Pruning Sparse Non-negative Matrix N-gram Language Models

Joris Pelemans[1,2], *Noam Shazeer*[1], *Ciprian Chelba*[1]

[1]Google, Inc., 1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
[2]Dept. ESAT, KU Leuven, Kasteelpark Arenberg 10
B-3001 Leuven, Belgium

{noam,jpeleman,ciprianchelba}@google.com
joris.pelemans@esat.kuleuven.be

Abstract

In this paper we present a pruning algorithm and experimental results for our recently proposed Sparse Non-negative Matrix (SNM) family of language models (LMs). We show that when trained with only n-gram features SNMLM pruning based on a mutual information criterion yields the best known pruned model on the One Billion Word Language Model Benchmark, reducing perplexity with 18% and 57% over Katz and Kneser-Ney LMs, respectively. We also present a method for converting an SNMLM to ARPA back-off format which can be readily used in a single-pass decoder for Automatic Speech Recognition.

Index Terms: sparse non-negative matrix, language modeling, n-grams, pruning

1. Introduction

Recently, neural network (NN) smoothing [1], [2], [3], and in particular recurrent neural networks (RNNs) [4], [5] have shown excellent performance in language modeling [6]. Although these models are currently the state of the art, they are too computationally expensive to be applied directly in an Automatic Speech Recognition (ASR) decoder. Instead, decoding is necessarily done using a multi-pass approach: in a first pass a less advanced, but efficient n-gram model is used to generate the most likely hypotheses which are then rescored using the NN-based model. In practical applications where there are memory and latency constraints, even this is insufficient and single-pass approaches using heavily pruned n-grams are a necessity [7]. Unfortunately, it turns out that pruning severely reduces the predictive power of the state-of-the-art Kneser-Ney (KN) family of n-gram smoothing techniques [8] and we have to resort to suboptimal techniques such as Katz smoothing [9].

We have recently proposed a novel LM paradigm based on Sparse Non-negative Matrix (SNM) estimation [10]. When trained with n-gram features, the SNMLMs perform almost as well as KN ones, whereas the addition of skip-gram features resulted in perplexity (PPL) results on par with RNNLMs. In fact, linear interpolation of RNN and SNM LMs yielded the best known result on the One Billion Word Language Modeling Benchmark [6], with a PPL approximately equal to the interpolation of several other models. Moreover, the computational advantages of SNM over both Maximum Entropy and RNNLM estimation promise an approach that has the same flexibility in combining arbitrary features effectively and yet should scale to very large amounts of data as gracefully as n-gram LMs do.

In this work we show that n-gram SNM models do not degrade with pruning as fast as KN, or Katz. We also show that they can be converted to ARPA [11] back-off LMs, which allows them to be applied in the first pass of an ASR decoder.

In the remainder of this paper we discuss pruning related work (Section 2), describe our new SNMLM paradigm (Section 3), compare pruning performances of Katz, Kneser-Ney and SNMLM (Section 5) and describe how an SNMLM can be converted to a back-off LM (Section 6). We end with conclusions and future work.

2. Related Work

The simplest and still widely used way of pruning n-gram LMs is to specify a count cut-off threshold below which n-grams are not added to the model. Though intuitive, this has the disadvantage that it does not give good control over the size of the model and that it is not self-contained i.e. to prune an existing LM one also needs to have access to the original counts. Moreover, it has been shown to be inferior to most if not all other pruning criteria.

One of the first alternatives was proposed by [12] in the form of a variable context length LM. Instead of enlarging the context length globally for all contexts the author proposed to do this only selectively, based on an approximate entropy criterion. This idea was further developed by [13] into a simple, self-contained thresholding algorithm for n-gram pruning which is still very popular today. The author also pointed out that the earlier work of [14], who proposed a pruning based on the weighted difference between lower and higher-order n-gram probabilities, is in fact a very good practical approximation of his relative entropy criterion. The only problem is that when an application requires pruning the LM aggressively, e.g. to less than 10% of its unpruned size (a case frequently encountered in ASR), entropy pruning turns out to be poorly suited for the family of KN smoothing techniques, as was pointed out by [15] and [7].

Pruning can also be achieved by clustering words into classes which can then be used to build a class-based n-gram model, as first proposed by [16]. Moreover, the idea of clustering can be combined with the above-mentioned techniques which is illustrated in [17] who report size reductions by a factor three at the same perplexity.

Finally, another interesting way of pruning based on statistical significance was recently proposed by [18] and shows that pruning can actually lead to better models. It would be interesting to know though whether this idea extends to aggressive pruning of KN models.

