Distributed training of large scale exponential language models

Abhinav Sethy Stanley Chen Bhuvana Ramabhadran

IBM

May 25, 2011

Outline

Exponential n-gram models

- Exponential *n*-gram models comprise a powerful statistical framework that can model complex dependencies and utilize rich feature spaces [Wu and Khudanpur, 2002, Rosenfeld, 1996]
- For a set of target symbols $y \in Y$ and input symbols $x \in X$, an exponential model with parameters $\Lambda = \{\lambda_i\}$ and corresponding features $f_1(x, y) \ldots f_F(x, y)$ has the form

$$P_\Lambda(y|x) = \frac{\exp(\sum_{i=1}^{F} \lambda_i f_i(x, y))}{Z(x)} \tag{1}$$

$$Z(x) = \sum_{y \in Y} \exp(\sum_{i=1}^{F} \lambda_i f_i(x, y)) \tag{2}$$

We refer to x as the *event history* and y as the *target*.

Exponential n-gram models

- In an *exponential n-gram model* (for $n = 3$), we have binary features $f_{(\mathbf{x},\mathbf{y})}(\cdot)$ for $(\mathbf{x}, \mathbf{y})$ of the forms

$$(\epsilon, w_j), (w_{j-1}, w_j), (w_{j-2} wj - 1, w_j)$$

  where $f_{(\mathbf{x},\mathbf{y})}(x, y) = 1$ iff the event history x ends in $\mathbf{x}$ and the target word y is $\mathbf{y}$.

- Parameter estimation involves ℓ_1 and ℓ_2^2 regularization. The training objective function can be written as

$$\emptyset_{\ell_1+\ell_2^2}(Ł) = \log \mathrm{PP}_{\mathrm{train}} + \frac{\alpha}{D} \sum_i |\lambda_i| + \frac{1}{2\sigma^2 D} \sum_i \lambda_i^2 \qquad (3)$$

  for some α and σ, where $\mathrm{PP}_{\mathrm{train}}$ is training set perplexity and D is the size of the training set in words.

Training exponential n-gram models

- Exponential n-gram models are trained using iterative update algorithms which compare the expected feature counts to observed feature counts and update parameters accordingly
- For a feature $f_{(\mathbf{x},\mathbf{y})}(\cdot)$, the expected feature count can be computed as

$$E(f_{(\mathbf{x},\mathbf{y})}) = \sum_{(x_d,y_d)\in\mathcal{D}:f_{(\mathbf{x},\mathbf{y})}(x_d,\mathbf{y})=1} \frac{\exp(\sum_{i=1}^{F}\lambda_i f_i(x_d,\mathbf{y}))}{Z(x_d)} \quad (4)$$

where the training data $\mathcal{D}$ is expressed as a sequence of event histories and targets (x_d, y_d). Given feature expectations, it is generally inexpensive to update the λ_i's using regularized versions of iterative scaling [Darroch and Ratcliff, 1972] or other iterative update algorithms [Malouf, 2002].

Model M

- In [Chen, 2009], a novel class-based exponential n-gram language model, Model M was proposed. Gains over a wide range of domains, including **Wall Street Journal**, **Hub4 English Broadcast News** and **GALE Arabic** (around **5%** Relative improvements in WER and PPL)
- Model M can be expressed as a combination of two exponential n-gram models

$$p(w_1 \cdots w_l) = \prod_{j=1}^{l+1} p(c_j | c_1 \cdots c_{j-1}, w_1 \cdots w_{j-1}) \times \prod_{j=1}^{l} p(w_j | c_1 \cdots c_j, w_1 \cdots w_{j-1}) \quad (5)$$

Model M

- We can define (the trigram version of) Model M as

$$\begin{aligned}\textbf{ClassModel} &: p(c_j|c_1 \cdots c_{j-1}, w_1 \cdots w_{j-1}) \equiv p_{\text{ng}}(c_j|c_{j-2}c_{j-1}, w_{j-2}w_{j-1}) \\ \textbf{WordModel} &: p(w_j|c_1 \cdots c_j, w_1 \cdots w_{j-1}) \equiv p_{\text{ng}}(w_j|w_{j-2}w_{j-1}c_j)\end{aligned} \tag{6}$$

  The class model includes features induced from both word and class histories.
- The two exponential n-gram models can be trained independently of each other.
- Although Model M provides significant clear gains on a range of tasks, it comes with a steep computational cost at training time. Our reference implementation required around 30 hours of computation and 10GB of memory for 100M words of training data on a typical Xeon CPU.

