INTERSPEECH 2009 BRIGHTON

A Parallel Training Algorithm for Hierarchical Pitman-Yor Process Language Models

Songfang Huang, Steve Renals

The Centre for Speech Technology Research
University of Edinburgh, Edinburgh, EH8 9AB, UK
{s.f.huang, s.renals}@ed.ac.uk

Abstract

The Hierarchical Pitman Yor Process Language Model (HPYLM) is a Bayesian language model based on a non-parametric prior, the Pitman-Yor Process. It has been demonstrated, both theoretically and practically, that the HPYLM can provide better smoothing for language modeling, compared with state-of-the-art approaches such as interpolated Kneser-Ney and modified Kneser-Ney smoothing. However, estimation of Bayesian language models is expensive in terms of both computation time and memory; the inference is approximate and requires a number of iterations to converge. In this paper, we present a parallel training algorithm for the HPYLM, which enables the approach to be applied in the context of automatic speech recognition, using large training corpora with large vocabularies. We demonstrate the effectiveness of the proposed algorithm by estimating language models from corpora for meeting transcription containing over 200 million words, and observe significant reductions in perplexity and word error rate.
Index Terms: language model, Pitman-Yor processes, hierarchical Bayesian models, parallel training, meetings

1. Introduction

The task of a language model (LM) is to estimate a probability distribution over sentences, which can be formed by piecing together a sequence of conditional probability distributions over each word in the sentence given the histories. An n-gram LM, which approximates the histories by the immediately preceding $n-1$ words, is the most widely used language model in many applications such as automatic speech recognition (ASR) and statistical machine translation (SMT). Due to the curse of dimensionality, *smoothing* plays an essential role when estimating n-gram language models. A large number of smoothing methods have been proposed in the literature [1, 2], among which interpolated Kneser-Ney [3] and modified Kneser-Ney [1] are the best.

In recent years, a hierarchical Bayesian language model based on Pitman-Yor processes has been introduced as a potentially better smoothing method, and *theoretically* shown to recover the interpolated Kneser-Ney smoothing for language modeling [4]. Our previous work [5], furthermore, *practically* demonstrated that hierarchical Pitman-Yor process language models (HPYLM) can offer consistent and significant reductions in perplexity and word error rate (WER), compared with both interpolated Kneser-Ney LMs (IKNLM) and modified Kneser-Ney LMs (MKNLM), for large vocabulary automatic speech recognition (LVASR) of conversational speech in multiparty meetings.

Training an HPYLM typically uses Markov chain Monte Carlo (MCMC) sampling methods, which are significantly more computationally expensive than training an IKNLM or MKNLM. There are two things that lead to increased computational complexity when training an HPYLM. First, the inference itself can take several hundred sampling iterations to converge. Second, the memory requirement for training an HPYLM is large and grows linearly with corpus size. It is therefore important to make the HPYLM scale to work on the large corpora used in LVASR and SMT.

In this paper, we present a parallel training algorithm for the HPYLM. The parallel training algorithm alleviates the limitation of computational time and memory constrained by a single machine, by dividing the inference into sub-tasks. The sub-tasks, thereafter, can be either parallelly submitted to a computing cluster, or sequentially run on a single server machine. In this way, we are able to train in parallel HPYLMs on corpora of more than 200 hundred millions of words, which in turn reduce the perplexity and word error rate significantly on meeting transcription tasks. We show that any approximations resulting from the proposed parallel algorithm have a negligible effect on performance.

2. Hierarchical Pitman-Yor Process LMs

In this section we briefly recap the hierarchical Pitman-Yor process language model first introduced in [4]. In the Bayesian framework for language modeling, a prior distribution is placed over the predictive distribution of interest in LMs, and the posterior distribution is inferred from the training data (observations). The final predictive probability can then be estimated from the posterior by marginalizing out the latent variables and/or hyper-parameters.

