Skip-gram Language Modeling Using Sparse Non-negative Matrix Probability Estimation

Noam Shazeer Joris Pelemans
Ciprian Chelba
Google, Inc., 1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
{noam,jpeleman,ciprianchelba}@google.com

Abstract

We present a novel family of language model (LM) estimation techniques named Sparse Non-negative Matrix (SNM) estimation.

A first set of experiments empirically evaluating it on the One Billion Word Benchmark [Chelba et al., 2013] shows that SNM n-gram LMs perform almost as well as the well-established Kneser-Ney (KN) models. When using skip-gram features the models are able to match the state-of-the-art recurrent neural network (RNN) LMs; combining the two modeling techniques yields the best known result on the benchmark.

The computational advantages of SNM over both maximum entropy and RNN LM estimation are probably its main strength, promising 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.

1 Introduction

A statistical language model estimates the prior probability values $P(W)$ for strings of words W in a vocabulary $\mathcal{V}$ whose size is in the tens, hundreds of thousands and sometimes even millions. Typically the string W is broken into sentences, or other segments such as utterances in automatic speech recognition, which are assumed to be conditionally independent; we will assume that W is such a segment, or sentence.

Estimating full sentence language models is computationally hard if one seeks a properly normalized probability model[1] over strings of words of finite length in

[1]We note that in some practical systems the constraint on using a properly normalized language

$\mathcal{V}^*$. A simple and sufficient way to ensure proper normalization of the model is to decompose the sentence probability according to the chain rule and make sure that the end-of-sentence symbol `</s>` is predicted with non-zero probability in any context. With $W = w_1, w_2, \ldots, w_n$ we get:

$$P(W) = \prod_{i=1}^{n} P(w_i | w_1, w_2, \ldots, w_{i-1}) \tag{1}$$

Since the parameter space of $P(w_k|w_1, w_2, \ldots, w_{k-1})$ is too large, the language model is forced to put the *context* $W_{k-1} = w_1, w_2, \ldots, w_{k-1}$ into an *equivalence class* determined by a function $\Phi(W_{k-1})$. As a result,

$$P(W) \cong \prod_{k=1}^{n} P(w_k | \Phi(W_{k-1})) \tag{2}$$

The word strings encountered in a practical application are of finite length. The probability distribution $P(W)$ should assign probability 0.0 to strings of words of infinite length, and thus sum up to 1.0 over the set of strings of finite length—the support of $P(W)$. From a modeling point of view in a practical situation, the text gets broken into sentences, and the language model needs to predict the distinguished end-of-sentence symbol `</s>`. It can be easily shown that if the language model is smooth, i.e. $P(w_k|\Phi(W_{k-1})) > \epsilon > 0, \forall w_k, W_{k-1}$, then we also have $P(\texttt{</s>}|\Phi(W_{k-1})) > \epsilon > 0, \forall W_{k-1}$ which in turn ensures that the model assigns probability 1.0 to the set strings of words of finite length.

Research in language modeling consists of finding appropriate equivalence classifiers Φ and methods to estimate $P(w_k|\Phi(W_{k-1}))$. The most successful paradigm in language modeling uses the $(n-1)$-*gram* equivalence classification, that is, defines

$$\Phi(W_{k-1}) \doteq w_{k-n+1}, w_{k-n+2}, \ldots, w_{k-1}$$

Once the form $\Phi(W_{k-1})$ is specified, only the problem of estimating $P(w_k|\Phi(W_{k-1}))$ from training data remains.

Perplexity as a Measure of Language Model Quality

A *statistical language model* can be evaluated by how well it predicts a string of symbols W_t—commonly referred to as *test data*—generated by the source to be modeled.

model is side-stepped at a gain in modeling power and simplicity.

