10-605 HW 7: Gibbs Sampling LDA using a Parameter Server

Due Tuesday, December 6, 23:59 via Autolab

Out Tuesday November 29, 2016

Policy on Collaboration among Students

These policies are the same as were used in Dr. Rosenfeld's previous version of 10601 from 2013. The purpose of student collaboration is to facilitate learning, not to circumvent it. Studying the material in groups is strongly encouraged. It is also allowed to seek help from other students in understanding the material needed to solve a particular homework problem, provided no written notes are shared, or are taken at that time, and provided learning is facilitated, not circumvented. The actual solution must be done by each student alone, and the student should be ready to reproduce their solution upon request. The presence or absence of any form of help or collaboration, whether given or received, must be explicitly stated and disclosed in full by all involved, on the first page of their assignment. Specifically, each assignment solution must start by answering the following questions in the report:

- Did you receive any help whatsoever from anyone in solving this assignment? Yes / No. If you answered 'yes', give full details: ________________ (e.g. "Jane explained to me what is asked in Question 3.4")
- Did you give any help whatsoever to anyone in solving this assignment? Yes / No. If you answered 'yes', give full details: ________________ (e.g. "I pointed Joe to section 2.3 to help him with Question 2".

Collaboration without full disclosure will be handled severely, in compliance with CMU's Policy on Cheating and Plagiarism. As a related point, some of the homework assignments used in this class may have been used in prior versions of this class, or in classes at other institutions. Avoiding the use of heavily tested assignments will detract from the main purpose of these assignments, which is to reinforce the material and stimulate thinking. Because some of these assignments may have been used before, solutions to them may be (or may have been) available online, or from other people. It is explicitly forbidden to use any such sources, or to consult people who have solved these problems before. You must solve the homework assignments completely on your own. I will mostly rely on your wisdom and honor to follow this rule, but if a violation is detected it will be dealt with harshly. Collaboration with other students who are currently taking the class is allowed, but only under the conditions stated below.

1 Important Note

Yulan Huang (yulanh@andrew.cmu.edu), and Jingyuan Liu (jingyual@andrew.cmu.edu) are the contact TAs for this assignment. Please post clarification questions to Piazza, and the instructors can be reached at the following email address: `10605-Instructors@cs.cmu.edu`.

2 Background

2.1 Parameter Server and JBosen

Parameter server (PS) is an abstraction for distributed, data-parallel ML. Each worker holds a partition of the dataset but has access to the full model parameters stored on the server machines through a fully connected bipartite network topology (Fig. 1). ML application programmers are presented with a distributed shared memory (DSM) abstraction through a key-value store interface. In Petuum's Bosen PS we expose a table interface where each row is accessed via a row ID. Note that servers and workers are logical components. Since a worker uses more CPU while a server uses more network, in practice it is often convenient to co-locate a server and a client on each node to increase resource utilization, which is currently what we do in Bosen.

In this assignment we will implement Gibbs sampling for LDA using JBosen, a minimal Java version of Bosen. JBosen exposes the same interface as Bosen, which is a hash-table view into the parameter server. Each table maps row IDs (integers) to rows, where each row contains columns of elements. Rows can be either sparse (implemented using a map) or dense (implemented using an array), depending on the performance requirements of the application. Custom rows can also be defined, but this is a more advanced feature that we will not use in this assignment. A table has two main API methods:

- `table.get(row_id, col_id)` retrieves a value from the table. This value might be a stale value within the staleness bound set by the application.
- `table.inc(row_id, col_id, i)` increments a value in the table by `i`. Because of Stale Synchronous Parallel (SSP) execution, this change may not immediately be reflected in the table. Even when using BSP, the change will only be reflected in the table on the next iteration.

anyJBosen also provides two important methods that are not associated with table:

- `PsTableGroup.clock()` informs JBosen to proceed to the next clock/iteration. JBosen will take care of managing consistency so that values in the table are not more than `s` clocks out of date. More specifically, when a worker thread calls clock(), the system will:

  1) Send all updates applied to tables by the thread to the servers. (Whenever you call table.inc(), the update is actually cached locally until you call clock());

  2) Block until all other worker threads have reached clock $c - s + 1$, where c is the clock at which clock() was called for the current thread, and s is the SSP staleness bound. Doing this maintains proper SSP execution.

- `PsTableGroup.globalBarrier()` forces tables to be completely synchronized. After this call exits, all workers will be at the same spot in the code and will have the freshest values in the tables.

