A Topic Modeling Toolbox Using Belief Propagation

Jia Zeng
School of Computer Science and Technology
Soochow University
Suzhou 215006, China
`j.zeng@ieee.org`

Abstract

Latent Dirichlet allocation (LDA) is an important hierarchical Bayesian model for probabilistic topic modeling, which attracts worldwide interests and touches on many important applications in text mining, computer vision and computational biology. This paper introduces a topic modeling toolbox based on the belief propagation (BP) algorithms referred to as TMBP, which is implemented by MEX C++/Matlab/Octave for either Windows 7 or Linux. When compared with existing topic modeling packages, the novelty of this toolbox lies in the BP algorithms for learning LDA-based topic models. TMBP v1.0 includes BP algorithms for latent Dirichlet allocation (LDA), author-topic models (ATM), relational topic models (RTM), and labeled LDA (LaLDA). This toolbox is an ongoing project and more BP-based algorithms for various topic models will be added in the near future. Interested users may also extend BP algorithms for learning more complicated topic models. The source codes are freely available under the GNU General Public Licence, Version 1.0 at *http://mloss.org/software/*.

Contents

1 Introduction **2**

2 Installation **3**
2.1 Octave 3
2.2 Matlab 3
2.3 Folders and Paths 4

3 Quick Start **4**
3.1 quickstart 4
3.2 demo1 4
3.3 demo2 5
3.4 test 5

4 Data Sets **6**
4.1 Document-word Matrix 6
4.2 Vocabulary 7

4.3 Document-author Matrix . . . 7
4.4 Author Names . . . 8
4.5 Document Citation Matrix . . . 8
4.6 MAT and UCI Text Files . . . 8

5 Functions and Arguments **9**
5.1 Default Arguments . . . 9
5.2 VBtrain, VBpredict, LDAVBtrain and LDAVBpredict . . . 9
5.3 GStrain, GSpredict, FGStrain and FGSpredict . . . 11
5.4 sBPtrain, sBPpredict, aBPtrain and aBPpredict . . . 12
5.5 ssiBPtrain, ssiBPpredict, asiBPtrain and asiBPpredict . . . 13
5.6 sCVB0train, sCVB0predict, aCVB0train and aCVB0predict . . . 14
5.7 RBPtrain and RBPpredict . . . 15
5.8 FBPtrain and FBPpredict . . . 15
5.9 ABPtrain and ABPpredict . . . 16
5.10 sTBPtrain, sTBPpredict, aTBPtrain and aTBPpredict . . . 17
5.11 ATMGStrain, ATMGSpredict, ATMBPtrain and ATMBPpredict . . . 18
5.12 RTMBPtrain and RTMBPpredict . . . 19
5.13 LaLDABPtrain . . . 19
5.14 Other Functions . . . 20

6 Message Passing Algorithms for LDA **21**
6.1 Collapsed Gibbs Sampling (GS) . . . 21
6.2 Loopy Belief Propagation (BP) . . . 22
6.3 Variational Bayes (VB) . . . 22
6.4 Synchronous and Asynchronous Message Passing . . . 23

7 Implementation Details **23**

References **25**

Index **26**

1 Introduction

The past decade has seen rapid development of latent Dirichlet allocation (LDA) [1] for solving topic modeling problems because of its elegant three-layer graphical representation as well as two efficient approximate inference methods such as Variational Bayes (VB) [1] and collapsed Gibbs Sampling (GS) [2]. Both VB and GS have been widely used to learn variants of LDA-based topic models until our recent work [3] reveals that there is yet another learning algorithm for LDA based on loopy belief propagation (BP). The basic idea of BP is inspired by the collapsed GS algorithm, in which the three-layer LDA can be interpreted as being collapsed into a two-layer Markov random field (MRF) represented by a factor graph [4]. The BP algorithm such as the sum-product operates well on the factor graph [5]. Extensive experiments confirm that BP is faster and more accurate than both VB and GS, and thus is a strong candidate for becoming the standard topic modeling algorithm. For example, we show how to learn three typical variants of LDA-based topic models, such as author-topic models (ATM) [6], relational topic models (RTM) [7], and labeled LDA (LaLDA) [8] using BP based on the novel factor graph representations [3]. We have implemented the topic modeling toolbox called TMBP by MEX C++ in the Matlab/Octave interface based on VB, GS and BP algorithms. Compared with other topic modeling packages,[1,2,3,4,5,6,7] the novelty of this toolbox lies in the BP algorithms for topic modeling. This paper introduces how to use this toolbox for basic topic modeling tasks. The source codes are freely available under the GNU General Public Licence.

Section 2 shows how to install TMBP. Section 3 shows several examples on how to use TMBP. Section 4 shows the data structure. Section 5 lists all functions in TMBP. Section 6 discusses the basic algorithms implemented in TMBP. Section 7 shows implementation details based on Section 6.

2 Installation

This toolbox is installed in Octave/Matlab environment. Detailed instructions can be also found in "installation.txt".

2.1 Octave

For Octave, we have tested TMBP toolbox in Windows 7 (64bit) system. We use the Octave and Octave-Forge Windows installer (http://octave.sourceforge.net/). For convenience, we provide `make_octave.m` to compile all MEX files:

```
>> make_octave
```

We place all compiled functions in folder `/toolbox`. After changing the current folder to `/toolbox`, users can use `quickstart.m` to examine if all compiled functions can work properly:

```
>> quickstart
```

2.2 Matlab

For Matlab, we have tested TMBP toolbox in Windows 7 (64bit) and Linux Linux RHEL5 (64bit) systems. We use the Matlab R2010a (64bit) environment. We provide `make_matlab64.m` to compile all MEX files:

```
>> make_Matlab64
```

For the 32bit Matlab environment, we provide `make_matlab32.m` for compilation:

```
>> make_Matlab32
```

We place all compiled functions in folder `/toolbox`. After changing the current folder to `/toolbox`, users can use `quickstart.m` to examine if all compiled functions can work properly:

```
>> quickstart
```

[1] http://www.cs.princeton.edu/~blei/lda-c/index.html
[2] http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm
[3] http://nlp.stanford.edu/software/tmt/tmt-0.3/
[4] http://CRAN.R-project.org/package=lda
[5] http://mallet.cs.umass.edu/
[6] http://www.arbylon.net/projects/
[7] http://CRAN.R-project.org/package=topicmodels

2.3 Folders and Paths

There are four folders: `/common`, `/datasets`, `/source` and `/toolbox`. Folder `/common` contains the common library file used by all functions in this toolbox. Folder `/datasets` contains all data sets stored in "*.mat" or "*.txt" format. Readers can refer to Section 4 for more details on data structures. Folder `/source` contains all C++ MEX source files "*.cpp" in this toolbox. Notice that all compiled functions are output in folder `/toolbox`.

When loading the data sets in Matlab/Octave environment, we need to change the current folder to `/datasets`. When using the functions in this toolbox, we need to change the current folder to `/toolbox`.

3 Quick Start

After changing the current folder to `/toolbox`, we provide three demos, `quickstart.m`, `demo1.m` and `demo2_matlab.m`, on how to use functions in TMBP toolbox. In later sections, we will explain all functions and their arguments.

3.1 quickstart

In the Octave/Matlab environment, the first demo `quickstart.m` confirms if all functions work properly:

```
>> quickstart
```

This script tests all functions with default arguments (see subsection 5.1) in 20 iterations. Users may use the same arguments but with more iterations for specific topic modeling tasks.

