Streaming for large scale NLP: Language Modeling

Amit Goyal, Hal Daumé III, and Suresh Venkatasubramanian
University of Utah, School of Computing
{amitg,hal,suresh}@cs.utah.edu

Abstract

In this paper, we explore a streaming algorithm paradigm to handle large amounts of data for NLP problems. We present an efficient low-memory method for constructing high-order approximate n-gram frequency counts. The method is based on a deterministic streaming algorithm which efficiently computes approximate frequency counts over a stream of data while employing a small memory footprint. We show that this method easily scales to billion-word monolingual corpora using a conventional (8 GB RAM) desktop machine. Statistical machine translation experimental results corroborate that the resulting high-n approximate *small* language model is as effective as models obtained from other count pruning methods.

1 Introduction

In many NLP problems, we are faced with the challenge of dealing with large amounts of data. Many problems boil down to computing relative frequencies of certain items on this data. Items can be words, patterns, associations, n-grams, and others. Language modeling (Chen and Goodman, 1996), noun-clustering (Ravichandran et al., 2005), constructing syntactic rules for SMT (Galley et al., 2004), and finding analogies (Turney, 2008) are examples of some of the problems where we need to compute relative frequencies. We use language modeling as a canonical example of a large-scale task that requires relative frequency estimation.

Computing relative frequencies seems like an easy problem. However, as corpus sizes grow, it becomes a highly computational expensive task. Brants et al. (2007) used 1500 machines for a day to compute the relative frequencies of n-grams (summed over all orders from 1 to 5) from 1.8TB of web data. Their resulting model contained 300 million unique n-grams.

Cutoff	Size	BLEU	NIST	MET
Exact	367.6m	**28.73**	**7.691**	**56.32**
2	229.8m	28.23	7.613	56.03
3	143.6m	28.17	7.571	**56.53**
5	59.4m	28.33	7.636	56.03
10	18.3m	27.91	7.546	55.64
100	1.1m	28.03	7.607	55.91
200	0.5m	27.62	7.550	55.67

Table 1: Effect of count-based pruning on SMT performance using EAN corpus. Results are according to BLEU, NIST and METEOR (MET) metrics. Bold #s are not statistically significant worse than exact model.

It is not realistic using conventional computing resources to use all the 300 million n-grams for applications like speech recognition, spelling correction, information extraction, and statistical machine translation (SMT). Hence, one of the easiest way to reduce the size of this model is to use count-based pruning which discards all n-grams whose count is less than a pre-defined threshold. Although count-based pruning is quite simple, yet it is effective for machine translation. As we do not have a copy of the web, we will use a portion of gigaword i.e. EAN (see Section 4.1) to show the effect of count-based pruning on performance of SMT (see Section 5.1). Table 1 shows that using a cutoff of 100 produces a model of size 1.1 million n-grams with a Bleu score of 28.03. If we compare this with an exact model of size 367.6 million n-grams, we see an increase of 0.8 points in Bleu (95% statistical significance level

Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the ACL, pages 512–520,
Boulder, Colorado, June 2009.

ϵ	Size	BLEU	NIST	MET
Exact	367.6m	**28.73**	**7.691**	**56.32**
1e-10	218.4m	**28.64**	**7.669**	**56.33**
5e-10	171.0m	**28.48**	**7.666**	**56.38**
1e-9	148.0m	**28.56**	7.646	**56.51**
5e-9	91.9m	28.27	7.623	56.16
1e-8	69.4m	28.15	7.609	56.19
5e-7	28.5m	28.08	7.595	55.91

Table 2: Effect of entropy-based pruning on SMT performance using EAN corpus. Results are as in Table 1

is ≈ 0.53 Bleu). However, we need 300 times bigger model to get such an increase. Unfortunately, it is not possible to integrate such a big model inside a decoder using normal computation resources.

A better way of reducing the size of n-grams is to use entropy pruning (Stolcke, 1998). Table 2 shows the results with entropy pruning with different settings of ϵ. We see that for three settings of ϵ equal to $1e$-10, $5e$-10 and $1e$-9, we get Bleu scores comparable to the exact model. However, the size of all these models is not at all small. The size of smallest model is 25% of the exact model. Even with this size it is still not feasible to integrate such a big model inside a decoder. If we take a model of size comparable to count cutoff of 100, i.e., with ϵ = 5e-7, we see both count-based pruning as well as entropy pruning performs the same.