Goldwater *et al.* claimed that the Pitman-Yor process is potentially a more suitable prior distribution for language modeling [6], because of its ability to generate power-law distributions. The power-law distribution – a few outcomes have very high probability and most outcomes occur with low probability – closely resembles the statistical properties of word frequencies observed in natural languages.

The Pitman-Yor process [7] $\mathrm{PY}(d, \theta, G_b)$ is a distribution over distributions, where d is a discount parameter, θ a strength parameter, and G_b a base distribution that can be understood as a mean of draws from $\mathrm{PY}(d, \theta, G_b)$. The procedure for generating draws from G that is distributed according to a Pitman-Yor process, $\mathrm{G} \sim \mathrm{PY}(d, \theta, G_b)$, can be described using the Chinese restaurant metaphor. Imagine a Chinese restaurant containing an infinite number of tables, each with infinite seating capacity. Customers enter the restaurant and seat themselves. The first customer sits at the first available table, while each of the

subsequent customers sits at an occupied table with probability proportional to the number of customers already sitting there $c_k - d$, or at a new unoccupied table with probability proportional to $\theta + dt_\bullet$, where c_k the number of customers sitting at table k, and $t_\bullet$ is the current number of occupied tables. This metaphor is helpful for the inference of an HPYLM.

Consider a vocabulary $\mathcal{V}$ with $|\mathcal{V}|$ word types. Let $G_\emptyset(w)$ be the unigram probability of w, and $G_\emptyset = [G_\emptyset(w)]_{w\in\mathcal{V}} = [G_\emptyset(w_1), G_\emptyset(w_2), G_\emptyset(w_3), \ldots, G_\emptyset(w_{|\mathcal{V}|})]$ represents the vector of word probability estimates for unigrams. A Pitman-Yor process prior is placed over $G_\emptyset \sim \mathrm{PY}(d, \theta, G_b)$ with an uninformative base distribution $G_b(w) = 1/|\mathcal{V}|$ for all $w \in \mathcal{V}$. According to the Chinese restaurant metaphor, customers (word tokens) enter the restaurant and seat themselves at tables. Those c_w customers that correspond to the same word label w, can sit at different tables, with t_w denoting the number of tables occupying customers w. Given the seating arrangement $\mathcal{S}$ of customers, and hyperparameters d and θ, the predictive probability of a new word w is given by:

$$P(w|\mathcal{S}, d, \theta) = \frac{c_w - dt_w}{\theta + c_\bullet} + \frac{\theta + dt_\bullet}{\theta + c_\bullet} G_b(w) \qquad (1)$$

where $c_\bullet = \sum_w c_w$ is the total number of customers, and $t_\bullet = \sum_w t_w$ is the total number of tables, in the restaurant for unigrams. Averaging over the posterior distribution over seating arrangements and hyperparameters, we can obtain the probability $P(w)$ for a unigram LM.

Similarly we can generalize the above unigram example to the n-gram case. An n-gram LM defines a probability distribution over the current word given a context $\mathbf{u}$ consisting of $n-1$ words. Let $G_\mathbf{u}(w)$ be the probability of the current word w and $G_\mathbf{u} = [G_\mathbf{u}(w)]_{w\in\mathcal{V}}$ be the target probability distribution for n-gram. A Pitman-Yor process is served as the prior over $G_\mathbf{u}$, with discounting parameter $d_{|\mathbf{u}|}$ and strength parameter $\theta_{|\mathbf{u}|}$ specific to the length of the context $|\mathbf{u}|$. The base distribution is $G_{\pi(\mathbf{u})}$, the lower order model of probabilities of the current word given all but the earliest word in the context. That is,

$$G_\mathbf{u} \sim \mathrm{PY}(d_{|\mathbf{u}|}, \theta_{|\mathbf{u}|}, G_{\pi(\mathbf{u})}) \qquad (2)$$