3.2 demo1

The second demo `demo1.m` shows a quick running (around 14 seconds) of the synchronous BP algorithm:

```
>> demo1
```

The results (the training perplexity at every 10 iterations, training time and the top five words in ten topics) are printed on the screen:

```
********************
The sBP Algorithm
********************
    Iteration 10 of 500:    1041.620873
    ...
    ...
    Iteration 490 of 500:   741.946849
Elapsed time is 13.246747 seconds.

********************
Top five words in each of ten topics by sBP
********************
design system reasoning case knowledge
model models bayesian data markov
genetic problem search algorithms programming
algorithm learning number function model
learning paper theory knowledge examples
learning control reinforcement paper state
model visual recognition system patterns
research report technical grant university
network neural networks learning input
data decision training algorithm classification
```

3.3 demo2

The third demo `demo2_matlab` performs the five-fold cross-validation using the synchronous BP algorithm:

```
>> demo2_matlab
```

Notice: This demo can be used only in Matlab because the function `cvpartition.m` is unavailable in Octave.

3.4 test

To verify the correctness of computational results, we compare the BP algorithm with two widely-used learning algorithms for LDA including GS [2] and VB [1]. We re-implement GS and VB in TMBP toolbox, and compare the output with other GS[8] and VB[9] implementations. The output results are almost the same using the same random initialization. So, our implementations are consistent with other implementations.

In folder `/toolbox`, we provide `test.m` to verify the results of synchronous BP when compared with GS and VB:

```
>> test
```

The results (the training perplexity at every 10 iterations, training time and the top five words of ten topics) are printed on the screen:

```
********************
The GS Algorithm
********************
    Iteration 10 of 1000:   1078.770457
    ...
    ...
    Iteration 990 of 1000:  790.647638
Elapsed time is 28.9777 seconds.

********************
Top five words in each of ten topics by GS
********************
model visual network recognition neural
learning algorithm model show examples
system paper knowledge learning design
learning reinforcement control robot environment
bayesian belief theory probability revision
genetic problem search algorithms algorithm
model models algorithm data method
learning algorithm training method decision
neural network networks learning input
research report grant university science

********************
The VB Algorithm
********************
    Iteration 10 of 1000:   1032.941456
    ...
    ...
    Iteration 990 of 1000:  886.938053
Elapsed time is 229.6280 seconds.
```

[8] http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm
[9] http://www.cs.princeton.edu/~blei/lda-c/index.html

```
********************
Top five words in each of ten topics by VB
********************
system paper design case reasoning
method algorithm network model networks
problem genetic learning paper system
algorithm learning model algorithms results
learning paper decision problem algorithms
learning algorithm problem paper method
learning network model networks feature
research learning grant paper models
neural networks network learning paper
paper algorithm data bayesian learning

********************
The sBP Algorithm
********************
    Iteration 10 of 1000:   1032.620873
    ...
    ...
    Iteration 990 of 1000:  741.712532
Elapsed time is 25.7835 seconds.

********************
Top five words in each of ten topics by sBP
********************
design system reasoning case knowledge
model bayesian models data markov
genetic problem search algorithms programming
algorithm learning function number model
learning paper theory knowledge examples
learning control reinforcement paper state
model visual recognition system patterns
research report technical grant university
network neural networks learning input
data decision training algorithm classification
```

4 Data Sets

In folder `/datasets`, we provide four publicly available document data sets [3]: 1) BLOG, 2) CORA, 3) MEDLINE and 4) NIPS. Without loss of generality, we will use CORA as an example to introduce the data structure. For CORA, there are five `*.mat` files. We can load these files into Matlab/Octave environment:

```
>> load cora_wd
>> load cora_voc
>> load cora_ad
>> load cora_author
>> load cora_dd
```

4.1 Document-word Matrix

The variable `cora_wd` is a $W \times D$ sparse document-word matrix, where W is the vocabulary size, D is the total number of documents in the corpus, and each nonzero element is the word count $x_{w,d} \neq 0$. We can visualize this sparse matrix by `spy` command:

```
>> spy(cora_wd);
```

Here, we show partial contents in the first document:

```
>> cora_wd(1:10,1)

ans =

     (1,1)          1
     (2,1)          4
     (3,1)          2
     (4,1)          2
     (5,1)          1
     (6,1)          2
     (7,1)          1
     (8,1)          2
     (9,1)          1
    (10,1)          1
```

We may transform this sparse matrix to full matrix and vice versa,

```
>> cora_wd = full(cora_wd);
>> cora_wd = sparse(cora_wd);
```

Notice: All functions process only sparse matrices. Full matrices may cause errors.

4.2 Vocabulary

The variable `cora_voc` contains a W-length cell array for the vocabulary words. Here, we show the first ten words in the vocabulary:

```
>> cora_voc(1:10)

ans =

      'computer'
      'algorithms'
      'discovering'
      'patterns'
      'groups'
      'protein'
      'sequences'
      'based'
      'fitting'
      'parameters'
```

4.3 Document-author Matrix

The variable `cora_ad` contains an $A \times D$ sparse document-author matrix, where A is the total number of unique authors and D is the total number of documents in the corpus. The nonzero element $x_{a,d} = 1$ denotes the author a is associated with the document d. Here, we show authors of the first document:

```
>> cora_ad(:,1)

ans =

(1481,1)          1
(2225,1)          1
```

Two authors with indices 1481 and 2225 are associated with the document 1.

4.4 Author Names

The variable `cora_author` contains an A-length cell array for author names. Here, we show two author names with indices 1481 and 2225:

```
>> cora_author([1481,2225])

ans =

      'M Gribskov'
      'T Bailey'
```

4.5 Document Citation Matrix

The variable `cora_dd` contains a $D \times D$ sparse document citation matrix, where the non-zero element $x_{d,d'} = 1$ denotes two documents d and d' have a citation link. The document 1 has two citation links with documents 389 and 484:

```
>> cora_dd(:,1)

ans =

  (389,1)        1
  (484,1)        1
```

Notice: We consider citations as undirected links, where citing and cited documents are not distinguished in topic modeling.

4.6 MAT and UCI Text Files

In folder `/datasets/utilities`, we provide two functions `MAT2WD` and `UCI2WD` to transform MAT[10] and UCI[11] text files into Matlab/Octave document-word sparse matrices. After changing the current folder to `/toolbox`, we run

```
>> kos_wd = MAT2WD('../datasets/kos_mat.txt');
#Document: 3430
#Vocabulary: 6906
#NNZ: 353160
```

We see that the KOS data set has $D = 3430$ documents and the vocabulary size $W = 6906$. The number of non-zero elements (NNZ) $NNZ = 353160$ in the document-word matrix.

The text file `kos_mat.txt` can be generated using Linux/Unix tool "doc2mat"[12] in folder `/datasets/utilities`. The output shows that there are 3430 documents with 6906 vocabulary words. `NNZ = 353160` denotes that there are 353160 nonzero elements in the document-word matrix. Similarly, we run

```
>> kos_wd = UCI2WD('../datasets/kos_uci.txt');
#Document: 3430
#Vocabulary: 6906
#NNZ: 353160
```

The file `kos_uci.txt` has the standard "bag-of-words" format[13] in UCI repository.

---

[10] http://glaros.dtc.umn.edu/gkhome/files/fs/sw/cluto/doc2mat.html
[11] http://archive.ics.uci.edu/ml/datasets/Bag+of+Words
[12] http://glaros.dtc.umn.edu/gkhome/files/fs/sw/cluto/doc2mat.html
[13] http://archive.ics.uci.edu/ml/datasets/Bag+of+Words

5 Functions and Arguments

In folder `/toolbox`, there are many compiled functions, whose source codes are in folder `/source`. For example, the source codes of functions `sBPtrain` and `sBPpredict` are in folder `/source/lda/bp/bp`. Please refer to `make_matlab64.m`, `make_matlab32.m` or `make_octave.m` to find the corresponding source codes.

