Implementation of an Alternative Scaling for Baum-Welch Algorithm for HMM Training in MapReduce

Manogna Vemulapati

Introduction

The Baum-Welch algorithm is used for unsupervised training of a Hidden Markov Model (HMM). The Apache Mahout project has an implementation of a basic sequential version of Baum-Welch algorithm. The limitation of this algorithm is that in order to train the HMM on a large number of training sequences, the training sequences have to be processed serially. An implementation of Baum-Welch algorithm for parallel HMM training which is based on Hadoop's MapReduce has been proposed for Apache Mahout in [1]. One common problem with Baum-Welch algorithm is that when it trains on long observation sequences, numerical underflow errors occur. Both the Apache Mahout's basic implementation and the MapReduce version of Baum-Welch in [1] offer a scaling option which is known as log-scaling in order to make the Baum-Welch algorithm numerically stable. In spite of this, the MapReduce version still has numerical stability issues when training on long sequences. The goal of this project is to implement an alternative scaling method for MapReduce version of Baum-Welch algorithm.

Scaling Method for HMM Training

The scaling method implemented in this project is described in detail in [2]. The first part of this project is to implement the new scaling method in the basic Baum-Welch implementation in Mahout.

The Baum-Welch algorithm is an iterative algorithm which revises the HMM model parameters in each iteration until a local maximum is reached. In each iteration, it computes the forward variables and backward variables. The forward variables are denoted $\alpha_t(i)$ where $0 \leqslant t \leqslant T-1$ and $0 \leqslant i \leqslant N-1$ for an observation sequence of length T and N hidden states. The backward variables $\beta_t(t)$ are defined similarly. In each iteration, the Baum-Welch algorithm uses the forward and backward variables to revise the HMM model parameters, namely, the transition probability matrix, the emission probability matrix and the initial probability matrix.

In the scaling scheme described in [2], the forward variables are scaled using scaling factors c_t in each iteration. There are T scaling factors where each scaling factor depends only on the time step t and is independent of the hidden state. In each iteration, the same scaling factors computed for forward variables are used to scale the backward variables. The scaled forward variables are denoted by $\hat{\alpha}_t(i)$ and the scaled backward variables are denoted by $\hat{\beta}_t(i)$.

The full implementation details for Mahout's implementation are given in [3]. The code is hosted in the GitHub repository [4].

Scaling Method for HMM Training in MapReduce

The parallel version of Baum-Welch algorithm to run on MapReduce is described in [1]. The basic idea is as follows. The unit of parallelism is an independent training observation sequence. Each iteration of the Baum-Welch algorithm is a MapReduce job. Each map task works on an independent training sequence. In other words, the key value pair input to each map task consists of a unique Key id and the observation sequence. The map task computes the forward and backward variables and emits intermediate key value pairs. Each map task emits $2N+1$ unique intermediate keys and their associated values. These intermediate keys are: an intermediate key called INITIAL which contains the array of initial state probabilities. There are N intermediate keys with the prefix TRANSIT_. The intermediate key of the form TRANSIT_i has the value which is array of transition probabilities to each of the N hidden states. Similarly, N intermediate keys of the form EMIT_ are output by each map task. Each EMIT_ key has the probability distribution for output symbols. The reduce tasks process the intermediate keys emitted by map tasks and revise the HMM's model parameters (transition probabilities matrix, emission probabilities matrix and the initial state probabilities) for the next iteration. There are $2N+1$ reduce tasks each working in parallel on an intermediate key.

The formula to revise HMM's model parameters in the case of multiple observation sequences is given in [2]. When the scaling described above is used, the equations are given below.

When there are L independent observation sequences the equation to update the element a_{ij} of the transition probabilities matrix with scaling is given below. The updated value is $\bar{a}_{ij}$.

$$\bar{a}_{ij} = \frac{\sum_{l=0}^{L-1} \sum_{t=0}^{T_l-2} \hat{\alpha}_t^l(i) a_{ij} b_j(O_{t+1}^l) \hat{\beta}_{t+1}^l(j)}{\sum_{l=0}^{L-1} \sum_{t=0}^{T_l-2} \hat{\alpha}_t^l(i) \hat{\beta}_t^l(i) / c_t^l}$$

Each map task now emits an intermediate key of the form TRANSIT_i for hidden state i which is an array of pairs of doubles of the form (numerator, denominator) with one pair of doubles for each hidden state j. The numerator and denominator are as given below:

$$numerator_{ij} = \sum_{t=0}^{T_l-2} \hat{\alpha}_t^l(i) a_{ij} b_j(O_{t+1}^l) \hat{\beta}_{t+1}^l(j)$$

$$denominator_{ij} = \sum_{t=0}^{T_l-2} \hat{\alpha}_t^l(i) \hat{\beta}_t^l(i) / c_t^l$$

Each reduce task performs the outer summation for all numerators and the outer summation for all denominators and performs the division to obtain the value $\bar{a}_{ij}$ as below.

$$\bar{a}_{ij} = \frac{\sum_{l=0}^{L-1} numerator_{ij}^l}{\sum_{l=0}^{L-1} denominator_{ij}^l}$$

The emission probabilities matrix and the initial probabilities matrix are also updated similarly.

The code is hosted in the GitHub repository in [5].

An Example: Finding CpG Islands in a DNA Sequence

In order to validate if the new scaling method is numerically stable when the HMM trained on a large training dataset, I have written a program to find the location of CpG islands in a large DNA sequence such as human chromosome.

A DNA sequence is a sequence of symbols: C,G,T and A. In order to locate the CpG islands in a given DNA sequence, a HMM is modeled with 8 states: C+, G+, T+, A+ (for C,G,T and A within a CpG island) and C-, G-, T- and A- (for bases outside an island). A hidden state C+ or C- emits output symbol c with probability 1 and similarly for other hidden states.

A given training sequence such as a human chromosome is split into chunks of fixed size and each chunk is treated as an independent observation sequence to be processed by a map task.

The HMM is trained with MapReduce with the new scaling method and it has successfully trained on chromosomes of the order of hundred million bases long without any numerical underflow errors. The trained HMM has been used to successfully decode the hidden state sequence (CpG islands) for a given DNA sequence.

The CpG island program is hosted in GitHub at [6].

References

1. Dhruv Kumar, “Baum-Welch Algorithm on Map-Reduce for Parallel Hidden Markov Model Training”, https://issues.apache.org/jira/browse/MAHOUT-627.
2. Dawei Shen, “Some Mathematics for HMM”, http://courses.media.mit.edu/2010fall/mas622j/ProblemSets/ps4/tutorial.pdf.
3. Manogna Vemulapati, "An alternative scaling method for Baum Welch for HMM.", https://manognavemulapati.github.io/mahout/HMMScaling.pdf.
4. Manogna Vemulapati, "Implementation of Alternative Scaling for HMM in Mahout.", https://github.com/manognavemulapati/mahout.
5. Manogna Vemulapati, "Implementation of Alternative Scaling for HMM in Training in MapReduce in Mahout.", https://github.com/manognavemulapati/Baum-Welch.
6. Manogna Vemulapati, "A program to find CpG islands in a DNA sequence using a HMM trained in MapReduce.", https://github.com/manognavemulapati/CpGIsland.