A commonly used quality measure for a given model M is related to the entropy of the underlying source and was introduced under the name of *perplexity* (PPL) [Jelinek, 1997]:

$$PPL(M) = exp(-\frac{1}{N}\sum_{k=1}^{N}\ln\left[P_M(w_k|W_{k-1})\right]) \tag{3}$$

To give intuitive meaning to perplexity, it represents the number of guesses the model needs to make in order to ascertain the identity of the next word, when running over the test word string from left to right. It can be easily shown that the perplexity of a language model that uses the uniform probability distribution over words in the vocabulary $\mathcal{V}$ equals the size of the vocabulary; a good language model should of course have lower perplexity, and thus the vocabulary size is an upper bound on the perplexity of any sensible language model.

Very likely, not all words in the test string W_t are part of the language model vocabulary. It is common practice to map all words that are out-of-vocabulary to a distinguished *unknown word* symbol, and report the out-of-vocabulary (OOV) rate on test data—the rate at which one encounters OOV words in the test string W_t—as yet another language model performance metric besides perplexity. Usually the unknown word is assumed to be part of the language model vocabulary—*open vocabulary* language models—and its occurrences are counted in the language model perplexity calculation, Eq. (3). A situation less common in practice is that of *closed vocabulary* language models where all words in the test data will always be part of the vocabulary $\mathcal{V}$.

2 Skip-gram Language Modeling

Recently, neural network (NN) smoothing [Bengio et al., 2003], [Emami, 2006], [Schwenk, 2007], and in particular recurrent neural networks [Mikolov, 2012] (RNN) have shown excellent performance in language modeling [Chelba et al., 2013]. Their excellent performance is attributed to a combination of leveraging long-distance context, and training a vector representation for words.

Another simple way of leveraging long distance context is to use skip-grams. In our approach, a skip-gram feature extracted from the context W_{k-1} is characterized by the tuple (r, s, a) where:

- r denotes number of remote context words
- s denotes the number of skipped words
- a denotes the number of adjacent context words

relative to the target word w_k being predicted. For example, in the sentence, `<S> The quick brown fox jumps over the lazy dog </S>` a $(1, 2, 3)$ skip-gram feature for the target word `dog` is:
`[brown skip-2 over the lazy]`

For performance reasons, it is recommended to limit s and to limit either $(r+a)$ or limit both r and s; not setting any limits will result in events containing a set of skip-gram features whose total representation size is quintic in the length of the sentence.

We configure the skip-gram feature extractor to produce all features $\mathbf{f}$, defined by the equivalence class $\Phi(W_{k-1})$, that meet constraints on the minimum and maximum values for:

- the number of context words used $r + a$;
- the number of remote words r;
- the number of adjacent words a;
- the skip length s.

We also allow the option of not including the exact value of s in the feature representation; this may help with smoothing by sharing counts for various skip features. Tied skip-gram features will look like:
`[curiousity skip-* the cat]`

In order to build a good probability estimate for the target word w_k in a context W_{k-1} we need a way of combining an arbitrary number of skip-gram features $\mathbf{f}_{k-1}$, which do not fall into a simple hierarchy like regular n-gram features. The following section describes a simple, yet novel approach for combining such predictors in a way that is computationally easy, scales up gracefully to large amounts of data and as it turns out is also very effective from a modeling point of view.

3 Sparse Non-negative Matrix 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 words, respectively. Hence, the training data consists of $|E||Pos(\mathbf{f})|$ positive and $|E||Pos(\mathbf{f})|(|\mathcal{V}| - 1)$ negative training examples, where $Pos(\mathbf{f})$ denotes the number of positive elements in the vector $\mathbf{f}$.

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 $\mathbf{y}$:

$$\mathbf{y} = \mathbf{M}\mathbf{f} \approx \mathbf{t} \tag{4}$$

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 feature vector $\mathbf{f}$ that is defined by the equivalence class Φ applied to the history $h(w)$ of that word in a text, the conditional probability $P_{\mathbf{M}}(w|\Phi(h(w)))$ then becomes:

$$P_{\mathbf{M}}(w|\Phi(h(w))) = P_{\mathbf{M}}(t_j|\mathbf{f}) = \frac{y_j}{\sum_{u=1}^{|\mathcal{V}|} y_u} = \frac{\sum_{i \in Pos(\mathbf{f})} M_{ij}}{\sum_{i \in Pos(\mathbf{f})} \sum_{u=1}^{|\mathcal{V}|} M_{iu}} \tag{5}$$

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. (5), 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. Moreover, only the positive entries in M_i need to be considered, making the range of the sum sparse.