TMBP toolbox contains various algorithms for learning LDA including Variational Bayes (VB) [1], collapsed Gibbs Sampling (GS) [2], and Belief Propagation (BP) [3]. Interested users may refer to related papers for theoretical details of these algorithms. Each algorithm often has two functional implementations, one for training and the other for prediction. For BP, there are often two scheduling techniques: synchronous and asynchronous schedules [3]. As a result, some BP algorithms have two slightly different implementations.

5.1 Default Arguments

In `quickstart.m`, we provide default arguments used in almost all functions:

```
ALPHA = 1e-2;
BETA = 1e-2;
N = 20;
SEED = 1;
OUTPUT = 1;
J = 10;
```

1. `ALPHA` and `BETA` are Dirichlet hyperparamters. In real-world applications, the asymmetric prior `ALPHA` may have substantial advantages over the symmetric prior, while the asymmetric prior `BETA` may not. Generally, the hyperparameters determine the sparseness of output multinomial parameters `phi` and `theta` which influence the topic modeling performance. However, for simplicity, we assume that the hyperparameters are symmetric and provided by users as prior knowledge. In many cases, we often use the smoothed LDA with fixed symmetric Dirichlet hyperparameters ALPHA $= 0.01$ and BETA $= 0.01$ [9].
2. `J` is the total number of topics provided by users.
3. `N` is the total number of learning iterations. Generally, `N = 500` is enough to produce a good topic modeling performance.
4. `SEED` is used for initializing random number generation. It should be a positive odd number. For example, `SEED = 1` or `SEED = 3`.
5. `OUTPUT = 0` means no output printed on the screen. `OUTPUT = 1` prints the number of iterations and training or predictive perplexity on the screen.

For all functions, there are three common output parameters:

1. `phi` is a $J \times W$ matrix for the unnormalized multinomial parameters over fixed vocabulary.
2. `theta` is a $J \times D$ matrix for the unnormalized document-specific topic proportions.
3. `mu` is a $J \times NNZ$ matrix for topic distributions over word indices, where NNZ is the total number of non-zero elements in the document-word matrix.

5.2 VBtrain, VBpredict, LDAVBtrain and LDAVBpredict

We re-implement the variational Bayes (VB) algorithm for training LDA [1].[14]

```
[phi, theta, mu] = VBtrain(cora_wd, J, N, M,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, mu] = VBtrain(cora_wd, J, N, M,
ALPHA, BETA, SEED, OUTPUT, muin);
```

[14] `http://www.cs.princeton.edu/~blei/lda-c/index.html`

```
[theta, mu] = VBpredict(cora_wd, phi, N, M,
ALPHA, BETA, SEED, OUTPUT);

[theta, mu] = VBpredict(cora_wd, phi, N, M,
ALPHA, BETA, SEED, OUTPUT, muin);
```

VB is a synchronous variational message passing algorithm [3]. `cora_wd` is the input document-word matrix. The difference between `VBtrain` and `VBpredict` is that `VBpredict` estimates `theta` for unseen test set while holding `phi` fixed. If we do not provide the initialized messages `muin`, both functions will randomly initialize messages `mu`.

Unlike other functions, users need to provide the inner number of iterations `M`. Since VB converges rapidly, `M` is often less than 10 in practice. Other arguments are default in subsection 5.1. Source codes are in folder `/source/lda/vb`.

Below shows the unnormalized variational parameters `phi` and `theta` for the first vocabulary word and the first document, respectively.

```
>> phi(:,1)

ans =

   52.8449
    3.1601
    7.6788
   17.7290
    9.6497
    7.9235
   20.9194
   19.0359
    0.0107
    0.0481

>> theta(:,1)

ans =

    0.0000
    3.4453
    4.5136
   78.8480
    0.0000
    5.1930
    0.0000
    0.0000
    0.0000
    0.0000
```

The normalized variational *message* `mu` for the first word index is

```
>> mu(:,1)

ans =

    0.0000
    0.0055
    0.0181
    0.9507
    0.0000
    0.0257
```

```
    0.0000
    0.0000
    0.0000
    0.0000
```

We also provide two pure Matlab codes for VB: `LDAVBtrain.m` and `LDAVBpredict.m`. Their speed are slower than the above MEX C++ implementations.

```
[phi, theta] = LDAVBtrain(cora_wd, J, N, M, ALPHA, BETA, OUTPUT);

theta = LDAVBpredict(cora_wd, phi, N, M, ALPHA, BETA, OUTPUT);
```

The arguments are the same with the above `VBtrain` and `VBpredict` functions. *Notice:* `SEED` is not required because Matlab has function `rand` for random number generation. Both functions cannot be used in Octave environment because function `psi` is unavailable.

5.3 GStrain, GSpredict, FGStrain and FGSpredict

We re-implement collapsed Gibbs sampling (GS) for LDA [2].[15]

```
[phi, theta, z] = GStrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, z] = GStrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, zin);

[theta, z] = GSpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);

[theta, z] = GSpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, zin);
```

GS is an asynchronous message passing algorithm but with random sampling [3]. `cora_wd` is the input document-word matrix. The difference between `GStrain` and `GSpredict` is that `GSpredict` estimates `theta` for unseen test set while holding `phi` fixed. `z` is a "$1 \times$ Number of tokens" vector for the topic labeling configuration over all word tokens. If we do not provide the initialized topic labeling configuration `zin`, both functions will randomly initialize the topic labeling configuration `z`. Other arguments are default in subsection 5.1. Source codes are in folder `/source/lda/gs`. Because GS works on word tokens, both `phi` and `theta` are sparse integer matrices:

```
>> phi(:,1)

ans =

   (2,1)        21
   (4,1)        26
   (6,1)         8
   (9,1)        41
  (10,1)        43

>> theta(:,1)

ans =

   (1,1)        16
   (2,1)        13
   (5,1)         9
```

[15] http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm

```
(7,1)        38
(10,1)       16
```

Notice: `z` is a vector for discrete topic label assignment over word tokens:

```
>> z(1:10)

ans =

   10     2    10     5     7     7     7     7     1    10
```

Fast Gibbs sampling (FGS) for LDA is an important improvement over GS. It is often faster than GS (maximum 8 times faster) when the number of topics is very large [9] (For example, `J = 900`).[16]

```
[phi, theta, z] = FGStrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, z] = FGStrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, zin);

[theta, z] = FGSpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);

[theta, z] = FGSpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, zin);
```

The arguments are the same with the above `GStrain` and `GSpredict` functions. Source codes are in folder `/source/lda/gs`.

5.4 sBPtrain, sBPpredict, aBPtrain and aBPpredict

The major contribution of TMBP toolbox lies in BP-based algorithms for LDA [3, 10, 11, 12], including BP, simplified BP (siBP), collapsed variational Bayes 0 (CVB0) [13], residual BP (RBP) [14], fast BP (FBP) [11], active BP (ABP) [11] and tiny BP (TBP) [12]. The slight difference between BP and siBP lies in the message update equation. CVB0 resembles BP but passes messages over word tokens rather than word indices. RBP, FBP, ABP and TBP are four important extensions of BP for training LDA.

For each algorithm, there are usually two scheduling schemes called synchronous and asynchronous schedules. For the synchronous schedule, we update parameters `PHI` and `THETA` until all messages have been updated at each iteration. For the asynchronous schedule, we randomly determine the route for message passing, and update parameters `PHI` and `THETA` immediately after the current message updating at each iteration [3]. Users may develop other BP-based algorithms for more complicated topic models based on our source codes `/source/lda/bp`.

