Stream-based Randomised Language Models for SMT

Abby Levenberg
School of Informatics
University of Edinburgh
a.levenberg@sms.ed.ac.uk

Miles Osborne
School of Informatics
University of Edinburgh
miles@inf.ed.ac.uk

Abstract

Randomised techniques allow very big language models to be represented succinctly. However, being batch-based they are unsuitable for modelling an unbounded stream of language whilst maintaining a constant error rate. We present a novel randomised language model which uses an *online perfect hash* function to efficiently deal with unbounded text streams. Translation experiments over a text stream show that our online randomised model matches the performance of batch-based LMs without incurring the computational overhead associated with full retraining. This opens up the possibility of randomised language models which continuously adapt to the massive volumes of texts published on the Web each day.

1 Introduction

Language models (LM) are an integral feature of statistical machine translation (SMT) systems. They assign probabilities to generated hypotheses in the target language informing lexical selection. The most common form of LMs in SMT systems are smoothed n-gram models which predict a word based on a contextual history of $n-1$ words. For some languages (such as English) trillions of words are available for training purposes. This fact, along with the observation that machine translation quality improves as the amount of monolingual training material increases, has lead to the introduction of randomised techniques for representing large LMs in small space (Talbot and Osborne, 2007; Talbot and Brants, 2008).

Randomised LMs (RLMs) solve the problem of representing large, static LMs but they are *batch* oriented and cannot incorporate new data without fully retraining from scratch. This property makes current RLMs ill-suited for modelling the massive volume of textual material published daily on the Web. We present a novel RLM which is capable of incremental (re)training. We use random hash functions coupled with an online perfect hashing algorithm to represent n-grams in small space. This makes it well-suited for dealing with an unbounded stream of training material. To our knowledge this is the first stream-based RLM reported in the machine translation literature. As well as introducing the basic stream-based RLM, we also consider adaptation strategies. Perplexity and translation results show that populating the language model with material chronologically close to test points yields good results. As with previous randomised language models, our experiments focus on machine translation but we also expect that our findings are general and should help inform the design of other stream-based models.

Section 2 introduces the incrementally retrainable randomised LM and section 3 considers related work; Section 4 then considers the question of how unbounded text streams should be modelled. Sections 5 and 6 show stream-based translation results and properties of our novel data-structure. Section 7 concludes the paper.

2 Online Bloomier Filter LM

Our online randomised LM (O-RLM) is based on the dynamic Bloomier filter (Mortensen et al., 2005). It is a variant of the batch-based Bloomier filter LM of Talbot and Brants (2008) which we refer to as the TB-LM henceforth. As with the TB-LM, the O-RLM uses random hash functions to represent n-grams as *fingerprints* which is the main source of space savings for the model.

2.1 Online Perfect Hashing

The key difference in our model as compared to the TB-LM is we use an *online perfect hashing*

Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 756–764,
Singapore, 6-7 August 2009.

