CS 476/676, Spring 2015
Problem Set #3: Posterior Inference
Version 1.0
Due by **11:59pm** on **Tuesday, April 28**

Contents

1 Instructions

1.1 What You are Given

We have provided text files contained in `hw3-files.zip` which contain the input files you will use.

1.2 What to Hand In

All of your submission files should be handed in as a single archive named `hw3-username.zip`, where `username` is replaced with your JHED ID. This zip archive will include:

- The code you write, your program as an executable, and various input files that you create. Instructions for what to hand in are given under the section headings titled **Deliverables**.
- Written answers to questions should go in a single PDF document named `writeup.pdf`. Questions that require a written response are given under the section headings **Analytical Questions** or **Empirical Questions**. We recommend using LaTeX to typeset your writeup.

1.3 Submission Policies

Please note the following:

- **Collaboration:** Please work in groups of size 2, 3 (preferred), or 4 people. The homeworks are a way for you to work through the material you're learning in this class on your own. But, by working in a group, and debugging each other's solutions, you'll have a chance to learn the material in more depth. The recommended format for tackling these problem sets is the following. Write a high level sketch of the solution for all of the problems on your own. Meet as a group to brainstorm your solutions and converge on a solution as a group. It is important that you have a good understanding of how you'd have approached the problem independently before discussing your solution with the other group members. Developing this intuition will serve you well in the final exam where you will be required to work on your own. Pursuant to your group meeting, write up the solutions on your own. Thereafter, meet as a group to clean up and submit a final write up as a group. By now, each of you should have a solid understand of the concepts involved, and by meeting as a group, you've had a chance to see common ways in which one can make mistakes. Submit your final solution as a final writeup for the group. Your submission should include the names of every team member. Also, name your file as `hw3-username1-username2-username3.zip`.
- **Late Submissions:** We allow each student to use up to 3 late days over the semester. You have late days, not late hours. This means that if your submission is late by any amount of time past the deadline, then this will use up a late day. If it is late by any amount beyond 24 hours past the deadline, then this will use a second late. **If you jointly submit an assignment as a team, then every team member will lose late days if the assignment is submitted late.** If you collaborate with team members but independently submit your own version, then late days will only apply to you.

1.4 How to Hand In

As with the previous homework, one person per group should submit the assignment on Piazza. A private note to the instructors should be submitted with the subject *Submission 3 from* `<list`

`of names of team members>` with the .zip submission as an attachment. The note should be submitted to the `submission3` folder.

2 Variational Inference on a Simple Network [40 points]

In this question, we will explore the difference between mean-field and structured mean-field variational inference. We want to perform inference in the network shown below, where each node has values in $\{1, 2\}$ with the following conditional probability distributions (the conditional distributions have already been implemented for you in the file `probs.R`). We are interested in computing the marginal distribution over E and F.

For this problem, we have provided starter code in three files: `probs.R`, `inference.R`, and `demo.R`. `probs.R` contains the conditional probability distributions (implemented as functions) along with some utility functions used in `inference.R`. `inference.R` contains two functions: `mean.field.inference` and `struct.mean.field.inference`. Both of these functions are only partially implemented. You will complete the implementation using update equations that you derive in the following questions. Finally, `demo.R` is a script that can be run by typing `Rscript demo.R` at the command line. Initially, this will only print the true marginal distribution over E and F and will fail with an error message indicating that update functions have not yet been implemented. Run this script once to see what the true marginal looks like.

$P(A = 2)$	0.5
$P(B = 2\|A = 1)$	0.1
$P(B = 2\|A = 2)$	0.9
$P(C = 2\|A = 1, B = 1)$	0.1
$P(C = 2\|A = 1, B = 2)$	0.5
$P(C = 2\|A = 2, B = 1)$	0.5
$P(C = 2\|A = 2, B = 2)$	0.9
$P(D = 2\|B = 1)$	0.4
$P(D = 2\|B = 2)$	0.6
$P(E = 2\|C = 1, D = 1)$	0.1
$P(E = 2\|C = 1, D = 2)$	0.9
$P(E = 2\|C = 2, D = 1)$	0.3
$P(E = 2\|C = 2, D = 2)$	0.7
$P(F = 2\|D = 1)$	0.1
$P(F = 2\|D = 2)$	0.9

Table 1: Conditional probability distributions for the simple network.

2.1 Empirical Questions [40 points]

a. **[10 points]** Derive the mean-field variational update equations for this network. Start with the energy functional, and explicitly derive the update equations for each of the marginal variational distributions (e.g. $Q(A)$).

b. **[10 points]** Implement the update equations in the function `mean.field.inference` in `inference.R`. Note that you only need to implement the `{a,b,c,d,e,f}.up` functions; everything else has already been written for you. Once you have completed the implementation, re-run `demo.R`. The script should print the marginal over E and F obtained using mean-field inference, and will display the KL divergence between the true and approximate marginal. Answer the following questions:

- Does this look like a reasonable approximation?
- What does the KL divergence mean in this case?

c. [**10 points**] To improve the approximation, we can use structured mean-field variational inference. Instead of using a completely factored joint over A, B, C, D, E and F, we will use two factors (A, B, C) and (D, E, F) (i.e. we have two variational distributions $Q(A, B, C)$ and $Q(D, E, F)$). Derive the update equations for this new factorization. Again, begin with the energy functional and explicitly derive the update equations for each factor.

d. [**10 points**] Implement the update equations in the function `struct.mean.field.inference` in `inference.R`. You only need to complete the functions `abc.up` and `def.up`. Re-run `demo.R` after completing the update functions. Again, the script should print the mean-field and structured mean-field approximate marginals below the true marginal along with the KL divergence for each. Answer the following questions:

- Does this look like a better approximation than mean-field?
- What does the KL divergence mean in this case?
- Give a brief explanation (2-3 sentences) for why structured mean-field performs better for this Bayesian network.

The functions you need to implement only require assigning variables and calling basic math functions; we expect you to be able to do this even if you have no previous experience with R. If you need an introduction to R, we recommend looking at `http://cran.r-project.org/doc/manuals/R-intro.html`.

Deliverables Submit the script `inference.R` with your implementation of the required functions.