There also have been prior work on maintaining approximate counts for higher-order language models (LMs) ((Talbot and Osborne, 2007a; Talbot and Osborne, 2007b; Talbot and Brants, 2008)) operates under the model that the goal is to store a compressed representation of a disk-resident table of counts and use this compressed representation to answer count queries approximately.

There are two difficulties with scaling all the above approaches as the order of the LM increases. Firstly, the computation time to build the database of counts increases rapidly. Secondly, the initial disk storage required to maintain these counts, prior to building the compressed representation is enormous.

The method we propose solves both of these problems. We do this by making use of the *streaming algorithm* paradigm (Muthukrishnan, 2005). Working under the assumption that multiple-GB models are infeasible, our goal is to instead of estimating a large model and then compressing it, we directly estimate a small model. We use a deterministic streaming algorithm (Manku and Motwani, 2002) that computes approximate frequency counts of frequently occurring n-grams. This scheme is considerably more accurate in getting the actual counts as compared to other schemes (Demaine et al., 2002; Karp et al., 2003) that find the set of frequent items without caring about the accuracy of counts.

We use these counts directly as features in an SMT system, and propose a direct way to integrate these features into an SMT decoder. Experiments show that directly storing approximate counts of frequent 5-grams compared to using count or entropy-based pruning counts gives equivalent SMT performance, while dramatically reducing the memory usage and getting rid of pre-computing a large model.

2 Background

2.1 n-gram Language Models

Language modeling is based on assigning probabilities to sentences. It can either compute the probability of an entire sentence or predict the probability of the next word in a sequence. Let w_1^m denote a sequence of words $(w_1, \ldots, w_m)$. The probability of estimating word w_m depends on previous n-1 words where n denotes the size of n-gram. This assumption that probability of predicting a current word depends on the previous words is called a Markov assumption, typically estimated by relative frequency:

$$P(w_m \mid w_{m-n+1}^{m-1}) = \frac{C(w_{m-n+1}^{m-1} w_m)}{C(w_{m-n+1}^{m-1})} \qquad (1)$$

Eq 1 estimates the n-gram probability by taking the ratio of observed frequency of a particular sequence and the observed frequency of the prefix. This is precisely the relative frequency estimate we seek.

2.2 Large-scale Language modeling

Using higher order LMs to improve the accuracy of SMT is not new. (Brants et al., 2007; Emami et al., 2007) built 5-gram LMs over web using distributed cluster of machines and queried them via network requests. Since the use of cluster of machines is not always practical, (Talbot and Osborne, 2007b; Talbot and Osborne, 2007a) showed a randomized data structure called Bloom filter, that can be used to construct space efficient language models

for SMT. (Talbot and Brants, 2008) presented randomized language model based on perfect hashing combined with entropy pruning to achieve further memory reductions. A problem mentioned in (Talbot and Brants, 2008) is that the algorithm that computes the compressed representation might need to retain the entire database in memory; in their paper, they design strategies to work around this problem. (Federico and Bertoldi, 2006) also used single machine and fewer bits to store the LM probability by using efficient prefix trees.

(Uszkoreit and Brants, 2008) used partially class-based LMs together with word-based LMs to improve SMT performance despite the large size of the word-based models used. (Schwenk and Koehn, 2008; Zhang et al., 2006) used higher language models at time of re-ranking rather than integrating directly into the decoder to avoid the overhead of keeping LMs in the main memory since disk lookups are simply too slow. Now using higher order LMs at time of re-ranking looks like a good option. However, the target n-best hypothesis list is not diverse enough. Hence if possible it is always better to integrate LMs directly into the decoder.

2.3 Streaming

Consider an algorithm that reads the input from a read-only *stream* from left to right, with no ability to go back to the input that it has already processed. This algorithm has working storage that it can use to store parts of the input or other intermediate computations. However, (and this is a critical constraint), this working storage space is significantly smaller than the input stream length. For typical algorithms, the storage size is of the order of $\log^k N$, where N is the input size and k is some constant.

