JMLR: Workshop and Conference Proceedings 21:249–253, 2012 The 11th ICGI

Treba: Efficient Numerically Stable EM for PFA

Mans Hulden MHULDEN@EMAIL.ARIZONA.EDU
Ikerbasque (Basque Science Foundation)

Abstract

Training probabilistic finite automata with the EM/Baum-Welch algorithm is computationally very intensive, especially if random ergodic automata are used initially, and additional strategies such as deterministic annealing are used. In this paper we present some optimization and parallelization strategies to the Baum-Welch algorithm that often allow for training of much larger automata with a larger number of observations. The tool, *treba*, which implements the optimizations, is available open-source and its results were used to participate in the PAutomaC PFA/HMM competition.

Keywords: EM, Baum-Welch, probabilistic automata, deterministic annealing.

1. Introduction

Treba is a basic tool for training and decoding probabilistic finite automata (PFA), in particular with variants of the EM/Baum-Welch algorithm.[1] The main design goals have been efficiency, numerical stability, and good parallelization of the learning algorithms. The EM algorithm requires some care in implementation and different implementation choices can produce tenfold differences in execution speed within the same implementation languages and hundredfold differences in execution speed across languages[2]. This, and the fact that open-source implementations of EM for PFSA are unavailable, have motivated this effort for a simple, generic, parallel implementation in the spirit of UNIX command-line tools. The tool is written in C and was originally designed to provide a strong EM baseline for the PAutomaC PFA/HMM learning competition (Verwer et al., 2012). In the following, we discuss some of the implementation choices, particularly from the point of view of efficiency as well as report on its participation in PAutomaC.

2. Features

Treba is a command line utility that contains the following features:

- PFA training (Baum-Welch, Viterbi), decoding, generation, and likelihood calculation.
- Various strategies to escape local optima of EM, including deterministic annealing (Rose, 1998), random restarts, and random initialization with various automaton topologies.
- Re-estimation algorithms with a fully concurrent implementation.

1. http://treba.googlecode.com
2. See e.g. the comparison between 11 implementations of the algorithm done by Aurélien Garivier at http://perso.telecom-paristech.fr/~garivier/code/index.php

3. Baum-Welch for PFSA

Treba implements a straightforward EM/Baum-Welch training for PFA with some augmentations and exceptions as discussed below. The Baum-Welch algorithm for PFA is conceptually slightly more straightforward than for HMMs as we need not worry about separate emission and transition probabilities. We refer the reader to de la Higuera (2010) for a detailed exposition of the algorithm in a PFA context, and for situations where training occurs with multiple observation sequences.

3.1. Deterministic annealing

Deterministic annealing (Rose, 1998) is a strategy whereby a globally concave function is first maximized, then gradually changed into a more non-concave function which is optimized progressively. In a Baum-Welch context, this is often implemented by including in the formulas a mutable parameter β representing how much weight is given to the training observations (Smith and Eisner, 2004). Re-estimation can then be performed as:

$$\hat{c}(q \xrightarrow{a} q') = \frac{[\text{expected count}(q \xrightarrow{a} q')]^{\beta}}{[\text{expected count}(q \xrightarrow{\Sigma} Q)]^{\beta}}$$

During training, the β parameter is initialized to a low value (≈ 0), and is increased by some predefined amount each time EM converges, until $\beta = 1$, when the above equation equals that of standard EM.

3.2. Parallellization

The Baum-Welch algorithm is eminently parallellizable with very few bottlenecks for concurrent implementation. In general, two straightforward options exist as to the choice of parallellization. Assuming a standard trellis representation of the states visited at each step in time, the necessary forward, backward, and re-estimation steps can be parallellized in this trellis-filling task with respect to the different states since at the states in the column representing time $t + 1$ depend only on the contents of the trellis at time t. Another option when training on multiple observations (as is needed for PAutomaC), is to launch n stand-alone threads where each separately updates the estimates based on c/n observations, where c is the number of observations used for training. Treba is parallelized according to the latter scheme, and in principle allows an unlimited number of threads to be launched.