Normalization sum computation: Overview

- For any two event histories x_1 and x_2, we can write the difference between their associated normalization terms as

$$Z(x_1) - Z(x_2) = \sum_{y \in Y} (\alpha(x_1, y) - \alpha(x_2, y)) \tag{7}$$

  where we define $\alpha(x, y) = \exp(\sum_{i=1}^{F} \lambda_i f_i(x, y))$, or the numerator on the right-hand side in eq. (1).
- For "sparse" models, most terms in eq. (7) will be 0 for most x_1 and x_2. For example, for an infrequent word y, only the unigram feature $f_{(\epsilon,y)}(\cdot)$ will be active for most x_1 and x_2, in which case $\alpha(x_1, y) = \alpha(x_2, y)$.

Normalization sum computation

- For the sequence of training set histories $x_1, x_2, \ldots, x_D$, we can compute normalization terms efficiently by just computing the difference between $Z(x_d)$ and $Z(x_{d+1})$.
- **Key Idea** Sort the corpus such that consecutive histories differ only in small number of words thus reducing the number of target words to sum over.

Normalization sum computation: history sorting

- Define a *feature history* $\mathbf{x}$ for each $\mathbf{x}$ occuring in a feature of the form $f_{(\mathbf{x},\mathbf{y})}(\cdot)$. For an exponential trigram model, for instance, there will be a feature history for each bigram, unigram, and empty history (*i.e.*, ϵ) occurring in the training data.
- The set of n-gram feature histories can be viewed as forming a tree with the empty history at the root. We assign each feature history a unique ID by doing a prefix traversal of the tree, so that lower-order histories are assigned lower ID's.
- In models with multiple sets of n-gram features, we can form a single tree by connecting each separate feature tree to a common root node

Normalization sum computation: history sorting

- For each training event history x_d, associate a list of active feature history ID's, sorted in ascending order.
- Sort event histories by sorting the associated (sorted) feature history ID lists lexicographically.
- In this way, consecutive event histories will tend to have very similar sets of active feature histories, so that $Z(x_{d+1}) - Z(x_d)$ can be computed efficiently. That is, consecutive event histories will tend to differ only in feature histories with high ID's, which tend to correspond to higher-order n-gram histories, which tend to co-occur in fewer features $f_{(\mathbf{x},\mathbf{y})}(\cdot)$.

Normalization sum computation

- Once event histories are sorted, we can compute all normalization terms $Z(x_d)$ in each training iteration by processing each event history in turn.
- For each event history x_d, we keep track of the values $\alpha(x_d, y)$ for all $y \in Y$ as well as $Z(x_d) = \sum_{y \in Y} \alpha(x_d, y)$.
- Moving from one event history to the next, we identify all feature histories $\mathbf{x}$ that differ, and compute $\alpha(x_{d+1}, y) - \alpha(x_d, y)$ for only those y such that the feature $f_{(\mathbf{x},y)}(\cdot)$ exists for one of these $\mathbf{x}$.
- For all other y, we have $\alpha(x_{d+1}, y) = \alpha(x_d, y)$, and $Z(x_{d+1})$ can be computed efficiently using eq. (7).

Expectation computation

- Given a feature $f_{(\mathbf{x},\mathbf{y})}(\cdot)$, from eq. (4) we see that its expectation is the sum of $\alpha(x_d, \mathbf{y})/Z(x_d)$ over those training event histories x_d where the feature history $\mathbf{x}$ is active.
- Consider a sequence of event histories $x_{d_1}, \ldots, x_{d_2}$ such that $\mathbf{x}$ is active in each history and $\alpha(x_d, \mathbf{y})$ is constant. Then, we have

$$\sum_{d=d_1}^{d_2} \frac{\alpha(x_d, \mathbf{y})}{Z(x_d)} = \alpha(x_{d_1}, \mathbf{y}) \times \sum_{d=d_1}^{d_2} Z^{-1}(x_d) \tag{8}$$

- By taking advantage of this property, instead of updating feature expectations for every $\mathbf{y}$ at every event history x_d, we need only update feature expectations for those $\mathbf{y}$ where the set of features active for the event $(x_d, \mathbf{y})$ changes
- The sum $\sum_{d=d_1}^{d_2} Z^{-1}(x_d)$ is independent of $\mathbf{y}$ and thus can be shared across target words.