Stream algorithms were first developed in the early 80s, but gained in popularity in the late 90s as researchers first realized the challenges of dealing with massive data sets. A good survey of the model and core challenges can be found in (Muthukrishnan, 2005). There has been considerable work on the problem of identifying high-frequency items (items with frequency above a threshold), and a detailed review of these methods is beyond the scope of this article. A new survey by (Cormode and Hadjieleftheriou, 2008) comprehensively reviews the literature.

3 Space-Efficient Approximate Frequency Estimation

Prior work on approximate frequency estimation for language models provide a "no-false-negative" guarantee, ensuring that counts for n-grams in the model are returned exactly, while working to make sure the false-positive rate remains small (Talbot and Osborne, 2007a). The notion of approximation we use is different: in our approach, it is the actual count values that will be approximated. We also exploit the fact that low-frequency n-grams, while constituting the vast majority of the set of unique n-grams, are usually smoothed away and are less likely to influence the language model significantly. Discarding low-frequency n-grams is particularly important in a stream setting, because it can be shown in general that any algorithm that generates approximate frequency counts for all n-grams requires space linear in the input stream (Alon et al., 1999).

We employ an algorithm for approximate frequency counting proposed by (Manku and Motwani, 2002) in the context of database management. Fix parameters $s \in (0, 1)$, and $\epsilon \in (0, 1), \epsilon \ll s$. Our goal is to approximately find all n-grams with frequency at least sN. For an input stream of n-grams of length N, the algorithm outputs a set of items (and frequencies) and guarantees the following:

- All items with frequencies exceeding sN are output (*no false negatives*).
- No item with frequency less than $(s - \epsilon)N$ is output (*few false positives*).
- All reported frequencies are less than the true frequencies by at most ϵN (*close-to-exact frequencies*).
- The space used by the algorithm is $O(\frac{1}{\epsilon} \log \epsilon N)$.

A simple example illustrates these properties. Let us fix $s = 0.01, \epsilon = 0.001$. Then the algorithm guarantees that all n-grams with frequency at least 1% will be returned, no element with frequency less than 0.9% will be returned, and all frequencies will be no more than 0.1% away from the true frequencies. The space used by the algorithm is $O(\log N)$, which can be compared to the much larger (close to N) space

needed to store the initial frequency counts. In addition, the algorithm runs in linear time by definition, requiring only one pass over the input. Note that there might be $\frac{1}{\epsilon}$ elements with frequency at least ϵN, and so the algorithm uses optimal space (up to a logarithmic factor).

3.1 The Algorithm

We present a high-level overview of the algorithm; for more details, the reader is referred to (Manku and Motwani, 2002). The algorithm proceeds by conceptually dividing the stream into *epochs*, each containing $1/\epsilon$ elements. Note that there are ϵN epochs. Each such epoch has an ID, starting from 1. The algorithm maintains a list of tuples[1] of the form (e, f, Δ), where e is an n-gram, f is its reported frequency, and Δ is the maximum error in the frequency estimation. While the algorithm reads n-grams associated with the current epoch, it does one of two things: if the new element e is contained in the list of tuples, it merely increments the frequency count f. If not, it creates a new tuple of the form $(e, 1, T-1)$, where T is the ID of the current epoch.

After each epoch, the algorithm "cleans house" by eliminating tuples whose maximum true frequency is small. Formally, if the epoch that just ended has ID T, then the algorithm deletes all tuples satisfying condition $f + \Delta \leq T$. Since $T \leq \epsilon N$, this ensures that no low-frequency tuples are retained. When all elements in the stream have been processed, the algorithm returns all tuples (e, f, Δ) where $f \geq (s-\epsilon)N$. In practice, however we do not care about s and return all tuples. At a high level, the reason the algorithm works is that if an element has high frequency, it shows up more than once each epoch, and so its frequency gets updated enough to stave off elimination.

4 Intrinsic Evaluation

We conduct a set of experiments with approximate n-gram counts (stream counts) produced by the stream algorithm. We define various metrics on which we evaluate the quality of stream counts compared with exact n-gram counts (true counts). To evaluate the quality of stream counts on these metrics, we carry out three experiments.