You may find the source code for JBosen, as well as detailed documentation, at `https://github.com/petuum/jbosen`.

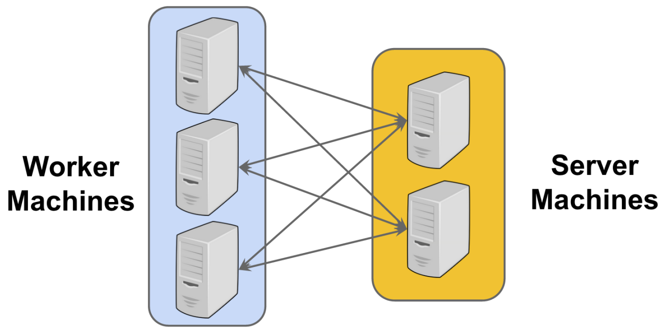

Figure 1: Parameter Server logically consists of a set of server nodes and worker nodes forming a fully connected bipartite graph.

2.2 Stale Synchronous Parallel (SSP)

Since the network is much slower than main memory reference ($10^4 \sim 10^5$x), reading parameters off server nodes for every access is a big bottleneck. We can cache them by storing a local copy in the main memory. There are various ways to synchronize different copies of parameters. Bulk-synchronous parallel (BSP) is a popular one used by Hadoop/Spark. However, BSP could suffer from stragglers and does not overlap communication and computation time effectively. Fully asynchrounous is fast but has no theoretical guarantee. This is where bounded delay consistency models come to the rescue.

JBosen implements Staleness Synchronous Parallel (SSP) [1]. The SSP consistency model ensures bounded staleness when reading the model state from the PS (Fig. 2).

Specifically, given a worker at logical clock c, reading any parameter must return a value that has been updated by all updates computed before and during clock $c - s - 1$, where s is the staleness threshold. Such a guarantee is provided by using a vector clock (one clock per worker) for each parameter in the client library. To reduce the memory overhead of one vector clock per parameter, we restrict clock to be called on all parameters and therefore only one vector clock is needed. When PS cannot provide parameter satisfying the staleness requirement (e.g,. a straggler is more than s clocks behind), the reader is blocked until the parameter is properly updated.

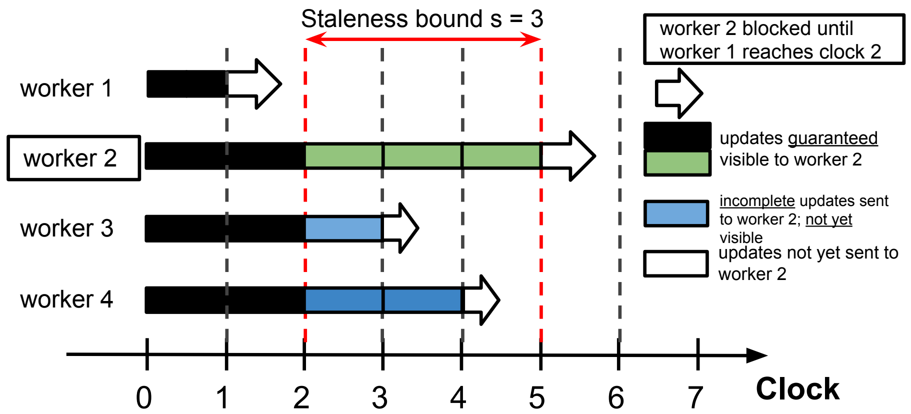
Staleness bound s = 3
worker 2 blocked until worker 1 reaches clock 2
worker 1
worker 2
worker 3
worker 4
updates guaranteed visible to worker 2
incomplete updates sent to worker 2; not yet visible
updates not yet sent to worker 2
0
1
2
3
4
5
6
7
Clock

Figure 2: An execution of PS under Stale Synchronous Parallel (SSP). Here staleness is 3, black and green denotes the updates that are visible to worker 2 (green updates are visible due to read-my-write consistency), while blue updates *could* be visible by worker 2 but is unspecified under SSP. Worker 2 is blocked at clock 5 because worker 1 hasn't finished clock 1, violating the SSP constraint. Specifically, worker 2 is blocked because updates from worker 1 clock 1 is not visible to worker 2 yet, and thus worker 2's read at clock 5 is blocked.