Expectation computation

- The sorting of event histories performed for the normalization term computation attempts to change as few $\alpha(x_d, \mathbf{y})$ as possible between consecutive event histories; thus, this sorting makes the expectation computation more efficient as well.
- Specifically, we compute feature expectations by processing each event history x_d in turn, keeping track of active features for each (x_d, y) (usually just the unigram feature for most words).
- For each x_d, we first compute $Z(x_d)$ using the algorithm described earlier. Let $Z_d^{-1} = \sum_{d'=1}^{d} Z^{-1}(x_{d'})$, the sum of inverse normalizers so far, and let $Z_d^{-1}(y) = Z_{d'}^{-1}$ for the most recent event history $x_{d'}$ such that $(x_{d'}, y)$ and (x_d, y) have different sets of features active.

Expectation computation

- Then, for each y such that (x_d, y) and (x_{d+1}, y) have different sets of features active and for each feature $f_{(\mathbf{x},y)}(\cdot)$ active for event (x_d, y), we update the running total for $E(f_{(\mathbf{x},y)})$ by adding the quantity $\alpha(y)[Z_d^{-1} - Z_d^{-1}(y)]$.
- This has the effect of identifying maximal event history sequences with shared expectation computation
- Once we have the feature expectations $E(f_{(\mathbf{x},\mathbf{y})})$ for a training iteration, we can use unnormalized iterative scaling [Chen et al., 2010] or some other update algorithm to update the parameters

Relationship to prior art

- Cluster expansion introduced in [Lafferty and Suhm, 1995] provides the basis for efficient expectation and normalization sum computation.
- In [Wu and Khudanpur, 2002, Wu and Khudanpur, 2000], the authors describe an efficient scheme for training exponential n-gram models where they propogate the normalization sums from lower order to higher order *n*-grams and expectations from higher order to lower order *n*-grams.
  - The approach proposed in this paper is based on sorting the *n*-gram events in the corpus in a particular order and computing only the differentials between consecutive sorted events. No need to keep full tree of *n*-gram features in memory.
  - Simpler formulation for models containing multiple sets of *n*-gram features (like the class prediction model in Model M)

Parallelized training: Target vocabulary splits

- The normalization sum can be computed by summing the normalization terms computed over each vocabulary partition.
- The expectation computation can be split perfectly by splitting the target words across machines
- Store a subset of the model features and parameters on each machine, the memory required on each machine is reduced by a factor equal to the number of machines.
- Parameter reestimation can also be split across machines since the feature expectation computations are disjoint.
- Need a master process to merge the partial normalization terms from each machine and to broadcast the complete normalization terms back.

Parallelized training: Split by corpus

- Distribute each partition of the training corpus to a different machine.
- Normalization sum computation can be done on a single machine with no merge required.
- Each worker machine needs to load parameters for the features corresponding to the set of event histories it will process
- The memory required for storing the model is not partitioned efficiently.
- Need to merge feature expectations across multiple machines
- Parameter reestimation also cannot be split across machines.