[1]We use hash tables to store tuples; however smarter data structures like suffix trees could also be used.

Corpus	Gzip-MB	M-wrds	Perplexity
EP	63	38	1122.69
afe	417	171	1829.57
apw	1213	540	1872.96
nyt	2104	914	1785.84
xie	320	132	1885.33

Table 3: Corpus Statistics and perplexity of LMs made with each of these corpuses on development set

4.1 Experimental Setup

The freely available English side of Europarl (EP) and Gigaword corpus (Graff, 2003) is used for computing n-gram counts. We only use EP along with two sections of the Gigaword corpus: Agence France Press English Service(afe) and The New York Times Newswire Service (nyt). The unigram language models built using these corpuses yield better perplexity scores on the development set (see Section 5.1) compared to The Xinhua News Agency English Service (xie) and Associated Press Worldstream English Service (apw) as shown in Table 3. The LMs are build using the SRILM language modelling toolkit (Stolcke, 2002) with modified Kneser-Ney discounting and interpolation. The evaluation of stream counts is done on EP+afe+nyt (EAN) corpus, consisting of 1.1 billion words.

4.2 Description of the metrics

To evaluate the quality of counts produced by our stream algorithm four different metrics are used. The accuracy metric measures the quality of top N stream counts by taking the fraction of top N stream counts that are contained in the top N true counts.

$$\text{Accuracy} = \frac{\text{Stream Counts} \cap \text{True Counts}}{\text{True Counts}}$$

Spearman's rank correlation coefficient or Spearman's rho(ρ) computes the difference between the ranks of each observation (i.e. n-gram) on two variables (that are top N stream and true counts). This measure captures how different the stream count ordering is from the true count ordering.

$$\rho = 1 - \frac{6 \sum d_i^2}{N(N^2-1)}$$

d_i is the difference between the ranks of corresponding elements X_i and Y_i; N is the number of elements found in both sets; X_i and Y_i in our case denote the stream and true counts.

Mean square error (MSE) quantifies the amount by which a predicted value differs from the true value. In our case, it estimates how different the stream counts are from the true counts.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\text{true}_i - \text{predicted}_i)^2$$

true and predicted denotes values of true and stream counts; N denotes the number of stream counts contained in true counts.

4.3 Varying ϵ experiments

In our first experiment, we use accuracy, ρ and MSE metrics for evaluation. Here, we compute 5-gram stream counts with different settings of ϵ on the EAN corpus. ϵ controls the number of stream counts produced by the algorithm. The results in Table 4 support the theory that decreasing the value of ϵ improves the quality of stream counts. Also, as expected, the algorithm produces more stream counts with smaller values of ϵ. The evaluation of stream counts obtained with ϵ = $50e$-8 and $20e$-8 reveal that the stream counts learned with this large value are more susceptible to errors.

If we look closely at the counts for ϵ = $50e$-8, we see that we get at least 30% of the stream counts from $245k$ true counts. This number is not significantly worse than the 36% of stream counts obtained from $4,018k$ true counts for the smallest value of $\epsilon = 5$e-8. However, if we look at the other two metrics, the ranking correlation ρ of stream counts compared with true counts on ϵ = $50e$-8 and $20e$-8 is low compared to other ϵ values. For the MSE, the error with stream counts on these ϵm values is again high compared to other values. As we decrease the value of ϵ we continually get better results: decreasing ϵ pushes the stream counts towards the true counts. However, using a smaller ϵ increases the memory usage. Looking at the evaluation, it is therefore advisable to use 5-gram stream counts produced with at most $\epsilon \leq 10e$-7 for the EAN corpus.

Since it is not possible to compute true 7-grams counts on EAN with available computing resources, we carry out a similar experiment for 7-grams on EP to verify the results for higher order n-grams [2]. The results in Table 5 tell a story similar to our results for 7-grams. The size of EP corpus is much smaller than EAN and so we see even better results on each of the metrics with decreasing the value of ϵ. The overall trend remains the same; here too, setting $\epsilon \leq 10e$-8 is the most effective strategy. The fact that these results are consistent across two datasets of different sizes and different n-gram sizes suggests that they will carry over to other tasks.