Since $G_{\pi(\mathbf{u})}$ is still an unknown probability distribution, a Pitman-Yor process is recursively placed over it with parameters specific to $|\pi(\mathbf{u})|$, $G_{\pi(\mathbf{u})} \sim \mathrm{PY}(d_{|\pi(\mathbf{u})|}, \theta_{|\pi(\mathbf{u})|}, G_{\pi(\pi(\mathbf{u}))})$. This is repeated until we reach $G_\emptyset$ for a unigram model discussed above. This results in a hierarchical prior, enabling us to generalize from the unigram model to the n-gram case. There are multiple restaurants (Pitman-Yor processes) in the prior hierarchy, with each corresponding to one context. By using the hierarchical framework of Pitman-Yor priors, different orders of n-gram can thus share information, similar to the traditional interpolation of higher order n-grams with lower order n-grams.

The inference algorithm for the HPYLM is Gibbs sampling, a special case of MCMC, which iteratively first removes customers, and then adds them again to the restaurants to obtain an updated seating arrangement [4]. A method based on auxiliary variables is used for sampling hyperparameters d and θ.

We implemented the hierarchical Pitman-Yor process language model on top of the SRILM toolkit [8], as an extended tool for Bayesian language modeling[1]. We take advantages of data structures available in SRILM for an efficient and extensible implementation of the HPYLM. Refer to our previous work in [5] for more detailed information.

[1]The HPYLM software is available from `http://homepages.inf.ed.ac.uk/s0562315/`.

3. A Parallel Training Scheme for HPYLM

Even with an efficient implementation of the HPYLM, however, it is still computationally expensive in terms of computing time and memory requirements to infer an HPYLM using a large corpus. According to our previous results in [5], increasing the size of either corpora or vocabulary increases the computational complexity of inferring an HPYLM, with a corpus of around 50 million words using a 50k vocabulary roughly taking ten minutes per iteration and occupying about 2.5 GB memory.

This motivated us to design a parallel training algorithm to efficiently estimate an HPYLM. We use a *divide-and-conquer* scheme. There are two steps: *data partition* and *model combination*. Generally speaking, we divide the inference task, which is normally infeasible or expensive using a single machine, into sub-tasks that fit well to the computational capacity of a single computing node – alleviating the memory requirement. Further combined with parallelism, we can also decrease the computational time for inference by running sub-tasks in parallel.

In the data partition step, we first divide word types in the vocabulary $\mathcal{V}$ into subsets $\mathcal{V}_k \subset \mathcal{V}$. For each subset $\mathcal{V}_k$, we then compose those bigrams beginning with words $w \in \mathcal{V}_k$, and their corresponding child n-grams with $n > 2$, as a sub-HPYLM (dotted rectangles), and put a pseudo $G_\emptyset$ (dotted circles) as the Pitman-Yor process for unigrams of the sub-HPYLM, as shown in Figure 1. Each sub-HPYLM can be inferred separately using the same routines as those for a normal HPYLM, except that the pseudo $G_\emptyset$ now additionally collects the number of insertion and deletion for customer $w \in \mathcal{V}_k$. The inference of sub-HPYLMs can be executed in parallel by submitting to a computing cluster.

In the model combination step, we combine all the sub-HPYLM models level-by-level in the HPY hierarchy. For each level, we accumulate auxiliary parameters, and sample the hyperparameters d and θ. For the global $G_\emptyset$ for unigrams, we infer the seating arrangements by using the insertion and deletion statistics accumulated by each pseudo $G_\emptyset$, to make sure the HPYLM is consistent regarding the *modified* counts for lower order n-grams [3, 4]. Depending on when to combine sub-HPYLMs, we explore two different versions of parallel training algorithm. The first version, iterative-synchronized or IterSync, combines sub-HPYLMs, and sample hyperparameters after each iteration, while the second one, final-synchronized or FinalSync, does the combination and samples hyperparameters only after each sub-HPYLM has finished all the predefined number of iterations. Due to the extra costs of submitting and queueing jobs at each iteration, it is much slower for IterSync to infer an HPYLM than FinalSync. It is understandable, however, that the second version, FinalSync, will lose more precisions than the first one, since the hyperparameters are not optimized globally, which in turn does harm to the inference of seating arrangements.

