Parallel Implementation of Baum-Welch Algorithm

Conference Paper · January 2006

CITATION
1

READS
62

2 authors, including:

Maxim Anikeev
Southern Federal University
35 PUBLICATIONS **71** CITATIONS

SEE PROFILE

Parallel Implementation of Baum-Welch Algorithm

M.V. Anikeev, O.B. Makarevich
Department of Information Security
Taganrog State University of Radio Engineering
Taganrog, Russia
e-mail: anikeev@users.tsure.ru, mak@tsure.ru

Abstract[1]

Besides the fact that hidden Markov models (HMMs) became a state-of-the-art technique for speech recognition applications, they find major use in other areas as well. Some problems require huge training sets for fitting HMMs to experimental data, which leads to increased complexity of training algorithms. We propose a simple strategy of organizing parallel HMM training, which can be effectively implemented using inexpensive network clusters.

1. Introduction

Hidden Markov models (HMMs) find major application in a variety of areas, which demand powerful pattern recognition techniques. These areas include speech processing and recognition [1, 2], communications [3], control [4, 5], image and character recognition [6, 7], and genetics [8]. HMMs are favoured due to their ability to describe many real-world processes, which do not strictly conform to the standard Markov assumption. At the same time, HMMs are not much more complex, than regular discrete Markov chains.

In recent years some attempts have been made to apply HMMs to various computer security problems [9, 10, 11, 12]. C. Warrender et al. [12] proposed the idea to use HMMs as normal behaviour profiles for security-critical processes. Such profiles are used to discover behaviour deviations of system processes, which are often caused by intrusion attempts. Unlike classic HMM applications, this approach demands much more training data to be collected to build adequate behaviour profiles. Programs behaviour often tends to be extremely diverse, thus in order to build a comprehensive profile one must observe program execution for a long time in different modes, and collect a lot of relevant data.

The authors of [12] applied a set of techniques to the detection of anomalies in traces of system calls. They compared HMMs with several other methods of data analysis, and found out that HMMs can usually provide slightly better detection/false positives rate, than the others. At the same time, all the algorithms related to the HMM (especially training algorithms) are much more computationally complex than their competitors. It took several months for authors to train all the HMMs, which were used for testing later. Of course if we think of applying this HMM-based approach to real problems, we should decrease this training time significantly; otherwise, there is a risk that the trained HMMs, corresponding to individual programmes become obsolete before their training is over.

The aim of our research is to design a faster way of HMM training, which could allow their effective use in intrusion detection as it is proposed in [12]. There is probably no need for faster HMM training in areas such as speech recognition, where collection of training data takes reasonably more time than the training itself. However, this acceleration might be useful for other areas, where big volumes of data should processed in real time.

2. Related Work

As soon as the necessity of constructing fast HMM training algorithms arises in selected applications only, there have been relatively few publications devoted to this topic.

W. Turin [13] proposed unidirectional and parallel Baum-Welch algorithm. The described parallel algorithm is mainly designed for signal processing applications. It is based on temporal splitting of a training sequence, and it relies on some features of observation sequences, such as continuous repetitions of identical observations. The author claims that the proposed algorithm reduces Baum-Welch training time if implemented on a massively parallel computer. However, no experimental data is given, which might show how well this algorithms works with different kinds of training sets.

A. Espinoza-Manzo et al. [14] made another attempt to design a fast implementation of Baum-Welch algorithm.

Proceedings of the 8th International Workshop on Computer Science and Information Technologies CSIT'2006
Karlsruhe, Germany, 2006

They made use of the fact that each iteration of this algorithm can be split into three separate stages, and designed the respective hardware pipeline architecture. It was synthesized on 40MHz FPGA chip, which executed Baum-Welch algorithm slightly slower than a serial software solution running on 500MHz PC.

3. Hidden Markov Model

3.1. Definition and notation

The comprehensive HMM tutorial can be found in [1]; here we focus on the basic definitions and training algorithm only.