3.2 Adjustment function and metafeatures

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

$$M_{ij} = e^{A(i,j)} \frac{C_{ij}}{C_{i*}} \tag{6}$$

where $\mathbf{C}$ is a feature-target count matrix, computed over the entire training corpus and $A(i,j)$ is a real-valued function, dubbed *adjustment function*. For each feature-target pair (f_i, t_j), the adjustment function extracts k new features α_k, called *metafeatures*, which are hashed as keys to store corresponding weights $\theta(hash(\alpha_k))$ in a huge hash table. To limit memory usage, we use a flat hash table and allow collisions, although this has the potentially undesirable effect of tying together the weights of different metafeatures. Computing the adjustment function for any (f_i, t_j) then amounts to summing the weights that correspond to its metafeatures:

$$A(i,j) = \sum_k \theta(hash[\alpha_k(i,j)]) \tag{7}$$

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

- feature identity, e.g. `[brown skip-2 over the lazy]`
- feature type, e.g. $(1, 2, 3)$ skip-grams
- feature count C_{i*}
- target identity, e.g. `dog`
- feature-target count C_{ij}

where we reused the example from Section 2. 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 $\alpha_k(i, j, C_{i*}, C_{ij})$. This will become important later when 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.

Before they are joined, count metafeatures are bucketed together according to their (floored) $\log_2$ value. As this effectively puts the lowest count values, of which there are many, into a different bucket, we optionally introduce a second (ceiled) bucket to assure smoother transitions. Both buckets are then weighted according to the $\log_2$ fraction lost by the corresponding rounding operation. Note that if we apply double bucketing to both the feature and feature-target count, the amount of metafeatures per input feature becomes $2^7 - 1$.

We will come back to these metafeatures in Section 4.4 where we examine their individual effect on the model.

3.3 Loss function

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. The most natural choice of loss function is one that is based on the multinomial distribution. That is, we consider $\mathbf{t}$ to be multinomially distributed with $|\mathcal{V}|$ possible outcomes. The loss function L_{multi} then is:

$$L_{multi}(\mathbf{y}, \mathbf{t}) = -log(P_{multi}(\mathbf{t}|\mathbf{f})) = -log(\frac{y_j}{\sum_{u=1}^{|\mathcal{V}|} y_u}) = log(\sum_{u=1}^{|\mathcal{V}|} y_u) - log(y_j) \tag{8}$$

Another possibility is the loss function based on the Poisson distribution[2]: we consider each t_j in t to be Poisson distributed with parameter y_j. The conditional probability of $P_{Poisson}(\mathbf{t}|\mathbf{f})$ then is:

$$P_{Poisson}(\mathbf{t}|\mathbf{f}) = \prod_{j \in \mathbf{t}} \frac{y_j^{t_j} e^{-y_j}}{t_j!} \tag{9}$$

and the corresponding Poisson loss function is:

$$\begin{aligned} L_{Poisson}(\mathbf{y}, \mathbf{t}) = -log(P_{Poisson}(\mathbf{t}|\mathbf{f})) &= -\sum_{j \in \mathbf{t}} [t_j \log(y_j) - y_j - log(t_j!)] \\ &= \sum_{j \in \mathbf{t}} y_j - \sum_{j \in \mathbf{t}} t_j \log(y_j) \end{aligned} \tag{10}$$

where we dropped the last term, since t_j is binary-valued[3]. 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.

3.4 Model Estimation

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 parameters of the metafeatures by calculating the gradient with respect to the adjustment function.