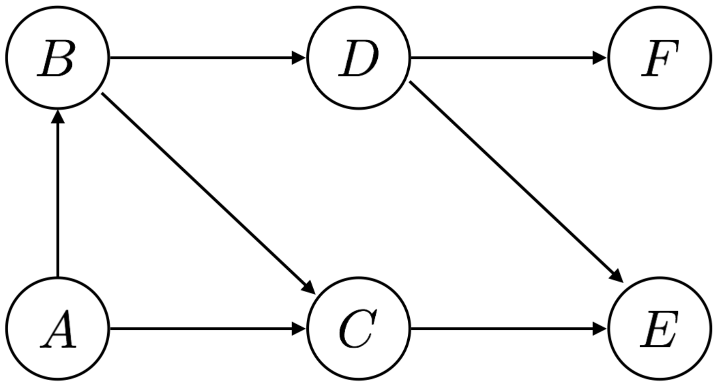

Figure 1: Simple network

3 Analyzing Multiple Text Corpora

In this assignment, we have provided you with abstracts from two research communities: 1) ACL which publishes papers in natural language processing, and 2) NIPS which publishes papers in machine learning. As someone who is new to these fields, you might be curious to understand how the research in these two communities has evolved. For example, what are the sets of topics that each of these communities have worked on? When did these topics emerge within each of these communities? Are there topics that are more prevalent in one? Below, we will walk you through *Multiple Corpora Latent Dirichlet Allocation (MCLDA)*, an example model that you might use to answer questions such as these. We also derive for you an approach for doing inference in MCLDA. You will be asked to implement a few different inference approaches for the MCLDA and analyze the inferred parameters to explore the given collections of abstracts.

3.1 Multiple Corpora Latent Dirichlet Allocation (MCLDA)

Our approach is motivated by a class of methods called Topic Models [1]. These employ a Bayesian approach for modeling the generation of documents within a collection. Griffiths et al. [3] use Latent Dirichlet Allocation (LDA) [1], the most popular variant of topic models to analyze abstracts from the Proceedings of the National Academy of Sciences (PNAS). Specifically, LDA posits that each document discusses a small set of topics; each word in the document is generated as a function of one of its topics. Topics are shared across documents in the collection. By analyzing prevalence of topics over time, they show nice results for when different scientific concepts emerged and periods during which different topics trended more than others. In our application, we wish to analyze two sets of collections jointly. Therefore, we develop a new model called Multiple Corpora Latent Dirichlet Allocation (MCLDA).

Let $d \in D$ index the set of documents. Each document is associated with a variable $c_d \in \{A, S\}$ which denotes whether the document is from the ACL or NIPS collection. $w_{d,i}$ indexes the set of words in a document. Let V be the total number of words in the vocabulary. v indexes the vth word in the vocabulary. We assume that there are K total topics. MCLDA posits that there are global and collection-specific parameters for each topic denoted by ϕ_k and $\phi_k^{(c)}$ respectively. Topics are a multinomial distribution over words in the vocabulary. The vth element of ϕ_k specifies how likely the vth word is under topic ϕ_k. For example, a topic on inference is more likely to heavily weight the words *estimation* and *parameters*.

Each document is composed from a mixture of topics; θ_d specifies the mixing proportions for the topics in document d. Note, the length of the vector θ_d equals K. A document is generated from MCLDA as follows: First, we sample θ_d for document d. Next, given N_d, the number of words in the document, we sample each of the words. To sample each word $w_{d,i}$, we sample the topic index $z_{d,i}$ from $\text{Mult}(\theta_d)$. Next, we sample whether the word is generated from the global or collection-specific topic distributions. The model posits that with probability λ the word $w_{d,i}$ is sampled according to the collection-specific distribution $\phi_k^{(c)}$, and with probability $1 - \lambda$, the word is sampled according to the global distribution ϕ_k. The choice of whether to use ϕ or $\phi^{(c)}$ is denoted by the binary random variable $x_{d,i}$ for each token (d, i); x is a Bernoulli random variable with parameter λ.

The graphical model for MCLDA is shown in Figure 2. The variables $w_{d,i}$ and c_d are observed. The remaining variables ϕ_k, θ_d, $z_{d,i}$, and $x_{d,i}$ are hidden variables and must be inferred from the data. α, β and λ are hyperparameters to specify the priors for the model. To specify a prior over θ, we use the Dirichlet distribution which is conjugate to the multinomial. More concretely, $\theta \sim \text{Dirichlet}(\alpha_1, \cdots, \alpha_k)$ where $\alpha_1 = \alpha_k = \alpha$. This is called a *symmetric* prior as the prior strengths are assumed to be the same for all elements in the distribution. Similarly, we use a symmetric Dirichlet distribution with hyperparameter β to specify the prior for word distributions for each of the topics ϕ_k. λ specifies what fraction of the words are likely to be sampled from the global vs. the collection-specific topics.

3.1.1 The data log-likelihood

Given the parameters of the word distributions ϕ, the hyperparameters of the document distributions α, and λ each document is generated conditionally independent of the others in our corpus. Therefore, the joint probability of the data decomposes as the product of the probability estimate for each document under the model. Let a document d be represented by the vector of words $\boldsymbol{w}$ that the document contains. The likelihood of each document is:

$$P(\boldsymbol{w}_d|\alpha, \boldsymbol{\phi}) = \int_\theta P(\theta|\alpha) \left(\prod_{n=1}^{N_d} \sum_{z_n} P(z_n|\theta) \Big((1-\lambda) P(w_n|z_n, \phi_{z_n}) + \lambda P(w_n|z_n, \phi_{z_n}^{(c_d)}) \Big) \right) \mathrm{d}\theta \quad (1)$$

Note that the document-specific likelihood in turn can be written as the product of individual word-specific likelihood terms. The joint likelihood is then written as:

$$P(\boldsymbol{w}|\alpha, \beta, \lambda) = \int_\phi P(\phi|\beta) \left(\prod_d P(\boldsymbol{w}_d|\alpha, \phi) \right) \mathrm{d}\phi \quad (2)$$

We assume α, β, and λ are always given. One can use cross-validation to set these parameters.

3.1.2 Inference

Exact inference in this model is intractable. Existing works have used various approximate inference techniques. See [2] for a comparison of different approximate inference schemes for LDA which is similar in computational complexity to our model. Collapsed Gibbs sampling and variation inference are among the most popular inference methods for LDA. In this assignment, we derive the Collapsed Gibbs sampler for the MCLDA for you. As we learnt in class, in (full) Gibbs sampling, each variable is sampled conditioned on all other variables in the model. In a collapsed Gibbs sampler, we marginalize out a subset of the variables instead of conditioning on all of them. This reduces the variance of our sampler and accelerates convergence. Blocked Gibbs samplers sample multiple variables at once, which introduces a tradeoff in complexity per sample and overall mixing time. Below, you will be deriving the full Gibbs sampler for the MCLDA and optionally (for extra credit) compare the sampler's performance against a blocked sampler.

4 Collapsed Gibbs Sampler Implementation [100 points]

Recall that in a typical (uncollapsed) Gibbs sampler, we iterate through each variable in the model and resample that variable conditioning on all the others. The equation we sample from is called the *full-conditional* for that variable.

In this section, you will implement a *collapsed* Gibbs sampler. In a collapsed sampler, the parameters θ and ϕ are eliminated through marginalization, so you only sample from the posteriors of the $\boldsymbol{z}$ and $\boldsymbol{x}$ variables. We will assume the hyperparameters α and β are given.

We will provide you with the sampling equations needed to implement a Gibbs sampler for the model described above. See the appendix, section 8, for a detailed derivation.