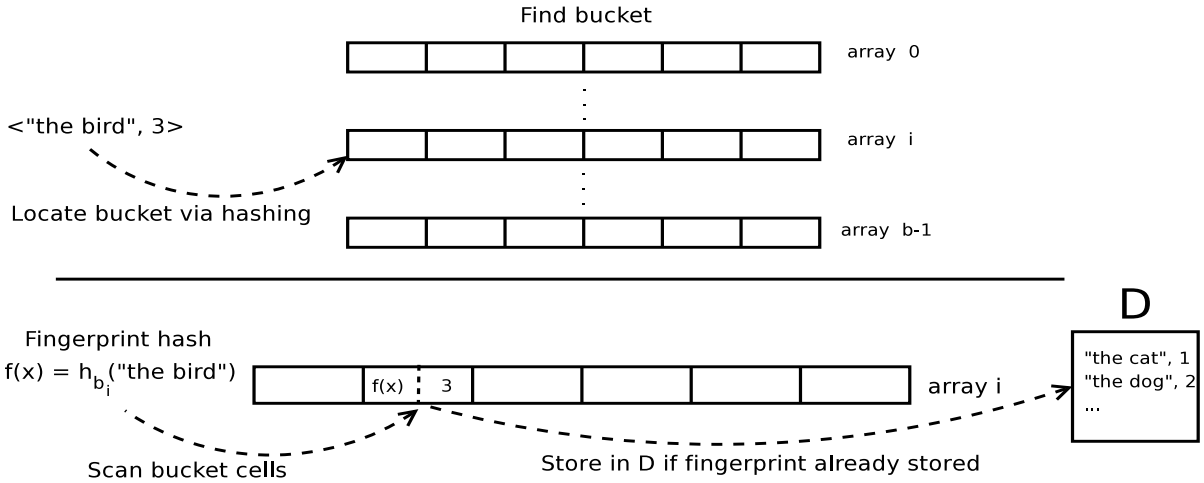

Figure 1: Inserting an n-gram into the dynamic Bloomier filter. Above: an n-gram is hashed to its target bucket. Below: the n-gram is transformed into a fingerprint and the same target bucket is scanned. If a collision occurs that n-gram is diverted to the overflow dictionary; otherwise the fingerprint is stored in the bucket.

function instead of having to precompute the perfect hash offline prior to data insertion.

The online perfect hash function uses two data structures: A and D. A is the main, randomised data structure and is an array of b dictionaries $A_0, \ldots, A_{b-1}$. D is a lossless data structure which handles collisions in A. Each of the dictionaries in A is referred to as a 'bucket'. In our implementation the buckets are equally sized arrays of w-bit cells. These cells hold the fingerprints and values of n-grams (one n-gram-value pair per cell).

To **insert** an n-gram x and associated value $v(x)$ into the model, we select a bucket A_i by hashing x into the range $i \rightarrow [0, \ldots, b-1]$. Each bucket has an associated random hash function, h_{A_i}, drawn from a universal hash function (UHF) family h (Carter and Wegman, 1977), which is then used to generate the n-gram fingerprint: $f(x) = h_{A_i}(x)$.

If the bucket A_i is not full we conduct a scan of its cells. If the fingerprint $f(x)$ is not already encoded in the bucket A_i we add the fingerprint and value to the first empty cell available. We allocate a preset number of the least significant bits of each w-bit cell to hold $v(x)$ and the remaining most significant bits for $f(x)$ but this is arbitrary. Any encoding scheme, such as the packed representation of Talbot and Brants (2008), is viable here.

However, if $f(x) \in A_i$ already (there is a collision) we store the n-gram x and associated value $v(x)$ in the lossless overflow dictionary D instead. D also holds the n-grams that were hashed to any buckets that are already full.

To **query** for the value of an n-gram, we first check if the gram is in the overflow dictionary D. If it is, we return the associated value. Otherwise we query A using the same hash functions and procedure as insertion. If we find a matching fingerprint in the appropriate bucket A_i we have a hit with high probability. **Deletions** and **updates** are symmetric to querying except we reset the cell to the null value or update its value respectively. As with other randomised models we construct queries with the appropriate *sanity checks* to lower the error rate efficiently (Talbot and Brants, 2008).

2.2 Data Insertion

Initially we seed the language model with a large corpus S in the usual manner associated with batch LMs. Then, when processing the stream, we aggregate n-gram counts for some consecutive portion, or *epoch*, of the input stream. We can vary the size of stream window. For example we might batch-up a day or week's worth of material. Intuitively, smaller windows produce results that are sensitive to small variation in the stream, while longer windows (corresponding to data over a longer time period) average out local spikes. The exact window size is a matter of experimentation. In our MT experiments (section 5) we can compute counts within the streaming window exactly but randomised approaches (such as the approximate counting schemes from section 3) can easily be employed instead.

These n-grams and counts are then considered for insertion into the online model. If we decide to insert an n-gram, we either update the count of that n-gram if we previously inserted it or else we insert it as a new entry. Note that there is some probability we may encounter a false positive and update some other n-gram in the model.