3.3. Numerical instability

The foremost practical implementation issue arising with the Baum-Welch algorithm is that of numerical instability (Rabiner, 1989). In calculating a large number of conditional probabilities, numerical underflow occurs very quickly as the values of the terms involved decrease exponentially. This is usually addressed in one of two ways: (1) scaling up the terms periodically to prevent underflow as probabilities are multiplied (as is documented in Rabiner (1989)), or, (2) performing all calculations in log space.

While the log space solution has the advantage of simplicity, in particular in a concurrent implementation of the algorithm, it itself entails efficiency and numerical stability

problems. The crux of calculating the necessary terms in log space in Baum-Welch lies in the addition step in the forward, backward, and the E-step of Baum-Welch where probabilities need to be summed. Assuming log space values of x and y, we need to calculate $log(exp(x) + exp(y))$. This is in practice calculated by the simpler $x + log(exp(y - x) + 1)$. However, at this point two problems present themselves. Firstly, evaluating logarithms and exponentiations is very costly compared with simple additions or probabilities, which would be the equivalent operation if the scaling approach is chosen. Secondly, as $y - x$ gets very small, numerical stability issues arise in the evaluation of $log(exp(y - x) + 1)$. In fact, if IEEE 754 single-precision floating points are used in conjunction with the standard C math library implementations of *log* and *exp*, Baum-Welch is numerically unreliable.[3] With double precision floating points, numerical accuracy is maintained until $y - x \approx -40$, which is usually sufficient for practical purposes. However, using library functions for evaluating the addition in log space results in that roughly 80% of the running time of Baum-Welch is spent on evaluating logs and exponentials. This is clearly the reason why some authors have reported the log approach to be, while simpler to implement, much slower in practice (Mann, 2006).

3.4. Efficiency concerns: fast logspace summation

The fact that we can express probability addition in log space by $x + log(exp(y - x) + 1)$ (where $y - x$ can effectively be assumed to be in the range $[-54, 0]$, as we can swap x and y if $y > x$) offers various options for speeding up the calculation.[4] The obvious choice is to approximate $log(exp(z) + 1)$ with a lookup table in the range $[-54, 0]$, perhaps with linear interpolation between lookup steps. However, as we will see below, this type of table lookup becomes drastically inefficient because of CPU caching issues if the table is to be large enough to not cause undue errors in the approximation. Instead, we have chosen to perform a polynomial approximation of $log(exp(z) + 1)$. The coefficients of the polynomial were calculated for various intervals using the Remez exchange algorithm (Remez, 1934). After experiments with speed and accuracy tradeoffs, we settled for a 4th-order polynomial approximation. In practice, we have chosen to use interval sizes between 1/32 and 1, the accuracy of which are listed in table 1.

w	**[-w,0]**	**[-60,-60+w]**
1	2.94×10^{-7}	4.63×10^{-24}
1/2	4.97×10^{-9}	1.21×10^{-25}
1/4	7.94×10^{-11}	3.47×10^{-27}
1/8	1.24×10^{-12}	1.03×10^{-28}
1/16	1.95×10^{-14}	3.17×10^{-30}
1/32	3.05×10^{-16}	4.81×10^{-32}

Table 1: Maximum error in 4th order minimax polynomial approximation of $log(exp(z) + 1)$ depending on interval width and range.

4. Evaluation

Table 2 shows a timing comparison of some implementations of EM, all running the first PAutomaC competition problem with randomly initialized ergodic automata/HMMs of var-

3. It is for this reason many mathematics libraries have a special function log1p() to evaluate log(1+x).
4. The range is motivated by the fact that if the difference between x and y is larger than -53.52, the value of the expression becomes smaller than a 64-bit floating point machine epsilon $2^{-53} \approx 1.11 \times 10^{-16}$.