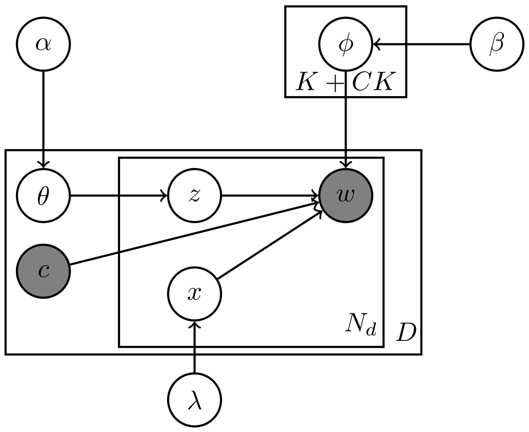

Figure 2: The graphical model for the LDA variant described here.

Full-Conditional for $z_{d,i}$ Given the current value assignments of all variables in the model (including the token's x variable, $x_{d,i}$, except for $z_{d,i}$, a new value for $z_{d,i}$ is sampled according to the following distribution:

$$\begin{aligned} P(z_{d,i} = k|\boldsymbol{z} - z_{d,i}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{w}, \alpha, \beta) &\propto \left(\frac{n_k^d + \alpha}{n_*^d + K\alpha}\right)\left(\frac{n_w^k + \beta}{n_*^k + V\beta}\right) && \text{if } x_{d,i} = 0 \\ P(z_{d,i} = k|\boldsymbol{z} - z_{d,i}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{w}, \alpha, \beta) &\propto \left(\frac{n_k^d + \alpha}{n_*^d + K\alpha}\right)\left(\frac{n_w^{c,k} + \beta}{n_*^{c,k} + V\beta}\right) && \text{if } x_{d,i} = 1 \end{aligned} \tag{3}$$

where the $\propto$ symbol means "proportional to." The notation n_k^d means the number of tokens in document d assigned to topic k, while n_*^d is the number of tokens assigned to any topic in document d (i.e. the length of document d). Similarly, n_w^k is the number of tokens assigned to topic k which are the word type w, and n_*^k is the total number of tokens of any word type that are assigned to topic k. The n_w^k counts are the corpus-independent counts (where $x = 0$) while $n_w^{c,k}$ are the corpus-dependent counts (where $x = 1$) for collection c. The n counts do not include the sampler assignment of the current token (d, i). V is the number of unique word types (the vocabulary size).

Full-Conditional for $x_{d,i}$ Similarly, given all other variable assignments, a new value for $x_{d,i}$ is sampled according to the following distribution:

$$\begin{aligned} P(x_{d,i} = 0|\boldsymbol{x} - x_{d,i}, \boldsymbol{z}, \boldsymbol{c}, \boldsymbol{w}, \lambda, \beta) &\propto (1 - \lambda)\left(\frac{n_w^{z_{d,i}} + \beta}{n_*^{z_{d,i}} + V\beta}\right) \\ P(x_{d,i} = 1|\boldsymbol{x} - x_{d,i}, \boldsymbol{z}, \boldsymbol{c}, \boldsymbol{w}, \lambda, \beta) &\propto \lambda\left(\frac{n_w^{c,z_{d,i}} + \beta}{n_*^{c,z_{d,i}} + V\beta}\right) \end{aligned} \tag{4}$$

MAP Estimates of the Marginalized Parameters Given the counts from the sampler assignments, MAP estimates of the multinomial parameters can be obtained as:

$$\hat{\theta}_{dk} = \frac{n_k^d + \alpha}{n_*^d + K\alpha} \tag{5}$$

$$\hat{\phi}_{kw} = \frac{n_w^k + \beta}{n_*^k + V\beta} \tag{6}$$

$$\hat{\phi}_{kw}^{(c)} = \frac{n_w^{c,k} + \beta}{n_*^{c,k} + V\beta} \tag{7}$$

Gibbs Sampling Algorithm In the limit, the sampled values of $\boldsymbol{z}$ and $\boldsymbol{x}$ variables, and these estimates for the $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ variables, will be distributed according to their posterior distribution. Your output files will give the expected value of the model parameters, which you will estimate as the sample average. Because the sampler will not be in a steady part of the posterior from the start, we run the sampler for some number of iterations, called the *burn-in* period, before we actually start saving the samples. Your program will run for a specified number of burn-in iterations before you begin saving values of $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\phi}}$ to obtain the sample mean.

The Gibbs sampling algorithm you will implement is thus:

1. Set all the $\boldsymbol{z}$ and $\boldsymbol{x}$ values to randomly chosen values in $\{0, \ldots, K-1\}$ and $\{0, 1\}$, respectively. There is one z and x value per token.
2. For each iteration t in $\{1, T\}$:
   (a) For each token (d, i) in each document d in the training set:
      i. Update the counts to exclude the assignments of the current token.
      ii. Randomly sample a new value for $z_{d,i}$. The probability of sampling a value $z_{d,i} = k$ should be proportional to Eq. 3.
      iii. Randomly sample a new value for $x_{d,i}$. The probability of sampling a value should be proportional to Eq. 4. Here, you should use the newly sampled value of $z_{d,i}$.
      iv. Update the counts to include the newly sampled assignments of the current token.
   (b) Estimate the parameters according to Eqs. 5-7
   (c) If the burn-in period is passed:
      i. Incorporate the estimated parameters into your estimate of the expected value.
   (d) Sample z and x of the test set in the same way, except that in Eqs. 3-4 we directly use $\hat{\phi}$ estimated from the current iteration, rather than recounting n_w^z from the test set.
   (e) Compute the train log-likelihood as described in section 4.0.3.
   (f) Compute the test log-likelihood as described in section 4.0.3.

4.0.3 Log-Likelihood

You will compute and print out the log-likelihood of the corpus, i.e. the log-likelihood of all the observed data $\boldsymbol{w}$, marginalizing out the hidden variables $\boldsymbol{z}$ and $\boldsymbol{x}$. For simplicity, rather than integrating over the parameters θ and ϕ, you will compute the data log-likelihood conditioned on the current MAP estimates of the parameters, $\hat{\theta}$ and $\hat{\phi}$, from Eqs. 5–7. This conditional log-likelihood is thus:

$$\log P(\boldsymbol{w}|\theta,\phi,\lambda,\alpha,\beta) = \log\left(\prod_d \prod_{i\in d}\sum_z \hat{\theta}_{dz}\left((1-\lambda)\hat{\phi}_{zw_{d,i}} + \lambda\hat{\phi}^{c_d}_{zw_{d,i}}\right)\right) \tag{8}$$
$$= \sum_d \sum_{i\in d} \log\left(\sum_z \hat{\theta}_{dz}\left((1-\lambda)\hat{\phi}_{zw_{d,i}} + \lambda\hat{\phi}^{(c_d)}_{zw_{d,i}}\right)\right)$$

4.1 Input Format

Inside `hw3-files.zip` is are two input files `input-train.txt` and `input-test.txt`. Each input file contains a set of documents from two corpora: abstracts from NIPS, a machine learning conference, and ACL, a natural language processing conference. There will a fair amount of topical overlap between the documents, but there will be some differences as well.