An HMM can be defined as a five-tuple $\lambda = \{S, V, A, B, \pi\}$, where S is a set of N states; V is a set of M observable symbols; A is an NxN transition probability matrix, whose elements are defined as $a_{ij} = P[q(t)=s_i \mid q(t-1)=s_j]$; B is an NxM observation probability matrix, whose elements are defined as $b_{jk} = P[o(t) = v_k \mid q(t) = s_j]$; and π is an N-element initial state distribution vector, whose elements are defined as $\pi_i = P[q(1)=s_i]$.

There are three basic problems commonly associated with HMMs:

- *Evaluation* problem implies finding the probability of generating an observation sequence given an HMM, i.e. $P(O|\lambda)$. Two standard algorithms with equal computational complexity are known for solving this problem; they are called forward and backward algorithms.
- *Recognition* problem implies finding the most probable hidden state sequence given an observation sequence and an HMM. This sequence can be found with Viterbi algorithm.
- *Training* problem aims to vary A, B and π elements of the given HMM in order to maximize conditional probability of generating a given observation sequence. Traditional search optimisation methods such as Newton-Raphson or conjugate gradients are not robust and difficult to adapt for using with HMM, especially when the latter have many hidden states. On the other hand, Baum-Welch algorithm is robust for tuning HMM parameters using either single or multiple observation sequences.

3.2. Sequential Baum-Welch Algorithm

HMMs can be trained using either single or multiple observation sequences.

Single Observation Sequences

Suppose we have an observation sequence $O=\{o_1, o_2,\ldots o_T\}$ and an HMM $\lambda=(A, B, \pi)$. In this case the aim of training procedure is adjusting the elements of λ to maximize $P(O|\lambda)$.

The first stage of Baum-Welch algorithm constitutes forward and backward algorithms, which are used for solving the evaluation problem described above. These algorithms compute forward variables α and backward variables β respectively according to the following recurrent formulae:

$$\alpha_1(i) = \pi_i b_i(o_1) \tag{1}$$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(o_{t+1}) . \tag{2}$$

$$\beta_T(i) = 1 \tag{3}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j) \tag{4}$$

On the next stage other auxiliary matrices ξ and γ are computed in the following way:

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \tag{5}$$

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} . \tag{6}$$

And finally it is possible to update all the HMM parameter values:

$$\pi'_i = \gamma_1(i) \tag{7}$$

$$a'_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{8}$$

$$b'_j(l) = \frac{\sum_{t=1}^{T-1} \gamma_t(j)}{\sum_{t=1}^{T-1} \gamma_t(j)} . \tag{9}$$

According to [1], the resulting model $\lambda'=(A', B', \pi')$ is either equivalent to λ, or such that $P(O|\lambda')>P(O|\lambda)$. Thus, if all the steps (1)-(9) are repeated until changes on a successive iteration are not significant, the resulting λ' will correspond to the local maximum of $P(O|\lambda)$ target function.

Multiple Observation Sequences

Oftentimes, HMMs have to be trained to match a set of observation sequences entirely. In this case, a training set is defined as $O=\{O^{(1)}, O^{(2)},\dots O^{(K)}\}$, where each $O^{(k)}$ is an observation sequence of an arbitrary length. The target function to be maximized is the product of all the separate probabilities $P(O^{(k)}|\lambda)$.

Expressions (1)-(6) remain the same for multiple observation training, however it should be taken into account that each $\alpha^{(k)}$, $\beta^{(k)}$, $\xi^{(k)}$, and $\gamma^{(k)}$ variables correspond to the respective members of the training set, and should be computed individually for each $O^{(k)}$. HMM update procedures take the following form:

$$\pi'_i = \frac{1}{K}\sum_{k=1}^{K}\gamma_1^{(k)}(i) \tag{10}$$

$$a'_{ij} = \frac{\sum_{k=1}^{K}\sum_{t=1}^{T-1}\xi_t^{(k)}(i,j)}{\sum_{k=1}^{K}\sum_{t=1}^{T-1}\gamma_t^{(k)}(i)} \tag{11}$$

$$b'_j(l) = \frac{\sum_{k=1}^{K}\sum_{t=1,\ s.t.\ O_t=v_l}^{T-1}\gamma_t^{(k)}(j)}{\sum_{k=1}^{K}\sum_{t=1}^{T-1}\gamma_t^{(k)}(j)} \tag{12}$$

Computation of intermediate values and successive HMM update procedures are enclosed in a loop, which ends whenever statistical difference between newly updated HMM is not notably different from the one obtained on the previous iteration.

