INTERSPEECH 2016
September 8–12, 2016, San Francisco, USA

Entropy Based Pruning for Non-negative Matrix Based Language Models with Contextual Features

Barlas Oğuz, Issac Alphonso, Shuangyu Chang

Microsoft Corporation, U.S.A.

barlaso@microsoft.com, issaca@microsoft.com, shchang@microsoft.com

Abstract

Non-negative matrix based language models have been recently introduced [1] as a computationally efficient alternative to other feature-based models such as maximum-entropy models. We present a new entropy based pruning algorithm for this class of language models, which is fast and scalable. We present perplexity and word error rate results and compare these against regular n-gram pruning. We also train models with location and personalization features and report results at various pruning thresholds. We demonstrate that contextual features are helpful over the vanilla model even after pruning to a similar size.

Index Terms: sparse non-negative matrix based language model, entropy based pruning, contextual features, personalization, adaptation, geolocation

1. Introduction

Classical n-gram models are computationally efficient. This single fact is probably the most likely explanation of their practical success. On the other hand they are also very limited and inflexible in terms of the features that these models make use of. Extending these models to make use of a wider variety of features (such as skip-grams, sub-word features, contextual features etc.) usually involves either coming up with new tailored smoothing solutions, or giving up the computational advantages of n-gram models.

For incorporating contextual features (such as geolocation, time of day or user id) maximum entropy models have been utilized to some degree of success [2]. The problem with these models is that training involves calculating normalized probabilities, which for large vocabularies become computationally prohibitive. As a result, approximations such as classes at the output layer and sampling have been tried [3, 4]. While they speed things up considerably, none of these approaches come close to the parallelizability and efficiency of an n-gram model.

Another approach, for contextual features such as geolocation and user id, is to train separate n-gram models for each context, and interpolate the context specific small LM with a larger generic LM [5]. This approach has been shown to provide perplexity reductions, but suffers from data sparsity and size heterogeneity issues.

Recently, a new class of language models have been proposed under the name of 'Sparse non-negative matrix based language models' (SNMLM) [1]. These models combine the flexibility of incorporating arbitrary features, like maximum entropy models, with the computational efficiency that comes from feature-count based models like n-gram models.

Combining arbitrary features in an SNMLM leads to a significant blow up of model size. In a production environment with memory constraints, these models need to be pruned to a reasonable size. Here we will outline an entropy based pruning approach suitable for SNMLM that has parallels to the often used entropy based pruning [6] for n-grams.

In the next section we will detail the workings of an SNMLM including training and evaluation. Then we will outline our pruning algorithm. In section 4 we compare the performance of SNMLM at various pruning levels to a production-grade model smoothed with modified absolute discounting [7], as well as a mutual information (MI) based pruning heuristic that has been proposed in [8]. Following this, we will add contextual features (geolocation and user id) to an SNMLM and report performance improvements in both perplexity (PPL) and word error rate (WER) metrics. Finally we will conclude with a summary and potential future directions.

2. Sparse non-negative matrix based language models

This section provides a summary of the model as described in [1], as well as establishing needed notation for the sections that follow.

A language model is a function which takes as input a context, and outputs a distribution over a vocabulary V:

$$P_{LM}(w|h)$$

Where h is the word context, including previous word history and any contextual information. We will represent h with a binary vector $\boldsymbol{f}$, which represents a sparse k-of-many encoding of a large feature set F. The set of features is part of the model design, but usually includes n-gram features, contextual features and combinations thereof. The cardinality of the feature set will be very large, but only a few entries in $\boldsymbol{f}$ will be non-zero at any one time. As an example, let F be composed of trigram, bigram and unigram features. Consider the following sentence:

[s] the quick brown fox [/s]

Active features in the probability calculations of the word 'brown' would be:

$\boldsymbol{f}$=Set{
Unigram (empty feature): ' ',
Bigram: 'quick',
Trigram: 'the quick' }