ϵ	5-**gram produced**	**Acc**	ρ	**MSE**
50e-8	245k	0.294	-3.6097	0.4954
20e-8	726k	0.326	-2.6517	0.1155
10e-8	1655k	0.352	-1.9960	0.0368
5e-8	4018k	0.359	-1.7835	0.0114

Table 4: Evaluating quality of 5-gram stream counts for different settings of ϵ on EAN corpus

ϵ	7-**gram produced**	**Acc**	ρ	**MSE**
50e-8	44k	0.509	0.3230	0.0341
20e-8	128k	0.596	0.5459	0.0063
10e-8	246k	0.689	0.7413	0.0018
5e-8	567k	0.810	0.8599	0.0004

Table 5: Evaluating quality of 7-gram stream counts for different settings of ϵ on EP corpus

4.4 Varying top K experiments

In the second experiment, we evaluate the quality of the top K (sorted by frequency) 5-gram stream counts. Here again, we use accuracy, ρ and MSE for evaluation. We fix the value of ϵ to $5e$-8 and compute 5-gram stream counts on the EAN corpus. We vary the value of K between $100k$ and $4,018k$ (i.e all the n-gram counts produced by the stream algorithm). The experimental results in Table 6 support the theory that stream count algorithm computes the exact count of most of the high frequency n-grams. Looking closer, we see that if we evaluate the algorithm on just the top $100k$ 5-grams (roughly 5% of all 5-grams produced), we see almost perfect results. Further, if we take the top $1,000k$ 5-grams (approximately 25% of all 5-grams) we again see excellent

[2]Similar evaluation scores are observed for 9-gram stream counts with different values of ϵ on EP corpus.

Top K	Accuracy	ρ	MSE
100k	0.994	0.9994	0.01266
500k	0.934	0.9795	0.0105
1000k	0.723	0.8847	0.0143
2000k	0.504	0.2868	0.0137
4018k	0.359	-1.7835	0.0114

Table 6: Evaluating top K sorted 5-gram stream counts for ϵ=5e-8 on EAN corpus

Top K	Accuracy	ρ	MSE
10k	0.996	0.9997	0.0015
20k	0.989	0.9986	0.0016
50k	0.950	0.9876	0.0016
100k	0.876	0.9493	0.0017
246k	0.689	0.7413	0.0018

Table 7: Evaluating top K sorted 7-gram stream counts for ϵ=10e-8 on EP corpus

performance on all metrics. The accuracy of the results decrease slightly, but the ρ and MSE metrics are not decreased that much in comparison. Performance starts to degrade as we get to $2,000k$ (over 50% of all 5-grams), a result that is not too surprising. However, even here we note that the MSE is low, suggesting that the frequencies of stream counts (found in top K true counts) are very close to the true counts. Thus, we conclude that the quality of the 5-gram stream counts produced for this value of ϵ is quite high (in relation to the true counts).

As before, we corroborate our results with higher order n-grams. We evaluate the quality of top K 7-gram stream counts on EP.[3] Since EP is a smaller corpus, we evaluate the stream counts produced by setting ϵ to $10e$-8. Here we vary the value of K between $10k$ and $246k$ (the total number produced by the stream algorithm). Results are shown in Table 7. As we saw earlier with 5-grams, the top $10k$ (i.e. approximately 5% of all 7-grams) are of very high quality. Results, and this remains true even when we increase K to $100k$. There is a drop in the accuracy and a slight drop in ρ, while the MSE remains the same. Taking all counts again shows a significant decrease in both accuracy and ρ scores, but this does not affect MSE scores significantly. Hence, the 7-gram stream counts i.e. $246k$ counts produced by ϵ = $10e$-8 are quite accurate when compared to the top $246k$ true counts.