Results

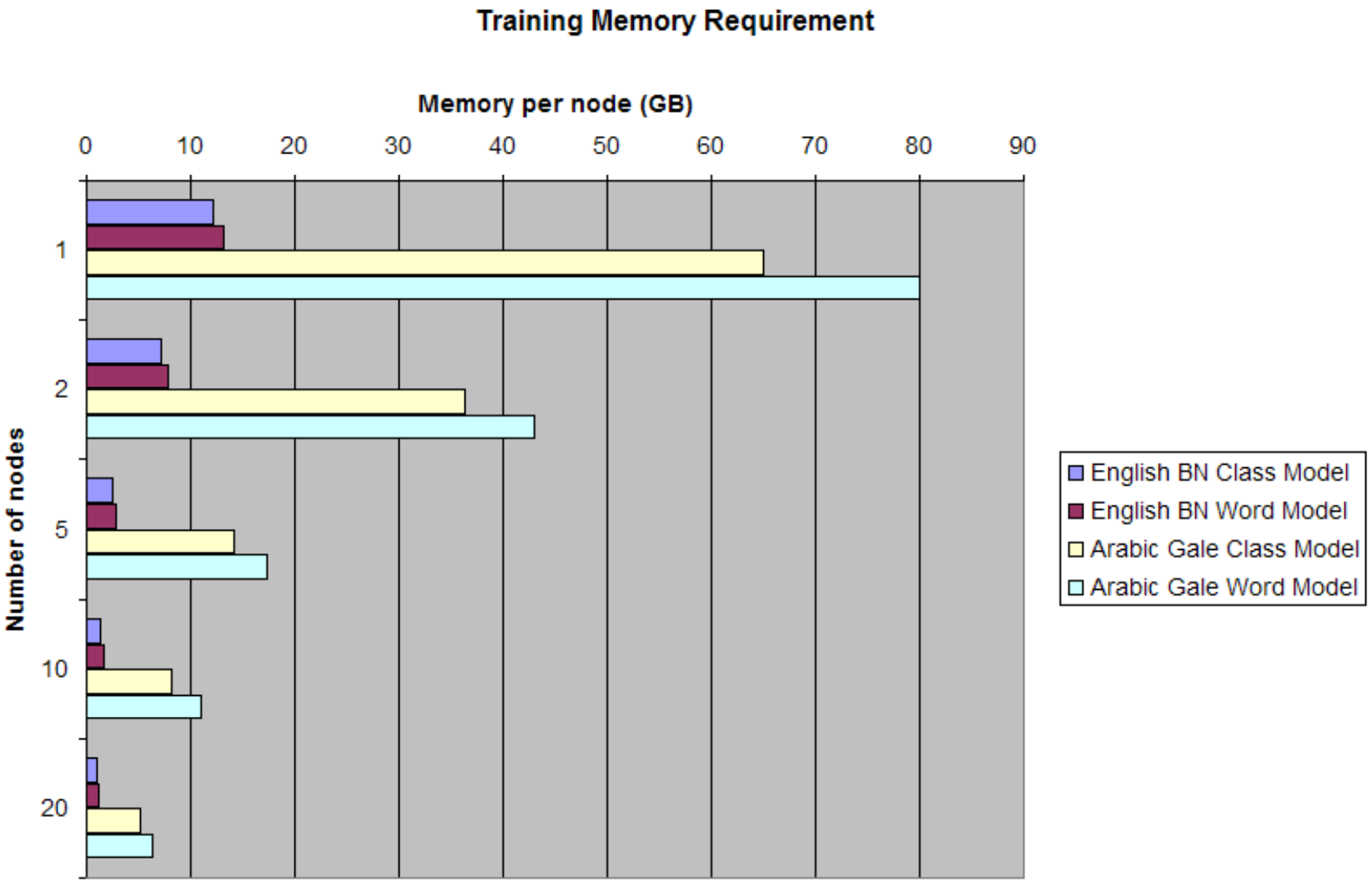
	Compute time
Direct implementation	4358
Unigram caching	84
Proposed Approach	6

Table 1: Compute time (in seconds) for estimating feature expectations with a direct implementation of Equation 4, unigram feature caching, and the proposed method.The corpus used for the results in Table 1 is the Hub4 AM training corpus with 1.5M words and a vocabulary of 87K.