2.3 Properties

The online perfect hash succeeds by associating each n-gram with only **one** cell in A rather than having it depend on cells (or bits) which may be shared by other n-grams as with the TB-LM. Since each n-gram's encoding in the model uses distinct bits and is independent of all other events it can not corrupt other n-grams when deleted.

Adding the overflow dictionary D means that we use more space than the TB-LM for the same support. It is shown in Mortensen et al. (2005) that the expected size of D is a small fraction of the total number of events and its space usage comprises less than $O(|S|)$ bits with high probability.

There is a nonzero probability for false positives. Since the overflow dictionary D has no errors, the expected error rate for our dynamic structure is the probability of a random collision in the hash range of each h_{A_i} for each bucket cell compared. In our setup we have

$$\Pr(falsepos) = \frac{|A_i|}{2^{|f(x)|}}$$

where $|f(x)|$ is the number of bits of each w-bit cell used for the fingerprint $f(x)$. w also primarily governs space used in the model. The O-RLM assumes only valid updates and deletions are performed (i.e. we do not remove or update entries that were never inserted prior).

The O-RLM takes time linear to the input size for training and uses worst-case constant time for querying and deletions where the constant is dependent on the number of cells per bucket in A. The number of bucket cells also effects the overall error rate significantly since smaller ranges reduce the probability of a collision. However, too few cells per bucket will result in many full buckets when the bucket hash function is not highly IID.

2.4 Basic RLM Comparisons

Table 1 compares expected versus observed false positive rates for the Bloom filter, TB-LM, and O-RLM obtained by querying a model of approximately 280M events with 100K unseen n-grams.

LM	Expected	Observed	RAM
Lossless	0	0	7450MB
Bloom	0.0039	0.0038	390MB
TB-LM	0.0039	0.0033	640MB
O-RLM	0.0039	0.0031	705MB

Table 1: Example false postive rates and corresponding memory usage for all randomised LMs.

We see the bit-based Bloom filter uses significantly less memory than the cell-based alternatives and the O-RLM consumes more memory than the TB-LM for the same expected error rate.

3 Related Work

3.1 Randomised Language Models

Talbot and Osborne (2007) used a *Bloom filter* (Bloom, 1970) to encode a smoothed LM. A Bloom filter (BF) represents a set S from arbitrary domain U and supports membership queries such as"Is $x \in S$?". The BF uses an array of m bits and k independent UHFs each with range $0, \ldots, m-1$. For insertion, each item is hashed through the k hash functions and the resulting target bits are set to one. During testing, an event $x \in U$ is passed through the same k hash functions and if any bit tested is zero then x was not in the support S.

The *Bloomier filter* directly represents key-value pairs by using a table of cells and a family of k associated hash functions (Chazelle et al., 2004). Each key-value pair is associated with k cells in the table via a perfect hash function. Talbot and Brants (2008) used a Bloomier filter to encode a LM. Before data can be added to the Bloomier filter, a greedy perfect hashing of all entries needs to be computed in advance; this attempts to associate each event in the support with one unique table cell so no other entry collides with it. The procedure can fail and might need to be repeated many times.

Neither of these two randomised language models are suitable for modelling a stream. Given the fact that the stream is of unbounded size, we are forced to delete items if we wish to maintain a constant error rate and account for novel n-grams. However, the Bloom filter LM nor the Bloomier Filter LM support deletions. The bit sharing of the Bloom filter (BF) LM (Talbot and Osborne, 2007) means deletions may corrupt shared stored events. The Bloomier filter LM (Talbot and Brants, 2008) has a precomputed matching of keys shared between a constant number of cells in the filter array.

Deleting items from a Bloomier Filter without recomputing the perfect hash will corrupt it.

3.2 Probabilistic Counting

Concurrent work has used approximate counting schemes based on Morris (1978) to estimate in small space frequencies over a high volume input text stream (Van Durme and Lall, 2009; Goyal et al., 2009). The space savings are due to compact storage of counts and retention of only a small subset of the available n-grams in the data stream. Since the final LMs are still lossless (modulo counts), the resulting LM needs significant space. It is trivial to use probabilistic counting within our framework.

