Sampling

1 Introducing

Consider a Nth-order tensor, $\boldsymbol{\mathcal{X}}$ with size $L_1 \times L_2 \times \ldots \times L_N$, the CP decomposition of this tensor is

$$\boldsymbol{\mathcal{X}} = \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] = \sum_{r=1}^{R} \boldsymbol{a}_r^{(1)} \circ \cdots \circ \boldsymbol{a}_r^{(N)} \tag{1}$$

where

$$\begin{aligned}
\mathbf{A}^{(1)} &= \left[\boldsymbol{a}_1^{(1)}, \boldsymbol{a}_2^{(1)}, \cdots, \boldsymbol{a}_r^{(1)}\right] \in R^{L_1 \times R} \\
\mathbf{A}^{(2)} &= \left[\boldsymbol{a}_1^{(2)}, \boldsymbol{a}_2^{(2)}, \cdots, \boldsymbol{a}_r^{(2)}\right] \in R^{L_2 \times R} \\
&\vdots \\
\mathbf{A}^{(N)} &= \left[\boldsymbol{a}_1^{(N)}, \boldsymbol{a}_2^{(N)}, \cdots, \boldsymbol{a}_r^{(1)}\right] \in R^{L_N \times R}
\end{aligned}$$

The column size R is the rank of this tenor, which means $\boldsymbol{\mathcal{X}}$ can be represented by a sum of R rank one tensor. $\boldsymbol{a}_r^{(n)}$ is the r-th column of matrix $\mathbf{A}^{(n)}$. In general, let $\boldsymbol{a}_{*r}^{(n)}$ be the k-th column of $\mathbf{A}^{(n)}$, $\boldsymbol{a}_{i_n *}^{(n)}$ be the i_n-th row vector of $\mathbf{A}^{(n)}$, and $a_{i_n k}^{(n)}$ be the element of $\mathbf{A}^{(n)}$.

The element in $\boldsymbol{\mathcal{X}}$ satisfies the equation:

$$x_{\boldsymbol{i}} = \sum_{r=1}^{R} a_{i_1 r}^{(1)} \cdot a_{i_2 r}^{(2)} \cdots a_{i_N r}^{(N)} \tag{2}$$

$\boldsymbol{i}$ is a shorthand for multi-index $(i_1, i_2, \ldots, i_N)$. We propose a method for estimating the maximum elements $x_{\boldsymbol{i}}$ given factor matrices $\mathbf{A}^{(n)}, n = 1, 2, \ldots, N$.

Notation	Explanation
$\boldsymbol{\mathcal{A}}$	tensor
$\mathbf{A}$	matrix
$\mathbf{A}^{(n)}$	n-th factor matrix of tensor
$\boldsymbol{a}_{*r}, \boldsymbol{a}_r$	k-th column of matrix
$\boldsymbol{a}_{i*}$	i-th row of matrix
$\boldsymbol{a}$	vector
a_{ir}	element of matrix

Table 1: Notation

2 Diamond Sampling

Without the loss of generality, suppose $\mathcal{X}$ is a three order tensor with size $I \times J \times K$, and the CP decomposition is given by

$$\mathcal{X} = [\![\mathbf{A}, \mathbf{B}, \mathbf{C}]\!] = \sum_{r=1}^{R} \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r \tag{3}$$

$$x_{ijk} = \sum_{r=1}^{R} a_{ir} \cdot b_{jr} \cdot c_{kr} \tag{4}$$

Where $\mathbf{A} \in R^{I \times R}, \mathbf{B} \in R^{J \times R}, \mathbf{C} \in R^{K \times R}$.

2.1 Graph representation

Those N factor matrices are represented by a weighted $(N+1)$-partite graph. And we call those N+1 partitions $\overline{V}, V_1, V_2, \ldots, V_N$ that V_n has L_n vertices, $\overline{V}$ has R vertices. Every two vertices from different partition classes V_i are nonadjacent. A vertice $\overline{v}_r$ in $\overline{V}$ is adjacent to the vertice v_i^n in V_n only when $a_{i_n r}^{(n)}$ is non-zero, and then assigned the weight to be $a_{i_n r}^{(n)}$.

Under the presentation, our sampling method can be expressed in plain English: we firstly pick an edge $e = (v_{i_1}^1, \overline{v}_k)$ with some probability, then walk N-1 times randomly from the $\overline{v}_k$ to the other partitions, record the N-1 vertices $i_2, i_3, \ldots, i_N$. Secondly, we walk from $v_{i_1}^1$ randomly to $\overline{V}$ and end in $\overline{v}'_k$. According to these N+2 vertices, we give a score to the coordinate $\boldsymbol{i} = (i_1, i_2, \ldots, i_N)$

2.2 Probability of edges and walks

- Walk with probability

  When we start from a vertice in V_1 to $\overline{V}$, or from a vertice in $\overline{V}$ to V_i, we choose the path according to the weight. That is given i_1, pick $r \in \{1, 2, \ldots, R\}$ with probability $|a_{i_1 r'}^{(1)}| / \parallel \boldsymbol{a}_{i_1 *}^{(1)} \parallel_1$ or given r, pick $i_n \in \{1, 2, \ldots, L_n\}$ with probability $|a_{i_n r}^{(n)}| / \parallel \boldsymbol{a}_{* r}^{(n)} \parallel_1$

