V}) \\ &= -\log p(\mathbf{V}|\mathbf{W},\mathbf{H}) - \log p(\mathbf{W}|\beta) - \log p(\mathbf{H}|\beta) - \log p(\beta).\end{aligned} \quad [5]$$

The first term is the log-likelihood which is derived from the probability. We express the negative log-likelihood of a single observation v_i, j as:

$$-\log p(v|\hat{v}) = -v\log\hat{v} + \hat{v} + \log v!. \quad [6]$$

Using the Stirling approximation, 6 can be written as:

$$-\log p(v|\hat{v}) \approx v\log\left(\frac{v}{\hat{v}}\right) + \hat{v} - v + \frac{1}{2}\log 2\pi v. \quad [7]$$

thus:

$$\begin{aligned}-\log p(\mathbf{V}|\hat{\mathbf{V}}) &= -\sum_{i=1}^{N}\sum_{j=1}^{N}\log p(v_{ij}|\hat{v}_{ij}) \\ &\simeq \sum_{i=1}^{N}\sum_{j=1}^{N}\left(v_{ij}\log\frac{v_{ij}}{\hat{v}_{ij}} + \hat{v}_{ij} - v_{ij} + \frac{1}{2}\log 2\pi v_{ij}\right) + \kappa.\end{aligned} \quad [8]$$

where κ is a constant.

Place independent half-normal prior over the columns of $\mathbf{W}$ and rows of $\mathbf{H}$, we get that the negative log priors over $\mathbf{W}$ and $\mathbf{H}$ are then given by:

$$\begin{aligned}-\log p(\mathbf{W}|\beta) &= -\sum_{i=1}^{N}\sum_{k=1}^{K}\log\mathcal{HN}(0,\beta_K^{-1}) \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K}\left(\frac{1}{2}\beta_k\omega_{ik}^2\right) - \frac{N}{2}\log\beta_k + \kappa,\end{aligned} \quad [9]$$

$$\begin{aligned}-\log p(\mathbf{H}|\beta) &= -\sum_{k=1}^{K}\sum_{j=1}^{N}\log\mathcal{HN}(0,\beta_K^{-1}) \\ &= \sum_{k=1}^{K}\sum_{j=1}^{N}\left(\frac{1}{2}\beta_k h_{kj}^2\right) - \frac{N}{2}\log\beta_k + \kappa.\end{aligned} \quad [10]$$

We can then perform a standard Gamma distribution over them with fixed hyper-hyperparameters a,b. The negative log hyper-priors are thus:

$$\begin{aligned}-\log p(\beta) &= -\sum_{k=1}^{K}\log\mathfrak{S}(\beta_k|a,b) \\ &= \sum_{k=1}^{K}\left(\beta_k b - (a-1)\log\beta_k\right) + \kappa.\end{aligned} \quad [11]$$

The objective function $\mathcal{U}$ of 5 can be expressed as the sum of 8 through 11:

$$\begin{aligned}\mathcal{U} = &\sum_i\sum_j\left[v_{ij}\log\frac{v_{ij}}{\hat{v}_{ij}} + \hat{v}_{ij}\right] \\ &+ \frac{1}{2}\sum_k\left[\left(\sum_i\beta_k w_{ik}^2\right) + \left(\sum_j\beta_k h_{kj}^2 - 2N\log\beta_k\right)\right] \\ &+ \sum_k(\beta_k b_k - (a_k-1)\log\beta_k) + \kappa.\end{aligned} \quad [12]$$