Scaling the iHMM: Parallelization versus Hadoop

Sébastien Bratières, Jurgen Van Gael
Department of Engineering
University of Cambridge
United Kingdom
Email: sb358, jv279 @cam.ac.uk

Andreas Vlachos
Computer Laboratory
University of Cambridge
United Kingdom
Email: av308@cl.cam.ac.uk

Zoubin Ghahramani
Department of Engineering
University of Cambridge
United Kingdom
Email: zoubin@eng.cam.ac.uk

***Abstract*—This paper compares parallel and distributed implementations of an iterative, Gibbs sampling, machine learning algorithm. Distributed implementations run under Hadoop on facility computing clouds. The probabilistic model under study is the infinite HMM [1], in which parameters are learnt using an instance blocked Gibbs sampling, with a step consisting of a dynamic program. We apply this model to learn part-of-speech tags from newswire text in an unsupervised fashion. However our focus here is on runtime performance, as opposed to NLP-relevant scores, embodied by iteration duration, ease of development, deployment and debugging.**

I. Introduction

Probabilistic models are at the core of many modern machine learning algorithms. Generally probabilistic models are specified up to a finite number of parameters which are subsequently learned by fitting the model to data. These models are generally called parametric models. Nonparametric models extend parametric models by allowing the number of parameters to grow when the number of data-points increases. As more parameters generally implies more flexible models, nonparametric models can easily adapt to the amount of data without overfitting.

As nonparametric models have the most potential in a context where much data is available, it is crucial that we can learn the parameters of these models efficiently. In this paper we compare two ways of scaling the training algorithm for a nonparametric model: using parallelization on a single machine versus using distributed computing on top of Hadoop. Similar effort has been undertaken by the Mahout community in order to build distributed versions of the parallelized machine learning algorithms described by Bradski et al. [2]

The broad context in which we will explore the applicability of the different scalability options is natural language processing (NLP). The explosion of the WWW has made available increasing amounts of text in digital format, thus presenting a growing challenge to the traditional single-processor or parallel approaches. Furthermore, it is a suitable domain for Bayesian non-parametric methods to demonstrate their potential, as it is hard to define in advance the right level of detail needed for varying amounts of text of different genres.

Concretely, the contribution of our paper is 4-fold:

1) We experiment how to apply the map-reduce paradigm to an iterative sampling algorithm,
2) We explore which paradigm (parallel/distributed) is best in which situation,
3) We report the lessons learned from our distributed implementation,
4) We will shortly release the Hadoop source code to encourage research in this area of machine learning.

II. The Infinite Hidden Markov Model

The hidden Markov model (or HMM) is a probabilistic modelling tool that is in wide use throughout machine learning, signal processing, bio-informatics and many other fields. The HMM describes a probability distribution over a sequence of observations $w_1, w_2, \cdots, w_T$ of length T. The HMM assumes there exists a Markov chain denoted by $s_1, s_2, \cdots, s_T$ where each s_t is in one of K possible states. The distribution of the state at time t only depends on the states before it through the state at time $t-1$. This dependency is governed by a K by K stochastic transition matrix π where $\pi_{ij} = p(s_t = j | s_{t-1} = i)$. This is the Markov property which gives the HMM its middle name. Generally, we do not directly observe the Markov chain, but rather an observation w_t whose distribution only depends on the state s_t, an observation model F (e.g. a Normal distribution) and a set of parameters θ (e.g. the mean and variances of the normal distribution).

For a fixed K many techniques (e.g. expectation maximization [3]) can be used to learn the parameters θ of the HMM. In this paper we focus on the issue of learning K using a nonparametric Bayesian approach. The *infinite hidden Markov model* (or iHMM), introduced in [1] is a probabilistic model which defines a distribution over *all* hidden Markov models with arbitrary large state space K. For a dataset of size N at most N states can be used. Nonetheless, the prior distribution puts most of its mass on state spaces much smaller than N (e.g. $\log N$ or $\log \log N$). Incidentally, after conditioning the model on N observations, the posterior will have most of its mass on hidden Markov models with state spaces much smaller than N. The nonparametric behavior of the iHMM stems from the fact that the more data is available, the more fine grained a state space it can learn *if the data calls for a more complex model*. As this paper focusses on the computational aspects of the iHMM, we point the interested reader to [1] and [4] for more theoretical background on the subject.