3. Sparse Non-negative Matrix Language Modeling

3.1. Model definition

In the Sparse Non-negative Matrix (SNM) paradigm, we represent the training data as a sequence of events $E = e_1, e_2, ...$ where each event $e \in E$ consists of a sparse non-negative feature vector $\mathbf{f}$ and a sparse non-negative target word vector $\mathbf{t}$. Both vectors are binary-valued, indicating the presence or absence of a feature or target word, respectively. Although SNM does not enforce it, for the purpose of language modeling, an event typically has multiple features, but only a single target word which effectively makes $\mathbf{t}$ a one-hot encoding of size $|\mathcal{V}|$ with $\mathcal{V}$ the vocabulary. The training data hence consists of $|E||Pos(\mathbf{f})||\mathcal{V}|$ training examples, where $Pos(\mathbf{f})$ denotes the set of positive elements in the vector $\mathbf{f}$. Of these, $|E||Pos(\mathbf{f})|$ are positive (presence of target word) and $|E||Pos(\mathbf{f})|(|\mathcal{V}|-1)$ are negative (absence of target word),

A language model is represented by a non-negative matrix $\mathbf{M}$ that, when applied to a given feature vector $\mathbf{f}$, produces a dense prediction vector $\mathbf{y}$:

$$\mathbf{y} = \mathbf{M}\mathbf{f} \approx \mathbf{t} \tag{1}$$

Upon evaluation, we normalize $\mathbf{y}$ such that we end up with a conditional probability distribution $P_{\mathbf{M}}(\mathbf{t}|\mathbf{f})$ for a model $\mathbf{M}$. For each word $w \in \mathcal{V}$ that corresponds to index j in $\mathbf{t}$, and its history that corresponds to feature vector $\mathbf{f}$, the conditional probability $P_{\mathbf{M}}(t_j|\mathbf{f})$ then becomes:

$$\begin{aligned} P_{\mathbf{M}}(t_j|\mathbf{f}) &= \frac{y_j}{\sum_{u=1}^{|\mathcal{V}|} y_u} \\ &= \frac{\sum_{i \in Pos(\mathbf{f})} M_{ij}}{\sum_{i \in Pos(\mathbf{f})} \sum_{u=1}^{|\mathcal{V}|} M_{iu}} \end{aligned} \tag{2}$$

For convenience, we will write $P(t_j|\mathbf{f})$ instead of $P_{\mathbf{M}}(t_j|\mathbf{f})$ in the rest of the paper.

As required by the denominator in Eq. (2), this computation involves summing over all of the present features for the entire vocabulary. However, if we precompute the row sums $\sum_{u=1}^{|\mathcal{V}|} M_{iu}$ and store them together with the model, the evaluation can be done very efficiently in only $|Pos(\mathbf{f})|$ time. Note also that the row sum precomputation involves only few terms due to the sparsity of $\mathbf{M}$.

3.2. Adjustment function and metafeatures

We let the entries of $\mathbf{M}$ be a slightly modified version of the relative frequencies:

$$M_{ij} = e^{A(i,j)} \frac{C_{ij}}{C_{i*}} \tag{3}$$

where $A(i,j)$ is a real-valued function, dubbed *adjustment function*, and $\mathbf{C}$ is a feature-target count matrix, computed over the entire training corpus. C_{ij} denotes the co-occurrence frequency of feature f_i and target t_j, whereas C_{i*} denotes the total occurrence frequency of feature f_i, summed over all targets.

For each feature-target pair (f_i, t_j), the adjustment function computes a sum of weights $\theta_k(i,j)$ corresponding to k new features, called *metafeatures*:

$$A(i,j) = \sum_k \theta_k(i,j) \tag{4}$$

From the given input features, such as regular n-grams and skip-grams, we construct the metafeatures as conjunctions of any or all of the following elementary metafeatures:

- feature identity, e.g. `[the quick brown]`
- feature type, e.g. 4-gram
- feature count C_{i*}
- target identity, e.g. `fox`
- feature-target count C_{ij}

Note that the seemingly absent feature-target identity is represented by the conjunction of the feature identity and the target identity. Since the metafeatures may involve the feature count and feature-target count, in the rest of the paper we will write $A(i, j, C_{i*}, C_{ij})$ when necessary. This will become important in Section 3.5 where we discuss leave-one-out training.