4.5 Analysis of tradeoff between coverage and space

In our third experiment, we investigate whether a large LM can help MT performance. We evaluate the coverage of stream counts built on the EAN corpus on the test data for SMT experiments (see Section 5.1) with different values of ϵm. We compute the recall of each model against 3071 sentences of test data where recall is the fraction of number of n-grams of a dataset found in stream counts.

$$\text{Recall} = \frac{\text{Number of } n\text{-grams found in stream counts}}{\text{Number of } n\text{-grams in dataset}}$$

We build unigram, bigram, trigram, 5-gram and 7-gram with four different values of ϵ. Table 8 contains the `gzip` size of the count file and the recall of various different stream count n-grams. As expected, the recall with respect to true counts is maximum for unigrams, bigrams, trigrams and 5-grams. However the amount of space required to store all true counts in comparison to stream counts is extremely high: we need $4.8GB$ of compressed space to store all the true counts for 5-grams.

For unigram models, we see that the recall scores are good for all values of ϵ. If we compare the approximate stream counts produced by largest ϵ (which is worst) to all true counts, we see that the stream counts compressed size is 50 times smaller than the true counts size, and is only three points worse in recall. Similar trends hold for bigrams, although the loss in recall is higher. As with unigrams, the loss in recall is more than made up for by the memory savings (a factor of nearly 150). For trigrams, we see a 14 point loss in recall for the smallest ϵ, but a memory savings of 400 times. For 5-grams, the best recall value is .020 ($1.2k$ out of $60k$ 5-gram stream counts are found in the test set). However, compared with the true counts we only loss a recall of 0.05 ($4.3k$ out of $60k$) points but memory savings of 150 times. In extrinsic evaluations, we will show that integrating 5-gram stream counts with an SMT system performs slightly worse than the true counts, while dramatically reducing the memory usage.

[3] Similar evaluation scores are observed for different top K sorted 9-gram stream counts with ϵ=10e-8 on EP corpus.

N-gram	unigram		bigram		trigram		5-gram		7-gram	
ϵ	Gzip MB	Recall	Gzip MB	Recall	Gzip MB	Recall	Gzip MB	Recall	Gzip MB	Recall
50e-8	.352	.785	2.3	.459	3.3	.167	1.9	.006	.864	5.6e-5
20e-8	.568	.788	4.5	.494	7.6	.207	5.3	.011	2.7	1.3e-4
10e-8	.824	.791	7.6	.518	15	.237	13	.015	9.7	4.1e-4
5e-8	1.3	.794	13	.536	30	.267	31	.020	43	5.9e-4
all	17	.816	228	.596	1200	.406	4800	.072	NA	

Table 8: Gzipped space required to store n-gram counts on disk and their coverage on a test set with different ϵm

For 7-gram we can not compute the true n-gram counts due to limitations of available computational resources. The memory requirements with smallest value of ϵ are similar to those of 5-gram, but the recall values are quite small. For 7-grams, the best recall value is $5.9e$-4 which means that stream counts contains only 32 out of $54k$ 7-grams contained in test set. The small recall value for 7-grams suggests that these counts may not be that useful in SMT. We further substantiate our findings in our extrinsic evaluations. There we show that integrating 7-gram stream counts with an SMT system does not affect its overall performance significantly.

5 Extrinsic Evaluation

5.1 Experimental Setup

All the experiments conducted here make use of publicly available resources. Europarl (EP) corpus French-English section is used as parallel data. The publicly available Moses[4] decoder is used for training and decoding (Koehn and Hoang, 2007). The news corpus released for ACL SMT workshop in 2007 consisting of 1057 sentences[5] is used as the development set. Minimum error rate training (MERT) is used on this set to obtain feature weights to optimize translation quality. The final SMT system performance is evaluated on a uncased test set of 3071 sentences using the BLEU (Papineni et al., 2002), NIST (Doddington, 2002) and METEOR (Banerjee and Lavie, 2005) scores. The test set is the union of the 2007 news devtest and 2007 news test data from ACL SMT workshop 2007.[6]

[4]http://www.statmt.org/moses/

[5]http://www.statmt.org/wmt07/

[6]We found that testing on Parliamentary test data was completely insensitive to large n-gram LMs, even when these LMs are exact. This suggests that for SMT performance, more data is better *only if* it comes from the right domain.

5.2 Integrating stream counts feature into decoder

Our method only computes high-frequency n-gram counts; it does not estimate conditional probabilities. We can either turn these counts into conditional probabilities (by using SRILM) or use the counts directly. We observed no significant difference in performance between these two approaches. However, using the counts directly consumes significantly less memory at run-time and is therefore preferable. Due to space constraints, SRILM results are omitted.