At present, there are no known deterministic algorithms for learning the parameters of the iHMM. In this paper we build on a Gibbs sampling method known as the beam sampler [5]. We refer to [5] for technical details of the beam sampler but line-out the algorithm here. Each iteration of the beam sampler consists of the following steps:

1) Compute the sufficient statistics for the transition and emission parameters,
2) Sample auxilary variables to dynamically truncate the infinite transition matrix,
3) Sample a finite representation of the transition and emission parameters,
4) Run forward-filtering backward-sampling to resample the hidden state sequence,
5) Resample any hyperparameters in the model.

In the application we describe below the sufficient statistics for the transition and emission parameters are the numbers n_{ij} which denote the number of transitions from i to j in our dataset and the numbers e_{iw} which denote the number of observations of type w emitted from state i. In other words, these numbers can be computed in time linear in the number of datapoints.

There is one auxilary variable for each adjacent pair of states in the dataset. Since sampling each auxilary variable is a constant time operation, we can compute these number in time linear in the number of datapoints.

Sampling the transition and emission parameters is generally very cheap: if K states are used in an iteration and the emission distribution is multinomial with E outcomes, K^2 entries in the transition matrix and KE entries in the emission distribution need to be computed.

The forward filtering backward-sampling procedure requires computing the table $p(s_t|w_{1:t})$. Using dynamic programming, we can efficiently compute this recursively using the equality $p(s_t|w_{1:t}) \propto p(w_t|s_t) \sum_{s_{t-1}} p(s_t|s_{t-1}) p(s_{t-1}|w_{1:t-1})$. This table has size KT for a sequence of length T, while each entry takes $\mathcal{K}$ operations to compute. In other words, the forward-filtering takes time $\mathcal{T}\mathcal{K}^2$ to compute. Once we have computed $p(s_T|w_{1:T})$ we can sample s_T and then backtrack to sample each other element in the state sequence. The complexity of this procedure is $\mathcal{T}\mathcal{K}^2$.

In other words, one iteration for the Gibbs sampler of the iHMM comprises of steps which are all linear in the length of the sequence T. Nonetheless, the additional K^2 factor for the dynamic programming step swamps the complexity for the other steps in the computation.

III. Unsupervised PoS tagging with the iHMM

Part-of-Speech (PoS) tagging is a standard component in NLP pipelines. PoS tags characterize words according to their syntactic (and sometimes semantic) behaviour, which allows us to perform syntactic parsing as well as use them in a variety of tasks, such as named entity recognition or determining intonation for text-to-speech systems [6].

Most of the work in PoS tagging has focused on the use of supervised machine learning methods, which require large amounts of labelled data. Using such methods (the supervised HMM being a very common choice), PoS tagging performance on English newswire text has reached high levels. However, when moving to new domains or languages which do not have labelled data readily available, such methods are unable to adapt. Therefore, recent work has focused on unsupervised methods that use unlabelled data and is available in large quantities.

In previous work on unsupervised PoS tagging using HMMs, a main question was how to set the number of hidden states appropriately. In particular, Johnson [7] reports results for different numbers of hidden states but it is unclear how to make this choice a priori, while Goldwater & Griffiths [8] leave this question as future work. It must be pointed here that this is a non-issue when using supervised machine learning methods, since there the model predicts a PoS tag from a fixed set that was provided with the training data. However in unsupervised PoS tagging the states learned by the iHMM do not correspond to PoS tags from a labelled corpus, therefore it is counter-intuitive to fix their number in advance. The fact that different authors use different versions with different number of PoS tags of the same dataset (e.g. Goldwater & Griffiths [8] versus Johnson [7]) supports this claim. To address the issue of selecting the number of states in unsupervised PoS tagging Van Gael et al. [9] applied the iHMM to the task and obtained competitive performance while allowing the model to pick the number of hidden states.