Each elementary metafeature is joined with the others to form more complex metafeatures which in turn are joined with all the other elementary and complex metafeatures, ultimately ending up with all 2^5-1 possible combinations of metafeatures.

As count metafeatures of the same order of magnitude carry similar information, we group them so they can share the same weight. We do this by bucketing the count metafeatures according to their (floored) $\log_2$ value.

3.3. Model estimation

Estimating a model $\mathbf{M}$ corresponds to finding optimal weights θ_k for all the metafeatures for all events in such a way that the average loss over all events between the target vector $\mathbf{t}$ and the prediction vector $\mathbf{y}$ is minimized, according to some loss function L.

In our previous work [10] we suggested a loss function based on the Poisson distribution: we consider each t_j in $\mathbf{t}$ to be Poisson distributed with parameter y_j. The conditional probability of $P_{Poisson}(\mathbf{t}|\mathbf{f})$ then is:

$$P_{Poisson}(\mathbf{t}|\mathbf{f}) = \prod_{j \in \mathbf{t}} \frac{y_j^{t_j} e^{-y_j}}{t_j!} \tag{5}$$

and the corresponding Poisson loss function is:

$$\begin{aligned} L_{Poisson}(\mathbf{y}, \mathbf{t}) &= -log(P_{Poisson}(\mathbf{t}|\mathbf{f})) \\ &= -\sum_{j \in \mathbf{t}} [t_j \log(y_j) - y_j - log(t_j!)] \\ &= \sum_{j \in \mathbf{t}} y_j - \sum_{j \in \mathbf{t}} t_j \log(y_j) \end{aligned} \tag{6}$$

where we dropped the last term, since t_j is binary-valued. Although this choice is not obvious in the context of language modeling, it is well suited to gradient-based optimization and, as we will see, the experimental results are in fact excellent. Moreover, the Poisson loss also lends itself nicely for multiple target prediction which might be useful in e.g. subword modeling.

The adjustment function is learned by applying stochastic gradient descent on the loss function. That is, for each feature-target pair (f_i, t_j) in each event we need to update the weights of the metafeatures by calculating the gradient with respect to the adjustment function which works out to:

$$\frac{\partial(L_{Poisson}(\mathbf{Mf}, \mathbf{t}))}{\partial(A(i,j))} = f_i M_{ij}(1 - \frac{t_j}{y_j}) \tag{7}$$

For the complete derivation, we refer the reader to [10].

We then use the Adagrad [19] adaptive learning rate procedure to update the metafeature weights. Rather than using a single fixed learning rate, Adagrad uses a separate adaptive learning rate $\eta_{k,N}(i,j)$ for each weight $\theta_k(i,j)$ at the Nth occurrence of (f_i, t_j):

$$\eta_{k,N}(i,j) = \frac{\gamma}{\sqrt{\Delta_0 + \sum_{n=1}^{N} \partial_n(ij)^2}} \tag{8}$$

where γ is a constant scaling factor for all learning rates, Δ_0 is an initial accumulator constant and $\partial_n(ij)$ is a short-hand notation for the Nth gradient of the loss with respect to $A(i,j)$.

3.4. Optimization

If we were to apply the gradient in Eq. (7) to each (positive and negative) training example, it would be computationally too expensive, because even though the second term is zero for all the negative training examples, the first term needs to be computed for all $|E||Pos(\mathbf{f})||\mathcal{V}|$ training examples.

However, since the first term does not depend on y_j, we are able to distribute the updates for the negative examples over the positive ones by adding in gradients for a fraction of the events where $f_i = 1$, but $t_j = 0$. In particular, instead of adding the term $f_i M_{ij}$, we add $f_i t_j \frac{C_{i*}}{C_{ij}} M_{ij}$ which lets us update the gradient only on positive examples. This is based on the observation that, over the entire training set, it amounts to the same thing:

$$\begin{aligned}\sum_{e=(f_i,t_j)\in E} f_i M_{ij} &= C_{i*} f_i M_{ij} \\ &= C_{ij} f_i M_{ij} + (C_{i*} - C_{ij}) f_i M_{ij} \\ &= C_{ij} f_i M_{ij} (1 + \frac{C_{i*} - C_{ij}}{C_{ij}}) \\ &= \sum_{e=(f_i,t_j)\in E} f_i t_j M_{ij} (1 + \frac{C_{i*} - C_{ij}}{C_{ij}})\end{aligned} \tag{9}$$

We note that this update is only strictly correct for batch training, and not for online training since M_{ij} changes after each update. Nonetheless, we found this to yield good results as well as seriously reducing the computational cost. The online gradient applied to each training example then becomes:

$$\frac{\partial(L_{Poisson}(\mathbf{Mf}, \mathbf{t}))}{\partial(A(i,j))} = f_i t_j \frac{C_{i*} - C_{ij}}{C_{ij}} M_{ij} + f_i t_j (1 - \frac{1}{y_j}) M_{ij} \tag{10}$$

which is non-zero only for positive training examples, hence speeding up computation by a factor of $|\mathcal{V}|$.