$\|Q\|$	**treba(MM)**	**treba(LUT)**	**treba(mathlib)**	**Ref HMM** (C++)	**PAutomaC baseline** (python)
5	**0.15s**	0.24s	0.50s	4.79s	29.6s
10	**0.61s**	0.83s	2.06s	17.34s	99.4s
20	**2.43s**	3.39s	8.72s	75.48s	339.11s

Table 2: Time taken per EM iteration of various implementations.

ious sizes.[5] All implementations were run single-threaded. For treba, we've included the minimax log summing implementation, a lookup table implementation (size 20,000), and the standard math library implementation. For comparison, Dekang Lin's C++ implementation of Baum-Welch for HMMs is included,[6] as well as the PAutomaC baseline, implemented in Python.

5. PAutomaC

Treba participated in the PAutomaC competition with EM-trained randomly initialized ergodic automata of various sizes: 5, 20, and 40 states. As is seen from table 3,[7] using perhaps even larger initial automata could have improved the results (larger ones were not calculated because of time constraints).[8]

5.1. Interpolating the test set

In addition to training the automata on the test set, a second training version was created where both the provided test sets and training sets were used for training, though with a twist. Since the test set had been stripped of duplicate words, we expanded the test set with an EM-like algorithm to estimate the number of duplicates in the original test set to avoid skewing the counts in favor of less frequent words. In effect, we create a new test set iteratively by calculating the *expected* number of occurrences (E-step) for each word in the test set based on how many times it occurred in the training set and test sets put together, and then modifying (M-step) the test set, adding occurrences to reflect the new estimate, i.e.:

$$\text{(E-step) } \hat{c}(w_i)^{Te} = \frac{c(w_i)^{Tr+Te}}{\sum_i (w_i)^{Tr+Te}} \quad \text{(M-step) } c(w_i)^{Te} = \text{Round}(\hat{c}(w_i))$$

This attempt to reconstruct the original duplicate-containing test set to glean new information is of course a general strategy that can be used together with any subsequent learning approach.[9]

5. On a 2.8MHz Intel Core 2 Duo.
6. http://webdocs.cs.ualberta.ca/~lindek/hmm.htm
7. The perplexity values given in table 3 are all non-interpolated (trained only on the training data) because of the natural unreliability of perplexity scores when the test set itself is used for training.
8. A 40-state automaton takes roughly an hour to train until log likelihood convergence ($\Delta \leq 0.1$)
9. For example, expanding the test set as above and using a simple trigram model consistently yielded better results than the baseline trigram approach provided by the PAutomaC competition, which also used the test set for training, but without any reconstruction of "missing duplicates."

References

C. de la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

T. P. Mann. Numerically stable hidden Markov model implementation. *Ms. Feb, 21. 2006.*, 2006.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

E. Y. Remez. Sur la détermination des polynômes d'approximation de degré donnée. *Comm. Soc. Math. Kharkov*, 10:41–63, 1934.

K. Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86(11):2210–2239, 1998.

N. A. Smith and J. Eisner. Annealing techniques for unsupervised statistical language learning. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 486. Association for Computational Linguistics, 2004.

S. Verwer, Rémi Eyraud, and Colin de la Higuera. PAutomaC: a PFA/HMM learning competition. In *ICGI 2012*, 2012.

Appendix A. Appendix

Problem #	**Number of states**		
	5	20	40
1	-9585	-9029	-8866
2	-6685	-6190	-6149
3	-6904	-6358	-6276
4	-5620	-4666	-4527
5	-8962	-4784	-4785
6	-11719	-8369	-7915
7	-7521	-4877	-4727
8	-8801	-7833	-7283
9	-13761	-6547	-5578
10	-16234	-14810	-12794

Table 3: Perplexity scores for the 10 first PAutomaC problems for various automata sizes. All automata are intialized randomly, and are fully connected.