The only remaining open question is: *how should we turn the counts into a feature that can be used in an SMT system?* We considered several alternatives; the most successful was a simple weighted count of n-gram matches of varying size, appropriately backed-off. Specifically, consider an n-gram model. For every sequence of words $w_i, \ldots, w_{i+N-1}$, we obtain a feature score computed recursively according to Eq (2).

$$\begin{aligned} f(w_i) &= \log\left(\frac{C(w_i)}{Z}\right) \\ f(w_i, \ldots, w_{i+k}) &= \log\left(\frac{C(w_i, \ldots, w_{i+k})}{Z}\right) \\ &+ \frac{1}{2} f(w_{i+1}, \ldots, w_{i+k}) \end{aligned} \quad (2)$$

Here, $\frac{1}{2}$ is the backoff factor and Z is the largest count in the count set (the presence of Z is simply to ensure that these values remain manageable). In order to efficiently compute these features, we store the counts in a suffix-tree. The computation proceeds by first considering w_{i+N-1} alone and then "expanding" to consider the bigram, then trigram and so on. The advantage to this order of computation is that the recursive calls can cease whenever a

n-gram(ϵ)	BLEU	NIST	MET	Mem GB
3 EP(exact)	25.57	7.300	54.48	2.7
5 EP(exact)	25.79	7.286	54.44	2.9
3 EAN(exact)	27.04	7.428	55.07	4.6
5 EAN(exact)	**28.73**	**7.691**	**56.32**	20.5
4(10e-8)	27.36	7.506	**56.19**	2.7
4(5e-8)	27.40	7.507	55.90	2.8
5(10e-8)	27.97	7.605	55.52	2.8
5(5e-8)	27.98	**7.611**	56.07	2.8
7(10e-8)	27.97	7.590	55.88	2.9
7(5e-8)	27.88	7.577	56.01	2.9
9(10e-8)	**28.18**	**7.611**	55.95	2.9
9(5e-8)	27.98	7.608	56.08	2.9

Table 9: Evaluating SMT with different LMs on EAN. Results are according to BLEU, NIST and MET metrics. Bold #s are not statistically significant worse than exact.

zero count is reached. (Extending Moses to include this required only about 100 lines of code.)

5.3 Results

Table 9 summarizes SMT results. We have 4 baseline LMs that are conventional LMs smoothed using modified Kneser-Ney smoothing. The first two trigram and 5-gram LMs are built on EP corpus and the other two are built on EAN corpus. Table 9 show that there is not much significant difference in SMT results of 5-gram and trigram LM on EP. As expected, the trigram built on the large corpus EAN gets an improvement of 1.5 Bleu Score. However, unlike the EP corpus, building a 5-gram LM on EAN (huge corpus) gets an improvement of 3.2 Bleu Score. (The 95% statistical significance boundary is about $\pm$ 0.53 Bleu on the test data, 0.077 Nist and 0.16 Meteor according to bootstrap resampling) We see similar gains in Nist and Meteor metrics as shown in Table 9.

We use stream counts computed with two values of ϵ, 5e-8 and 10e-8 on EAN corpus. We use all the stream counts produced by the algorithm. 4, 5, 7 and 9 order n-gram stream counts are computed with these settings of ϵ. These counts are used along with a trigram LM built on EP to improve SMT performance. The memory usage (Mem) shown in Table 9 is the full memory size required to run on the test data (including phrase tables).

Adding 4-gram and 5-gram stream counts as feature helps the most. The performance gain by using 5-gram stream counts is slightly worse than compared to true 5-gram LM on EAN. However, using 5-gram stream counts directly is more memory efficient. Also, the gains for stream counts are exactly the same as we saw for same sized count-based and entropy-based pruning counts in Table 1 and 2 respectively. Moreover, unlike the pruning methods, our algorithm directly computes a small model, as opposed to compressing a pre-computed large model.

Adding 7-gram and 9-gram does not help significantly, a fact anticipated by the low recall of 7-gram-based counts that we saw in Section 4.5. The results with two different settings of ϵ are largely the same. This validates our intrinsic evaluation results in Section 4.3 that stream counts learned using $\epsilon \leq$ 10e-8 are of good quality, and that the quality of the stream counts is high.