The input file assumes each line is one document. The first token of each line is the corpus ID (the value of c_d), where $c = 0$ for the NIPS corpus and $c = 1$ for the ACL corpus. The remaining tokens are the words of the document. For example:

```
0 this is a document from the nips corpus
1 this is a document from the acl corpus
0 this is another document from the nips corpus
... ...
```

We have already processed the documents to remove punctuation and to make the words lowercase. We also removed common words, known as stop words, such as "and" and "the", which do not contribute to topical content.

4.2 Output Format

Your program should write output files which contain the distributions θ and ϕ. You will pass in an argument to your program containing the name of an output file, e.g. `output.txt`, but you will actually append this filename to create 6 different output files: `output.txt-theta`, `output.txt-phi`, `output.txt-phi0`, `output.txt-phi1`, `output.txt-trainll`, `output.txt-testll`.

The file `output.txt-theta` will contain the values of θ for the training data where the dth line corresponds to the dth document (in the same order that the documents appear in the input file), and the kth token of the dth line will contain the value θ_{dk}.

The file `output.txt-phi` will contain the values of ϕ. Each line corresponds to a word, where the first token of the line is the word, and the remaining tokens contain the value of ϕ_{kw}. For example, if there are 3 topics, then `output.txt-phi` will look something like:

```
this 1.1561602396399e-03 4.3120265574119e-02 3.1088866898660e-02
document 3.6187399870030e-02 4.0462121563490e-02 2.1777333111282e-02
corpus 2.2541197497970e-02 4.3316990381425e-02 4.6417628944161e-02
nips 5.8867538741561e-03 1.2942924525683e-02 2.1800741469388e-02
acl 7.7708784812371e-03 2.8229868507249e-02 4.5927702097470e-02
... ...
```

The parameter values should be printed with 13 significant digits in scientific notation.

There should be two additional output files, `output.txt-phi0` and `output.txt-phi1`, which contain the corpus-dependent parameters $\phi^{(0)}$ and $\phi^{(1)}$. These files will have the same format as `output.txt-phi`.

The file `output.txt-trainll` will contain the log-likelihoods on the training data. You should write out the train log-likelihood after each iteration, with one log-likelihood value per line. Do not include anything else on that line. Compute it as described in section 4.0.3. The file `output.txt-testll` will be identical to `output.txt-trainll` except that it will contain the test log-likelihood after each iteration. You should write out these log-likelihood values in scientific notation with 13 digits of precision.

4.3 Analyzing the Topics

Included in `hw3-files.zip` is a Python script called `topwords.py`. It takes a ϕ output file as input, and prints out the highest probability words in each topic. You may find it helpful to visualize the posterior, and you will be asked to use this to analyze the topics later. You can run it with the following command:

```
python topwords.py output.txt-phi
```

4.4 Deliverables [100 points]

You will create and submit a program called `collapsed-sampler`. This program will take 9 arguments at the command line: a path to the input train file, a path to the input test file, the name of the output file, the number of topics (K), the value of λ, the value of α, the value of β, the number of total iterations, and the number of samples to use as burn-in. You can assume we will give you an input file with 2 corpora (i.e. the possible values of c are 0 or 1), although you might want to generalize your code to accept other inputs for your own benefit.

For example, the following command would run the sampler with $K = 10$ topics and would collect 100 samples after a 1000-iteration burn-in:

```
./collapsed-sampler input-train.txt input-test.txt output.txt 10 0.5 0.1 0.01 1100 1000
```

In addition to turning in your code, you should hand in all `output*` files associated with 5 different sampler inputs:

- `collapsed-output-5-0.5-0.1.txt` – $K = 5$ topics, $\lambda = 0.5, \alpha = 0.1, \beta = 0.01$
- `collapsed-output-5-0.8-0.1.txt` – $K = 5$ topics, $\lambda = 0.8, \alpha = 0.1, \beta = 0.01$
- `collapsed-output-25-0.5-0.1.txt` – $K = 25$ topics, $\lambda = 0.5, \alpha = 0.1, \beta = 0.01$
- `collapsed-output-25-0.2-0.1.txt` – $K = 25$ topics, $\lambda = 0.2, \alpha = 0.1, \beta = 0.01$
- `collapsed-output-25-0.5-1.0.txt` – $K = 25$ topics, $\lambda = 0.5, \alpha = 1.0, \beta = 0.01$

For each of these output files, you should run the sampler for a total of 1100 iterations with 1000 burn-in iterations, which means your θ and ϕ parameters will be averaged from 100 samples.

Creating the `collapsed-sampler` program As in previous homeworks, you can write your program in whatever programming language you'd like to, as long as it can be executed on the `ugrad` servers, but you must develop an executable named `collapsed-sampler`. As before, this executable should simply be a shell script which acts as a wrapper to your program. Remember that you need to run the command `chmod +x collapsed-sampler` in order to make the script an executable file that we can run. See Piazza for an example Matlab script.

Programming efficiency We note that you will get the best performance out of your implementation if you store the document and word counts and probabilities as arrays. Doing this requires creating a mapping from word types to integers, unlike if you simply were to use hash tables or dictionaries indexed on strings. The Gibbs sampler has low complexity, and we are giving you a small data set, so you should be able to run your program regardless and we will not place any requirements on efficiency, but note that if you want to create a program that can scale to larger data sets, then you should keep in mind the efficiency of your data structures.

Clarifications and Tips

- Typically the vocabulary would need to grow as needed. In this assignment, for simplicity, you can assume that the vocabulary is fixed; it is just the union of all of the words in both the training and testing sets.
- To improve efficiency, all counts are first initialized according to the initial random assignment of $z_{d,i}$, $x_{d,i}$, and they are then updated as the sampling iterations proceed.
- Remember to exclude the assignment of the current token before each sampling. Include the newly sampled value after sampling.
- When performing inference on the test data, you should not alter the parameters sampled on the training set. In particular, you should only have n_w^z counts (used to compute ϕ) from the training set.

  However, the θ parameters are document specific, so these don't come from the training set. To infer these, you need to run the sampler on the test data. When sampling z on the test data, you'll need to update the n_z^d counts to estimate θ for each document. You will also need to sample x (because it affects the sampling equation for z), although this won't alter any of the counts.
- Each Gibbs sampling iteration should sweep through the training data first, then sweep through the test data. That is, in each iteration you will
  1. Sweep through the training data once, updating the n_w^z and n_z^d counts
  2. Estimate ϕ from the training counts
  3. Sweep through the test data once, updating the n_z^d counts but keeping the n_w^z counts fixed (ϕ)
  4. Compute the likelihood for both the training and testing data, where θ is learned separately from the training and testing data, and where ϕ is learned from only the training data.
- For simplicity, only compute the likelihood using the current point estimates of θ and ϕ at each iteration. You don't need to use estimates that are averaged over multiple samples.
- To make sure your implementation is correct, run for only 100-200 iterations to see if the training data's likelihood is (roughly) increasing.