The synchronous BP (sBP) algorithm:

```
[phi, theta, mu] = sBPtrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, mu] = sBPtrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, muin);

[theta, mu] = sBPpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);

[theta, mu] = sBPpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

The asynchronous BP (aBP) algorithm:

[16] `http://www.ics.uci.edu/~iporteou/`

```
[phi, theta, mu] = aBPtrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[phi, theta, mu] = aBPtrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

```
[theta, mu] = aBPpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta, mu] = aBPpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

`cora_wd` is the input document-word matrix. If we do not provide initialized messages `muin`, all functions will randomly initialize messages `mu`. Other arguments are default in subsection 5.1.

5.5 ssiBPtrain, ssiBPpredict, asiBPtrain and asiBPpredict

The simplified BP (siBP) algorithm is an extension of the BP algorithm [3].

The synchronous siBP (ssiBP) algorithm:

```
[phi, theta, mu] = ssiBPtrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[phi, theta, mu] = ssiBPtrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

```
[theta, mu] = ssiBPpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta, mu] = ssiBPpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

The asynchronous siBP (asiBP) algorithm:

```
[phi, theta, mu] = asiBPtrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[phi, theta, mu] = asiBPtrain(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

```
[theta, mu] = asiBPpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta, mu] = asiBPpredict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

`cora_wd` is the input document-word matrix. If we do not provide initialized messages `muin`, all functions will randomly initialize messages `mu`. Other arguments are default in subsection 5.1.

The motivation of siBP is for the fast vectorized Matlab codes in functions `LDAssiBPtrain.m` and `LDAssiBPpredict.m`. Interested users may revise these two functions for more complicated topic modeling tasks.

```
[phi, theta] = LDAssiBPtrain(cora_wd, J, N, ALPHA, BETA, OUTPUT);
```

```
theta = LDAssiBPpredict(cora_wd, phi, N, ALPHA, BETA, OUTPUT);
```

`cora_wd` is the input document-word matrix. *Notice:* `SEED` is not required because Matlab has function `rand` for random number generation. The output parameters `phi` and `theta` are normalized multinomial parameters:

```
>> phi(:,1)

ans =

    0.0003
    0.0006
    0.0025
    0.0010
    0.0003
    0.0031
    0.0019
    0.0000
    0.0005
    0.0000

>> theta(:,1)

ans =

    0.0884
    0.0009
    0.0018
    0.4341
    0.0089
    0.1050
    0.0007
    0.2384
    0.0002
    0.1216
```

Other arguments are default in subsection 5.1.

5.6 sCVB0train, sCVB0predict, aCVB0train and aCVB0predict

In a recent review [13], CVB0 performs the best in terms of speed and accuracy among recent inference algorithms. We show its relation with BP [3]. The synchronous CVB0 (sCVB0) algorithm:

```
[phi, theta, mu] = sCVB0train(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, mu] = sCVB0train(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, muin);

[theta, mu] = sCVB0predict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);

[theta, mu] = sCVB0predict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

The asynchronous CVB0 (aCVB0) algorithm:

```
[phi, theta, mu] = aCVB0train(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, mu] = aCVB0train(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, muin);

[theta, mu] = aCVB0predict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta, mu] = aCVB0predict(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

cora_wd is the input document-word matrix. If we do not provide initialized messages muin, all functions will randomly initialize messages mu. Other arguments are default in subsection 5.1. Source codes are available in folder /source/lda/bp/cvb0.

5.7 RBPtrain and RBPpredict

Residual BP (RBP) [11] uses an informed scheduling strategy for asynchronous message passing, which converges significantly faster and more often than synchronous BP (sBP) in subsection 5.4. RBP has two different implementations.

The first is sorting residuals based on the document indices [11]:

```
[phi, theta, mu] = RBPtrain_doc(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, mu] = RBPtrain_doc(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, muin);

[theta, mu] = RBPpredict_doc(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);

[theta, mu] = RBPpredict_doc(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

When the number of documents D is not very large, this implementation is more efficient.

The second is sorting residuals based on the vocabulary indices [11]:

```
[phi, theta, mu] = RBPtrain_voc(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, mu] = RBPtrain_voc(cora_wd, J, N,
ALPHA, BETA, SEED, OUTPUT, muin);

[theta, mu] = RBPpredict_voc(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT);

[theta, mu] = RBPpredict_voc(cora_wd, phi, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

When the number of documents D is very large, for example, $D \geq 1,000,000$, this implementation is more efficient.

cora_wd is the input document-word matrix. If we do not provide initialized messages muin, all functions will randomly initialize messages mu. Other arguments are default in subsection 5.1. Source codes are available in folder /source/lda/bp/residual.

5.8 FBPtrain and FBPpredict

Fast BP [11] searches only a subset of the topic space at each iteration, saving enormous training time when the number of topics is large. It is often significantly faster while achieving a lower predictive perplexity than FGS [9] algorithm in subsection 5.3. FBP is extended from RBP in subsection 5.7. Similarly, it also has two different implementations.

The first is sorting topic residuals based on the document indices:

```
[phi, theta, mu] = FBPtrain_doc(cora_wd, J, LAMBDA, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[phi, theta, mu] = FBPtrain_doc(cora_wd, J, LAMBDA, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

```
[theta, mu] = FBPpredict_doc(cora_wd, phi, LAMBDA, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta, mu] = FBPpredict_doc(cora_wd, phi, LAMBDA, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

When the number of documents D is not very large, this implementation is more efficient.

The second is sorting topic residuals based on the vocabulary indices:

```
[phi, theta, mu] = FBPtrain_voc(cora_wd, J, LAMBDA, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[phi, theta, mu] = FBPtrain_voc(cora_wd, J, LAMBDA, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

```
[theta, mu] = FBPpredict_voc(cora_wd, phi, LAMBDA, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta, mu] = FBPpredict_voc(cora_wd, phi, LAMBDA, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

When the number of documents D is very large, for example, $D \geq 1,000,000$, this implementation is more efficient.

`cora_wd` is the input document-word matrix. If we do not provide initialized messages `muin`, all functions will randomly initialize messages `mu`. The parameter `LAMBDA` $\in (0, 1]$ controls the proportion of topic space to be searched. The smaller `LAMBDA` the faster FBP. When the number of topics `J>=100`, `LAMBDA=0.1` is enough to yield a comparable topic modeling accuracy as FGS in subsection 5.3. Other arguments are default in subsection 5.1. Source codes are available in folder `/source/lda/bp/fast`.

5.9 ABPtrain and ABPpredict

Active BP (ABP) [11] is currently one of the fastest algorithms for training LDA. ABP is extended from FBP in subsection 5.8. To speed up topic modeling, it scans only the subset of the documents in the corpus at each iteration. Similarly, ABP has two different implementations. The first is sorting documents based on residuals:

```
[phi, theta, mu] = ABPtrain_doc(cora_wd, J, TD, TK, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[phi, theta, mu] = ABPtrain_doc(cora_wd, J, TD, TK, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

```
[theta, mu] = ABPpredict_doc(cora_wd, phi, TD, TK, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta, mu] = ABPpredict_doc(cora_wd, phi, TD, TK, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

When the number of documents D is not very large, this implementation is more efficient.

The second is sorting vocabulary words based on residuals:

```
[phi, theta, mu] = ABPtrain_voc(cora_wd, J, TW, TK, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[phi, theta, mu] = ABPtrain_voc(cora_wd, J, TW, TK, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

```
[theta, mu] = ABPpredict_voc(cora_wd, phi, TW, TK, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta, mu] = ABPpredict_voc(cora_wd, phi, TW, TK, N,
ALPHA, BETA, SEED, OUTPUT, muin);
```

When the number of documents D is very large, for example, $D \geq 1,000,000$, this implementation is more efficient.