4. Organization of Parallel Computations in Baum-Welch Algorithm

The proposed parallel implementation of Baum-Welch algorithm is suitable for multiple-observation training only. It is based on the following prerequisites of effective parallelism:

- Each iteration of Baum-Welch algorithm for multiple observation sequences consists of complex computations of various intermediate values (1)-(6) followed by relatively simple update procedures (10)-(12).
- Calculation of each subset of intermediate values ($\alpha^{(k)}$, $\beta^{(k)}$, $\xi^{(k)}$, and $\gamma^{(k)}$) depends on current HMM parameters, and $O^{(k)}$ observation sequence only.

These two facts lead to a straightforward data decomposition strategy. That is the entire training set O is split into subsets, and a separate process can do most of the required computations for each subset. Only then, all the computed values should be used to update current HMM parameters, and the training either proceeds to the next iteration or stops.

Suppose the training set is split into the following disjoin subsets: $O = \Omega_1 \cup \Omega_2 \cup \dots \cup \Omega_U$. Denote the following intermediate parameters for an arbitrary subset Ω_u:

$$numA_{ij}(u) = \sum_{\Omega_u}\sum_{t=1}^{T-1}\xi_t(i,j) \tag{13}$$

$$numB_{jl}(u) = \sum_{\Omega_u}\sum_{t=1,\ s.t.\ O_t=v_l}^{T-1}\gamma_t(j) \tag{14}$$

$$den_j(u) = \sum_{\Omega_u}\sum_{t=1}^{T}\gamma_t(j) \tag{15}$$

These computations, as well as all the preceding computations according to (1)-(6) expressions, can be executed in context of a separate process, which takes training data from Ω_u subset only. Whenever each parallel process finishes calculation of all of its *numA*, *numB*, and *den* values, it is possible to update HMM parameters within a single sequential process, according to the following expressions:

$$a'_{ij} = \frac{numA_{ij}(1) + numA_{ij}(2) + \dots + numA_{ij}(U)}{den_j(1) + den_j(2) + \dots + den_j(U)} \tag{16}$$

$$b'_j(l) = \frac{numB_{jl}(1) + numB_{jl}(2) + \dots + numB_{jl}(U)}{den_j(1) + den_j(2) + \dots + den_j(U)} \tag{17}$$

Update procedure for π vector values is obvious, and it can be done according to (10) with respect to parallel execution.

It is important to note that for big training sets containing long observation sequences (i.e. for those demanding much faster training implementation), the total complexity of parallel steps (1)-(6) and (13)-(15) significantly exceeds total complexity of sequential steps (10), (16), and (17). This means that the theoretical speedup limit stated by Amdahl's law is likely to be very high for many applied problems.

5. Implementation and Results

Unlike the approach presented in [13], which is mainly designed for massively parallel computers, the method described in this paper is more suitable for distributed cluster systems based on message-passing paradigm. The fact, that computations executed within individual parallel processes are mostly independent from each other, and data exchange between processes is relatively low, makes it possible to implement this algorithm on inexpensive

network clusters, which use standard communication equipment, such as 100Mb/s Ethernet cards.

As soon as the complexity of Baum-Welch training depends on the number of observation symbols, rather than the number of observation sequences, it is important to organize training set decomposition in a way that Ω_u subsets contain approximately the same number of observation symbols. Otherwise, some parallel branches will complete their execution much earlier than the others, which will lead to the waste of computational resources.

The proposed parallel algorithm was implemented for cluster architecture using Message-Passing Interface (MPI) standard. As soon as the need for faster HMM training was mainly caused by intrusion detection problems, the designed implementation was tested using Computer Immune Systems dataset of the University of New Mexico (http://www.cs.unm.edu/~immsec).