5 Blocked Gibbs Sampler [20 points] [20 extra credit]

In this section, you will derive the *blocked* collapsed Gibbs sampler for the MCLDA. For extra credit, you may also implement the blocked Gibbs sampler that you derived.

5.1 Analysis Questions [20 points]

A standard Gibbs sampler samples only one variable at a time, from their distribution conditioned on all others. A *blocked* Gibbs sampling jointly samples multiple variables (a block of variables), conditioned on all others. Sampling larger blocks is more expensive because there are more combinations of values to be considered, but this may also improve the sampler mixing time (the time needed to reach the Markov chain's steady state distribution).

A natural block of variables in this assignment's model is the set of variables associated with a single token. Rather than alternately sampling $z_{d,i}$ and $x_{d,i}$, one could jointly sample values for the pair $(z_{d,i}, x_{d,i})$. Derive a blocked collapsed Gibbs sampler that does this. That is, come up with the distributions from which $P(z_{d,i} = k, x_{d,i} = 0)$ and $P(z_{d,i} = k, x_{d,i} = 1)$ are sampled, similar to Eqs. 3 and 4. As a starting point, you may refer to the appendix to see how these equations were derived for the single-variable case.

5.2 Deliverables [20 extra credit]

You will create and submit a program called `blocked-sampler`. This program will take the same 9 arguments at the command line as your `collapsed-sampler` program. It should accept input and output files of the exact same formats as described in sections 4.1 and 4.2, respectively.

For example, the following command would run the blocked sampler with $K = 10$ topics and would collect 100 samples after a 1000-iteration burn-in:

```
./blocked-sampler input-train.txt input-test.txt output.txt 10 0.5 0.1 0.01 1100 1000
```

In addition to turning in your code, you should hand in all `blocked-output*` files associated with the same 5 program inputs as the collapsed sampler, e.g. `blocked-output-5-0.5-0.1.txt`.

Creating the `blocked-sampler` program As in Homework 1 and 2, you can write your program in whatever programming language you like, as long as it can be executed on the `ugrad` servers, but you must develop an executable named `blocked-sampler`. As before, this executable should simply be a shell script which acts as a wrapper to your program. Remember that you need to run the command `chmod +x blocked-sampler` in order to make the script an executable file that we can run. See Piazza for an example Matlab script.

Log-Likelihood You will compute and print out the log-likelihood of the corpus, just as in your `collapsed-sampler` program using Eq. (8).

6 Text Analysis with MCLDA [40 points] [10 extra credit]

It is now time to do something meaningful with the parameters you have inferred. In this section, we will ask you to examine the parameters that are learned (along with the data likelihood), examine the topics (by looking at their most probable words), and analyze the data.

6.1 Empirical Questions [40 points] [10 extra credit]

For these questions, use $\alpha = 0.1, \beta = 0.01, \lambda = 0.5$ and 25 topics, unless otherwise stated the question.

1. [**10 points**] Plot both the **training and test** likelihood for the collapsed sampler as a function of number of iterations. That is, the x-axis should show the sampler iteration, while the y-axis will show the likelihood of data using the sampled parameters at that iteration. Show both curves (the train and test likelihoods) in the same graph so they can easily be compared. Do this three times for different sampling chains: that is, run the Gibbs sampler (starting from randomly initialized values) three times (3 plots total). Describe what you observe; how do the train and test likelihoods compare?

2. [**5 extra credit**] If you implemented the blocked sampler, compare its likelihood over time to the basic sampler. Plot the **training** likelihood for the two samplers as a function of number of iterations. That is, the x-axis should show the sampler iteration, while the y-axis will show the likelihood on the training data using the sampled parameters at that iteration. Show the likelihood from both the standard and blocked samplers on the same plot so that you can easily compare. You only need to do this for one sampling chain. Which of the two samplers appears to mix (the likelihood reaches a stable point) faster, if either?

3. [**5 extra credit**] Repeat the previous question, except the x-axis should be your program runtime instead of iterations. That is, you will plot the training likelihood against runtime for the two samplers. Which of the two samplers is faster per iteration, if either?

4. [**5 points**] Show how the **test** likelihood varies with the number of topics. Plot the likelihood on the test set, where the x-axis is the number of topics in $\{10, 20, 30, 40, 50\}$. Do this using the parameters averaged at the end of the collapsed sampling algorithm. You only need to do this for one chain.

5. [**5 points**] Show how the **test** likelihood varies with the value of λ. Plot the likelihood on the test set, where the x-axis is the value of λ in $\{0, 0.25, 0.5, 0.75, 1\}$. Do this using the parameters averaged at the end of the collapsed sampling algorithm. You only need to do this for one chain.

6. [**20 points**] Using the collapsed sampler, qualitatively examine the highest probability words for the various topics using the `topwords.py` script, and describe what you observe:

   (a) [**10 points**] Do you see topic indices where the global, NIPS and ACL word distributions all share a common theme? Similarly, do you see topic indices where the three word distributions are very different? Give examples, and try this with different numbers of topics.

   (b) [**5 points**] How does the value of λ affect the topics? In particular, how do the global and corpus-specific topics compare for large and small values of λ? Why? For this comparison, use 25 topics.

   (c) [**5 points**] The hyperparameters α and β can be thought of as smoothing parameters. How do these values affect the topics? In particular, compare a small value $\alpha = 0.001$ with a large value $\alpha = 10.0$, and describe what differences you observe in the topics. Why does this happen? Do the same for β. For this comparison, use 25 topics.

7 Variational Inference [100 extra credit]

Alternative approaches to posterior inference include variational methods. You may derive and implement a variational inference algorithm for the model described in this handout. This is purely optional, and you will be given extra credit for doing so.

7.1 Analytical Questions [30 extra credit]

Derive a variational inference algorithm for MCLDA (Figure 2) based on a mean field approximation.

Your algorithm should be based on the variational inference algorithm in the original LDA paper [1]. See Section 5 of [1] on how to derive such an algorithm. You will follow a similar approach, but it will be modified for the particular MCLDA model in this handout.

You should describe the variational parameters you introduce, their update equations, and the pseudocode for the complete algorithm.

7.2 Deliverables [50 extra credit]

You will create and submit a program called `variational`. This program will take 8 arguments at the command line: a path to the input train file, a path to the input test file, the name of the output file, the number of topics (K), the value of λ, the value of α, the value of β, and the number of iterations to loop the algorithm. The input and output formats are the same as for the Gibbs sampler program.

This program will implement the algorithm you derived in the previous section. In general, the algorithm is run until convergence criteria are reached; for example, the difference in parameters between two iterations is under some threshold. However, you should simply run your algorithm for a fixed number of iterations, given by the last parameter of the program.