6 Conclusion

We have proposed an efficient, low-memory method to construct high-order approximate n-gram LMs. Our method easily scales to billion-word monolingual corpora on conventional (8GB) desktop machines. We have demonstrated that approximate n-gram features could be used as a direct replacement for conventional higher order LMs in SMT with significant reductions in memory usage. In future, we will be looking into building streaming skip n-grams, and other variants (like cluster n-grams).

In NLP community, it has been shown that having more data results in better performance (Ravichandran et al., 2005; Brants et al., 2007; Turney, 2008). At web scale, we have terabytes of data and that can capture broader knowledge. Streaming algorithm paradigm provides a memory and space-efficient platform to deal with terabytes of data. We hope that other NLP applications (where we need to compute relative frequencies) like noun-clustering, constructing syntactic rules for SMT, finding analogies, and others can also benefit from streaming methods. We also believe that stream counts can be applied to other problems involving higher order LMs such as speech recognition, information extraction, spelling correction and text generation.

References

Noga Alon, Yossi Matias, and Mario Szegedy. 1999. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1).

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

S. Chen and J. Goodman. 1996. An Empirical Study of Smoothing Techniques for Language Modeling. In *Proceedings of 34th Annual Meeting of the Association for Computational Linguistics*, pages 310–318, Santa Cruz, CA, June.

Graham Cormode and Marios Hadjieleftheriou. 2008. Finding frequent items in data streams. In *VLDB*.

E.D. Demaine, A. Lopez-Ortiz, and J.I. Munro. 2002. Frequency estimation of internet packet streams with limited space.

George Doddington. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*.

Ahmad Emami, Kishore Papineni, and Jeffrey Sorensen. 2007. Large-scale distributed language modeling. In *Proceedings of the 2007 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages 37–40.

Marcello Federico and Nicola Bertoldi. 2006. How many bits are needed to store probabilities for phrase-based translation? In *Proceedings on the Workshop on Statistical Machine Translation at ACL06*.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proceedings of HLT/NAACL-04*.

D. Graff. 2003. English Gigaword. Linguistic Data Consortium, Philadelphia, PA, January.

Richard M. Karp, Christos H. Papadimitriou, and Scott Shenker. 2003. A simple algorithm for finding frequent elements in streams and bags.

Philipp Koehn and Hieu Hoang. 2007. Factored translation models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 868–876.

G. S. Manku and R. Motwani. 2002. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*.

S. Muthukrishnan. 2005. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2).

K. Papineni, S. Roukos, T. Ward, and W. Zhu. 2002. Bleu: a method for automatic evaluation of machine translation.

Deepak Ravichandran, Patrick Pantel, and Eduard Hovy. 2005. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*.

Holger Schwenk and Philipp Koehn. 2008. Large and diverse language models for statistical machine translation. In *Proceedings of The Third International Joint Conference on Natural Language Processing (IJCNP)*.

Andreas Stolcke. 1998. Entropy-based pruning of backoff language models. In *In Proc. DARPA Broadcast News Transcription and Understanding Workshop*.

A. Stolcke. 2002. SRILM – An Extensible Language Modeling Toolkit. In *Proceedings of the 7th International Conference on Spoken Language Processing*, pages 901–904, Denver, CO, September.

David Talbot and Thorsten Brants. 2008. Randomized language models via perfect hash functions. In *Proceedings of ACL-08: HLT*.

David Talbot and Miles Osborne. 2007a. Randomised language modelling for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*.

David Talbot and Miles Osborne. 2007b. Smoothed Bloom filter language models: Tera-scale LMs on the cheap. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EM NLP-CoNLL)*.

Peter D. Turney. 2008. A uniform approach to analogies, synonyms, antonyms, and associations. In *Proceedings of COLING 2008*.

Jakob Uszkoreit and Thorsten Brants. 2008. Distributed word clustering for large scale class-based language modeling in machine translation. In *Proceedings of ACL-08: HLT*.

Ying Zhang, Almut Silja Hildebrand, and Stephan Vogel. 2006. Distributed language modeling for n-best list re-ranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*.