2.3 LDA with Gibbs Sampling

Just a quick review of the problem and algorithm. Latent Dirichlet Allocation is a generative model that represents set of documents as mixtures of topics or clusters. In case of textual data, the likelihood of occurrence of words in the entire corpus can be well represented by a multinomial distribution. The Dirichlet distribution being a conjugate of the multinomial distribution blends well with the overall generative story of the mixture models in Latent Dirichlet Allocation. The plate notation is shown in Fig. 3. ϕ captures word-topic distribution for each topic k $\in$ K, θ captures document-topic distribution for each document d $\in$ M, z is the latent variable which estimates the probability of a word w in a document lying in cluster k. The probability distribution models and a Gibbs sampler over all the variables are described below

```
$n_{kv}, n_{dk} \leftarrow [][], n_k, n_d \leftarrow []$
for all documents $m \in [1, M]$ do
    for all words $n \in [1, N_m]$ in document m do
        sample topic index $z_{m,n} = k \sim Multi(1/K)$
        increment document-topic count : $n_{dk}[d, z] + 1$
        increment topic-word count : $n_{kv}[z][n] + 1$
        increment topic-word sum : $n_k[z] + 1$
    end for
end for
while $t \leq |iterations|$ do
    for all documents $m \in [1, M]$ do
        for all words $n \in [1, N_m]$ in document m do
            $z \leftarrow z_{m,n}$
            $n_{dk}[d, z]--$
            $n_{kv}[z, n]--$
            $n_k]k]--$
            $p(z_i = k|z_{\neg i,words}) = \frac{n_k v[][n]+\beta}{n_k+\beta*V}(n_{dk}[d][] + \alpha)$
            $p(z_i = k|z_{\neg i,words})/ = \sum p(z_i = k|z_{\neg i,words})$
            $z \leftarrow$ sample a topic from posterior p(z)
            $n_{dk}[d, z] + +$
            $n_{kv}[z, n] + +$
            $n_k]k] + +$
        end for
    end for
end while
```

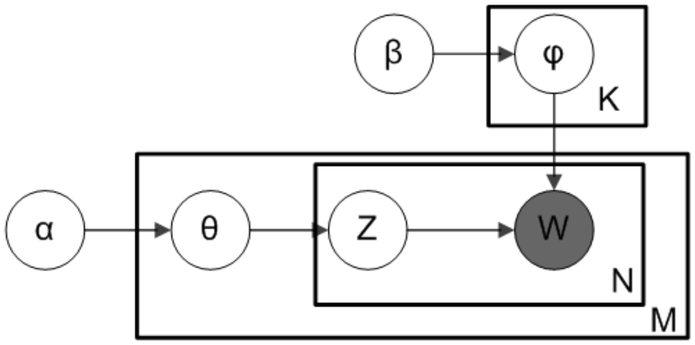

Figure 3: Plate notation for Latent Dirichlet Allocation.

3 Getting Started

You are provided with a starter kit for this assignment, which takes care of boilerplate code, and provides scripts to compile and run your implementation. It also contains a dataset you can use to test your code. You can find it at `http://curtis.ml.cmu.edu/w/courses/images/7/70/Lda.zip`.

- **Boilerplate code.** You are provided with two complete Java files, `Lda.java` and `DataLoader.java`. `Lda.java` provides some initialization code and `DataLoader.java` provides code for loading data. You are also provided with an incomplete Java file, `LdaApp.java`, which has some initialization and code stubs. This is where you should complete your implementation. By default, the provided code reads data from `20news.csv` and outputs the word-topic table to `word-topic.csv`, as well as the likelihood per iteration data to `likelihood.csv`. These can be changed using arguments to `lda_run.py`.

- **Gradle script.** The starter kit contains a stand-alone version of Gradle, which is a Java dependency management and compilation framework. Simply run

  ```
  ./gradlew fatJar
  ```

  to compile your program. This will automatically download dependencies, including JBosen, compile your program, and puts everything into a single jar file.

- **Run/kill scripts.** `lda_run.py` and `lda_kill.py` make it easy to test your program using multiple nodes. After compiling your program using Gradle, simply run

  ```
  python lda_run.py
  ```

  to launch a set of nodes specified in `hosts.txt`. `lda_run.py` also handles the command line arguments passed to your LDA implementation. For more information,

  ```
  python lda_run.py --help
  ```

  Hint: If you are getting an "address already in use" error, try using `lda_kill.py` to terminate any left-over processes first. If `lda_kill.py` does not work, SSH to each machine and run `killall java`.