For example, the following command would run the inference algorithm for 20 iterations with $K = 10$ topics.

```
./variational input-train.txt input-test.txt output.txt 10 0.5 0.1 0.01 20
```

In addition to turning in your code, you should hand in all `variational-output*` files associated with the same 5 program inputs as the sampler, e.g. `variational-output-5-0.5-0.1.txt`.

7.3 Empirical Questions [20 extra credit]

For these questions, use $\alpha = 0.1, \beta = 0.01, \lambda = 0.5$ and 25 topics, unless otherwise stated the question.

1. [**5 extra credit**] Plot both the **training and test** likelihoods from the variational algorithm over time, with the number of iterations on the x-axis. Put the two curves in the same graph so they can easily be compared.

2. [**5 extra credit**] Compare the **test** likelihood of the variational algorithm to the collapsed Gibbs sampler over time. Plot the test likelihood for the two algorithms as a function of number of iterations. That is, the x-axis should show the sampler iteration, while the y-axis will show the likelihood on the held-out test data using the inferred parameters at that iteration. Show the likelihood from both algorithms on the same plot so that you can easily compare. Do this three times for three different trials with random initializations (3 plots total). Does one algorithm generally yield better held-out likelihood than the other?

3. [**5 extra credit**] Repeat the previous question, except the x-axis should be your program runtime instead of iterations. That is, you will plot the test likelihood against runtime for the two algorithms. You only need to do this for one trial (1 plot total).

4. **[5 extra credit]** Different hyperparameter values may be optimal when using different inference algorithms. Show the likelihood on the test data for various values of α, and do this for both algorithms. How do the various α values compare across the two algorithms?

8 Appendix

8.1 Probability Background: The DCM Distribution

If a vector of variables $\boldsymbol{z}$ is distributed according to multinomial parameters θ, and θ is distributed according to Dirichlet(α), then the marginal probability of $\boldsymbol{z}$ has the following form:

$$P(\boldsymbol{z}|\alpha) = \int_\theta P(\boldsymbol{z}|\theta)P(\theta|\alpha)\mathrm{d}\theta = \frac{\Gamma(\sum_k \alpha_k)}{\Gamma(\sum_k n_k + \alpha_k)} \prod_k \frac{\Gamma(n_k + \alpha_k)}{\Gamma(\alpha_k)} \tag{9}$$

where n_k is the number of variables i such that $z_i = k$, and Γ is the Gamma function, a generalization of the factorial function. This distribution is called the Dirichlet compound multinomial (DCM) distribution.

In deriving the collapsed Gibbs sampler, we will use the DCM twice to marginalize over $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. Note that in this problem set, we have assumed symmetric Dirichlet parameters, so $\alpha_k = \alpha, \forall k$.

8.2 Collapsed Gibbs Sampler Derivation

Below, we describe in detail how the sampling distributions for the collapsed Gibbs sampler are derived. For the Gibbs sampler that results, we will iterate through each variable in the model and resample that variable conditioning on all the others.

The Variables In this section, we focus specifically on the case of the *collapsed* Gibbs sampler.

We often talk about the *data*, *variables*, *parameters*, and *hyperparameters* of the model separately, in order to provide a clearer differentiation between them. However, this distinction is somewhat artificial if we think of MCLDA as just another Bayes Net. In fact, just as in any other Bayes Net, our model simply consists of a network of random variables. At inference time, we observe some of these variables, we marginalize over others, and we sample from the posterior of the rest. In this sense, we can divide the variables into three sets specific to the *collapsed* Gibbs sampler:

	Variables
Observed	$\boldsymbol{w}, \alpha, \beta, \lambda$
Marginalized	$\boldsymbol{\theta}, \boldsymbol{\phi}$
Sampled	$\boldsymbol{z}, \boldsymbol{x}$

To derive the Gibbs sampler, we need to find the full-conditional for each sampled variable. We condition on all of the other observed and sampled variables. We call the resulting sampler *collapsed* because we have marginalized over some of the variables.

$$P(z_{d,i} = k|\boldsymbol{z} - z_{d,i}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{w}; \alpha, \beta, \lambda) \tag{10}$$

$$P(x_{d,i} = b|\boldsymbol{z}, \boldsymbol{x} - x_{d,i}, \boldsymbol{c}, \boldsymbol{w}; \alpha, \beta, \lambda) \tag{11}$$

We let $\boldsymbol{z} - z_{d,i}$ denote the vector of variables $\boldsymbol{z}$ except for $z_{d,i}$, and $\boldsymbol{x} - x_{d,i}$ denote the vector variables $\boldsymbol{x}$ except for $x_{d,i}$. Determining how to compute the full conditionals requires just a bit of algebra and calculus, plus some standard rules of probability.

The Uncollapsed Joint Likelihood Before we define the full-conditionals, we want to consider the likelihood and collapsed likelihood of the full model.

The uncollapsed likelihood is below. We define it as usual using the conditional independence assumptions given by our model, MCLDA. We can read these independence assumptions directly off our plate diagram.

$$P(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{w}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta, \lambda) = P(\boldsymbol{w}|\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{\phi})P(\boldsymbol{\phi}|\beta)P(\boldsymbol{z}|\boldsymbol{\theta})P(\boldsymbol{\theta}|\alpha)P(\boldsymbol{x}|\lambda) \tag{12}$$

The Collapsed Joint Likelihood To obtain the collapsed likelihood of the model, we must integrate out the marginalized variables.

$$\begin{aligned}
P(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{w}|\alpha, \beta, \lambda) &= \int_{\boldsymbol{\theta}} \int_{\boldsymbol{\phi}} P(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{w}, \boldsymbol{\theta}, \boldsymbol{\phi}|\alpha, \beta, \lambda)\, \mathrm{d}\boldsymbol{\phi}\, \mathrm{d}\boldsymbol{\theta} \\
&\quad \text{by definition of a marginalized probability.} \\
&= \int_{\boldsymbol{\theta}} \int_{\boldsymbol{\phi}} P(\boldsymbol{w}|\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{\phi})P(\boldsymbol{\phi}|\beta)P(\boldsymbol{z}|\boldsymbol{\theta})P(\boldsymbol{\theta}|\alpha)P(\boldsymbol{x}|\lambda)\, \mathrm{d}\boldsymbol{\phi}\, \mathrm{d}\boldsymbol{\theta} \\
&\quad \text{by substituting in our definition of the uncollapsed likelihood.} \\
&= \left(\int_{\boldsymbol{\phi}} P(\boldsymbol{w}|\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{\phi})P(\boldsymbol{\phi}|\beta)\mathrm{d}\boldsymbol{\phi}\right) \left(\int_{\boldsymbol{\theta}} P(\boldsymbol{z}|\boldsymbol{\theta})P(\boldsymbol{\theta}|\alpha)\mathrm{d}\boldsymbol{\theta}\right) P(\boldsymbol{x}|\lambda) \\
&\quad \text{by rearranging the terms.} \\
&= P(\boldsymbol{w}|\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c}, \beta)P(\boldsymbol{z}|\alpha)P(\boldsymbol{x}|\lambda) \\
&\quad \text{collapsing the integrals by definition of marginalization.}
\end{aligned}$$

The final form above of the collapsed likelihood should be fairly intuitive. That we collapsed the intregrals may be somewhat surprising. After all, how do we know whether the two terms $P(\boldsymbol{w}|\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c}, \beta)$ and $P(\boldsymbol{z}|\alpha)$ will have closed forms? To ensure this is the case, we will consider each one in turn.