The parallel training algorithm makes it possible to estimate an HPYLM on large corpora using a large vocabulary. The detailed parallel algorithm for the HPYLM is described in Algorithm 1. Since this is indeed a data parallelism, it is possible to port the parallel training algorithm for the HPYLM in Algorithm 1 to the MapReduce framework. This work on parallel computing for inferring HPYLMs has made use of the computing cluster managed by Sun Grid Engine, which is provided by the Edinburgh Compute and Data Facility[2].

[2]`http://www.ecdf.ed.ac.uk/`

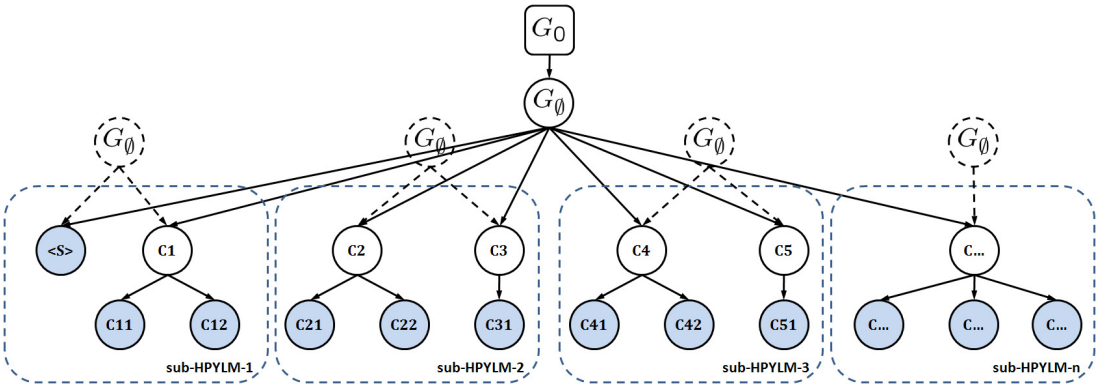

Figure 1: The partition of an HPYLM into sub-HPYLMs, denoted by dotted rectangles, for the parallel inference. The dotted circles represent pseudo $G_\emptyset$s to complete a Pitman-Yor process hierarchy, and collect additional insertion/deletion information. Each circle corresponds to a context, or a restaurant in the Chinese restaurant metaphor.

Algorithm 1 A Parallel Training Algorithm for the HPYLM

procedure PARALLELHPYLMITERSYNC(order n, $\mathcal{C}$, $\mathcal{V}$)
  divide vocab $\mathcal{V}$ into K subsets, $\mathcal{V}_k \subset \mathcal{V}$
  for $k = 1$ **to** K **do**
    read counts for subset $\mathcal{V}_k$ from count $\mathcal{C}$
    initialize sub-HPYLM$_k$
  end for
  for $i = 1$ **to** N iterations **do**
    PARALLEL
    for $k = 1$ **to** K **do**
      infer sub-HPYLM$_k$: INFERHPYLM(n)
    end for
    ENDPARALLEL
    infer HPYLM for unigrams
    collect statistics
    sample hyperparameters
  end for
  combine sub-HPYLMs
  estimate a final ARPA format LM
end procedure

procedure PARALLELHPYLMFINALSYNC(order n, $\mathcal{C}$, $\mathcal{V}$)
  divide vocab $\mathcal{V}$ into K subsets, $\mathcal{V}_k \subset \mathcal{V}$
  for $k = 1$ **to** K **do**
    read counts for subset $\mathcal{V}_k$ from count $\mathcal{C}$
    initialize sub-HPYLM$_k$
  end for
  PARALLEL
    for $i = 1$ **to** N iterations **do**
      for $k = 1$ **to** K **do**
        infer sub-HPYLM$_k$: INFERHPYLM(n)
      end for
    end for
  ENDPARALLEL
  infer HPYLM for unigrams
  collect statistics
  sample hyperparameters
  combine sub-HPYLMs
  estimate a final ARPA format LM
end procedure

4. Experiments and Results

In the following experiments, we trained HPYLMs using 100 iterations to burn in, and output standard ARPA format LMs, which were subsequently used in the first pass decoding using HDecode[3].