Parallelization Results

- English Hub4 system : 335M words, 80K vocabulary
- English Hub 4 Model M: Class prediction model had 238M features with 86M distinct event histories and the word prediction model has 276M features with 97M distinct event histories
- Arabic Gale system : 1.6 billion words
- Arabic Gale Model M : Class prediction model has 750M features with 390M distinct event histories and the word prediction model has 940M features with 410M distinct event histories

Results

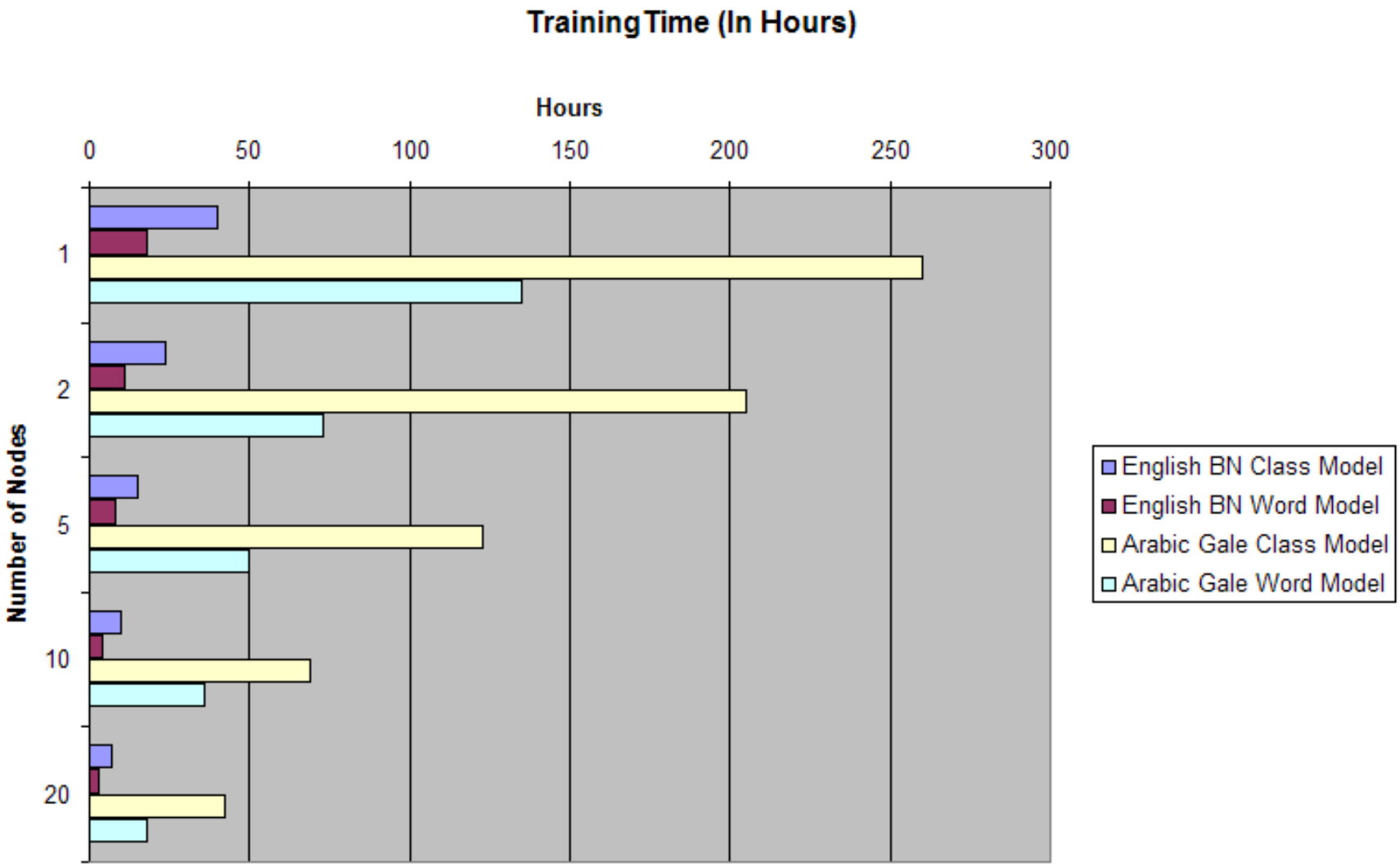

Training Memory Requirement
Memory per node (GB)
0
10
20
30
40
50
60
70
80
90
Number of nodes
1
2
5
10
20
English BN Class Model
English BN Word Model
Arabic Gale Class Model
Arabic Gale Word Model