- Assign probabilities to edges

  If $a_{i_1 k}^{(1)} \neq 0$ than assign the Assign pair $(v_{i_1}^1, \overline{v}_r)$ to be

  $$p(e) = \mid a_{i_1 k}^{(1)} \mid \parallel \boldsymbol{a}_{i_1 *}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{* r}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{* r}^{(2)} \parallel_1 \ldots \parallel \boldsymbol{a}_{* r}^{(N)} \parallel_1 / \parallel \mathbf{W} \parallel_1$$

  Where

  $$\parallel \mathbf{W} \parallel_1 = \sum_{i_1, r} \mid a_{i_1 k}^{(1)} \mid \parallel \boldsymbol{a}_{i_1 *}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{* r}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{* r}^{(2)} \parallel_1 \ldots \parallel \boldsymbol{a}_{* r}^{(N)} \parallel_1$$

2.3 Scoring samples

We walk S times, and each time we will get an coordinate $\boldsymbol{i} = (i_1, i_2, \ldots, \imath_N)$. If this coordinate has not been sampled previous, let the score $\mathbf{X}_{\boldsymbol{i},\ell}$ in the ℓ-th turn be $sgn(a_{i_1 r}^{(1)} \cdot a_{i_2 r}^{(2)} \cdots a_{i_N r}^{(N)} \cdot a_{i_1 r'}^{(1)}) a_{i_2 r'}^{(2)} \cdots a_{i_N r'}^{(N)}$, and assign $\widehat{x}_{\boldsymbol{i}} = \mathbf{X}_{\boldsymbol{i},\ell}$. Otherwise, assign $\widehat{x}_{\boldsymbol{i}}$ by $\mathbf{X}_{\boldsymbol{i},\ell}$.

2.4 Correctness and error bounds

We define the walk to be event $\varepsilon_{\boldsymbol{i},r',r}$, that is pick a pair $(v_{i_1}^1, \overline{v}_r)$ then pick pathes from $V_{i_1}^1$ in V_1 to $\overline{V}$ and an addition path from $\overline{v}_r$ in $\overline{V}$ to V_1.

Lemma 2.1. *The expectation of $\widehat{x}_{\boldsymbol{i}}$ equals to $s \cdot x_{\boldsymbol{i}} / \parallel \mathbf{W} \parallel_1$.*

Proof: Suppose each walk is independent. And the final score $\widehat{x}_{\boldsymbol{i}} = \sum_{\ell=1} \mathbf{X}_{\boldsymbol{i},\ell}$. So that

$$\mathbb{E}[\widehat{x}_{\boldsymbol{i}}] = \mathbb{E}[\sum_{\ell=1} \mathbf{X}_{\boldsymbol{i},\ell}] = s\mathbb{E}[\mathbf{X}_{\boldsymbol{i},1}] \tag{5}$$

The probability of $\varepsilon_{\boldsymbol{i},r',r}$ is

$$\begin{aligned}
Pr(\varepsilon_{\boldsymbol{i},r',r}) &= Pr(\text{pick } (v_{i_1}^1, \overline{v}_r) \cdot \prod_{n=2}^{n=N} Pr(\text{pick } (v_{i_n}^n, \overline{v}_r)|\text{given } \overline{v}_r) \cdot Pr(\text{pick } (v_{i_1}^1, \overline{v}_r')|\text{given } v_{i_1}^1) \\
&= \frac{w_{i_1 r}}{\parallel W \parallel_1} \cdot \frac{|a_{i_1 r'}^{(1)}|}{\parallel \boldsymbol{a}_{i_1 *}^{(1)} \parallel_1} \cdot \prod_{n=2}^{n=N} \frac{|a_{i_n r}^{(n)}|}{\parallel \boldsymbol{a}_{*r}^{(n)} \parallel_1} \\
&= \frac{|a_{i_1 r}^{(1)}| \parallel \boldsymbol{a}_{i_1 *}^{(1)} \parallel_1 \prod_{n=2}^{n=N} \parallel \boldsymbol{a}_{*r}^{(n)} \parallel_1}{\parallel W \parallel_1} \cdot \frac{|a_{i_1 r'}^{(1)}|}{\parallel \boldsymbol{a}_{i_1 *}^{(1)} \parallel_1} \cdot \prod_{n=2}^{n=N} \frac{|a_{i_n r}^{(n)}|}{\parallel \boldsymbol{a}_{*r}^{(n)} \parallel_1} \\
&= \frac{|a_{i_1 r'}^{(1)}| \prod_{n=1}^{n=N} |||a_{i_1 r'}^{(1)}|}{\parallel W \parallel_1}
\end{aligned}$$

We get the probability of one walk.

$$Pr(\varepsilon_{\boldsymbol{i},r',r}) = \frac{|a_{i_1 r'}^{(1)}| \prod_{n=1}^{n=N} |||a_{i_1 r'}^{(1)}|}{\parallel W \parallel_1} \tag{6}$$

Using equation in 5 and 6.