3.5. Leave-one-out training

A model with a huge amount of parameters is prone to overfitting the training data. The preferred way to deal with this issue is to use held-out data to estimate the parameters. Unfortunately the aggregated gradients in Eq. (10) do not allow us to use additional data to train the adjustment function, since they tie the update computation to the relative frequencies $\frac{C_{i*}}{C_{ij}}$ in the training data. Instead, we have to resort to leave-one-out training to prevent the model from overfitting. We do this by excluding the event that generates the gradients from the counts used to compute those gradients. So, for each positive example (f_i, t_j) of each event $e = (\mathbf{f}, \mathbf{t})$, we compute the gradient, excluding 1 from C_{i*} and C_{ij}. For the gradients of the negative examples on the other hand we only exclude 1 from C_{i*}, because we did not observe t_j. In order to keep the aggregate computation of the gradients for the negative examples, we distribute them uniformly over all the positive examples with the same feature; each of the C_{ij} positive examples will then compute the gradient of $\frac{C_{i*} - C_{ij}}{C_{ij}}$ negative examples.

To summarize, when we do leave-one-out training we apply the following gradient update rule on all positive training examples:

$$\begin{aligned}&\frac{\partial(L_{Poisson}(\mathbf{Mf}, \mathbf{t}))}{\partial(A(i,j))} \\ &= f_i t_j \frac{C_{i*} - C_{ij}}{C_{ij}} e^{A(i,j,C_{i*}-1,C_{ij})} \frac{C_{ij}}{C_{i*} - 1} \\ &+ f_i t_j (1 - \frac{1}{y'_j}) e^{A(i,j,C_{i*}-1,C_{ij}-1)} \frac{C_{ij} - 1}{C_{i*} - 1}\end{aligned} \tag{11}$$

where y'_j is the product of leaving one out for all the relevant features:

$$\begin{aligned}y'_j &= (\mathbf{M'f})_j \\ \mathbf{M}'_{ij} &= e^{A(i,j,C_{i*}-1,C_{ij}-1)} \frac{C_{ij} - 1}{C_{i*} - 1}\end{aligned}$$

4. Pruning

In our first implementation, we opted for a pruning statistic motivated by the inner term of the mutual information calculation:

$$\begin{aligned}MI(f_i, t_j) &= P(f_i, t_j) log(\frac{P(f_i, t_j)}{P(f_i)P(t_j)}) \\ &= \frac{C_{ij}}{C_{**}} log(\frac{C_{ij} C_{**}}{C_{i*} C_{*j}})\end{aligned} \tag{12}$$

A candidate n-gram defined by the feature-target pair (f_i, t_j) is kept in the final model if and only if its $MI(f_i, t_j)$ value is above a chosen pruning threshold. To choose the threshold that produces a model of desired size, we compute MI-based quantiles.

5. Experiments

Our experimental setup used the One Billion Word Benchmark corpus[1] made available by [6].

For completeness, here is a short description of the corpus, containing only monolingual English data:

- Total number of training tokens is about 0.8 billion
- The vocabulary provided consists of 793471 words including sentence boundary markers <S>, </S>, and was constructed by discarding all words with count below 3
- Words outside of the vocabulary were mapped to an <UNK> token, also part of the vocabulary
- Sentence order was randomized
- The test data consisted of 159658 words (without counting the sentence beginning marker <S> which is never predicted by the language model)
- The out-of-vocabulary (OoV) rate on the test set was 0.28%.

As a baseline, we trained 5-gram models with two of the most prevalent smoothing techniques i.e. Katz [9] and interpolated KN [8] and pruned the models by using Stolcke's entropy criterion [13]. The SNM models were pruned according

[1]http://www.statmt.org/lm-benchmark

Model	Params	PPL
KN 5-gram, unpruned	1.76B	67.6
Katz 5-gram, unpruned	1.74B	79.9
SNM 5-gram, unpruned	1.74B	70.3
KN 5-gram, pruned	30M	243
Katz 5-gram, pruned	30M	128
SNM 5-gram, pruned	30M	105

Table 1: Perplexity (PPL) results for pruned and unpruned Kneser-Ney (KN), Katz and SNM 5-grams.

to Eq. (12) where the pruning thresholds were chosen such that the sizes of the pruned models being compared had roughly the same number of parameters. For SNM n-gram LMs that means counting both the M_{ij} and the M_{i*} parameters in the numerator and denominator of Eq. (2), respectively, just as for a back-off n-gram LM we count the probabilities and back-off weights.