Now, each feature $f \in F$ has a weight associated with each target word $w \in V$. We represent this weight function, which depends on the tuple (w, f), by a matrix $M(w, f)$. We denote each entry in short by M_{ij}, where j is the target word index, and i the feature index. In matrix notation, this gives us:

$$\boldsymbol{y} = \boldsymbol{M}\boldsymbol{f}$$

where $\boldsymbol{y}$ is a dense score vector over the vocabulary V. To make this into a distribution, we normalize to get

$$P_{LM}(w|\boldsymbol{f}) = \frac{y}{|y|} = \frac{\sum_{i\in \boldsymbol{f}} M_{iw}}{\sum_{i\in \boldsymbol{f}} \sum_j M_{ij}} \equiv \frac{\sum_{i\in \boldsymbol{f}} M_{iw}}{\sum_{i\in \boldsymbol{f}} M_{i*}} \quad (1)$$

where M_{i*} is the column sum for feature i. So far, we have discussed little more than notation. Many classes of language models could be described in this matrix framework. The following form for the weights M_{ij} defines the SNMLM:

$$M_{ij} = e^{A(i,j)} \frac{C_{ij}}{C_{i*}} \quad (2)$$

Here $A(i,j)$ is an adjustment function, akin to a smoothing term, to be trained, C_{ij} is the occurrence count of the tuple (i,j) in the training data, and C_{i*} is the count of feature i. Thus, the weight M_{ij} is just the modified maximum likelihood probability of word j given feature i. The final probability estimation is calculated by interpolating the estimates from all active features, as given in eq. 1.

2.1. Training the adjustment function

The adjustment function $A(i,j)$ is trained against a Poisson loss function:

$$\begin{aligned} L_{Poisson}(\boldsymbol{y}, w) &= -\log\left(P_{Poisson}(w|\boldsymbol{f})\right) && (3) \\ &= -\log\left(y_w\right) + \sum_{j\in V} y_j && (4) \end{aligned}$$

This loss was chosen to avoid terms that sum over the entire vocabulary in the gradient calculation.

The gradient update is calculated to be (see [1] for details.):

$$\frac{\partial L_{Poisson}(y,w)}{\partial A(i,j)} = f_i M_{ij} \left(\frac{C_{i*}}{C_{ij}} - \frac{1}{y_j}\right), \quad \text{if } j = w \text{ and } 0 \text{ otherwise} \quad (5)$$

2.2. Meta-features

Having a separate feature $A(i,j)$ for each (i,j) would be prone to overfitting, so some parameter tying is necessary. This is accomplished by having the adjustment term depend on several meta-features $\alpha_k(i,j)$. A simple sum of these then define the $A(i,j)$:

$$A(i,j) = \sum_k \alpha_k(i,j) \quad (6)$$

Picking the metafeatures is a feature engineering problem, however the following have been proposed as candidates:

- Target word identity j
- Feature identity i
- Binned log count C_{ij}
- Binned log count C_{i*}
- Feature type (e.g. 2-gram, 3-gram, skip gram, location etc.) and combinations thereof.

2.3. Leave-one-out training

Since the counts that define the model are also being used to train the metafeatures, leave one out training is used to prevent overfitting to the counts. This is done by excluding the current training example from the counts when calculating the gradient. If we write $\alpha_k(i,j,C_{ij},C_{i*}), M_{ij}(C_{ij},C_{i*}), y_j(C_{ij},C_{i*})$ to show the explicit dependence on the counts, we modify the gradient as:

$$\begin{aligned} \frac{\partial L_{Poisson}(y,w)}{\partial A(i,j)} &= f_i M_{ij}(C_{ij}, C_{i*}-1)\frac{C_{i*}-C_{ij}}{C_{ij}} \\ &+ f_i M_{ij}(C_{ij}-1, C_{i*}-1)\left(1 - \frac{1}{y_j(C_{ij}-1, C_{i*}-1)}\right), \\ &\text{if } j = w \text{ and } 0 \text{ otherwise} \quad (7) \end{aligned}$$

2.4. Implementation details

The metafeature vector is implemented as a flat hash table of sufficient size. Collisions are present, but don't affect the final result significantly. Adagrad[9] is used for optimization, where the adaptive learning rate at the Nth occurrence of (i,j) is:

$$\mu_{k,N}(i,j) = \frac{\gamma}{\sqrt{\Delta_0 + \sum_1^N \delta_n(ij)^2}} \quad (8)$$

Here $\delta_n(ij)$ denotes the value of the gradient at the nth occurrence. We pick $\Delta_0 = 1$ and $\gamma = 10^{-3}$.

2.5. Training of hash table

In contrast to calculating counts, which can easily be parallelized, the training of the metafeature vector is an inherently sequential operation. While there are several possible approaches to get around this bottleneck, such as parallelization based on model averaging or mini-batch training, in practice we observe that the metafeatures do not need to be trained on the full data set. The most important metafeatures can be sufficiently trained with only a small fraction of data. Therefore we sub-sample the data to a subset of several million tokens for training the hash table, which proved sufficient for our purposes.