4.1. Meeting Corpora

Partially driven by the Rich Transcription (RT) evaluations organized by the U.S. National Institute of Standard and Technology (NIST), there is a growing research interest in the automatic transcription of multiparty meetings. European projects AMI and AMIDA [9] are examples of such efforts. This has, consequently, provided us a well-defined benchmark on which to evaluate a state-of-the-art LVASR system.

The experiments reported in this paper, therefore, were performed using two meeting transcription tasks. The first task is the NIST Rich Transcription 2006 spring meeting recognition evaluations (RT06s). We tested only on those audio data recorded from individual headset microphones (IHM), consisting of meeting data collecting by the AMI project, CMU, NIST, and VT (Virginia Tech). The training data sets for language models used in this paper, and the test transcription (*rt06seval*), are listed in Table 1. The web-data for meetings and conversational speech were collected from the world wide web using strategies described in [10].

Table 1: The statistics of the training and testing data sets for language models for the RT06s task.

No.	LM Data Set	#Sentences	#Words
1	*rt06strain*	205,814	1,847,201
2	Fisher (fisher-03-p1)	1,076,063	10,593,403
3	webdata (meetings)	3,218,066	36,073,718
4	webdata (conversational)	12,684,025	162,913,566
test	*rt06seval*	3,597	31,810

A second multiparty meeting corpus we consider in this paper is a domain-specific meeting corpus — the AMI Meeting Corpus[4] [11], which consists of 100 hours of multimodal meeting recordings with comprehensive annotations at a number of different levels. About 70% of the corpus was elicited using a design scenario, in which the participants play the roles of

[3] http://htk.eng.cam.ac.uk/
[4] http://corpus.amiproject.org

employees — project manager, marketing expert, user interface designer, and industrial designer — in an electronics company that decides to develop a new type of television remote control. We used the scenario part of the AMI Meeting Corpus for our experiments. There are 137 scenario meetings in total, of which we use 105 meetings (514,667 words) for training, and 32 meetings (175,302 words) for testing.

4.2. Validation of Parallelization Errors

We first conducted experiments to validate the errors/losses caused by parallelization. We trained three HPYLMs of order 3 on fisher-03-p1 shown in Table 1, one without using parallel training algorithm but the traditional inference, and the other two with parallelism of IterSync and FinalSync, respectively. We output standard ARPA format LMs, and evaluated these LMs for the transcription of *rt06seval*. Table 2 shows perplexity and WER results, which claims that there is no statistically significant difference between these results, although FinalSync has a bit higher perplexity than the other two. It is therefore safe for us to use the proposed parallel training algorithm for the HPYLM. In the following experiments, we all use the IterSync parallel training algorithm for inferring HPYLMs.

Table 2: Results of training on fisher-03-p1 and testing on *rt06seval*, to compare three cases of training an HPYLM: without parallelization, IterSync parallelism, and FinalSync parallelism, respectively.

Parallel	PPL	SUB	DEL	INS	WER
No	121.4	16.0	9.8	3.0	28.9
IterSync	121.8	16.1	9.8	3.0	28.9
FinalSync	126.6	16.1	9.7	3.1	28.9

4.3. Experiments on *rt06seval*

We trained three types of ARPA format trigram LMs – IKNLM, MKNLM, and HPYLM – on data sets No. 1–4 in Table 1, which is overall a corpus of around 210 million words. We observe in Table 3 that the HPYLM has a lower perplexity than both IKNLM and MKNLM. Moreover, the HPYLM produces significant lower WER ($p < 0.005$) compared with both the IKNLM and the MKNLM.