Table 1 shows the perplexity results of the pruned and unpruned models on the One Billion Word Benchmark test data along with model sizes.

As we indicated in our earlier work, unpruned SNM n-gram LMs are quite promising as they perform almost as well as the well-established KN models, while still having plenty of room for improvement. Both unpruned SNM and KN perform significantly better than Katz. Pruning the models to a size of 30 million parameters (1.7% of the unpruned size) drastically changes the picture: KN suffers so much from entropy pruning that its perplexity more than triples, making it the worst pruned model by far. Katz degrades less and does a lot better than KN, but the mutual information-pruned version of our SNM n-gram LM is clearly the best with a relative perplexity reduction of 18% over Katz and as much as 57% over KN.

6. Conversion to ARPA Back-off Format

It would be attractive to represent the SNM n-gram model in the ARPA back-off format, as we can then use it in ASR decoders based on Finite State Transducers and apply existing implementations of the pruning techniques mentioned in Section 2. Although we have not yet implemented such a conversion, we show here that it is indeed possible to do this by deriving the formulas for both the probabilities and back-off weights:

For the case of an SNMLM using only n-gram features we can write the probability assignment as:

$$\begin{aligned} P(w|h) &= \frac{M(h,w) + M(h',w) + \ldots + M(\cdot,w)}{M(h,\cdot) + M(h',\cdot) + \ldots + M(\cdot,\cdot)} \\ M(h,\cdot) &= \sum_{w \in V(h)} M(h,w) \end{aligned}$$

where h' denotes the back-off context obtained by dropping the leftmost word from h, and $V(h)$ denotes the set of predicted words observed in the context h in the training data. If we denote:

$$\begin{aligned} S(h,w) &= M(h,w) + M(h',w) + \ldots + M(\cdot,w) \\ S(h,\cdot) &= M(h,\cdot) + M(h',\cdot) + \ldots + M(\cdot,\cdot) \end{aligned}$$

we have:

$$\begin{aligned} P(w|h) &= \frac{S(h,w)}{S(h,\cdot)} \\ P(w|h') &= \frac{S(h',w)}{S(h',\cdot)} \end{aligned}$$

This means that the back-off weight computation for n-gram context h works out as follows:

$$\begin{aligned} BoW(h) &= \frac{1 - \sum_{w \in V(h)} P(w|h)}{1 - \sum_{w \in V(h)} P(w|h')} \\ &= \frac{1 - \sum_{w \in V(h)} \frac{S(h,w)}{S(h,\cdot)}}{1 - \sum_{w \in V(h)} \frac{S(h',w)}{S(h',\cdot)}} \end{aligned} \qquad (13)$$

The numerator of Eq. (13) works out to:

$$\begin{aligned} & 1 - \sum_{w \in V(h)} \frac{S(h,w)}{S(h,\cdot)} \\ &= 1 - \sum_{w \in V(h)} \frac{S(h',w) + M(h,w)}{S(h',\cdot) + M(h,\cdot)} \\ &= 1 - \sum_{w \in V(h)} \frac{S(h',w)}{S(h',\cdot)} \cdot \frac{S(h',\cdot)}{S(h',\cdot) + M(h,\cdot)} \\ &\quad - \sum_{w \in V(h)} \frac{M(h,w)}{S(h',\cdot) + M(h,\cdot)} \\ &= 1 - \frac{S(h',\cdot)}{S(h,\cdot)} \cdot \sum_{w \in V(h)} \frac{S(h',w)}{S(h',\cdot)} - \frac{M(h,\cdot)}{S(h',\cdot) + M(h,\cdot)} \\ &= 1 - \frac{M(h,\cdot)}{S(h,\cdot)} - \frac{S(h',\cdot)}{S(h,\cdot)} \cdot \sum_{w \in V(h)} \frac{S(h',w)}{S(h',\cdot)} \\ &= \frac{S(h',\cdot)}{S(h,\cdot)} - \frac{S(h',\cdot)}{S(h,\cdot)} \cdot \sum_{w \in V(h)} \frac{S(h',w)}{S(h',\cdot)} \\ &= \frac{S(h',\cdot)}{S(h,\cdot)} \cdot \left[1 - \sum_{w \in V(h)} \frac{S(h',w)}{S(h',\cdot)} \right] \end{aligned}$$

Substituting back in Eq. (13) we arrive at:

$$BoW(h) = \frac{S(h',\cdot)}{S(h,\cdot)} \qquad (14)$$

7. Conclusions and Future Work

We have presented an algorithm for pruning the SNMLM that applies generally to such models whether they use only n-gram features, or more complex features such as skip-grams.