Evaluating unsupervised PoS tagging is rather difficult mainly due to the fact that the output of such systems are not actual PoS tags but state identifiers. Therefore it is impossible to evaluate performance against a manually labelled dataset using accuracy, as in supervised PoS tagging. Nevertheless, the state identifiers provide a clustering of the instances which can be used for evaluation purposes by treating the instances with same state identifier as belonging to the same cluster.

Clustering evaluation measures assess and sometimes combine the two desirable properties that a clustering should have with respect to a manually labelled dataset: homogeneity and completeness. Homogeneity is the degree to which each cluster contains instances from a single class. Completeness is the degree to which each class is contained in a single cluster. While an ideal clustering should have both properties, naively improving one of them can be harmful for the other. For example, one can achieve better homogeneity by simply increasing the number of clusters discovered but this is likely to reduce completeness.

The most common approach followed in previous work is to evaluate unsupervised PoS tagging as clustering against a manually labelled dataset is the Variation of Information (VI) [10] which assesses homogeneity and completeness using the quantities $H(C|K)$ (the conditional entropy of the class distribution in the manually labelled dataset given the clustering) and $H(K|C)$ (the conditional entropy of clustering given the class distribution in the manually labelled dataset). The lower these quantities are, the better the clustering is with respect to the manually labelled dataset. The final score is obtained by summing them, which means that lower values are better.

While inducing a mapping between states and PoS tags and the use accuracy is an option, the quality of the mapping would affect the evaluation, which is undesirable. This is also the case with the commonly used F-measure clustering evaluation measure [11]. Information-theoretic measures like VI neither need nor attempt to infer such mappings, therefore they are more suitable to our purposes.

IV. Experiments

As a rough performance indicator, we measured the duration of a Gibbs iteration in different setups, one parallel, and three distributed Hadoop setups. In addition, we are contrasting the different settings in terms of ease of development, deployment, and debugging.

A. Algorithm and data

The implemented algorithm was the same in all settings. In particular, the initial value for K was fixed to 100, a value to which cluster number converges as reported in [9]. To measure iteration duration, we averaged across 10 iterations of the algorithm.

In all our experiments, the datasets were derived from the Wall Street Journal (WSJ) part of the Penn Treebank, which is one of the standard corpora used in NLP research. It consists of 1 million tokens of financial newswire text and it has been labelled manually with PoS tags.

We extracted subsets of the WSJ dataset of sizes 1e3, 1e4 and 1e5 tokens. Together with the full dataset of 1e6 tokens and a dataset with all datapoints duplicated 10 times (10e7 tokens) we are covering an interesting range. Note that the dataset with 1e7 is not interesting from a modelling point of view as we duplicated all data 10 times; nonetheless the computational analysis remains valid.

As a sanity check, we evaluated the output of our distributed implementation on the 1e5 subset of the WSJ and the performance in terms of VI was 4.5 bits, roughly equivalent to the ones achieved by the parallel one in [9]. It must be noted that these scores are not strictly comparable due to differences in the dataset size, and we leave it to future work to present a full NLP-oriented evaluation of our distributed implementation.

B. Configurations

Parallel is an implementation of the iHMM in .NET which uses multithreading on a quad core 2.4 GHz machine with 8GB of RAM.

hadoop-1 is an implementation of the iHMM on Hadoop, where each step of the Gibbs iteration is implemented as map-reduce. "Each step" means each operation which scales with the number of data points, K or V (as defined above). This gives a total of 9 map-reduce jobs for each iteration in hadoop-1. *hadoop-2* is entirely like hadoop-1, except that only the most CPU-intensive step, namely the dynamic program, was implemented as map-reduce. *hadoop-3* is exactly like hadoop-2 from the software point of view.