Table 3: The perplexity and word error rate results on *rt06seval*.

LMS	PPL	SUB	DEL	INS	WER
IKNLM	107.0	14.5	9.7	2.7	27.0
MKNLM	105.2	14.4	9.8	2.7	26.8
HPYLM	**98.9**	**14.2**	**9.8**	**2.6**	**26.5**

4.4. Experiments on AMI Corpus

In addition to the training data from the scenario AMI meetings , we included broadcast news (HUB-4) and conversational text (Fisher), which totally makes up a corpus of around 200 millions of words for training LMs. Again we trained IKNLM, MKNLM, and HPYLM on this combined data, and used these three LMs for the transcription of the testing data of 32 scenario AMI meetings . It is not surprising to find that the HPYLM is more accurate than both the IKNLM and the MKNLM, with lower perplexity and significantly lower WER ($p < 0.005$).

Table 4: The perplexity and word error rate results on the scenario AMI Meeting Corpus.

LMS	PPL	SUB	DEL	INS	WER
IKNLM	168.6	22.2	10.7	5.7	38.6
MKNLM	163.9	22.0	10.8	5.6	38.5
HPYLM	**158.8**	**21.9**	**10.8**	**5.5**	**38.2**

5. Conclusions

In conclusion we have demonstrated that it is feasible to infer a hierarchical non-parametric Bayesian language model from a large corpus, thus making it practical to use for large vocabulary speech recognition or machine translation. Our experimental results have shown that any approximations resulting from the parallel algorithm have a negligible effect on performance. Overall, the HPYLM results in significantly improved accuracy compared with the current state-of-the-art (IKNLM/MKNLM). The resulting language model may be interpreted as a smoothed n-gram model, can be implemented in a standard way (e.g., using an ARPA format language model file), and may be used in place of other smoothed n-gram language models.

6. Acknowledgements

This work is partially supported by the European IST Programme Project FP6-033812 (AMIDA). We thank the AMI-ASR team for providing the baseline ASR system for experiments. This work has made use of the resources provided by the Edinburgh Compute and Data Facility, which is partially supported by the eDIKT initiative.

7. References

[1] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," in *Proc. of the 34th Annual Meeting of the ACL*, June 1996, pp. 310–318.

[2] J. T. Goodman, "A bit of progress in language modeling," *Computer Speech and Language*, pp. 403–434, 2001.

[3] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *Proc. of ICASSP'95*, 1995, pp. 181–184.

[4] Y. W. Teh, "A hierarchical Bayesian language model based on Pitman-Yor processes," in *Proc. of the Annual Meeting of the ACL*, vol. 44, 2006.

[5] S. Huang and S. Renals, "Hierarchical Pitman-Yor language models for ASR in meetings," in *Proc. ASRU'07*, 2007, pp. 124–129.

[6] S. J. Goldwater, T. L. Griffiths, and M. Johnson, "Interpolating between types and tokens by estimating power-law generators," in *Advances in Neural Information Processing Systems 18*, 2006.

[7] J. Pitman and M. Yor, "The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator," *The Annals of Probability*, vol. 25, no. 2, pp. 855–900, 1997.

[8] A. Stolcke, "SRILM - an extensible language modeling toolkit," in *Proceedings of ICSLP'02*, Denver, Colorado, September 2002.

[9] S. Renals, T. Hain, and H. Bourlard, "Recognition and interpretation of meetings: The AMI and AMIDA projects," in *Proc. IEEE ASRU'07*, 2007.

[10] V. Wan and T. Hain, "Strategies for language model web-data collection," in *Proc. of ICASSP'06*, Toulouse, France, May 2006.

[11] J. Carletta, "Unleashing the killer corpus: experiences in creating the multi-everything AMI Meeting Corpus," *Language Resources and Evaluation Journal*, vol. 41, no. 2, pp. 181–190, 2007.