For the case of n-gram features, the algorithm significantly outperforms entropy pruning for the well-established Katz and interpolated Kneser-Ney models; relative perplexity reductions of 18% and 57%, respectively, were reported.

We have also shown that the n-gram SNMLM can be converted to the standard ARPA back-off format, making it easily usable in ASR decoders based on Finite State Transducers, or other implementations.

Future work includes model pruning based on various other criteria, e.g. using adjusted relative frequencies M_{ij} in Eq. (12), entropy pruning or significance pruning after conversion to ARPA back-off format. In a wider scope we would also like to explore richer features similar to [20], as well as richer metafeatures in the adjustment model, mixing SNM models trained on various data sources such that they perform best on a given development set, and estimation techniques that are more flexible in this respect.

8. References

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, “A neural probabilistic language model,” *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[2] A. Emami, “A neural syntactic language model,” Ph.D. dissertation, Johns Hopkins University, 2006.

[3] H. Schwenk, “Continuous space language models,” *Computer Speech and Language*, vol. 21, 2007.

[4] T. Mikolov, “Statistical language models based on neural networks,” Ph.D. dissertation, Brno University of Technology, 2012.

[5] M. Sundermeyer, R. Schlüter, and H. Ney, “LSTM neural networks for language modeling,” in *Proc. Interspeech*, 2012, pp. 194–197.

[6] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, “One billion word benchmark for measuring progress in statistical language modeling,” in *Proc. Interspeech*, 2014, pp. 2635–2639.

[7] C. Chelba, T. Brants, W. Neveitt, and P. Xu, “Study on interaction between entropy pruning and Kneser-Ney smoothing,” in *Proc. Interspeech*, 2010, pp. 2242–2245.

[8] R. Kneser and H. Ney, “Improved backing-off for m-gram language modeling,” in *Proc. ICASSP*, vol. I, 1995, pp. 181–184.

[9] S. M. Katz, “Estimation of probabilities from sparse data for the language model component of a speech recognizer,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, no. 3, pp. 400–401, 1987.

[10] N. Shazeer, J. Pelemans, and C. Chelba, “Skip-gram language modeling using sparse non-negative matrix probability estimation,” *CoRR*, vol. abs/1412.1454, 2014. [Online]. Available: http://arxiv.org/abs/1412.1454

[11] *ARPA back-off format, SRILM - The SRI Language Modeling Toolkit*, www ed., SRI International, 2011. [Online]. Available: http://www.speech.sri.com/projects/srilm/manpages/

[12] R. Kneser, “Statistical language modeling using a variable context length,” in *Proc. ICSLP*, vol. 1, 1996, pp. 494–497.

[13] A. Stolcke, “Entropy-based pruning of backoff language models,” in *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, 1998, pp. 270–274.

[14] K. Seymore and R. Rosenfeld, “Scalable backoff language models,” in *Proc. ICSLP*, vol. 1, 1996, pp. 232–235.

[15] V. Siivola, T. Hirsimäki, and S. Virpioja, “On growing and pruning Kneser-Ney smoothed n-gram models,” *IEEE Transactions on Audio, Speech & Language Processing*, vol. 15, no. 5, pp. 1617–1624, 2007.

[16] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai, “Class-based n-gram models of natural language,” *Computational Linguistics*, vol. 18, pp. 467–479, 1992.

[17] J. Goodman and J. Gao, “Language model size reduction by pruning and clustering,” in *Proc. ICSLP*, 2000, pp. 110–113.

[18] R. C. Moore and C. Quirk, “Less is more: Significance-based n-gram selection for smaller, better language models,” in *Proc. EMNLP*. ACL, 2009, pp. 746–755.

[19] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, Jul. 2011.

[20] J. T. Goodman, “A bit of progress in language modeling,” *Computer Speech & Language*, vol. 15, no. 4, pp. 403–434, 2001.