Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Accelerating distributed Expectation–Maximization algorithms with frequent updates[☆]

Jiangtao Yin[a,*], Yanfeng Zhang[b], Lixin Gao[a]

[a] University of Massachusetts Amherst, 151 Holdsworth Way, Amherst, MA 01003, USA
[b] Northeastern University, 11 Wenhua Road, Shenyang, Liaoning 110819, China

HIGHLIGHTS

- Propose two approaches to parallelize EM algorithms with frequent updates in a distributed environment to scale to massive data sets.
- Prove that both approaches maintain the convergence properties of the EM algorithms.
- Design and implement a distributed framework to support the implementation of frequent updates for the EM algorithms.
- Extensively evaluate our framework on both a cluster of local machines and the Amazon EC2 cloud with several popular EM algorithms.
- The EM algorithms with frequent updates implemented on our framework can converge much faster than traditional implementations.

ARTICLE INFO

Article history:
Received 29 December 2016
Received in revised form 1 June 2017
Accepted 20 July 2017
Available online 29 July 2017

Keywords:
Expectation–Maximization
Frequent updates
Concurrent updates
Distributed framework
Clustering
Topic modeling

ABSTRACT

Expectation–Maximization (EM) is a popular approach for parameter estimation in many applications, such as image understanding, document classification, and genome data analysis. Despite the popularity of EM algorithms, it is challenging to efficiently implement these algorithms in a distributed environment for handling massive data sets. In particular, many EM algorithms that frequently update the parameters have been shown to be much more efficient than their concurrent counterparts. Accordingly, we propose two approaches to parallelize such EM algorithms in a distributed environment so as to scale to massive data sets. We prove that both approaches maintain the convergence properties of the EM algorithms. Based on the approaches, we design and implement a distributed framework, FreEM, to support the implementation of frequent updates for the EM algorithms. We show its efficiency through two categories of EM applications, clustering and topic modeling. These applications include k-means clustering, fuzzy c-means clustering, parameter estimation for the Gaussian Mixture Model, and variational inference for Latent Dirichlet Allocation. We extensively evaluate our framework on both a cluster of local machines and the Amazon EC2 cloud. Our evaluation shows that the EM algorithms with frequent updates implemented on FreEM can converge much faster than those implementations with traditional concurrent updates.

1. Introduction

Discovering knowledge from a large collection of data sets is one of the most fundamental problems in many applications, such as image understanding, document classification, and genome data analysis. Expectation–Maximization (EM) [8] is one of the most popular approaches in these applications [5,9,32]. It estimates parameters for hidden variables by maximizing the likelihood. EM is an iterative approach that alternates between performing an Expectation step (E-step), which computes the distribution for the hidden variables using the current estimates for the parameters, and a Maximization step (M-step), which re-estimates parameters to be those maximizing the likelihood found in the E-step.

Due to the popularity, many methods for accelerating EM algorithms have been proposed. Some of them [18,21] show that a partial E-step may accelerate convergence. Such a partial E-step selects only a subset of data points for computing the distribution. The advantage of the partial E-step is to allow the M-step to be performed more frequently, so that the algorithm can leverage more up-to-date parameters to process data points and potentially accelerates convergence. Intuitively, updating the parameters frequently might incur additional overhead. However, the parameters typically depend on statistics of data sets, which can be computed

☆ Part of this work has been published in Proceedings of CLUSTER'12: 2012 IEEE International Conference on Cluster Computing (Yin et al., 2012).
* Corresponding author.
E-mail addresses: jyin@ecs.umass.edu (J. Yin), zhangyf@cc.neu.edu.cn (Y. Zhang), lgao@ecs.umass.edu (L. Gao).

http://dx.doi.org/10.1016/j.jpdc.2017.07.005
0743-7315/© 2017 Elsevier Inc. All rights reserved.

incrementally. That is, the cost of computing statistics grows linearly with the number of data points whose statistics have been changed in the E-step. As a result, performing frequent updates on the parameters does not necessarily introduce considerable additional cost. We refer to the EM algorithm that updates the parameters frequently as *the EM algorithm with frequent updates*. In contrast, the traditional EM algorithm, which computes the distribution for all data points and then updates the parameters, is referred to as *the EM algorithm with concurrent updates*.

Despite the fact that the EM algorithm with frequent updates has the potential to speedup convergence, parallelizing it can be challenging. Although computing the distribution and updating statistics can be performed concurrently, parameters such as centroids of clusters are global parameters. Updating these global parameters has to be performed in a centralized location and all workers have to be synchronized. Synchronization in a distributed environment may incur considerable overhead. Therefore, we have to control the frequency of parameter updates to achieve a good performance.

In this paper, we propose two approaches to parallelize the EM algorithm with frequent updates in a distributed environment: partial concurrent and subrange concurrent. In the *partial concurrent* approach, each E-step processes only a block of data points. The size of a block controls the frequency of parameter updates. In the *subrange concurrent* approach, each E-step computes the distribution in a subrange of hidden variables instead of the whole range. The subrange size can determine the frequency of parameter updates. We prove that both approaches maintain the convergence properties of EM algorithms. We control the parameter update frequency by setting the block/subrange size, and provide strategies to determine the optimal values. Additionally, both approaches can scale to any number of workers/processors.