3.3 Compact Exact Language Models

Randomised algorithms are not the only compact representation schemes. Church et al. (2007) looked at Golomb Coding and Brants et al. (2007) used tries in a distributed setting. These methods are less succinct than randomised approaches.

3.4 Adaptive Language Models

There is a large literature on adaptive LMs from the speech processing domain (Bellegarda, 2004). The primary difference between the O-RLM and other adaptive LMs is that we add and remove n-grams from the model instead of adapting only the parameters of the current support set.

3.5 Domain adaptation in Machine Translation

Within MT there has been a variety of approaches dealing with domain adaption (for example (Wu et al., 2008; Koehn and Schroeder, 2007). Typically LMs are interpolated with one another, yielding good results. These models are usually statically trained, exact and unable to deal with an unbounded stream of monolingual data. Domain adaptation has similarities with streaming, in that our stream may be non-stationary. A crucial difference however is that the stream is of unbounded length, whereas domain adaptation usually assumes some finite and fixed training set.

4 Stream-based translation

Streaming algorithms have numerous applications in mainstream computer science (Muthukrishnan, 2003) but to date there has been very little awareness of this field within computational linguistics.

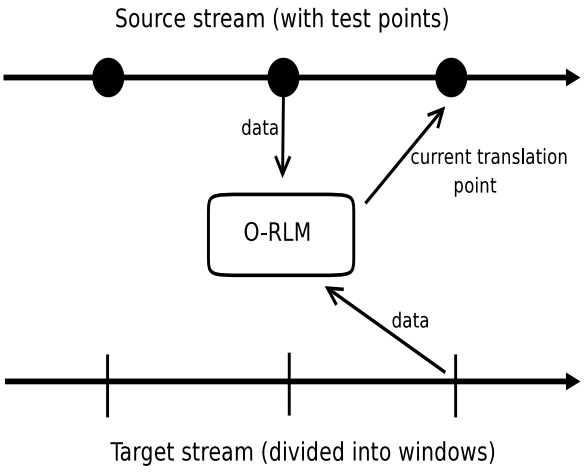

Figure 2: Stream-based translation. The online RLM uses data from the target stream and the last test point in the source stream for adaptation.

A *text stream* can be thought of as a unbounded sequence of documents that are time-stamped and we have access to them in strict chronological order. The volume of the stream is so large we can afford only a limited number of passes over the data (typically one).

Text streams naturally arise on the Web when millions of new documents are published each day in many languages. For instance, 18 thousand websites continuously publish news stories in 40 languages and there are millions of multilingual blog postings per day. There are over 30 billion e-mails sent daily and social networking sites, including services such as Twitter, generate an adundance of textual data in real time. Web crawlers that spidered all these new documents would produce an unbounded input stream.

The stream-based translation scenario is as follows: we assume that each day we see a source stream of many new newswire stories that need translation. We also assume a stream of newswire stories in the target language. Intuitively, since the concurrent streams are from the same domain, we can use the contexts provided in the target stream to help with the translation of the source stream (Figure 2). From a theoretical perspective, since we cannot represent the entirety of the stream and wish to maintain a constant error rate, we are forced to throw some information away.

Given that the incoming text stream contains far too much data to store in its entirety an immediate question we would like to answer is: within our LM, which subset of the target text stream should

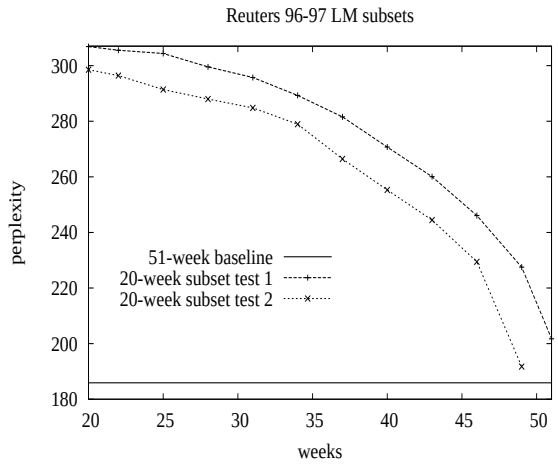

Figure 3: Perplexity results using streamed data. Perplexity decreases as we retrain LMs using data chronologically closer to the (two) test dates.

we represent in our model?