The Hadoop experiments were implemented in Java using the Hadoop map-reduce library. They ran on Amazon's Elastic MapReduce computing cloud. For hadoop-1 experiments, they ran on clusters of different sizes: each cluster had one master node (the job tracker, in map-reduce terminology) and one or several slave nodes: 1, 2, 3, 4, 8 or 16 depending on the experiment. For hadoop-2 experiments, the cluster size was kept constant: one master and one slave node. For both hadoop-1 and hadoop-2, we used nodes of the Amazon "small" type, i.e. 32-bit platforms with one CPU equivalent to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor, 1.7 GB of memory, with 160 GB storage. For *hadoop-3*, and in order to beat the parallel setting, we resorted to Amazon "extra large" nodes, 64-bit platforms with 8 virtual cores, each equivalent to 2.5 times the reference 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor, 7 GB of memory, 1690 GB of storage.

C. Results

Figures 1 to 3 represent the iteration duration (in seconds) across different settings against a range of data set sizes. Data set size is measured in number of data points, that is, numbers of tokens (words). In the plots, each point marker represents an experiment with a given software and hardware setting, and a given data set size. Iteration duration was averaged over the 10 first iterations of the iHMM learning algorithm. Lines connect point markers to denote that they belong to the same experimental setting. In the legends, the letter H stands for *hadoop*.

log-log representation implies that a linear increase in computing cost with the number of data points should be reflected in a line. This is indeed the general trend of all experiments. The initially lower slope of some curves reflects the overhead of parts of the algorithm, or parts of the distribution process, which does not scale linearly with the data size.

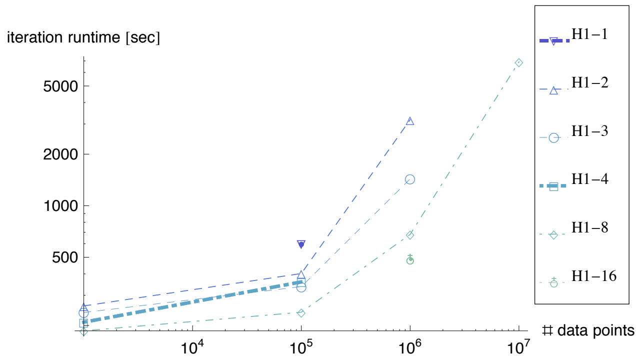

Figure 1. All *hadoop-1* experiments. The name of each result set is hadoop-1-*, with * indicating the number of slave nodes in the cluster. Iteration duration scales only slightly with cluster size.

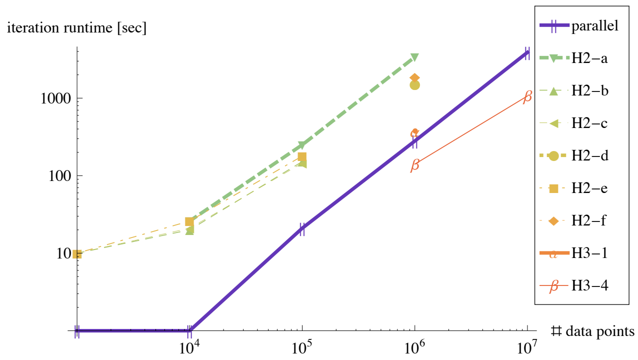

Figure 2. All *hadoop-2*, *hadoop-3*, and the *parallel* experiment side-by-side. All hadoop-2-* experiments use the same setting, and are just different runs of the same experiment. This demonstrates that there is little variability between runs. The hadoop-3 experiments demonstrate that scaling, and resorting to a more powerful machine type, finally beats the parallel setup.