We design and implement a distributed framework, FreEM, for implementing the EM algorithm with frequent updates based on the two proposed approaches. FreEM eases the process of programming EM algorithms in a distributed environment. Programmers only need to specify the E-step and the M-step. The detailed mechanisms, such as data distribution, communication among workers, and frequency of M-step, are all handled automatically. As a result, it facilitates the process of implementing EM algorithms and accelerates the algorithms through frequent updates. We evaluate FreEM in the context of a wide class of well-known EM applications: k-means clustering, fuzzy c-means clustering, parameter estimation for the Gaussian Mixture Model, and Latent Dirichlet Allocation for topic modeling. Our results show that the EM algorithm with frequent updates can run much faster than that with traditional concurrent updates. In addition, EM algorithms can be implemented on FreEM in a more efficient way than on Hadoop [13], an open source implementation of the popular programming model MapReduce [7].

The rest of this paper is organized as follows. Section 2 describes the EM algorithm with frequent updates. Section 3 exemplifies frequent updates through EM applications. Section 4 presents our approaches to parallelize the EM algorithm with frequent updates. In Section 5, we present the design, implementation and API of FreEM. Section 6 is devoted to the evaluation results. Finally, we discuss related work in Section 7 and conclude the paper in Section 8.

2. EM algorithms

In a statistical model, suppose that we have observed the value of one random variable, X, which comes from a parameterized family, $P(X|\theta)$. The value of another variable, Z, is hidden. Given the observed data, we wish to find θ such that $P(X|\theta)$ is the maximum. In order to estimate θ, it is typical to introduce the log likelihood function: $L(\theta) = \log P(X|\theta)$. Suppose the data consists of n independent data points $\{x_1, \ldots, x_n\}$, and thereby the hidden variable can be decomposed as $\{Z_1, Z_2, \ldots, Z_n\}$. Then, $L(\theta) = \sum_{i=1}^{n} \log P(x_i|\theta)$. We assume that Z has a finite range for simplicity, but the result can be generalized. Thus, the probability $P(x_i|\theta)$ can be written in terms of possible value (z_i) of the hidden variable Z_i as: $P(x_i|\theta) = \sum_{z_i} P(x_i, z_i|\theta)$. When it is hard to maximize $L(\theta)$ directly, an EM algorithm is usually used to maximize $L(\theta)$ iteratively.

The EM algorithm leverages an iterative process to maximize $L(\theta)$. Each iteration consists of an E-step and a M-step. The E-step leverages the data points and the current estimates of the parameters to estimate the distribution of hidden variables. The M-step updates the parameters to be those maximizing the likelihood found in the E-step.

One classic example of the EM algorithm is k-means clustering [16]. It aims to partition n data points $\{x_1, x_2, \ldots, x_n\}$ into k ($k \le n$) clusters $\{c_1, c_2, \ldots, c_k\}$ so as to minimize the objective function:

$$f = \sum_{i=1}^{k} \sum_{x_j \in c_i} \|x_j - \mu_{c_i}\|^2, \tag{2.1}$$

where $\mu_{c_i} = \frac{1}{|c_i|}\sum_{x_j \in c_i} x_j$ is the centroid of cluster c_i.

The E-step of k-means assigns points to the cluster with the closest mean. That is, a data point x_j is assigned to cluster c if $c = \arg\min_j \|x_i - \mu_{c_j}\|^2$. Its M-step updates the centroids (parameters) for all clusters.

2.1. The EM algorithm with concurrent updates

The EM algorithm with concurrent updates computes the distribution for all data points in its E-step. Formally, let Q_i be some distribution over z_i ($\sum_{z_i} Q_i(z_i) = 1$, $Q_i(z_i) \ge 0$). Such an EM algorithm starts with some initial guess at the parameters $\theta^{(0)}$, and then seeks to maximize $L(\theta)$ by iteratively applying the following two steps:

E-step: For each $x_i \in X$, set $Q_i(z_i) = P(z_i|x_i, \theta^{(t-1)})$.

M-step: Set $\theta^{(t)}$ to be the θ that maximizes $\sum_{i=1}^{n} E_{Q_i}[\log P(x_i, z_i|\theta)]$.

Here, the expectation E_{Q_i} is taken with respect to the distribution $Q_i(\cdot)$ over the range of Z in the E-step.

The vanilla k-means (Lloyd's algorithm [15]) belongs to this category. Its E-step performs cluster assignment for each data point, and it M-step updates the centroids along the direction of minimizing the objective function.

2.2. The EM algorithm with frequent updates

The EM algorithm with frequent updates attempts to accelerate the convergence by frequently updating the parameters. This algorithm can provide more up-to-date parameters to process data points and to potentially speedup convergence. However, updating parameters frequently may incur significant overhead if the update is done in the original way. In order to conquer this obstruction, we introduce a way of updating parameters incrementally. In the EM algorithm, the distribution influences the likelihood of the parameters via some sufficient statistics. The statistics is usually the summation over the statistics on each individual data point, and a summation can be incrementally updated. As a result, the cost of computing the sufficient statistics grows linearly with the number of data points whose statistics have been changed in the E-step.

Take k-means for instance. Let S_i ($S_i = \sum_{x_j \in c_i} x_j$) and W_i ($W_i = |c_i|$) be the statistics. The centroid of one cluster (*e.g.*, i) can be easily