3. Entropy based pruning

Our approach is similar to entropy based pruning for regular n-gram models [6]. For each feature-target pair, we calculate the change in entropy resulting from removing this parameter from the model. We then rank parameters according to this score, and use a suitable threshold to prune our model. Thus, the effect of each parameter is considered independently, and joint effects are ignored, as in the n-gram case.

In contrast to regular n-gram models, SNMLM models are not completely generative. $p(w,f)$ is not specified by the model because the joint probability of the co-occurrences of various features are not modeled. It only produces probabilities $p(w|f)$, conditioned on a given set of features in a discriminative manner. Therefore, it is not possible to do entropy based pruning as a strictly post-processing step. We need to start from the training data to learn about the prior co-occurrence probabilities of the features.

From eq. 1 we can see that the train set entropy of a given model is given by

$$H_T = \sum_{(w,f)} -\log \frac{\sum_{k\in f} M_{kw}}{\sum_{k\in f} M_{k*}} \quad (9)$$

where the sum is running over all the words in the training set. Removing a feature-target pair (i,j) from the model affects all terms in which M_{ij} appears, as well as all the terms in which M_{i*} appears. Running through the entire data for each feature is obviously infeasible. Some effort is necessary to calculate the entropy change for all feature-target pairs in linear time with respect to the size of the training data.

Let p' be the probability after removing a particular M_{ij} from the model. We can write (all the sums are over $k \in \boldsymbol{f}$):

$$\begin{aligned}\log p'(w,f) &= \log \frac{\sum M_{kj} - M_{ij}}{\sum M_{k*} - M_{ij}} \\ &= \log \frac{\sum M_{kj}}{\sum M_{k*}} - \log\left(1 - \frac{M_{ij}}{\sum M_{k*}}\right) \\ &+ \log\left(1 - \frac{M_{ij}}{\sum M_{kj}}\right) \\ &= \log p(w,f) - \log\left(1 - \frac{M_{ij}}{\sum M_{k*}}\right) \\ &+ \log\left(1 - \frac{M_{ij}}{\sum M_{kj}}\right) \end{aligned} \tag{10}$$

if w is equal to j and i is in $\boldsymbol{f}$. Otherwise if w is not equal to j, but i is in $\boldsymbol{f}$,

$$\begin{aligned}\log p'(w,f) &= \log \frac{\sum M_{kj}}{\sum M_{k*} - M_{ij}} \\ &= \log \frac{\sum M_{kj}}{\sum M_{k*}} - \log\left(1 - \frac{M_{ij}}{\sum M_{k*}}\right) \\ &= \log p(w,f) - \log\left(1 - \frac{M_{ij}}{\sum M_{k*}}\right) \end{aligned} \tag{11}$$

Therefore the entropy difference can be written as

$$\begin{aligned}\sum_{(w,f)} \log p(w,f) - \log p'(w,f) = &\sum_{(w,f):i\in f} \log\left(1 - \frac{M_{ij}}{\sum M_{k*}}\right) \\ &- \sum_{w=j,i\in f} \log\left(1 - \frac{M_{ij}}{\sum M_{kj}}\right)\end{aligned} \tag{12}$$

The second term can be calculated for all M_{ij} with only a single pass through the data, since the sum only runs over the terms where M_{ij} appears. The first term however, potentially runs over a large portion of the data, if a particular feature is very common (e.g. the empty feature appears in all entries). To get around this issue, we make the approximation $\log\left(1 - \frac{M_{ij}}{\sum M_{k*}}\right) \approx -\frac{M_{ij}}{\sum M_{k*}}$. We justify this by observing that $\sum M_{k*}$ always includes the empty feature, which is very large, therefore the fraction in question is almost always very small, making this a good approximation. With this we get

$$\begin{aligned}\sum_{(w,f)} \log p(w,f) - \log p'(w,f) \approx -M_{ij} &\sum_{(w,f):i\in f} \frac{1}{\sum M_{k*}} \\ &- \sum_{w=j,i\in f} \log\left(1 - \frac{M_{ij}}{\sum M_{kj}}\right)\end{aligned} \tag{13}$$

Here the quantity $\sum_{(w,f):i\in f} \frac{1}{\sum M_{k*}}$ can be calculated for all i with only a single pass through the data. We use the absolute value of this quantity to put a threshold on M_{ij}.