$$\begin{aligned}
\mathbb{E}[x_{\boldsymbol{i}}/s] &= \mathbb{E}[\mathbf{X}_{\boldsymbol{i},1}] \\
&= \sum_r \sum_{r'} Pr(\varepsilon_{\boldsymbol{i},r',r}) \cdot sgn(a_{i_1 r}^{(1)} \cdot a_{i_2 r}^{(2)} \cdots a_{i_N r}^{(N)} \cdot a_{i_1 r'}^{(1)}) a_{i_2 r'}^{(2)} \cdots a_{i_N r'}^{(N)} \\
&= \frac{1}{\parallel \mathbf{W} \parallel_1} \sum_r \sum_{r'} |a_{i_1 r}^{(1)}| \cdot |a_{i_2 r}^{(2)}| \cdots |a_{i_N r}^{(N)}| \cdot |a_{i_1 r'}^{(1)}| sgn(a_{i_1 r}^{(1)} \cdot a_{i_2 r}^{(2)} \cdots a_{i_N r}^{(N)} \cdot a_{i_1 r'}^{(1)}) a_{i_2 r'}^{(2)} \cdots a_{i_N r'}^{(N)} \\
&= \frac{1}{\parallel \mathbf{W} \parallel_1} \sum_r \sum_{r'} a_{i_1 r}^{(1)} \cdot a_{i_2 r}^{(2)} \cdots a_{i_N r}^{(N)} \cdot a_{i_1 r'}^{(1)} a_{i_2 r'}^{(2)} \cdots a_{i_N r'}^{(N)} \\
&= \frac{1}{\parallel \mathbf{W} \parallel_1} \{\sum_r a_{i_1 r}^{(1)} \cdot a_{i_2 r}^{(2)} \cdots a_{i_N r}^{(N)}\}^2 \\
&= \frac{x_{\boldsymbol{i}}^2}{\parallel \mathbf{W} \parallel_1}
\end{aligned}$$

$\square$

Lemma 2.2. *Fix $\varepsilon > 0$ and error probability $\sigma \in (0,1)$. Assuming all entries in factor matrices are nonnegative and at most K. If the number of samples*

$$s \leq 3K^N \parallel \mathbf{W} \parallel_1 \log 2/\sigma/(\varepsilon^2 {x_{\boldsymbol{i}}}^2),$$

then

$$Pr[|\widehat{x}_{\boldsymbol{i}} \parallel \mathbf{W} \parallel_1 /s - {x_{\boldsymbol{i}}}^2| \geq \varepsilon {x_{\boldsymbol{i}}}^2] \leq \sigma$$

Theorem 2.1. *Fix some threshold τ and error probability $\sigma \in (0,1)$. Assume all entries in factor matrices are nonnegative and at most K. Suppose $s \geq 12K^N \parallel \mathbf{W} \parallel_1 \log(2L_1 L_2 \cdots L_N/\sigma)/\tau^2$. Then with probability at least $1-\sigma$, the following holds for all indices $\boldsymbol{i} = (i_1, i_2, \ldots, i_N)$ and $\boldsymbol{i'} = (i_1', i_2', \ldots, i_N')$: if $x_{\boldsymbol{i}} > \tau$ and $x_{\boldsymbol{i'}} < \tau/4$, then $\widehat{x}_{\boldsymbol{i}} > \widehat{x}_{\boldsymbol{i'}}$.*

2.5 Finding top-t largest value

2.6 Finding k-NN for query

At this situation, we use on row of matrix $\mathbf{A}^{(1)}$ each time called $\boldsymbol{u}$. And the tenor of N-1 order called rank tensor for query $\boldsymbol{u}$.

$$\mathcal{X}_{\boldsymbol{u}} = \left[\!\left[\boldsymbol{u}, \mathbf{A}^{(2)} \ldots, \mathbf{A}^{(N)}\right]\!\right] = \sum_{r=1}^{R} u_r \cdot \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(N)} \tag{7}$$

And find the k most revelent vector sets, each vector set consist N-1 vectors and the

The biggest difference between different queries in sampling processing is the probability for picking vertice in partition $\overline{V}$, or the frequency sequence $(c_1, c_2, \ldots, c_R)$ of $\overline{v}_r$ to be sampled. Notice that, $\sum_{r=1}^{R} c_r = s$.

For effectively implementation, we use the lists $\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_R$ to record the sub-path(walk from vertice $\overline{v}_r$ in $\overline{V}$ to $V_i, i \in \{2, \ldots, N\}$.

2.6.1 Weight assigning and frequency generating

For each $1 \leq r \leq R$, let $u'_r \leftarrow\mid u_r \mid\parallel \boldsymbol{a}_{*r}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{*r}^{(2)} \parallel_1 \ldots \parallel \boldsymbol{a}_{*r}^{(N)} \parallel_1$. Instead of sampling r directly s times, choosing the c_r randomly such that c_r has the expected value $su'_r / \parallel u' \parallel_1$ will still works.

$$c_r = \begin{cases} \lfloor su'_r / \parallel u' \parallel_1 \rfloor, & \text{with probability } \lceil su'_r / \parallel u' \parallel_1 \rceil - su'_r / \parallel u' \parallel_1 \\ \lceil su'_r / \parallel u' \parallel_1 \rceil, & \text{with probability } \lfloor su'_r / \parallel u' \parallel_1 \rfloor - su'_r / \parallel u' \parallel_1 \end{cases}$$

2.6.2 Sub-walks

When given some r, sampling method need to pick the left indices $\boldsymbol{i'} = (i'_1, i'_2, \ldots, i'_N)$, which has been stored in the previous query, it saves a lot of computation.