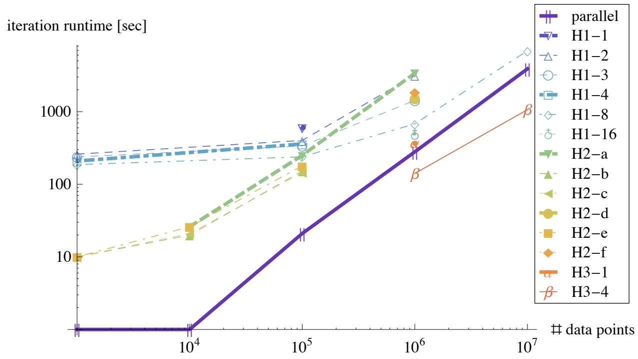

Figure 3. All experiments in the same plot. The general scaling trend follows that of the parallel setup. Increasing data size even further than what was tested here, we expect that hadoop-1 and -2 setups will become faster than the parallel setup. However, the point where e.g. lines for experiments *hadoop-1-8* and *parallel* intersect seems several orders of magnitude above present experiments.

data size	parallel	hadoop1-1	hadoop1-2	hadoop1-3	hadoop1-4	hadoop1-8	hadoop1-16	hadoop2-a	hadoop2-b	hadoop2-c	hadoop2-d	hadoop2-e	hadoop2-f
1,000	1		260	238	209	186			10	10		10	
10,000	1							26	20	21		26	
100,000	21	600	402	337	359	239		250	152	145		179	
1,000,000	280		3179	1434		680	487	3390			1515		1877
10,000,000	3893					6910							
# slave nodes	n/a	1	2	3	4	8	16	1	1	1	1	1	1
AEMR machine type	n/a	S	S	S	S	S	S	S	S	S	S	S	S

Figure 4. Experimental data for Figures 1-3. Iteration duration in seconds.

None of the hadoop-1 and hadoop-2 implementations were faster, in absolute terms, than parallel for the data sizes demonstrated here. This can be attributed to the different performance of the CPUs used in the parallel case, and on the Amazon cluster used for the distributed experiments.

The parallel implementations scales perfectly well with data size, but cannot accommodate data sizes beyond those shown, i.e. from 10 million data points onwards, because it is entirely memory-based (and therefore incurs no disk I/O costs). It is therefore necessary, for large datasets, to resort to the distributed implementations.

hadoop-1's cost is roughly stable for a wide range of settings, and does not scale well with the number of nodes. This implementation, where each Gibbs iteration contains 9 map-reduce jobs, is apparently badly suited to a Hadoop implementation because of the heavy overhead each map-reduce job incurs: about 30 seconds notwithstanding the number of nodes it runs on.

hadoop-2's cost is well beyond that of hadoop-1 and does not suffer from the large overhead effect. It therefore scales well with the amount of data.

The isolated hadoop-3 experiments used hardware which is comparable with the parallel experiment, and demonstrate a definite speedup when running from a cluster.

D. *Qualitative comparison*

This section completes the computing cost comparison with lessons from the software development exercise of all implementations.

1) Development: Using the parallel extensions under .Net proved relatively easy, and since no file-level or data-splitting is expected from the developer, was a one-off change from a reference, non-parallel implementation.

Turning to Hadoop implied learning the framework through tutorials and books, a much longer process. The developer writes his own reader/writer for the file format in which he intends to store map input, intermediate data, and reduce output. Portions of code corresponding to the reference implementation can be reused inside the relevant, corresponding map-reduce job. Shared parameters represent a special challenge in this shared-nothing framework, and here they were written to disk. Map-reduce jobs which needed to update them had to funnel all data processing through a single reducer, in order to obtain a single updated value for the parameter.

Quite some effort had to be expended in tuning, considering the large overhead that a Hadoop job setup incurs, which is constant on each slave node. In particular, in hadoop-1, independent map-reduce jobs were parallelized using the JobControl feature of Hadoop. In spite of these efforts, the job setup overhead remained large, with respect to the fact that our algorithm is data-light but CPU intensive in only one of its phases.

2) Deployment: Deployment of the parallel .Net version presented no particular difficulties. Deployment on the Amazon Elastic MapReduce (AEMR) platform proved acceptably easy once all the necessary admninistrative configuration had been performed, and the command-line tool for cluster startup and termination, job startup and termination had been learnt.