Since the pruning is based on the change in train set entropy, this approach is prone to over-fitting the training data and under-performs for this reason. The fundamental problem is that we're taking an expectation over the train set distribution $p_T(w,f)$ instead of the model distribution $p_M(w,f)$, which is not fully defined. However the conditional distribution $p_M(w|f)$ is defined, so we can use $p_M(w|f)p_T(f)$ to replace $p_T(w|f)p_T(f) = p_T(w,f)$ in taking the expectation. This means adjusting each term in eq. 13 with the factor

$$\frac{p_M(w|f)}{p_T(w|f)} = \frac{\sum M_{kj}}{\sum M_{k*}} \frac{|f|}{\sum \frac{C_{kj}}{C_{k*}}} \equiv \theta(w,f).$$

The new estimate for the change in entropy becomes

$$-M_{ij} \sum_{(w,f):i\in f} \frac{\theta(w,f)}{\sum M_{k*}} - \sum_{w=j,i\in f} \theta(w,f) \log\left(1 - \frac{M_{ij}}{\sum M_{kj}}\right) \tag{14}$$

With this adjustment, pruning performs as well as regular entropy based n-gram pruning.

3.1. Mutual information based pruning

We will compare our approach to the heuristic that was proposed in [8]. There, features are pruned based on the mutual information-like threshold function:

$$MI_{ij} = \frac{C_{ij}}{C_{**}} \log \frac{C_{ij}C_{**}}{C_{i*}C_{*j}} \tag{15}$$

4. Experiments

As pointed out in the introduction, the main draw of SNMLMs is their flexibility in incorporating arbitrary features efficiently. We were also motivated by this, since in a production setting, many contextual signals are available to us and could be used to improve the performance of our language models. Therefore we chose to test on real data from practical applications that have contextual features available for experimentation. Nevertheless, for reference, we will present a few numbers comparing our SNMLM with only n-gram features to classical n-gram models at various pruning thresholds. After that, we will add location labels and user id's to the feature set and report improvements. Then we prune our contextual SNMLM model down to a reasonable size and show that the improvements persist. The WER experiments were conducted as N-best re-scoring on first-pass recognition outputs, which were taken from production results where each utterance was recognized with a large production LM. The production LM was a 5-gram model trained from a large body of text covering all domains relevant to the application. The acoustic model was a sequence trained context-dependent DNN model with a front-end of 29 logfilter bank features and their first and second derivatives. The N-best oracle WER relative to the 1-best baseline is around 40%. Results are reported as percentage WER reduction (WERR) over this baseline. Absolute perplexity difference of around 2 and WERR difference of 0.5% are significant at a p level of 0.05. Before we summarize our results, we describe our data set.

4.1. Data

Our data set is a combination of text queries and unsupervised recognition output from voice utterances that are sent to Cortana, Microsoft's virtual voice assistant. We work with a set of around 220 million US-English sentences, which consist of nearly 1.2 billion tokens. Each sentence has a geolocation and anonymized user id associated with it. The geolocation coordinates are pre-clustered into 500 clusters by a k-means algorithm. We use the resulting cluster labels as our location feature. This is similar to using 'market area' descriptors that has been reported in [10], but is slightly less arbitrary. We use the user id's

	PPL		
#params	n-gram	entropy-pruned	MI-pruned
630M	73.1	75.1	75.1
280M	74.1	77.1	78.3
140M	74.6	79.3	80.4
77M	75.3	81.6	83.1
37M	77.2	86.2	86.7
17M	79.4	93.5	92.6
11M	82.7	99.0	98.5
4M	97.3	107.6	110.0

Table 1: Comparing the perplexity of an entropy pruned n-gram ARPA model to SNMLM under entropy-based pruning and MI-based pruning.

	WERR		
#params	n-gram	entropy-pruned	MI-pruned
630M	4.82	5.17	5.17
280M	5.36	4.77	4.55
140M	5.40	4.65	4.60
77M	5.55	4.62	4.67
37M	5.02	4.72	4.22
17M	4.93	4.34	4.10
11M	4.56	4.25	4.01
4M	4.15	4.04	3.97

Table 2: Comparing the percentage WER reduction over a baseline (WERR) of an entropy pruned n-gram ARPA model to SNMLM under entropy-based pruning and MI-based pruning.

as is. The test set is similar, but comes from a later, disjoint date range, and consists of around 50,000 manually transcribed Cortana voice utterances. We restrict the vocabulary to the top 600k words.