- for $r = 1, 2, \ldots, R$:

  sample r' c_r times with the probability $|u_r| / \parallel u \parallel_1$.

- for $r = 1, 2, \ldots, R$:

-

3 Two Approaches

In this section, we introduce wedge sampling and the advanced edition diamond sampling.

3.1 Wedge Approach

A random-sampling based algorithm that identifies high-valued entries of nonnegative matrix products, by assigning scores to each entry of $\boldsymbol{q}^T\mathbf{A}$, where $\mathbf{A} \in R^{d\times n} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n]$ is a matrix whose rows are the instance, and d is the dimension of feature vector. Our task is to find the most relevant instance $\boldsymbol{a}_i$ that maximize dot product $\boldsymbol{q}^T\boldsymbol{a}_i$,

Algorithm 1 Wedge Sampling

Given matrix $\mathbf{A}$ and query$\boldsymbol{q}$

Let S be the number of samples.

Preprocessing stage
for k=1,...,d **do**
    compute $a_k = \sum_{h=1}^{n} a_{kh}$ and $q'_k = q_k a_k$
end for
for k=1,...,d **do** determine the number of samples c_i needed for

$$c_i = \begin{cases} \lfloor \frac{Sq'_k}{\sum q'_k} \rfloor, & \text{w/prob. } \lceil \frac{Sq'_k}{\sum q'_k} \rceil - \frac{Sq'_k}{\sum q'_k} \\ \lceil \frac{Sq'_k}{\sum q'_k} \rceil, & \text{w/prob. } \lfloor \frac{Sq'_k}{\sum q'_k} \rfloor - \frac{Sq'_k}{\sum q'_k} \end{cases}$$

end for
for $k = 1, \ldots, d$ **do**
    Sample $j \in \{1, \ldots, n\}$ c_k times with probability $p(e_{kj}) = a_{kj} / \sum_{h=1}^{n} a_{kh}$.
    Increase the counter of S_j.
end for
Postprocessing

3.2 Graph representation

Consider a product $\mathbf{M} = \mathbf{QA}$, where $\mathbf{Q} \in R^{m \times d}, \mathbf{A} \in R^{d \times n}$ are represented by a layered graph with three layers, and treat matrix $\mathbf{Q}, \mathbf{A}$ as weighted directed bipartite graph. There are m nodes $\{v_1^1, v_2^1, \ldots, v_m^1\}$ in first layer, d nodes $\{v_1^2, v_2^2, \ldots, v_d^2\}$ in second layer , and n nodes $\{v_1^2, v_2^2, \ldots, v_d^2\}$in the third. For a query vector $\boldsymbol{q}^T$, see it as the i-th row of $\mathbf{Q}$, and $\boldsymbol{m} = \boldsymbol{q}^T \mathbf{A}$, then define a trace: random walks from the i-th node, which is the query vector $\boldsymbol{q}^T$ in first layer, and terminates at the j-th node in third layer. In this situation, the first layer has one node.

Walk S times randomly will terminate in third layer S times, and record the time node $j \in \{1, 2, \ldots, n\}$ has been reached to S_j, clearly $\sum_{j=1}^{n} S_j = S$. If $j \in \{1, 2, \ldots, n\}$ is select with probability $m_j / \sum_{h=1}^{n} m_h$, then the expected value of S_j equals $m_j S / \sum_{h=1}^{n} m_h$. Hence, $S_j \sum_{h=1}^{n} m_h / S$ is an unbiased estimator for m_j.

For every pair of nodes $\{v_k^2, v_j^3\}$, assign the probability to the edge

$$p(e_{kj}) = a_{kj} / \sum_{h=1}^{d} a_{kh}$$

Processing goes in Algorithm 1. Think the trace as first walk to node v_k^2, and then v_j^3. So firstly, compute the times c_k expected to go through node v_k^2, and sample c_k times to arrive v_j^3.

- The expect of c_k.

$$E[c_k] = \frac{Sq'_k}{\sum q'_k}$$

- The expect of S_j.

$$\begin{aligned}
E[S_j] &= \sum_{k=1}^{d} c_k p(e_{kj}) \\
&= \sum_{k=1}^{d} \frac{S q_k a_k}{\sum q_k a_k} \frac{a_{kj}}{a_k} \\
&= \frac{S}{\sum q_k a_k} \sum_{k=1}^{d} q_k a_{kj} \\
&= \frac{S m_j}{\sum q_k a_k} = \frac{S m_j}{\sum m_h}
\end{aligned}$$

As we can see, $S_j \sum_{h=1}^{n} m_h / S$ is an unbiased estimator for m_j.

3.3 Implementing details for the wedge sampling

To find the maximum of $\mathbf{M} = \mathbf{Q}\mathbf{A}$, see $\mathbf{Q}$ as a bunch of query vector $\boldsymbol{q_i}$. Replace the matrix $\mathbf{A}$ by lists $L_1, L_2, \ldots, L_d$ of indices $\{1, 2, \ldots, n\}$. For each input query $\boldsymbol{q}$, the algorithm first processes $\boldsymbol{q}$ to determine how many samples are needed for each of the d lists L_k ,or the times c_k expected to go through node v_k^2. It then retrieves these samples from the appropriate stored lists and counts how many times each of $\{1, 2, \ldots, n\}$ was sampled. It saves the time for repeating sampling of different query in line 9 of Algorithm 1.