`cora_wd` is the input document-word matrix. If we do not provide initialized messages `muin`, all functions will randomly initialize messages `mu`. The parameters `TD, TW, TK` $\in (0, 1]$ control the proportion of documents, vocabulary words and topics to be searched. The smaller `TD, TW, TK` the faster ABP. When the number of topics `J>=100`, `TD=TW=TK=0.2` is enough to yield a comparable topic modeling accuracy as FGS [9] in subsection 5.3. Other arguments are default in subsection 5.1. Source codes are available in folder `/source/lda/bp/active`.

5.10 sTBPtrain, sTBPpredict, aTBPtrain and aTBPpredict

For massive data sets, existing topic modeling algorithms often consume huge memory, which is often unavailable for a common desktop computer. To save memory, tiny BP (TBP) [12] does not save messages during computation, and thus saving enormous memory usage.

The synchronous TBP (sTBP) algorithm:

```
[phi, theta] = sTBPtrain(cora_wd, J, N, ALPHA, BETA, SEED, OUTPUT);
```

```
[theta] = sTBPpredict(cora_wd, phi, N, ALPHA, BETA, SEED, OUTPUT);
```

The asynchronous TBP (aTBP) algorithm:

```
[phi, theta] = aTBPtrain(cora_wd, J, N, ALPHA, BETA, SEED, OUTPUT);
```

```
[theta] = aTBPpredict(cora_wd, phi, N, ALPHA, BETA, SEED, OUTPUT);
```

`cora_wd` is the input document-word matrix. Other arguments are default in subsection 5.1. Source codes are available in folder `/source/lda/bp/tiny`.

When massive data sets cannot fit into the computer memory, we may load the part of the document file as blocks from hard disk into memory [15].

The synchronous TBP for file (`sTBP_file`):

```
[phi, theta] = sTBPtrain_file(filename, J, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta] = sTBPpredict_file(filename, phi, N,
ALPHA, BETA, SEED, OUTPUT);
```

The aynchronous TBP for file (`aTBP_file`):

```
[phi, theta] = aTBPtrain_file(filename, J, N,
ALPHA, BETA, SEED, OUTPUT);
```

```
[theta] = aTBPpredict_file(filename, phi, N,
ALPHA, BETA, SEED, OUTPUT);
```

`filename` is the file name with the MAT format in subsection 4.6. As an example, we use the file `/datasets/kos_mat.txt` in `quickstart.m`. Other arguments are default in subsection 5.1. Source codes are available in folder `/source/lda/bp/tiny`.

Notice: Because we load partial document file as blocks from hard disk into memory, `sTBP_file` and `aTBP_file` are often slower than the corresponding `sTBP` and `aTBP`.

5.11 ATMGStrain, ATMGSpredict, ATMBPtrain and ATMBPpredict

Unlike LDA, ATM [6] treats each author rather than document as a mixture of topics. In this way, the multinomial parameter `theta` is a $J \times A$ matrix, where A is the total number unique authors. We implement GS and synchronous BP algorithms for learning ATM in this toolbox.

We re-implement the GS algorithm for ATM (ATMGStrain and ATMGSpredict)[17]:

```
[phi, theta, z, x] = ATMGStrain(cora_wd, cora_ad, J, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, z, x] = ATMGStrain(cora_wd, cora_ad, J, N,
ALPHA, BETA, SEED, OUTPUT, zin, xin);

[theta, z, x] = ATMGSpredict(cora_wd, cora_ad, phi, N,
ALPHA, BETA, SEED, OUTPUT);

[theta, z, x] = ATMGSpredict(cora_wd, cora_ad, phi, N,
ALPHA, BETA, SEED, OUTPUT, zin, xin);
```

`cora_wd` is the input document-word matrix. `cora_ad` is the input document-author matrix. Other arguments are default in subsection 5.1. `z` and `zin` are the same as in subsection 5.3. Source codes are available in folder `/source/atm/gs`. The output `x` is a $1 \times NNZ$ vector for the author labeling configuration over word tokens. The input `xin` is the user-defined initialization of this configuration. Below shows the author labels for the first five word tokens:

```
x(1:5)

ans =

        2225        2225        2225        2225        2225
```

It means the first five word tokens are generated by author with index 2225.

```
>> cora_author(2225)

ans =

    'T Bailey'
```

The synchronous BP algorithms for ATM (ATMBPtrain and ATMBPpredict) can be viewed as a soft version of the GS algorithm:

```
[phi, theta, mu, x] = ATMBPtrain(cora_wd, cora_ad, J, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, mu, x] = ATMBPtrain(cora_wd, cora_ad, J, N,
ALPHA, BETA, SEED, OUTPUT, muin, xin);

[theta, mu, x] = ATMBPpredict(cora_wd, cora_ad, phi, N,
ALPHA, BETA, SEED, OUTPUT);

[theta, mu, x] = ATMBPpredict(cora_wd, cora_ad, phi, N,
ALPHA, BETA, SEED, OUTPUT, muin, xin);
```

`cora_wd` is the input document-word matrix. `cora_ad` is the input document-author matrix. `z` and `zin` are the same as in subsection 5.3. The other arguments are set to the default values listed

[17] `http://psiexp.ss.uci.edu/research/programs_data/toolbox.htm`

in 5.1. The output `x` is a $NMAX \times NNZ$ matrix, where $NMAX$ is the maximum number of co-authors per document. In this way, we assume that all co-authors contribute to generate word index with different probabilities. Below shows the first two authors' contributions for the first five word indices:

```
x(1:2,1:5)

ans =

    0.5239    0.4868    0.4139    0.5140    0.4727
    0.4761    0.5132    0.5861    0.4860    0.5273
```

5.12 RTMBPtrain and RTMBPpredict

RTM [7] models citation links in document networks based on LDA. The GS algorithm for RTM can be found in the package lda.[18] In this toolbox, we implement only the synchronous BP algorithm for RTM (RTMBPtrain and RTMBPpredict):

```
[phi, theta, gamma, mu] = RTMBPtrain(cora_wd, cora_dd, J, N, OMEGA,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, gamma, mu] = RTMBPtrain(cora_wd, cora_dd, J, N, OMEGA,
ALPHA, BETA, SEED, OUTPUT, muin);

[theta, mu] = RTMBPpredict(cora_wd, cora_dd, phi, gamma, N, OMEGA,
ALPHA, BETA, SEED, OUTPUT);

[theta, mu] = RTMBPpredict(cora_wd, cora_dd, phi, gamma, N, OMEGA,
ALPHA, BETA, SEED, OUTPUT, muin);
```

`cora_wd` is the input document-word matrix. `cora_dd` is the input document-document citation matrix. OMEGA $\in [0, 1]$ is a balancing weight for relational topic models (RTM) [3], which balances messages from document contents and document links. When OMEGA $= 0$, RTM reduces to the standard LDA. When OMEGA $= 1$, RTM uses only link information for topic modeling. The other arguments are set to the default values listed in 5.1. The source code is in folder `/source/rtm/bp`.

The output `gamma` is a $J \times J$ matrix for topic dependencies over links. Below shows such dependencies of the first five topics:

```
>> gamma(1:5,1:5)

ans =

    0.1077    0.0991    0.0995    0.0930    0.0961
    0.1150    0.1160    0.1123    0.1055    0.1123
    0.1128    0.1096    0.1115    0.1096    0.1080
    0.0981    0.0958    0.1021    0.1121    0.1045
    0.1015    0.1022    0.1006    0.1047    0.1058
```

We see that the document has a higher likelihood to cite other documents with the same topic.

5.13 LaLDABPtrain

LaLDA [8] is a supervised topic model for multi-label classification based on LDA. The GS learning algorithm for LaLDA can be found in the Stanford Topic Modeling Toolbox. [19] We provide only the synchronous BP algorithm for learning LaLDA.