Using perplexity, we investigated this question using a text stream based on Reuter's RCV1 text collection (Rose et al., 2002). This contains 800k time-stamped newswire stories from a full calender year (8.20.1996 - 8.19.1997). We used the SRILM (Stolcke, 2002) to construct an exact trigram model built using all the RCV1 data with the exception of the final week which we held out as test data. This served as an oracle since we store all of the stream.

We then trained multiple exact LMs of much smaller sizes, coined *subset LMs*, to simulate memory constraints. For a given date in the RCV1 stream, these subset LMs were trained using a fixed window of previously seen documents up to that data. Then we obtained perplexity results for each subset LM against our test set.

Figure 3 shows an example. For this experiment subset LMs were trained using a sliding window of 20 weeks with the window advancing over a period of three weeks each time. The two arcs correspond to two different test sets drawn from different days. The arcs show that **recency** has a clear effect: populating LMs using material closer to the test data date produces improved perplexity performance. The LM chronologically closest to a given test set has perplexity closest to the results of the significantly larger baseline LM which uses all the stream. As expected, using all of the data yields the lowest perplexity.

We note that this is a robust finding, since we also observe it in other domains. For example, we conducted the same tests over a stream of 18 billion tokens drawn from 80 million time-stamped blog posts downloaded from the web with matching results. The effect of recency on perplexity has also been observed elsewhere (see, for example, Rosenfeld (1995) and Whittaker (2001)).

Epoch	Stream Window
1	08.20.1996 to 01.01.1997
2	01.02.1997 to 04.23.1997
3	04.24.1997 to 08.18.1997

Table 2: The stream timeline is divided into windowed epochs for our recency experiments.

Our experiments show that a possible way to tackle stream-based translation is to always focus the attention of the LM on the most recent part of the stream. This means we remove data from the model that came from the receding parts of the stream and replace it with the present.

5 SMT Experiments

5.1 Experimental Setup

We used publicly available resources for all our tests: for decoding we used Moses (Koehn and Hoang, 2007) and our parallel data was taken from the Spanish-English section of Europarl. For test material, we translated 63 documents (800 sentences) from three randomly selected dates spaced throughout the RCV1 year (January 2nd, April 24, and August 19).[1] This effectively divided the stream into three *epochs* between the test dates (table 2). We held out 300 sentences for minimum error rate training (MERT) (Och, 2003) and optimised the parameters of the feature functions of the decoder for each experimental run.

The RCV1 is not a large corpus when compared to the entire web but it is multilingual, chronological, and large enough to enable us to test the effect of recency in a translation setting.

5.2 Adaption

We looked at a number of ways of adapting the O-RLM:

1. (**Random**) Randomly sample the stream and for each new n-gram encountered, insert

[1] As RCV1 is not a parallel corpus we translated the reference documents ourselves. This parallel corpus is available from the authors.

Order	Full	Epoch 1	Epoch 3
1	1.25M	0.6M	0.7M
2	14.6 M	6.8M	7.0M
3	50.6 M	21.3M	21.7M
4	90.3 M	34.8M	35.4M
5	114.7M	41.8M	42.6M
Total	271.5M	105M	107.5M

Table 3: Distinct n-grams (in millions) encountered in the full stream and example epochs.

it and remove some previously inserted n-gram, irrespective of whether it was ever requested by the decoder or is a prefix.

2. (**Conservative**) For each new n-gram encountered in the stream, insert it in the filter and remove one previously inserted n-gram which was never requested by the decoder. To preserve consistency we do not remove lower-order grams that are needed to estimate backoff probability for higher-order smoothing. Counts are updated for n-grams already in the model if the new count observed is larger than the current one.