3.4 Diamond Approach

For $\boldsymbol{m} = \boldsymbol{q}^T \mathbf{A}$, the diamond sampling aim to find two intersecting wedges, so that the probability to terminate at $j = \{1, 2, \ldots, n\}$ is proportional to m_j^2.

4 Sampling in Tensor

We can consider the problem as finding some pair of vectors that from two different bunch of vector set, in previous, to have the the top-t maximum dot products.

Now, we extend the problem from two vector set to N vector sets. And find a list of vectors $\{\boldsymbol{a}_{i_1}^{(1)}, \ldots, \boldsymbol{a}_{i_N}^{(N)}\}$, where $\boldsymbol{a}_{i_n}^{(n)}$ is a vector from the n-th vector set $\mathbf{A}^{(n)}$ and $i_n \in \{1, \ldots, L^{(n)}\}$, according to the value of

$$\sum_{k=1}^{R} a_{i_1 k}^{(1)} a_{i_2 k}^{(2)} \cdots a_{i_N k}^{(N)}$$

A more mathematic interpretation is: given factor matrixes $\mathbf{A}^{(n)} \in R^{L^{(n)} \times R}, n = \{1, 2, \ldots, N\}$, which is the CP decomposition of tensor $\boldsymbol{\mathfrak{X}}$. Find the top-t value in tensor $\boldsymbol{\mathfrak{X}}$.

4.1 Wedge Sampling in Tensor

We separated those matrixes into three lapsers, the first layer has L^1 nodes, indicated by the first factor matrix $\mathbf{A}^{(1)}$. And the second layer called the middle layer has R nodes, which picture the feature dimension or the rank of CP decomposition. The layer has $N-1$ parallel layers, each layer has L^n nodes.Under this assumption, we define a trace: walk from the first layers at node $v_{i_1}^1$ to middle layer v_k^m , then terminates at each parallel layers with nodes $\{v_{i_2}^2, v_{i_3}^3, \ldots, v_{i_N}^N\}$.

As similar, define the probability of $\{v_k^m, v_{i_n}^n\}$:

$$P(e_{ki_n}) = a^{(n)}_{i_n k} / \sum_{h=1}^{R} a^{(n)}_{i_n h}$$

4.1.1 Find the most relevant vector lists of query q

Given a query $\boldsymbol{q}$, find the most relevant vector lists. It is the same when $\mathbf{A}^{(1)}$ is a vector or see $\boldsymbol{q}$ as one column of $\mathbf{A}^{(1)}$, since we define the row of factor matrix is the number of instances.

- Preprocessing
  $q'_k = q_k \parallel \boldsymbol{a}^{(2)}_{*r} \parallel_1 \ldots \parallel \boldsymbol{a}^{(N)}_{*r} \parallel_1$, where $\parallel \boldsymbol{a}^{(n)}_{*r} \parallel_1 = \sum_{h=1}^{R} a^{(n)}_{i_n h}$.
- Compute the times c_k expected to go through node v^m_k

$$c_i = \begin{cases} \lfloor \frac{Sq'_k}{\sum q'_k} \rfloor, & \text{w/prob. } \lceil \frac{Sq'_k}{\sum q'_k} \rceil - \frac{Sq'_k}{\sum q'_k} \\ \lceil \frac{Sq'_k}{\sum q'_k} \rceil, & \text{w/prob. } \lfloor \frac{Sq'_k}{\sum q'_k} \rfloor - \frac{Sq'_k}{\sum q'_k} \end{cases}$$

  As we can see

$$E[c_k] = \frac{Sq'_k}{\sum q'_k}$$

- Sample each node in parallel layers c_k times with probability $P(e_{ki_n})$, and increase the counter of $S_{i_2,\ldots,i_N}$
  The expect of $S_{i_2,\ldots,i_N}$.

$$\begin{aligned} E[S_{i_2,\ldots,i_N}] &= \sum_{k=1}^{R} c_k p(e_{ki_2}) p(e_{ki_3}) \cdots p(e_{ki_N}) \\ &= \frac{S}{\sum q'_k} \sum_{k=1}^{R} q_k \cdot a^{(2)}_{i_2 k} \cdots a^{(N)}_{i_N k} \\ &= \frac{S m_{i_2,\ldots,i_N}}{\sum q'_k} \\ &= \frac{S m_{i_2,\ldots,i_N}}{\sum m_{i_2,\ldots,i_N}} \end{aligned}$$

  So the $S_{i_2,\ldots,i_N} \sum m_{i_2,\ldots,i_N} / S$ is an unbiased estimator for $m_{i_2,\ldots,i_N}$.

4.1.2 Find the maximum value in tensor

The counter can only indicate the relatively relation of one vector list to query, which can not be used to compare different query. So we need to consider the value overall. Query a single vector , however, is still useful in some application like recommendation system, for at most time we only need to find the most relevant item for on user(query), and the global maximum is useless and time consuming. The trick of replace data instance by a list of indices used in pervious is also helpful when we consider a number of user at a time.