[18] `http://CRAN.R-project.org/package=lda`

[19] `http://nlp.stanford.edu/software/tmt/tmt-0.3/`

```
[phi, theta, z] = LaLDABPtrain(cora_wd, cora_ad, N,
ALPHA, BETA, SEED, OUTPUT);

[phi, theta, z] = LaLDABPtrain(cora_wd, cora_ad, N,
ALPHA, BETA, SEED, OUTPUT, zin);
```

`cora_wd` is the input document-word matrix. We use the document-author matrix `cora_ad` as class labels for each document. The other arguments are set to the default values listed in 5.1. `z` is a $1 \times$ NNZ vector for the topic labeling configuration over all word indices. If we do not provide the initialized topic labeling configuration `zin`, both functions will randomly initialize the topic labeling configuration `z`. Generally, each document will have multiple class labels. For the unseen test set, we do not know class labels for each document, so that all labels in the training set should be considered. In this case, we can simply infer `theta` using `sBPpredict` in subsection 5.4. The output `z` is the class labeling assignment over word indices:

```
>> z(1:5)

ans =

        1481        2225        1481        1481        1481
```

We also provide a pure Matlab code `LaLDAssiBPtrain.m` for labeled LDA:

```
[phi, theta] = LaLDAssiBPtrain(cora_wd, cora_ad, N,
ALPHA, BETA, OUTPUT);
```

`cora_wd` is the document-word sparse matrix. `cora_ad` is the document-label sparse matrix (We assume that author names are labels). *Notice:* When the total number of labels is large, this function becomes very slow due to matrix multiplications. The source code is available in folder `/source/lalda/bp`.

5.14 Other Functions

We also provide two functions `topicshow.m` and `perplexity.m`. The former shows the top N words in each topic, and the latter calculates the training and predictive perplexity values [13, 3], which are widely-used performance measures for topic modeling.

```
>> topicshow(phi, cora_voc, 5)
network model neural networks data
learning algorithm problem model algorithms
paper learning control planning state
learning paper system neural networks
bayesian algorithm show decision paper
learning problem algorithm search results
model models data bayesian algorithms
learning algorithm algorithms results decision
neural network networks algorithm algorithms
design problem system genetic research
```

The first input is the multinomial parameter `phi` generated by training algorithms, the second input is the vocabulary `cora_voc`, and the third input is the top $N = 5$ words in each topic.

```
>> perplexity(cora_wd, phi, theta, ALPHA, BETA)

ans =

  1.0020e+003
```

The input `cora_wd` is the document-word matrix, `phi` and `theta` are multinomial parameters generated by prediction algorithms, and `ALPHA` and `BETA` are Dirichlet hyperparameters.

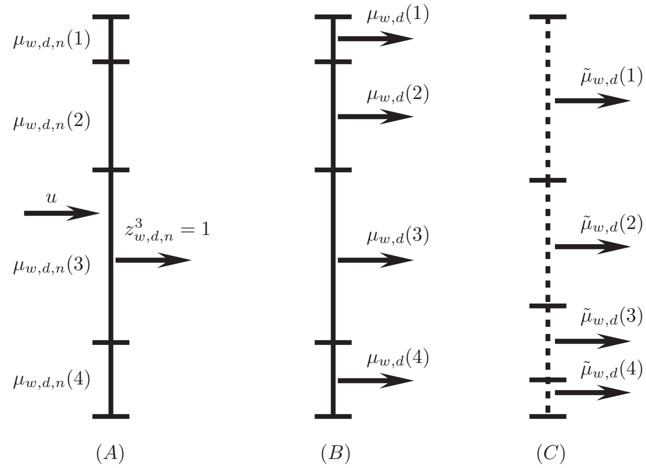

Figure 1: Message passing for training LDA: (A) collapsed Gibbs sampling (GS), (B) loopy belief propagation (BP), and (C) variational Bayes (VB).

6 Message Passing Algorithms for LDA

LDA allocates a set of semantic topic labels, $\mathbf{z} = \{z^k_{w,d}\}$, to explain non-zero elements in the document-word co-occurrence matrix $\mathbf{x}_{W\times D} = \{x_{w,d}\}$, where $1 \le w \le W$ denotes the word index in the vocabulary, $1 \le d \le D$ denotes the document index in the corpus, and $1 \le k \le K$ denotes the topic index. Usually, the number of topics K is provided by users. The topic label satisfies $z^k_{w,d} = \{0,1\}, \sum_{k=1}^K z^k_{w,d} = 1$. After inferring the topic labeling configuration over the document-word matrix, LDA estimates two matrices of multinomial parameters: topic distributions over the fixed vocabulary $\boldsymbol{\phi}_{W\times K} = \{\phi_{\cdot,k}\}$, where $\theta_{\cdot,d}$ is a K-tuple vector and $\phi_{\cdot,k}$ is a W-tuple vector, satisfying $\sum_k \theta_{k,d} = 1$ and $\sum_w \phi_{w,k} = 1$. From a document-specific proportion $\theta_{\cdot,d}$, LDA independently generates a topic label $z^k_{\cdot,d} = 1$, which further combines $\phi_{\cdot,k}$ to generate a word index w, forming the total number of observed word counts $x_{w,d}$. Both multinomial vectors $\theta_{\cdot,d}$ and $\phi_{\cdot,k}$ are generated by two Dirichlet distributions with hyperparameters α and β. For simplicity, we consider the smoothed LDA with fixed symmetric hyperparameters provided by users [2]. To illustrate the generative process, we refer the readers to the original three-layer graphical representation for LDA [1] and the two-layer factor graph for the collapsed LDA [3].

6.1 Collapsed Gibbs Sampling (GS)

After integrating out the multinomial parameters $\{\phi, \theta\}$, LDA becomes the collapsed LDA in the collapsed hidden variable space $\{\mathbf{z}, \alpha, \beta\}$. GS [2] is a Markov Chain Monte Carlo (MCMC) sampling technique to infer the marginal distribution or *message*, $\mu_{w,d,n}(k) = p(z^k_{w,d,n} = 1)$, where $1 \le n \le x_{w,d}$ is the word token index. The message update equation is

$$\mu_{w,d,n}(k) \propto \frac{\mathbf{z}^k_{\cdot,d,-n} + \alpha}{\sum_k [\mathbf{z}^k_{\cdot,d,-n} + \alpha]} \times \frac{\mathbf{z}^k_{w,\cdot,-n} + \beta}{\sum_w [\mathbf{z}^k_{w,\cdot,-n} + \beta]}, \tag{1}$$

where $\mathbf{z}^k_{\cdot,d,-n} = \sum_w z^k_{w,d,-n}$, $\mathbf{z}^k_{w,\cdot,-n} = \sum_d z^k_{w,d,-n}$, and the notation $-n$ denotes excluding the current topic label $z^k_{w,d,n}$. After normalizing the message $\sum_k \mu_{w,d,n}(k) = 1$, GS draws a random number $u \sim \text{Uniform}[0,1]$ and checks which topic segment will be hit as shown in Fig. 1A, where $K = 4$ for example. If the topic index $k = 3$ is hit, then we assign $z^3_{w,d,n} = 1$. The sampled topic label will be used immediately to estimate the message for the next word token. If we view the sampled topic labels as *particles*, GS can be interpreted as a special case of non-parametric belief propagation [16], in which only particles rather than complete messages are updated and passed at each iteration. Eq. (1) sweeps all word tokens for training iterations $1 \le t \le T$ until the convergence criterion is satisfied. Based on inferred topic configuration $z^k_{w,d,n}$ over word tokens, the multinomial parameters can be estimated as follows,

$$\phi_{w,k} = \frac{\mathbf{z}^k_{w,\cdot,\cdot} + \beta}{\sum_w [\mathbf{z}^k_{w,\cdot,\cdot} + \beta]}, \tag{2}$$

$$\theta_{k,d} = \frac{\mathbf{z}^k_{\cdot,d,\cdot} + \alpha}{\sum_k [\mathbf{z}^k_{\cdot,d,\cdot} + \alpha]}. \tag{3}$$

These equations look similar to Eq. (1) except including the current topic label $z^k_{w,d,n}$ in both numerator and denominator.