3. (**Severe**) Differs from the conservative approach only in that we delete *all* unused n-grams (i.e. all those not requested by the decoder in the previous translation task) from the O-RLM before adapting with data from the stream. This means the data structure is sparsely populated for all runs.

All the TB-LMs and O-RLMs were unpruned 5-gram models and used *Stupid-backoff* smoothing (Brants et al., 2007) [2] with the backoff parameter set to 0.4 as suggested. The number of distinct n-grams encountered in the stream for two epochs is shown in Table 3.

Table 6 shows translation results using these adaption strategies. In practice, the random approach does not work while the conservative and severe adaption techniques produce equivalent results due to the small proportion of data in the model that is queried during decoding. All the MT experiments that follow use the severe method and the overflow dictionary always holds less than 1% of the total elements in the model.

[2] Smoothing text input data streams poses an interesting problem we hope to investigate in the future.

Date	Lossless	TB-LM	O-RLM
Jan	37.83	37.12	37.17
Apr	34.88	34.21	34.79
Aug	29.05	28.52	28.44
Avg	33.92	33.28	33.46

Table 4: Baseline translation results in BLEU using data from the first stream epoch with a lossless LM (4.5GB RAM), the TB-LM and the O-RLM (300MB RAM). All LMs are static.

5.3 Training Regimes

We now consider stream-based translation. Our first *naive approach* is to continually add new data from the stream to the training set without deleting anything. Given a constant memory bound this strategy only increases the error rate over time as discussed. Our second, computationally demanding approach is, before each test point, to rebuild the TB-LM from scratch using the stream data from the most recent epoch as the training set. This is *batch retraining*. The final approach incrementally retrains online. This utilizes the same training data as above (the stream data from the last epoch) but instead of full retraining it replaces n-grams currently in the model with unseen n-grams and counts encountered in the data stream.

5.4 Streaming Translation Results

Each table shows translation results for the three different test times in the stream. All results reported use the case-sensitive BLEU score.

For our baselines we use *static* LMs trained on the first epoch's data to test all three translation points in the source stream. This is the traditional approach. We trained an exact, modified Kneser-Ney smoothed LM (here we do not enforce a memory constraint) and also used the TB-LM and O-RLM to verify our structures adequecy. Results are shown in table 4. The exact model gives better performance overall due to the more sophisticated smoothing used.

Table 5 shows results for a set of *stream-based* LMs using the TB-LM and the O-RLM with memory bounds of 200MB and 300MB. As expected, the naive models performance degrades over time as we funnel more data into the TB-LM and the error rises. The batch retrained TB-LMs and O-RLMs have constant error rates of $\frac{1}{2^8}$ and $\frac{1}{2^{12}}$ and so outperform the naive approach. Since the training data is identical we see (approximately) equal

	Naive TB-LM		Batch Retrained TB-LM		O-RLM	
Date	200MB	300MB	200MB	300MB	200MB	300MB
Jan	35.94	37.12	35.94	37.12	36.44	37.17
Apr	33.55	35.79	36.01	35.99	35.87	36.10
Aug	22.44	26.07	28.97	29.38	29.00	29.18
Avg	30.64	32.99	33.64	34.16	33.77	34.15

Table 5: Translation results for stream-based LMs in BLEU. Performance degrades with time using the Naive approach. The batch retrained TB-LM and stream-based O-RLM use constant error rates of $\frac{1}{2^8}$ and $\frac{1}{2^{12}}$.

performance from the batch retrained and online models. We also see some improvement compared to the static baselines when the LMs use the most recent data from the target language stream with respect to the current translation point.

The key difference is that each time we batch retrain the TB-LM, we must compute a perfect hashing of the new training set. This is computationally demanding since the perfect hashing algorithm uses Monte Carlo randomisation which fails routinely and must be repeated. To make the algorithm tractable the training data set must be divided into lexically sorted subsets as well. This requires extra passes over the data which may not be trivial in a streaming environment.

In contrast, the O-RLM is incrementally retrained online. This makes it more resource efficient since we find bits in the model for the n-grams dynamically without using more memory than we intially set. Note that even though the O-RLM is theoretically less space efficient than the TB-LM, when using the same amount of memory translation performance is comparable.