3) Debugging: Debugging the .Net implementation consisted of fixing dependencies on shared parameters, which can be spotted in the source code.

Running on AEMR presented a number of difficulties. Several bugs and crashes, some unsolved to day, which appeared exclusively on AEMR, not on our development cluster, made deployment hard. Diagnostic tools sum up to console and log consultation on running clusters. The algorithm received a large amount of logging statements, and logging configuration itself had to be brought in agreement with Hadoop requirements, to allow some amount of debugging.

V. Conclusion

We have implemented an iterative learning algorithm under four different settings, one parallel and three distributed ones with Hadoop. The runtime duration of an iteration is crucial for such a Gibbs sampling algorithm, since convergence typically takes several thousands of iterations.

We presented the algorithm (beam sampling on the infinite HMM) and the application (part-of-speech tagging) we were applying it on. Iteration durations were compared for the different settings, and the effort involved in developing the implementations was contrasted.

The parallel deployment had better performance than the distributed ones running on "small" cores; the overall best performance is obtained on a distributed setup running the "extra-large", more powerful, machines. The parallel setup would not scale up to larger data sizes, neither would it scale very much with number of cores used. Therefore a distributed implementations is required when turning to web-scale data.

Our first experiment with Hadoop, in which all loops were turned into map-reduce jobs, incurred high map-reduce job setup overhead, in spite of the tuning we applied. Had it been know to us that Hadoop is not well suited to iterative applications containing several map-reduce jobs per iteration, we would not have tried this; we hope this finding is a contribution of this paper. Our Hadoop implementation with only one map-reduce job per iteration had acceptable scaling behaviour with data size. Ongoing experiments investigate its scaling across number of nodes, which seems roughly linear, making it a good candidate for Gibbs sampling on a very large scale.

Acknowledgment

The authors would like to thank Amazon Education Services for a generous grant to use the Amazon Web Services. Jurgen Van Gael is supported by a Microsoft Research Scholarhsip.

References

[1] M. J. Beal, Z. Ghahramani, and C. E. Rasmussen, "The Infinite Hidden Markov Model," *Advances in Neural Information Processing Systems*, vol. 14, pp. 577 – 584, 2002.

[2] G. Bradski, C.-T. Chu, A. Ng, K. Olukotun, S. K. Kim, Y.-A. Lin, and Y. Yu, "Map-reduce for machine learning on multicore," in *NIPS*, 2006.

[3] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970.

[4] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Hierarchical Dirichlet processes," *Journal of the American Statistical Association*, vol. 101, no. 476, pp. 1566–1581, 2006.

[5] J. Van Gael, Y. Saatci, Y. W. Teh, and Z. Ghahramani, "Beam Sampling for the Infinite Hidden Markov Model," in *Proceedings of the 25th International Conference on Machine learning*, Helsinki, 2008.

[6] C. D. Manning and H. Schuetze, *Foundations of Statistical Natural Language Processing*, 1st ed. The MIT Press, Jun. 1999.

[7] M. Johnson, "Why Doesn't EM Find Good HMM POS-Taggers?" in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007, pp. 296–305.

[8] S. Goldwater and T. Griffiths, "A fully Bayesian approach to unsupervised part-of-speech tagging," in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 744–751.

[9] J. Van Gael, A. Vlachos, and Z. Ghahramani, "The infinite HMM for unsupervised PoS tagging," in *Proceedings of 2009 Conference on Empirical Methods in Natural Language Processing*, Singapore, 2009, pp. 678–687.

[10] M. Meilă, "Comparing clusterings—an information based distance," *Journal of Multivariate Analysis*, vol. 98, no. 5, pp. 873–895, 2007.

[11] B. C. M. Fung, K. Wang, and M. Ester, “Hierarchical document clustering using frequent itemsets,” in *Proceedings of SIAM International Conference on Data Mining*, 2003, pp. 59–70.