6.2 Loopy Belief Propagation (BP)

Similar to GS, BP [3] performs in the collapsed hidden variable space of LDA called collapsed LDA. The basic idea is to integrate out the multinomial parameters $\{\theta, \phi\}$, and infer the marginal posterior probability in the collapsed space $\{\mathbf{z}, \alpha, \beta\}$. The collapsed LDA can be represented by a factor graph, which facilitates the BP algorithm for approximate inference and parameter estimation. Unlike GS, BP infers messages, $\mu_{w,d}(k) = p(z^k_{w,d} = 1)$, without sampling in order to keep all uncertainties of messages. The message update equation is

$$\mu_{w,d}(k) \propto \frac{\boldsymbol{\mu}_{-w,d}(k) + \alpha}{\sum_k [\boldsymbol{\mu}_{-w,d}(k) + \alpha]} \times \frac{\boldsymbol{\mu}_{w,-d}(k) + \beta}{\sum_w [\boldsymbol{\mu}_{w,-d}(k) + \beta]}, \tag{4}$$

where $\boldsymbol{\mu}_{-w,d}(k) = \sum_{-w} x_{-w,d}\mu_{-w,d}(k)$ and $\boldsymbol{\mu}_{w,-d}(k) = \sum_{-d} x_{w,-d}\mu_{w,-d}(k)$. The notation $-w$ and $-d$ denote all word indices except w and all document indices except d. After normalizing $\sum_k \mu_{w,d}(k) = 1$, BP updates other messages iteratively. Fig. 1B illustrates the message passing in BP when $K = 4$, slightly different from GS in Fig. 1A. Eq. (4) differs from Eq. (1) in two aspects. First, BP infers messages based on word indices rather than word tokens. Second, BP updates and passes complete messages without sampling. In this sense, BP can be viewed as a *soft* version of GS. Obviously, such differences give Eq. (4) two advantages over Eq. (1). First, it keeps all uncertainties of messages for high topic modeling accuracy. Second, it scans a total of NNZ word indices for message passing, which is significantly less than the total number of word tokens $\sum_{w,d} x_{w,d}$ in $\mathbf{x}$. So, BP is often faster than GS by scanning a significantly less number of elements ($NNZ \ll \sum_{w,d} x_{w,d}$) at each training iteration [3]. Eq. (4) scans NNZ in the document-word matrix for training iterations $1 \le t \le T$ until the convergence criterion is satisfied. Based on the normalized messages, the multinomial parameters can be estimated by

$$\phi_{w,k} = \frac{\boldsymbol{\mu}_{w,\cdot}(k) + \beta}{\sum_w [\boldsymbol{\mu}_{w,\cdot}(k) + \beta]}, \tag{5}$$

$$\theta_{k,d} = \frac{\boldsymbol{\mu}_{\cdot,d}(k) + \alpha}{\sum_k [\boldsymbol{\mu}_{\cdot,d}(k) + \alpha]}. \tag{6}$$

These equations look similar to Eq. (4) except including the current message $\mu_{w,d}(k)$ in both numerator and denominator.

6.3 Variational Bayes (VB)

Unlike BP in the collapsed space, VB [1, 17] passes variational messages, $\tilde{\mu}_{w,d}(k) = \tilde{p}(z^k_{w,d} = 1)$, derived from the approximate variational distribution $\tilde{p}$ to the true joint distribution p by minimizing the KL divergence, $KL(\tilde{p}||p)$. The variational message update equation is

$$\tilde{\mu}_{w,d}(k) \propto \frac{\exp[\Psi(\tilde{\boldsymbol{\mu}}_{\cdot,d}(k) + \alpha)]}{\exp[\Psi(\sum_k [\tilde{\boldsymbol{\mu}}_{\cdot,d}(k) + \alpha])]} \times \frac{\tilde{\boldsymbol{\mu}}_{w,\cdot}(k) + \beta}{\sum_w [\tilde{\boldsymbol{\mu}}_{w,\cdot}(k) + \beta]}, \tag{7}$$

where $\tilde{\boldsymbol{\mu}}_{\cdot,d}(k) = \sum_w x_{w,d}\tilde{\mu}_{w,d}(k)$, $\tilde{\boldsymbol{\mu}}_{w,\cdot}(k) = \sum_d x_{w,d}\tilde{\mu}_{w,d}(k)$, and the notation $\exp$ and Ψ are exponential and digamma functions, respectively. After normalizing the variational message $\sum_k \tilde{\mu}_{w,d}(k) = 1$, VB passes this message to update other messages. There are two major differences between Eq. (7) and Eq. (4). First, Eq. (7) involves computationally expensive digamma functions. Second, it include the current variational message $\tilde{\mu}_{w,d}$ in the update equation. The digamma function significantly slows down VB, and also introduces bias in message passing [13, 3]. Fig. 1C shows the variational message passing in VB, where the dashed line illustrates that the variational message is derived from the variational distribution. Based on the normalized variational messages, VB estimates the multinomial parameters as

$$\phi_{w,k} = \frac{\tilde{\boldsymbol{\mu}}_{w,\cdot}(k) + \beta}{\sum_w [\tilde{\boldsymbol{\mu}}_{w,\cdot}(k) + \beta]}, \tag{8}$$

$$\theta_{k,d} = \frac{\tilde{\boldsymbol{\mu}}_{\cdot,d}(k) + \alpha}{\sum_k [\tilde{\boldsymbol{\mu}}_{\cdot,d}(k) + \alpha]}. \tag{9}$$

These equations are almost the same as Eqs. (5) and (6) but using variational messages.

6.4 Synchronous and Asynchronous Message Passing

Message passing algorithms for LDA first randomly initialize messages, and then pass messages according to two schedules: the synchronous and the asynchronous update schedules [18]. The synchronous message passing schedule uses all messages at training iteration $t-1$ to update current messages at training iteration t, while the asynchronous schedule immediately uses the updated messages to update other remaining messages within the same training iteration t. Empirical results demonstrate that the asynchronous schedule is slightly more efficient than the synchronous schedule [3] for topic modeling. However, the synchronous schedule is much easier to extend for parallel computation.

GS is naturally an asynchronous message passing algorithm. The sampled topic label will immediately influence the topic sampling process at the next word token. Both synchronous and asynchronous schedules of BP work equally well in terms of topic modeling accuracy, but the asynchronous schedule converges slightly faster than the synchronous one [19]. VB is a synchronous variational message passing algorithm, updating messages at iteration t using messages at iteration $t-1$.

7 Implementation Details

Most topic modeling algorithms can be formulated within the unified message passing framework in section 6. So, in folder `/source`, most codes can be extended from BP codes in subsection 5.4. We shall use `sBPtrain.cpp` in folder `/source/lda/bp/bp` as an example to show the basic structure of programs.

First, there is a main function in Matlab/Octave MEX format:

```
/* main function */
void mexFunction(int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[])
```

In the main function, we load input data `cora_wd` and all arguments provided by users including `J, N, ALPHA, BETA, SEED, OUTPUT` in subsection 5.1. Also, we output two parameters `phi` and `theta` in subsection 5.1.