6 O-RLM Properties

The previous experiments confirm that the O-RLM can be employed as a LM in an SMT setting but it is useful to get insight into the intrinsic properties of the data structure. Many of the properties of the model, such as the number of bits per fingerprint, follow directly from the TB-LM but the relationship between the overflow dictionary and the randomised buckets is novel.

Figures 4 and 5 shows properties of the O-RLM while varying only the number of cells in each bucket and keeping all other model parameters constant. We test membership of n-grams in an unseen corpus against those stored in the table. Our tests were conducted over a larger stream of 1.25B n-grams from the Gigaword corpus(Graff, 2003). We set our space usage to match the 3.08 bytes per n-gram reported in Talbot and Brants (2008) and held out just over 1M unseen n-grams to test the error rates of our models.

Date	Severe	Random	Conservative
Jan	36.44	36.44	36.44
Apr	35.87	31.08	35.51
Aug	29.00	19.31	29.14
Avg	33.77	29.11	33.70

Table 6: Adaptation results measured in BLEU. Random deletions degrade performance when adapting a 200MB O-RLM.

In Figure 4 we see a direct correlation between model error and cells per buckets. As the number of cells decreases the false positive rate drops as well since fewer cells to compare against per bucket means a lower chance of producing collisions. If the range is decreased too much though more data is diverted to the overflow dictionary due to many buckets reaching capacity when inserting and adapting. Clearly this is less space efficient. Figure 5 shows the relationship between the percent of data in the overflow dictionary and the total cells per bucket.

7 Conclusions

Our experiments have shown that for stream-based translation, using recent data can benefit performance but simply adding entries to a randomised representation will only reduce translation performance over time. We have presented a novel randomised language model based on dynamic perfect hashing that supports online insertions and deletions. As a consequence, it is considerably faster and more efficient than batch retraining.

While not advocating the idea that only small amounts of data are needed for language mod-

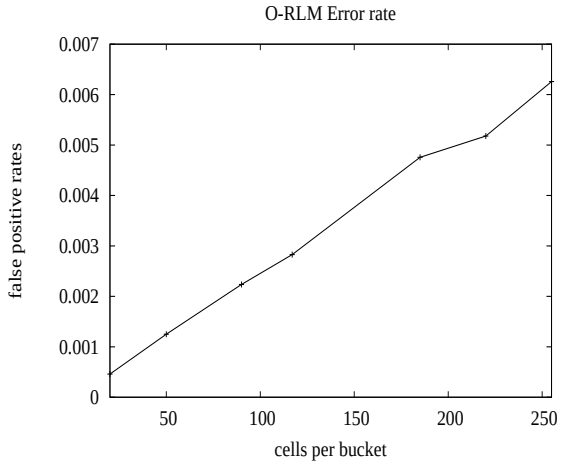

Figure 4: The O-RLM error rises in correlation with the number of cells per bucket.

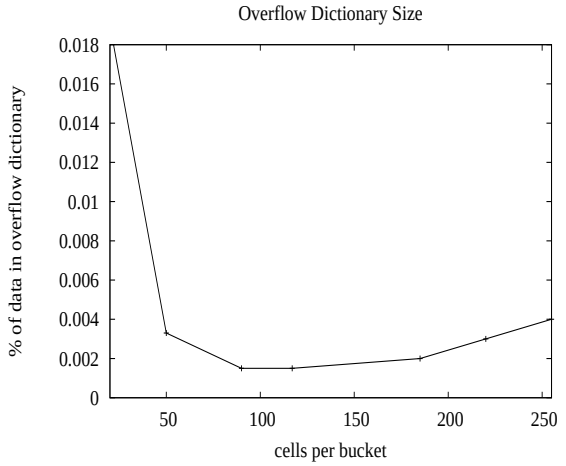

Figure 5: Too few cells per bucket causes a higher percentage of the data to be stored in the overflow dictionary due to full buckets.

elling, within a bounded amount of space our results show that it is better to have a low error rate and store a wisely chosen fraction of the data than having a high error rate and storing more of it. Clearly tradeoffs will vary between applications.