To find the maximum is tensor $\boldsymbol{\mathfrak{X}}$, the diamond sampling supply referenced method.

- ***Preprocessing***
  For every i_1, k compute the weight
  $w_{i_1 k} = \mid a^{(1)}_{i_1 k} \mid \parallel \boldsymbol{a}^{(2)}_{*r} \parallel_1 \ldots \parallel \boldsymbol{a}^{(N)}_{*r} \parallel_1$, where $\parallel \boldsymbol{a}^{(n)}_{*r} \parallel_1 = \sum_{h=1}^{R} a^{(n)}_{i_n h}$.
- ***Random walk to v^m_k.***
  Sample S times with the probability $w_{i_1 k} / \parallel \mathbf{W} \parallel_1$

Algorithm 2 Diamond Sampling with factor matrixes

Given factor matrix $\mathbf{A}^{(n)}, n = 1, 2, \ldots, N$.
Let s be the number of samples.

for all $a_{i_1k}^{(1)} \neq 0$ **do**
  $w_{i_1k} \leftarrow \mid a_{i_1k}^{(1)} \mid \parallel \boldsymbol{a}_{i_1*}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{*r}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{*r}^{(2)} \parallel_1 \ldots \parallel \boldsymbol{a}_{*r}^{(N)} \parallel_1$
end for
$\mathcal{X} \leftarrow$ all-zeros tensor of size $L^{(1)} \times L^{(2)} \ldots \times L^{(N)}$
for $\ell = 1, \ldots, s$ **do**
  Sample (i_1, k) with probability $w_{i_1k} / \parallel \mathbf{W} \parallel_1$
  for $n = 2, ..., N$ **do**
    Sample i_n with probability $|a_{i_nk}^{(n)}| / \parallel \boldsymbol{a}_{*r}^{(n)} \parallel_1$
  end for
  Sample k' with probability $|a_{i_1k'}^{(1)}| / \parallel \boldsymbol{a}_{i_1*}^{(1)} \parallel_1$
  $x_{i_1,i_2,\cdots,i_N} \leftarrow x_{i_1,i_2,\cdots,i_N} + sgn(a_{i_1k}^{(1)} \cdot a_{i_2k}^{(2)} \cdots a_{i_Nk}^{(N)} \cdot a_{i_1k'}^{(1)}) a_{i_2k'}^{(2)} \cdots a_{i_Nk'}^{(N)}$
end for
Postprocessing

- Sample each node in parallel layers
  Given k sample $i_2, \ldots, i_N$ with probability $P(e_{ki_n})$, and increase the value of $c_{i_1,i_2,\ldots,i_N}$ by $a_{i_2k}^{(2)} \cdots a_{i_Nk}^{(N)}$
  The expect of $x_{i_1,i_2,\ldots,i_N}$.

$$\begin{aligned}
E[x_{i_1,i_2,\ldots,i_N}] &= \sum_{k=1}^{R} p(i_1, k) p(e_{ki_2}) p(e_{ki_3}) \cdots p(e_{ki_N}) \\
&= \frac{S}{\parallel \mathbf{W} \parallel_1} \sum_{k=1}^{R} a_{i_1k}^{(1)} \cdot a_{i_2k}^{(2)} \cdots a_{i_Nk}^{(N)} \\
&= \frac{S}{\parallel \mathbf{W} \parallel_1} x_{i_1,i_2,\ldots,i_N}
\end{aligned}$$

  So the $c_{i_1,i_2,\ldots,i_N} \frac{S}{\|\mathbf{W}\|_1}$ is an unbiased estimator for $x_{i_2,\ldots,i_N}$.

4.2 Diamond Sampling in Tensor

Diamond sampling is the advanced edition of wedge sampling, which can deal with the negative value.

4.2.1 Find the maximum value in tensor

A direct extension for diamond sampling in tensor is stated in Algorithm 2.

4.2.2 Find the most relevant vector lists of query u

Consider query $\boldsymbol{u}$ is one column of $\mathbf{A}^{(1)}$, and the retrieve tensor

$$\mathcal{X}_u = \left[\!\left[\boldsymbol{u}, \mathbf{A}^{(2)}, \ldots, \mathbf{A}^{(N)} \right]\!\right] = \sum_{r=1}^{R} \boldsymbol{a}_r^{(2)} \circ \cdots \circ \boldsymbol{a}_r^{(N)} \circ u_r$$

We can use the counter to indicate the value, but is will not deal with the negative value, so a degenerate method of diamond sampling was proposed. The way to find the maximum of $x_{u,\boldsymbol{i}}$ may be more concise when sampling starts with $\boldsymbol{u}$ as we can see in line 2 of Algorithm 3.