Second, there is a specific function for the synchronous BP algorithm:

```
/* run the learning algorithm */
sBP(ALPHA, BETA, W, J, D, NN, OUTPUT,
sr, ir, jc, phi, theta, mu, startcond);
```

Notice: Only sparse matrices can be processed. In `sBP` function, after random initializing messages and parameters, the major program for message passing is as follows,

```
/* passing message mu */
for (di=0; di<D; di++) {
    for (i=jc[di]; i<jc[di + 1]; i++) {
        wi = (int) ir[i];
        xi = sr[i];
```

```
        mutot = 0;
        for (j=0; j<J; j++) {
            mu[i*J + j] = (phi[wi*J + j] - xi*mu[i*J + j] + BETA)/
            (phitot[j] - xi*mu[i*J + j] + WBETA)*
            (theta[di*J + j] - xi*mu[i*J + j] + ALPHA);
            mutot += mu[i*J + j];
        }
        for (j=0; j<J; j++) {
            mu[i*J + j] /= mutot;
        }
    }
}
```

The message passing process iterates all non-zero elements in the document-word matrix. In the meanwhile, each message is locally normalized to $[0, 1]$. Usually, in our view, implementations of different topic models mainly differ in the following message updating part as shown in Eq. (4):

```
mu[i*J + j] = (phi[wi*J + j] - xi*mu[i*J + j] + BETA)/
(phitot[j] - xi*mu[i*J + j] + WBETA)*
(theta[di*J + j] - xi*mu[i*J + j] + ALPHA);
```

Interested users may focus on revising the above code for more complicated topic models. Here, we compare with the message passing codes in `ATMBPtrain.cpp` (author-topic model [6] in folder `/source/atm/bp`):

```
/* message passing */
for (di=0; di<D; di++) {
    for (i=jcwd[di]; i<jcwd[di + 1]; i++) {
        wi = (int) irwd[i]; // current word index
        xi = srwd[i]; // current word counts
        // message
        for (a=0; a<(jcad[di + 1]-jcad[di]); a++) xprob[a] = 0;
        for (j=0; j<J; j++) zprob[j] = 0;
        totprob = 0;
        for (a=0; a<(jcad[di + 1]-jcad[di]); a++) {
            ai = (int) irad[jcad[di] + a];
            for (j=0; j<J; j++) {
                probs = (phi[wi*J + j] - xi*muz[i*J + j] + BETA)/
                    (phitot[j] - xi*muz[i*J + j] + WBETA) *
                    (theta[ai*J + j] - xi*muz[i*J + j] *
                    mux[i*MA + a] + ALPHA)/
                    (thetad[ai] - xi*mux[i*MA + a] + JALPHA);
                xprob[a] += probs;
                zprob[j] += probs;
                totprob += probs;
            }
        }
        for (a=0; a<(jcad[di + 1]-jcad[di]); a++) {
            mux[i*MA + a] = xprob[a]/totprob;
        }
        for (j=0; j<J; j++) {
            muz[i*J + j] = zprob[j]/totprob;
        }
    }
}
```

We see that most codes resemble the message passing codes in `sBPtrain.cpp`. The major difference lies in the message updating part like Eq. (4), where the author information is incorporated in ATM as follows:

```
probs = (phi[wi*J + j] - xi*muz[i*J + j] + BETA)/
```

```
(phitot[j] - xi*muz[i*J + j] + WBETA) *
(theta[ai*J + j] - xi*muz[i*J + j]*mux[i*MA + a] + ALPHA)/
(thetad[ai] - xi*mux[i*MA + a] + JALPHA);
```

So, if users want to develop and implement message passing algorithms for new topic models, it is a good choice to revise the message updating part from Eq. (4) to (6) in source codes `sBPtrain.cpp` or `aBPtrain.cpp` in folder `/source/lda/bp/bp`.

References

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003. 2, 5, 9, 21, 22

[2] T. L. Griffiths and M. Steyvers, "Finding scientific topics," *Proc. Natl. Acad. Sci.*, vol. 101, pp. 5228–5235, 2004. 2, 5, 9, 11, 21

[3] J. Zeng, W. K. Cheung, and J. Liu, "Learning topic models by belief propagation," *arXiv:1109.3437v4 [cs.LG]*, 2011. 2, 3, 6, 9, 10, 11, 12, 13, 14, 19, 20, 21, 22, 23

[4] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on Inform. Theory*, vol. 47, no. 2, pp. 498–519, 2001. 2

[5] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006. 2

[6] M. Rosen-Zvi, T. Griffiths, M. Steyvers, and P. Smyth, "The author-topic model for authors and documents," in *UAI*, 2004, pp. 487–494. 3, 18, 24

[7] J. Chang and D. M. Blei, "Hierarchical relational models for document networks," *Annals of Applied Statistics*, vol. 4, no. 1, pp. 124–150, 2010. 3, 19

[8] D. Ramage, D. Hall, R. Nallapati, and C. D. Manning, "Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora," in *Empirical Methods in Natural Language Processing*, 2009, pp. 248–256. 3, 19

[9] I. Porteous, D. Newman, A. Ihler, A. Asuncion, P. Smyth, and M. Welling, "Fast collapsed Gibbs sampling for latent Dirichlet allocation," in *KDD*, 2008, pp. 569–577. 9, 12, 15, 17

[10] J. Zeng, "A topic modeling toolbox using belief propagation," *arXiv:1201.0838v1 [cs.LG]*, 2012. 12

[11] J. Zeng, Z.-Q. Liu, and X.-Q. Cao, "A new approch to speeding up topic modeling," *arXiv:1204.0170v1 [cs.LG]*, 2012. 12, 15, 16

[12] ——, "Memory-efficient topic modeling," *arXiv:1206.1147v1 [cs.LG]*, 2012. 12, 17

[13] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh, "On smoothing and inference for topic models," in *UAI*, 2009, pp. 27–34. 12, 14, 20, 23

[14] J. Zeng, X.-Q. Cao, and Z.-Q. Liu, "Residual belief propagation for topic modeling," *arXiv:1204.6610v1 [cs.LG]*, 2012. 12

[15] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin, "Large linear classification when data cannot fit in memory," in *KDD*, 2010, pp. 833–842. 17

[16] E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky, "Nonparametric belief propagation," in *CVPR*, 2003, pp. 605–612. 21

[17] J. Winn and C. M. Bishop, "Variational message passing," *J. Mach. Learn. Res.*, vol. 6, pp. 661–694, 2005. 22

[18] M. F. Tappen and W. T. Freeman, "Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters," in *ICCV*, 2003, pp. 900–907. 23

[19] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: Informed scheduling for asynchronous message passing," in *UAI*, 2006, pp. 165–173. 23

Index

ABP, 12
ABPpredict, 16
aBPpredict, 12
ABPtrain, 16
aBPtrain, 12
asiBPpredict, 13
asiBPtrain, 13
asynchronous, 12, 23
aTBPpredict, 17
aTBPtrain, 17
ATM, 3, 18, 24
ATMBPpredict, 18
ATMBPtrain, 18
ATMGSpredict, 18
ATMGStrain, 18

BP, 2, 22

data structure, 4, 6
demo, 4, 5

factor graph, 2
FBP, 12
FBPpredict, 15
FBPtrain, 15
FGSpredict, 11
FGStrain, 11

GS, 2, 21
GSpredict, 11
GStrain, 11

LaLDA, 3
LaLDABPtrain, 19
LDA, 2, 5, 9, 11, 12, 16, 18, 19, 21
LDAVBpredict, 9
LDAVBtrain, 9

Matlab/Octave, 3
MRF, 2

quick start, 4

RBP, 12
RBPpredict, 15
RBPtrain, 15
RTM, 3, 19
RTMBPpredict, 19
RTMBPtrain, 19

sBPpredict, 12
sBPtrain, 12
ssiBPpredict, 13
ssiBPtrain, 13
sTBPpredict, 17
sTBPtrain, 17
synchronous, 12, 23

TBP, 12
test, 5
TMBP, 3–5, 9, 12

VB, 2, 9, 22
VBpredict, 9
VBtrain, 9