- **Host file.** `hosts.txt` is the default location that `lda_run.py` looks for a list of hosts. Each line of the file specifies an endpoint, in the form `<ip>:<port>`, that you wish to launch a JBosen node on.

  Hint: If you are testing with multiple processes on the same machine, make sure to space out the ports used. JBosen may bind to several ports adjacent to the port specified.

- **Dataset.** `20news.csv` and `vocab.csv` contain a processed version of the 20 newsgroups dataset, with stop words and E-mail headers removed. Words in `20news.csv` are expressed as the index of the word in `vocab.csv`, so that `0` represents the first word in `vocab.csv`, `1` represents the second word, and so on. You may use this dataset to test your program and it will also be used in the questions you will answer.

- **AWS Experiments.** Please follow the guidelines given here `https://goo.gl/fVTzI5` for running experiments on AWS. If you have almost used up your AWS credits, please send an email to Bhuwan (bdhingra@andrew.cmu.edu) with your detailed AWS credit usage, including left credits and which homeworks you spent the credits on. You may get more credits based on this.

4 Tasks and Questions

1. **Implementation** Starting from the code in the starter kit, complete the implementation of LDA Gibbs sampling using the algorithm described above. To test your program, you may use the dataset provided with the starter kit.

2. **Topics learned by LDA [20 pts]** Run your implementation of LDA on the dataset provided in the starter kit, `20news.csv`, using 20 topics, $\alpha = 0.1$ and $\beta = 0.1$. You should run it using 1 worker with 1 thread for 100 iterations, 25 clocks per iteration, with a staleness bound of 0.

   (a) Report the plot log-likelihood values vs. the iteration number. In your plot, logarithmically scale the likelihood.

   (b) For each topic learned by your implementation, list the 5 words with the highest probability. You may use `vocab.txt` to retrieve the textual words. The actual topics of each newsgroup can be found at `http://qwone.com/~jason/20Newsgroups/`. Can you identify any of these from the topics you learned (you should be able to identify a few of them)? If so, which ones? As a sanity check for your results, the top 3 terms learned by our solution for a selection of topics are:

      - bike, car, bmw
      - hocky, season, teams
      - religion, atheism, atheists

3. **Effect of staleness [20 pts]** In this question, you will explore how the staleness bound affects the convergence rate of LDA. Run five experiments, using `staleness` settings 0, 2, 4, 8, and 16 on 4 m3.xlarge instances. Use the following settings for running your experiments:

   - `alpha = 0.1`

- `beta = 0.1`
- `num_topics = 20`
- `num_iterations = 30`
- `num_clocks_per_iteration = 25`

(a) Plot the log-likelihood values vs. iteration number. Which staleness setting converges the fastest per iteration?

(b) Plot the log-likelihood values vs. the total time in seconds. Which staleness setting converges the fastest per second?

(c) Explain the reason behind your results in terms of freshness of parameters, overhead of synchronization, and convergence per iteration.

(d) What results would you expect if the application is more sensitive to staleness? Less sensitive to staleness?

4. **Effect of clock frequency [20 pts]** In this question, you will explore how the frequency of synchronization affects the convergence rate of LDA. Run five experiments on 4 m3.xlarge machines, using `num_clocks_per_iteration` settings 1, 5, 10, 50, and 100.

   - `alpha = 0.1`
   - `beta = 0.1`
   - `num_topics = 20`
   - `num_iterations = 30`
   - `staleness = 0`

   (a) Plot the log-likelihood values vs. iteration number. Which clock frequency setting converges the fastest per iteration?

   (b) Plot the log-likelihood values vs. the total time in seconds. Which clock frequency setting converges the fastest per second?

   (c) Explain the reason behind your results in terms of freshness of parameters, overhead of synchronization, and convergence per iteration.

   (d) What results would you expect if the application is more sensitive to staleness? Less sensitive to staleness?

5. **Collaboration Policy** Answer the questions in the collaboration policy on page 1.

5 Deliverables

Set the argument `num_local_worker_threads` to 4 and submit the four likelihood files generated. Submit a tar file containing 5 files: `LdaApp.java, likelihood*.csv` all at the top level directory via AutoLab to HW7: LDA. Submit your report.pdf to HW7: Report. You should implement the algorithm by yourself instead of using any existing machine learning toolkit.

References

[1] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2013.