The Form of $P(\boldsymbol{z}|\alpha)$ First, consider the form of $P(\boldsymbol{z}|\alpha)$.

$$\begin{aligned}
P(\boldsymbol{z}|\alpha) &= \int_{\boldsymbol{\theta}} P(\boldsymbol{z}|\boldsymbol{\theta})P(\boldsymbol{\theta}|\alpha)\mathrm{d}\boldsymbol{\theta} \\
&= \prod_d \left(\int_{\boldsymbol{\theta}_d} P(\boldsymbol{z}_d|\boldsymbol{\theta}_d)P(\boldsymbol{\theta}_d|\alpha)\mathrm{d}\boldsymbol{\theta}_d\right) \\
&\quad \text{since the likelihood is the product over documents.} \\
&= \prod_d \left(\frac{\Gamma(\sum_k \alpha)}{\Gamma(\sum_k n_k^d + \alpha)} \prod_k \frac{\Gamma(n_k^d + \alpha)}{\Gamma(\alpha)}\right) \\
&\quad \text{since each term is a DCM, substitute Eq. (9).}
\end{aligned}$$

Since each term in the product over documents was exactly a DCM distribution, we already knew exactly what form it should take. It was given in Eq. (9) and we have simply updated it to use the appropriate document-specific count variables. We've also written it to reflect that α parameterizes a symmetric Dirichlet. The notation n_k^d means the number of tokens in document d assigned to topic k.

The Form of $P(\boldsymbol{w}|\boldsymbol{z},\boldsymbol{x},\boldsymbol{c},\beta)$ Second, consider the form of $P(\boldsymbol{w}|\boldsymbol{z},\boldsymbol{x},\boldsymbol{c},\beta)$. Let $I(x)$ be an indicator function that returns 1 if the expression x is true, and 0 otherwise. (Here the algebra looks a bit trickier, the basic ideas are almost the same as our derivation of $P(\boldsymbol{z}|\alpha)$.)

$$\begin{aligned}
P(\boldsymbol{w}|\boldsymbol{z},\boldsymbol{x},\boldsymbol{c},\beta) &= \int_{\boldsymbol{\phi}} P(\boldsymbol{w}|\boldsymbol{z},\boldsymbol{x},\boldsymbol{c},\boldsymbol{\phi}) P(\boldsymbol{\phi}|\beta)\,\mathrm{d}\boldsymbol{\phi} \\
&= \int_{\boldsymbol{\phi}} \prod_{(d,i)} P(w_{d,i}|z_{d,i},x_{d,i},c_d,\boldsymbol{\phi}) P(\boldsymbol{\phi}|\beta)\,\mathrm{d}\boldsymbol{\phi} \\
&\qquad \text{since the likelihood is a product over each word.} \\
&= \int_{\boldsymbol{\phi}} \prod_k P(\phi_k|\beta) \prod_{(d,i)} P(w_{d,i}|\phi_k)^{I(z_{d,i}=k,x_{d,i}=0)} \\
&\qquad \times \prod_c \prod_k P(\phi_k^{(c)}|\beta) \prod_{(d,i)} P(w_{d,i}|\phi_k^{(c)})^{I(z_{d,i}=k,x_{d,i}=1,c_d=c)}\,\mathrm{d}\boldsymbol{\phi} \\
&\qquad \text{using an indicator function to break up into the two cases of sampling} \\
&\qquad \text{the word from the global vs. corpus-specific topic.} \\
&= \prod_k \int_{\phi_k} P(\phi_k|\beta) \prod_{(d,i)} P(w_{d,i}|\phi_k)^{I(z_{d,i}=k,x_{d,i}=0)}\,\mathrm{d}\phi_k \\
&\quad \times \prod_c \prod_k \int_{\phi_k^{(c)}} P(\phi_k^{(c)}|\beta) \prod_{(d,i)} P(w_{d,i}|\phi_k^{(c)})^{I(z_{d,i}=k,x_{d,i}=1,c_d=c)}\,\mathrm{d}\phi_k^{(c)} \\
&\qquad \text{by rearranging the integrals.} \\
&= \prod_k \left(\frac{\Gamma(\sum_w \beta)}{\Gamma(\sum_w n_w^k + \beta)} \prod_w \frac{\Gamma(n_w^k + \beta)}{\Gamma(\beta)} \right) \\
&\quad \times \prod_c \prod_k \left(\frac{\Gamma(\sum_w \beta)}{\Gamma(\sum_w n_w^{c,k} + \beta)} \prod_w \frac{\Gamma(n_w^{c,k} + \beta)}{\Gamma(\beta)} \right) \\
&\qquad \text{since each term is a DCM, substitute Eq. (9).}
\end{aligned}$$

where $I(x)$ is an indicator function that returns 1 if the expression x is true, and 0 otherwise.

Notice that we actually had two separate DCM distributions. The final step replaced the terms for each of those DCM distributions. The term n_w^k is the number of tokens assigned to topic k which are the word type w. The n_w^k counts are the corpus-independent counts (where $x = 0$) while $n_w^{c,k}$ are the corpus-dependent counts (where $x = 1$) for collection c.

Back to the Collapsed Joint Likelihood Now finally, we can substitute our simplified forms of $P(\boldsymbol{z}|\alpha)$ and $P(\boldsymbol{w}|\boldsymbol{z},\boldsymbol{x},\boldsymbol{c},\beta)$ into the collapsed joint likelihood as below. The only other term we need is the expanded form of $P(\boldsymbol{x}|\lambda) = \prod_{(d,i)} P(x_{d,i}|\lambda)$. This allows us to obtain the full expanded form of the likelihood.

$$\begin{aligned}
P(\boldsymbol{z},\boldsymbol{x},\boldsymbol{c},\boldsymbol{w}|\alpha,\beta,\lambda) =& P(\boldsymbol{w}|\boldsymbol{z},\boldsymbol{x},\boldsymbol{c},\beta)P(\boldsymbol{z}|\alpha)P(\boldsymbol{x}|\lambda) \\
=& \prod_k \left(\frac{\Gamma(\sum_w \beta)}{\Gamma(\sum_w n_w^k + \beta)} \prod_w \frac{\Gamma(n_w^k + \beta)}{\Gamma(\beta)} \right) \\
& \times \prod_c \prod_k \left(\frac{\Gamma(\sum_w \beta)}{\Gamma(\sum_w n_w^{c,k} + \beta)} \prod_w \frac{\Gamma(n_w^{c,k} + \beta)}{\Gamma(\beta)} \right) \\
& \times \prod_d \left(\frac{\Gamma(\sum_k \alpha)}{\Gamma(\sum_k n_k^d + \alpha)} \prod_k \frac{\Gamma(n_k^d + \alpha)}{\Gamma(\alpha)} \right) \times \prod_{(d,i)} P(x_{d,i}|\lambda)
\end{aligned} \tag{13}$$

8.2.1 The Full Conditionals

Now that we have the full expanded form of the collapsed joint likelihood. Deriving the full-conditionals is fairly straightforward.