Algorithm 3 Diamond Sampling with a query vector

Given factor matrix $\mathbf{A}^{(n)} \in R^{L^{(n)} \times R}, n = 2, \dots, N, \boldsymbol{u} \in R^{1 \times R}$.
Let s be the number of samples.

```
for all u_k ≠ 0 do
    w_k ← | u_k ||| u ||_1|| a_{*r}^{(2)} ||_1|| a_{*r}^{(3)} ||_1 ... || a_{*r}^{(N)} ||_1
end for
X_u ← all-zeros tensor of size L^{(2)} × L^{(3)} ... × L^{(N)}
for ℓ = 1, ..., s do
    Sample k with probability w_k/ || w ||_1
    for n = 2, ..., N do
        Sample i_n with probability |a_{i_n k}^{(n)}|/ || a_{*r}^{(n)} ||_1
    end for
    Sample k' with probability | u_{k'} | / || u ||_1
    x_{i_2,i_3,...,i_N} ← x_{i_2,i_3,...,i_N} + sgn(a_{i_2 k}^{(2)} · a_{i_3 k}^{(3)} ··· a_{i_N k}^{(N)} · u_k · u_{k'}) a_{i_2 k'}^{(2)} a_{i_3 k'}^{(3)} ··· a_{i_N k'}^{(N)}
end for
Postprocessing
```

5 Implementation issues

****

5.1 Determined Sample Numbers Online

Pre-sample a number of instance, and it can be used with the accuracy bound to determine a reasonable number of samples.

5.2 Dealing with query

When we want to find the maximus relevant vectors for a number of queries. It is effective to save the the sampled instances into lists $L_1, L_2, \dots, L_d$. When processes a query, sampled the nodes in middle layer. It then retrieves these samples form the stored lists.

6 Experiments

****

---

7 Sampling in Fashion Recommendation

7.1 Structure of U and $A^{(n)}$

Using sampling to solve a fashion recommendation problem.

$$\begin{aligned} r_{u,\mathfrak{O}_t} = & \sum_{n=1}^{N} \boldsymbol{u}^{(n)} \cdot \boldsymbol{h}^{(n,n)}(\boldsymbol{x}_{i_n}) + \\ & \sum_{n=1}^{N} \sum_{m=n+1}^{N} \boldsymbol{h}^{(n,m)}(\boldsymbol{x}_{i_n}) \cdot \boldsymbol{h}^{(m,n)}(\boldsymbol{x}_{i_m}) \end{aligned}$$

Given a user $\boldsymbol{u}^{(n)} \in R^D$, and the latent space vector $\boldsymbol{h}^{(n,m)}(\boldsymbol{x}_{i_n}) \in R^D$, we need to find the maximum value in tensor $r_{u,\mathfrak{O}_t}$, which is the outfit most be like by this user. To do so, we need to reform the vectors into factor matrixes before hand.

We introduce a matrix called $Matrix(\boldsymbol{a}^{(n)}_{i_n*})$ with size $N \times ND$, which is the reshaped matrix of vector $\boldsymbol{a}^{(n)}_{i_n*} \in R^{N^2D}$.

$$Matrix(\boldsymbol{a}^{(n)}_{i_n*}) = \begin{array}{cc} & \begin{array}{cccccc} \phantom{\tfrac{1}{2}\boldsymbol{h}^{(n,1)}(\boldsymbol{x}_{i_n})} & \phantom{\tfrac{1}{2}\boldsymbol{h}^{(n,2)}(\boldsymbol{x}_{i_n})} & & i_n\text{-th column} & & \phantom{\tfrac{1}{2}\boldsymbol{h}^{(n,N)}(\boldsymbol{x}_{i_n})} \end{array} \\ \begin{array}{c} \\ \\ \\ i_n\text{-th row} \\ \\ \\ \end{array} & \begin{pmatrix} \boldsymbol{e} & \boldsymbol{e} & \cdots & \frac{1}{2}\boldsymbol{h}^{(n,1)}(\boldsymbol{x}_{i_n}) & \cdots & \boldsymbol{e} \\ \boldsymbol{e} & \boldsymbol{e} & \cdots & \frac{1}{2}\boldsymbol{h}^{(n,2)}(\boldsymbol{x}_{i_n}) & \cdots & \boldsymbol{e} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{2}\boldsymbol{h}^{(n,1)}(\boldsymbol{x}_{i_n}) & \frac{1}{2}\boldsymbol{h}^{(n,2)}(\boldsymbol{x}_{i_n}) & \cdots & \boldsymbol{h}^{(n,n)}(\boldsymbol{x}_{i_n}) & \cdots & \frac{1}{2}\boldsymbol{h}^{(n,N)}(\boldsymbol{x}_{i_n}) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{e} & \boldsymbol{e} & \cdots & \frac{1}{2}\boldsymbol{h}^{(n,N)}(\boldsymbol{x}_{i_n}) & \cdots & \boldsymbol{e} \end{pmatrix} \end{array} \tag{8}$$

Where $\boldsymbol{x}_{\boldsymbol{i_n}}$ is the i_n-th item in n-th category, $\boldsymbol{h}^{(n,m)}(\boldsymbol{x}_{i_n})$, a row vector, is the function to map the n-th category's items to the latent space for matching with the m-th category's items, $\boldsymbol{e}$ a row vector with all ones. And every vector can be see as a block of this matrix. Use $\boldsymbol{a}^{(n)}_{i_n*}$ to form the factor matrix $\mathbf{A}^{(n)} \in R^{L^{(n)} \times N^2D}$.