Testing was performed on the network cluster consisting of Celeron 2.4GHz computers with 256Mb RAM each, and connected using 100Mbit/s Ethernet. Experimental speedup values were around 1.93-1.96 for the cluster made of two computers, and they were as high as 2.7-2.8 for the cluster consisting of three computers.

6. Conclusion

The described parallel implementation of Baum-Welch algorithm for multiple observation sequences has advantages compared with the sequential execution when training data is big enough. Although the need for faster HMM training was caused by intrusion detection problems, the designed implementation does not depend on any features of a particular applied area, and thus it can be used for other applications as well.

Compared with the approach presented in [13], this implementation strategy seems to be more suitable for inexpensive network clusters, rather than for massively parallel computers. Also, it does not require any dedicated hardware modules as in [14].

Acknowledgments

This work was supported by grants from Russian Foundation for Basic Research (04-07-90137 and 06-07-89010-a).

References

1. Rabiner, L. R. “A tutorial on Hidden Markov Models and selected applications in speech recognition”. In: *Proc. IEEE*, 77(2), 1986, pp. 257-286.
2. “Noise reduction in speech application”. Edited by G. M. Davis, CRC Press LLC, 2002.
3. Turin, W., van Nobelen, R. “Hidden Markov modeling of flat fading channels”. *IEEE Journal on Selected Areas is Communications*, Vol. 16, Dec. 1998, pp. 1809-1817.
4. Nechyba, M.C., Xu, Y. “Stochastic similarity for validating human control strategy models”. *IEEE Trans. Robotics and Automation*, Vol. 14, Issue 3, Jun 1998, pp. 437-451.
5. Mangold, S., Kyriazakos, S. “Applying pattern recognition techniques based on hidden Markov models for vehicular position location in cellular networks”. In: *Proc. IEEE Vehicular Technology Conference*, Vol. 2, 1999, pp. 780-784.
6. Eickeller, S., Müller, S., Rigoll, G. “Recognition of JPEG compressed face images based on statistical methods”. *Image and Vision Computing*, Vol. 18, 2000, pp. 279-287.
7. Elms, A. J., Procter, S., Illingworth, J. “The advantage of using and HMM-based approach for faxed word recognition”. *International Journal on Document Analysis and Recognition (IJDAR)*, 1998; 1(1), pp. 18-36.
8. Kulp, D., Haussler, D., Reese, M. G., Eeckman, F. H. “A generalized hidden Markov model for the recognition of human genes in DNA”. In: Proc. 4th Intl. Conf. on Intelligent Systems for Molecular Biology, 1996, pp. 134-142.
9. Lane, T. “Hidden Markov models for human/computer interface modeling”. In: *Proc. of the IJCAI-99 Workshop on Learning About Users*, 1999, pp. 35-44.
10. Park, H.-J., Cho, S.-B. “Efficient anomaly detection by modeling privilege flows using hidden Markov model”. *Computers & Security*, 2003; 1(22), pp. 45-55.
11. Ye, N. “A Markov chain model of temporal behavior for anomaly detection”. In: *Proc. 2000 IEEE Workshop on Information Assurance and Security*, 2000, pp. 171-174.
12. Warrender, C., Forrest, S., Pearlmutter, B. “Detecting intrusions using system calls: alternative data models”. In: *Proc. 1999 IEEE Symposium on Security and Privacy*, 1999, pp. 133-145.
13. Turin, W. “Unidirectional and parallel Baum-Welch algorithms”. In: *IEEE Trans. Of Speech and Audio Processing*, Nov. 1998, pp. 516-523.
14. Espinosa-Manzo, A., López-López, A., Arias-Estrada, M. O. “Implementing Hidden Markov Models in a Hardware Architecture”. In: *Proc. International Meeting of Computer Science ENC ’01*, Vol. II, Aguascalientes, México, September 15-19 2001, pp. 1007-1016.