4.2. SNMLM vs. n-gram

SNMLM can be trained with only n-gram features, providing a direct comparison with smoothed n-gram models. Tables 1 and 2 compare the perplexity and WER reduction (WERR) of an entropy pruned n-gram ARPA model to SNMLM under entropy-based pruning and MI-based pruning of [8] at various pruning levels. When compared directly to a 5-gram model with modified absolute discounting, SNMLM n-gram model performs on average 10% worse in terms of perplexity. The entropy based pruning works as well in both cases, limiting the perplexity degradation to around 10% for every 8-9x reduction in model size. The pruning heuristic described in [8] does remarkably well, almost matching and sometimes even beating the entropy-pruned SNMLM in perplexity, however falls short in WER metrics.

4.3. SNMLM with contextual features

The main value proposition of SNMLM is the ability to incorporate contextual features cheaply. We experiment with two of these: location and personalization features, as described in section 4.1 above. In table 3 we compare an n-gram SNMLM model with location features only, with personalization features only, and with the combination of both. Since each contextual feature produces its own set of n-gram features, adding contextual features expands the feature space greatly. The unpruned n-gram SNMLM has 630 million parameters. For SNMLM with location features, this number grows to 1.98 billion. With personalization features, it is 3.4 billion. And with both features added, the model has 4.8 billion features. We see that adding each feature results in both perplexity and WER improvements. Stacking features give additional gain, although not completely additive.

model	PPL	WERR
n-gram SNMLM	75.1	5.17
+location	69.2	6.26
+user id	59.8	5.45
+location+user id	**55.4**	**6.80**

Table 3: Comparison of n-gram SNMLM with contextual features added.

	PPL		WERR	
#params	entropy-based	MI based	entropy-based	MI based
4.8B	55.4	55.4	6.80	6.80
1.1B	58.4	60.6	6.11	5.90
460M	60.8	63.2	6.02	6.11
290M	63.6	65.0	6.04	5.62
140M	69.6	70.1	5.90	5.29
78M	77.5	73.7	5.62	4.91

Table 4: Comparing entropy-based vs. MI-based Pruning of SNMLM with location and user id features.

While the gains from adding contextual features are significant, the huge expansion in model size can be prohibitive in production settings. Furthermore, it is not clear if the gains are simply a result of adding more parameters to the model, or if the novel contextual information is actually helping. After pruning the contextual model (location+user id) to a more reasonable size, we can see in table 4 that contextual models do have an advantage of pure n-gram SNMLM models even when pruned down to a comparable size. Comparison with the MI-based pruning yields similar conclusions as in the n-gram case.

5. Conclusion

An efficient entropy-based pruning algorithm for non-negative matrix-based language models has been demonstrated. We showed that the algorithm performs competitively with a commonly used entropy pruned n-gram model smoothed with modified absolute discounting. We also showed that SNMLM trained with contextual features, such as location and personalization features, outperform vanilla SNMLM models, even when pruned down to similar size. Future work includes incorporating more contextual features, such as time, gender, app context etc. Also, deeper investigation into the training of meta-features may lead to further optimizations in speed and performance.

6. References

[1] N. Shazeer, J. Pelemans, and C. Chelba, “Sparse non-negative matrix language modeling for skip-grams,” in *Proceedings of Interspeech*, 2015, pp. 1428–1432.

[2] G. Zweig and S. Chang, “Personalizing model m for voice-search.” in *INTERSPEECH*, 2011, pp. 609–612.

[3] S. F. Chen, “Performance prediction for exponential language models,” in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009, pp. 450–458.

[4] S. F. Chen and S. M. Chu, “Enhanced word classing for model m.” in *INTERSPEECH*, 2010, pp. 1037–1040.

[5] C. Chelba, X. Zhang, and K. Hall, “Geo-location for voice search language modeling,” in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[6] A. Stolcke, “Entropy-based pruning of backoff language models,” *arXiv preprint cs/0006025*, 2000.

[7] P. Nguyen, J. Gao, and M. Mahajan, “Msrlm: a scalable language modeling toolkit,” *Microsoft Research MSR-TR-2007-144.2007*, 2007.

[8] J. Pelemans, N. Shazeer, and C. Chelba, “Pruning sparse non-negative matrix n-gram language models,” in *Proceedings of Interspeech*, 2015, pp. 1433–1437.

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

[10] C. Chelba and N. Shazeer, “Sparse non-negative matrix language modeling for geo-annotated query session data,” *ASRU*, 2015.