Results

	Number of compute nodes				
	1	2	5	10	20
English BN (Class Model)	12.1	7.2	2.5	1.4	0.95
English BN (Word Model)	13.2	7.8	2.8	1.6	1.2
Arabic Gale (Class Model)	65	36.3	14.1	8.1	5.2
Arabic Gale (Word Model)	80	43	17.3	11	6.3

Table 2: Memory requirement (GB) per compute node

Results and conclusion



Results

	Number of compute nodes				
	1	2	5	10	20
English BN (Class Model)	40	24	15	10	7
English BN (Word Model)	18	11	8	4	3
Arabic Gale (Class Model)	260	205	123	69	42
Arabic Gale (Word Model)	135	73	50	36	18

Table 3: Compute time in hours for training word and class prediction components of Model M

Related work on parallelization

- Parallel training for maxent models is a well researched area. One of the first papers to discuss parallelized computation of exponential n-gram models was [Rosenfeld, 1996]
- Large scale distributed n-gram models are an active area of research [Brants et al., 2007, Emami et al., 2007]
- In [Mann et al., 2009] the authors investigated methods to minimize communication load while training large scale maximum entropy models in a distributed environment. The algorithms presented will likely hold for exponential n-gram training as well.

Some notes on computational cost

- We used unnormalized iterative scaling [Chen, 2009] for parameter reestimation. Other choices for parameter reestimation such as Conjugate gradient, LBFGS, RPROP were not studied.
- The timing results are with a strict convergence criteria of 0.0001 difference in objective function delta between iterations
- The computation cost is both a function of number of unique features and unique histories. In a typical case with unpruned models we can typically process around 1.2 million unique n-grams per second with 5 compute nodes.

Conclusion

- Efficient training algorithm for exponential n-gram models with binary features (such as Model M)
- Parallelization by vocabulary subsets and data subsets
  - Vocabulary subsets partition the model perfectly leading to better memory usage
  - Data subsets allow for higher number of splits

IEEE 2011 WORKSHOP ON AUTOMATIC SPEECH RECOGNITION AND UNDERSTANDING

Hilton Waikoloa Village, Big Island, Hawaii
11-15 December 2011
www.asru2011.org

Paper submission deadline	**1 July 2011**
Paper acceptance/rejection	20 August 2011
Early registration deadline	15 October 2011
Workshop	11-15 December 2011

IEEE Advancing Technology for Humanity
IEEE Signal Processing Society

References I

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007).

Large language models in machine translation.

In *Empirical Methods in Natural Language Processing*.

Chen, S. F. (2009).

Shrinking exponential language models.

In *Proceedings of NAACL-HLT*.

Chen, S. F., Mangu, L., Ramabhadran, B., Sarikaya, R., and Sethy, A. (2010).

Scaling shrinkage-based language models.

Technical Report RC 24970, IBM Research Division.

Darroch, J. and Ratcliff, D. (1972).

Generalized iterative scaling for log-linear models.

The Annals of Mathematical Statistics, 43:1470–1480.

References II

Emami, A., Papineni, K., and Sorensen, J. (2007).

Large-scale distributed language modeling.

In *Proceedings of ICASSP*.

Lafferty, J. and Suhm, B. (1995).

Cluster expansions and iterative scaling for maximum entropy language models.

In Hanson, K. and Silver, R., editors, *Maximum Entropy and Bayesian Methods*, pages 195–202. Kluwer Academic Publishers.

Malouf, R. (2002).

A comparison of algorithms for maximum entropy parameter estimation.

In *proceedings of the 6th conference on Natural language learning - Volume 20*, COLING-02.

References III

Mann, G., McDonald, R., Mohri, M., Silberman, N., and Walker, D. D. (2009).

Efficient large-scale distributed training of conditional maximum entropy models.

In *Proceedings of NIPS*.

Rosenfeld, R. (1996).

A maximum entropy approach to adaptive statistical language modeling.

Computer Speech and Language, 10:187–228.

Wu, J. and Khudanpur, S. (2000).

Efficient training methods for maximum entropy language modeling.

In *Proceedings of ICSLP2000*.

References IV

Wu, J. and Khudanpur, S. (2002).

Building a topic-dependent maximum entropy language model for very large corpora.

In *Proceedings of ICASSP2002*.