Sampling z To sample a value for $z_{d,i}$, we need to derive the following conditional probability:

$$P(z_{d,i}|\boldsymbol{z}-z_{d,i},\boldsymbol{x},\boldsymbol{c},\boldsymbol{w};\alpha,\beta,\lambda) = \frac{P(\boldsymbol{z},\boldsymbol{x},\boldsymbol{c},\boldsymbol{w}|\alpha,\beta,\lambda)}{P(\boldsymbol{z}-z_{d,i},\boldsymbol{x},\boldsymbol{c},\boldsymbol{w}|\alpha,\beta,\lambda)} \tag{14}$$

Note that we have eliminated the parameters θ and ϕ.

We want the conditional probability that $z_{d,i} = k$, which is the joint distribution of all variables including $z_{d,i} = k$ divided by the joint of all variables except for $z_{d,i}$ (Eq. 14). All terms in Eq. 13 will cancel except for those which involve the value of $z_{d,i}$. This yields:

$$\begin{aligned}
P(z_{d,i}=k|\boldsymbol{z}-z_{d,i},\boldsymbol{x},\boldsymbol{c},\boldsymbol{w}|\alpha,\beta,\lambda) &= \frac{\frac{\Gamma(n^k_{w_{d,i}}+1+\beta)}{\Gamma(1+\sum_w n^k_w+\beta)}\frac{\Gamma(n^d_k+1+\alpha)}{\Gamma(1+\sum_{k'} n^d_{k'}+\alpha)}}{\frac{\Gamma(n^k_{w_{d,i}}+\beta)}{\Gamma(\sum_w n^k_w+\beta)}\frac{\Gamma(n^d_k+\alpha)}{\Gamma(\sum_{k'} n^d_{k'}+\alpha)}} \quad \text{if } x_{d,i}=0 \\
P(z_{d,i}=k|\boldsymbol{z}-z_{d,i},\boldsymbol{x},\boldsymbol{c},\boldsymbol{w}|\alpha,\beta,\lambda) &= \frac{\frac{\Gamma(n^{c,k}_{w_{d,i}}+1+\beta)}{\Gamma(1+\sum_w n^{c,k}_w+\beta)}\frac{\Gamma(n^d_k+1+\alpha)}{\Gamma(1+\sum_{k'} n^d_{k'}+\alpha)}}{\frac{\Gamma(n^{c,k}_{w_{d,i}}+\beta)}{\Gamma(\sum_w n^{c,k}_w+\beta)}\frac{\Gamma(n^d_k+\alpha)}{\Gamma(\sum_{k'} n^d_{k'}+\alpha)}} \quad \text{if } x_{d,i}=1
\end{aligned} \tag{15}$$

where the n variables denote counts which exclude $z_{d,i}$. The counts in the numerator have an additional $+1$ because the counts in the joint distribution do include the assignment $z_{d,i} = k$.

A property of the Gamma function is that $\Gamma(x + 1) = x\Gamma(x)$. Using this property, we can rearrange Eq. 15 as:

$$\frac{\frac{\Gamma(a+1)}{\Gamma(b+1)}\frac{\Gamma(c+1)}{\Gamma(d+1)}}{\frac{\Gamma(a)}{\Gamma(b)}\frac{\Gamma(c)}{\Gamma(d)}} = \frac{\frac{a\Gamma(a)}{b\Gamma(b)}\frac{c\Gamma(c)}{d\Gamma(d)}}{\frac{\Gamma(a)}{\Gamma(b)}\frac{\Gamma(c)}{\Gamma(d)}} = \frac{a}{b}\frac{c}{d} \tag{16}$$

Substituting a, b, c, d with the count expressions in Eq. 15 gives the sampling equation 3.

Sampling x To sample a value for $x_{d,i}$, we need to derive the following conditional probability:

$$P(x_{d,i}|\boldsymbol{z}, \boldsymbol{x} - x_{d,i}, \boldsymbol{c}, \boldsymbol{w}; \alpha, \beta, \lambda) = \frac{P(\boldsymbol{z}, \boldsymbol{x}, \boldsymbol{c}, \boldsymbol{w}|\alpha, \beta, \lambda)}{P(\boldsymbol{z}, \boldsymbol{x} - x_{d,i}, \boldsymbol{c}, \boldsymbol{w}|\alpha, \beta, \lambda)} \tag{17}$$

As before, all of the terms in the joint (Eq. 13) will cancel out except those which involve $x_{d,i}$. This leaves us with:

$$P(x_{d,i} = 0|\boldsymbol{z}, \boldsymbol{x} - x_{d,i}, \boldsymbol{c}, \boldsymbol{w}; \alpha, \beta, \lambda) = \frac{P(x = 0|\lambda)\frac{\Gamma(n^k_{w_{d,i}}+1+\beta)}{\Gamma(1+\sum_w n^k_w+\beta)}}{\frac{\Gamma(n^k_{w_{d,i}}+\beta)}{\Gamma(\sum_w n^k_w+\beta)}} \tag{18}$$

$$P(x_{d,i} = 1|\boldsymbol{z}, \boldsymbol{x} - x_{d,i}, \boldsymbol{c}, \boldsymbol{w}; \alpha, \beta, \lambda) = \frac{P(x = 1|\lambda)\frac{\Gamma(n^{c,k}_{w_{d,i}}+1+\beta)}{\Gamma(1+\sum_w n^{c,k}_w+\beta)}}{\frac{\Gamma(n^{c,k}_{w_{d,i}}+\beta)}{\Gamma(\sum_w n^{c,k}_w+\beta)}}$$

where $P(x = 0|\lambda) = 1 - \lambda$ and $P(x = 1|\lambda) = \lambda$. As before, by using the property that $\Gamma(x + 1) = x\Gamma(x)$ and canceling terms, we obtain the sampling equation 4.

References

[1] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. Journal of Machine Learning Research, 3:993–1022, Jan 2003.

[2] A. Asuncion, M. Welling, P. Smyth, and Y.W. Teh. On Smoothing and Inference for Topic Models. Uncertain in Artificial Intelligence, 2009.

[3] T. Griffiths, and M. Steyvers. Finding scientific topics. Proceedings of the National Academy of Sciences, Apr 2004.