And similarly, we define the $Matrix(\mathbf{U})$, as the reshaped matrix of row vector $\boldsymbol{u}$.

$$Matrix(\boldsymbol{u}) = \begin{pmatrix} \boldsymbol{u}^{(1)} & \boldsymbol{e} & \cdots & \boldsymbol{e} \\ \boldsymbol{e} & \boldsymbol{u}^{(2)} & \cdots & \boldsymbol{e} \\ \vdots & \vdots & \vdots & \vdots \\ \boldsymbol{e} & \boldsymbol{e} & \cdots & \boldsymbol{u}^{(N)} \end{pmatrix} \tag{9}$$

With the definition of equation 8 and equation 9, we can conclude that:

$$\begin{aligned} r_{u,\mathfrak{O}_t} = & \sum_{n=1}^{N} \boldsymbol{u}^{(n)} \cdot \boldsymbol{h}^{(n,n)}(\boldsymbol{x}_{i_n}) + \\ & \sum_{n=1}^{N} \sum_{m=n+1}^{N} \boldsymbol{h}^{(n,m)}(\boldsymbol{x}_{i_n}) \cdot \boldsymbol{h}^{(m,n)}(\boldsymbol{x}_{i_m}) \\ = & \sum_{k=1}^{N^2D} u_k a^{(1)}_{i_1k} a^{(2)}_{i_2k} \cdots a^{(N)}_{i_Nk} \\ = & x_{\boldsymbol{i}} \end{aligned}$$

Suppose $k-1 = xND + yD + z$, then $(x+1, y+1)$ represents the block vector's position in $Matrix(\boldsymbol{a}_{i_n *}^{(n)})$ or $Matrix(\boldsymbol{u})$, and $z+1$ represents the element's index in $(x+1, y+1)$-th block. i.e

$$a_{i_n k}^{(n)} = \begin{cases} 1, & x+1 \neq n, y+1 \neq n; \\ (1/2) h_{z+1}^{(n,y+1)}(\boldsymbol{x}_{i_n}), & x+1 = n, y+1 \neq n; \\ (1/2) h_{z+1}^{(n,x+1)}(\boldsymbol{x}_{i_n}), & x+1 \neq n, y+1 = n; \\ h_{z+1}^{(n,n)}(\boldsymbol{x}_{i_n}), & x+1 = y+1 = n. \end{cases} \quad u_k = \begin{cases} 1, & x+1 \neq n \text{ or } y+1 \neq n; \\ u_{z+1}^{(n)}, & x+1 = y+1 = n. \end{cases}$$

Base on the special structure of $\boldsymbol{u}$ and $\mathbf{A}^{(n)}$, the Algorithm 3 can have more concise formation.

7.2 Compute the weight u_k

See the line 2 of Algorithm 3

- $\parallel \mathbf{U} \parallel_1$ is redundant.
- Only two of $\{| u_k |, \parallel \boldsymbol{a}_{*r}^{(1)} \parallel_1, \parallel \boldsymbol{a}_{*r}^{(2)} \parallel_1, \ldots, \parallel \boldsymbol{a}_{*r}^{(N)} \parallel_1\}$ need to be computed. For robustness, the weight w_k is replaced by $| u_k | \parallel \boldsymbol{a}_{*r}^{(1)} \parallel_1 \parallel \boldsymbol{a}_{*r}^{(2)} \parallel_1 \ldots \parallel \boldsymbol{a}_{*r}^{(N)} \parallel_1 / (L^{(1)} \times L^{(2)} \ldots \times L^{(N)})$. In each turn, two components is calculated, and divided by the corresponding length.

7.3 Compute the $x_{i_1, i_2, \cdots, i_N}$

- In each round of line 8 of Algorithm 3, only two indexes of $\{i_1, i_2, ..., i_N\}$ need to be sampled with particular distribution ($\boldsymbol{p}_n = \boldsymbol{a}_{*r}^{(n)} / \parallel \boldsymbol{a}_{*r}^{(n)} \parallel_1$).
- Update values Δx

$$\Delta x = sgn(a_{i_1 k}^{(1)} a_{i_2 k}^{(2)} \cdot a_{i_3 k}^{(3)} \cdots a_{i_N k}^{(N)} \cdot u_k \cdot u_{k'}) a_{i_1 k'}^{(1)} a_{i_2 k'}^{(2)} a_{i_3 k'}^{(3)} \cdots a_{i_N k'}^{(N)}$$

And only four of elements need to be calculated.

- Suppose $k-1 = xND + yD + z$ and $x+1 = y+1 = n$, then only i_n need to be sampled with specific distribution. After sampled i_n, k',we update $x_{i_1, i_2, i_3, \cdots, i_N}$ for $i_m = \{1, \ldots, L^{(m)}\}, m \neq n$

$$x_{i_1, i_2, i_3, \cdots, i_N} \leftarrow x_{i_1, i_2, i_3, \cdots, i_N} + \frac{1}{\prod_{m \neq n} L^{(m)}} \Delta x$$

- Or only i_n, i_m need to be sampled with specific distribution. After sampled i_n, i_m, k', then we update $x_{i_1, i_2, i_3, \cdots, i_N}$ for $i_h = \{1, \ldots, L^{(h)}\}, h \neq m, n$

$$x_{i_1, i_2, i_3, \cdots, i_N} \leftarrow x_{i_1, i_2, i_3, \cdots, i_N} + \frac{1}{\prod_{h \neq m, n} L^{(h)}} \Delta x$$