This is the first stream-based randomised language model and associated machine translation system reported in the literature. Clearly there are many interesting open questions for future work. For example, can we use small randomised representations called *sketches* to compactly represent side-information on the stream telling us which aspects of it we should insert into our data? How can we efficiently deal with smoothing in this setting? Our adaptation scheme is simple and our data stream is tractable. Currently we are conducting tests over much larger, higher variance text streams from crawled blog data. In the future we will also consider randomised representations of other adaptive LMs in the literature using a static background LM in conjunction with our online one. We ultimately hope to deploy large-scale LMs which continuously adapt to the vast amount of material published on the Web without incurring significant computational overhead.

Acknowledgements

The authors would like to thank David Talbot, Adam Lopez and Phil Blunsom for their valuable comments and insight. This work was supported in part under the GALE program of the Defense Advanced Research Projects Agency, Contract No. HR0011-06-C-0022.

References

Jerome R. Bellegarda. 2004. Statistical language model adaptation: review and perspectives. *Speech Communication*, 42:93–108.

Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426.

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)*, pages 858–867.

J. Lawrence Carter and Mark N. Wegman. 1977. Universal classes of hash functions (extended abstract). In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, New York, NY, USA. ACM Press.

Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. 2004. The bloomier filter: an efficient data structure for static support lookup tables. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 30–39, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Kenneth Church, Ted Hart, and Jianfeng Gao. 2007. Compressing trigram language models with Golomb coding. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 199–207, Prague, Czech Republic, June. Association for Computational Linguistics.

Amit Goyal, Hal Daumé III, and Suresh Venkatasubramanian. 2009. Streaming for large scale NLP: Language modeling. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, Boulder, CO.

David Graff. 2003. English Gigaword. Linguistic Data Consortium (LDC-2003T05).

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.

Philipp Koehn and Josh Schroeder. 2007. Experiments in domain adaptation for statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 224–227, Prague, Czech Republic, June. Association for Computational Linguistics.

Robert Morris. 1978. Counting large numbers of events in small registers. *Commun. ACM*, 21(10):840–842.

Christian Worm Mortensen, Rasmus Pagh, and Mihai Pătraçcu. 2005. On dynamic range reporting in one dimension. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 104–111, New York, NY, USA. ACM.

S. Muthukrishnan. 2003. Data streams: algorithms and applications. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 413–413, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 160–167, Morristown, NJ, USA. Association for Computational Linguistics.

Tony Rose, Mark Stevenson, and Miles Whitehead. 2002. The reuters corpus volume 1 - from yesterdays news to tomorrows language resources. In *In Proceedings of the Third International Conference on Language Resources and Evaluation*, pages 29–31.

Ronald Rosenfeld. 1995. Optimizing lexical and n-gram coverage via judicious use of linguistic data. In *In Proc. European Conf. on Speech Technology*, pages 1763–1766.

A. Stolcke. 2002. Srilm – an extensible language modeling toolkit. In *Proc. Intl. Conf. on Spoken Language Processing, 2002*.

David Talbot and Thorsten Brants. 2008. Randomized language models via perfect hash functions. In *Proceedings of ACL-08: HLT*, pages 505–513, Columbus, Ohio, June. Association for Computational Linguistics.

David Talbot and Miles Osborne. 2007. 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 (EMNLP-CoNLL)*, pages 468–476.

Benjamin Van Durme and Ashwin Lall. 2009. Probabilistic counting with randomized storage. In *Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, Pasadena, CA, July.

E. W. D. Whittaker. 2001. Temporal adaptation of language models. In *In Adaptation Methods for Speech Recognition, ISCA Tutorial and Research Workshop (ITRW)*, pages 203–206.

Hua Wu, Haifeng Wang, and Chengqing Zong. 2008. Domain adaptation for statistical machine translation with domain dictionary and monolingual corpora. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 993–1000. Coling 2008 Organizing Committee, August.