For the multinomial loss, this gradient is:

$$\begin{aligned} \frac{\partial(L_{multi}(\mathbf{Mf}, \mathbf{t}))}{\partial(A(i,j))} &= \frac{\partial(log(\sum_{u=1}^{|\mathcal{V}|}(\mathbf{Mf})_u) - log(\mathbf{Mf})_j)}{\partial(M_{ij})} \frac{\partial(M_{ij})}{\partial(A_{ij})} \\ &= [\frac{\partial(log(\sum_{u=1}^{|\mathcal{V}|}(\mathbf{Mf})_u))}{\partial(M_{ij})} - \frac{\partial(log(\mathbf{Mf})_j)}{\partial(M_{ij})}]M_{ij} \\ &= [\frac{\partial(\sum_{u=1}^{|\mathcal{V}|}(\mathbf{Mf})_u)}{\sum_{u=1}^{|\mathcal{V}|}(\mathbf{Mf})_u \partial(M_{ij})} - \frac{\partial(\mathbf{Mf})_j}{(\mathbf{Mf})_j \partial(M_{ij})}]M_{ij} \\ &= (\frac{f_i}{\sum_{u=1}^{|\mathcal{V}|}(\mathbf{Mf})_u} - \frac{f_i}{y_j})M_{ij} \\ &= f_i M_{ij}(\frac{1}{\sum_{u=1}^{|\mathcal{V}|} y_u} - \frac{1}{y_j}) \end{aligned} \tag{11}$$

[2] Although we do not use it at this point, the Poisson loss also lends itself nicely for multiple target prediction which might be useful in e.g. subword modeling.

[3] In fact, even in the general case where t_k can take any non-negative value, this term will disappear in the gradient, as it is independent of $\mathbf{M}$.

The problem with this update rule is that we need to sum over the entire vocabulary $\mathcal{V}$ in the denominator. For most features f_i, this is not a big deal as $C_{iu} = 0$, but some features occur with many if not all targets e.g. the empty feature for unigrams. Although we might be able to get away with this by re-using these sums and applying them to many/all events in a mini batch, we chose to work with the Poisson loss in our first implementation.

If we calculate the gradient of the Poisson loss, we get the following:

$$\begin{aligned}
\frac{\partial(L_{Poisson}(\mathbf{Mf}, \mathbf{t}))}{\partial(A(i,j))} &= \frac{\partial(\sum_{u=1}^{|\mathcal{V}|}(\mathbf{Mf})_u - \sum_{u=1}^{|\mathcal{V}|} t_u \log(\mathbf{Mf})_u)}{\partial(M_{ij})} \frac{\partial(M_{ij})}{\partial(A(i,j))} \\
&= [\frac{\partial(\sum_{u=1}^{|\mathcal{V}|}(\mathbf{Mf})_u)}{\partial(M_{ij})} - \frac{\partial(\sum_{u=1}^{|\mathcal{V}|} t_u \log(\mathbf{Mf})_u)}{\partial(M_{ij})}] M_{ij} \\
&= [f_i - \frac{t_j}{(\mathbf{Mf})_j} \frac{\partial(\mathbf{Mf})_j}{\partial(M_{ij})}] M_{ij} \\
&= [f_i - \frac{t_j f_i}{(\mathbf{Mf})_j}] M_{ij} \\
&= f_i M_{ij} (1 - \frac{t_j}{y_j}) \qquad (12)
\end{aligned}$$

If we were to apply this gradient 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 $f_i t_j \frac{C_{i*}}{C_{ij}} M_{ij}$:

$$\frac{C_{i*}}{C_{ij}} M_{ij} \sum_{e=(f_i,t_j)\in E} f_i t_j = \frac{C_{i*}}{C_{ij}} M_{ij} C_{ij} = M_{ij} \sum_{e=(f_i,t_j)\in E} f_i \qquad (13)$$

which lets us update the gradient only on positive examples. 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:

$$\frac{\partial(L_{Poisson}(\mathbf{Mf}, \mathbf{t}))}{\partial(A(i,j))} = f_i t_j M_{ij} (\frac{C_{i*}}{C_{ij}} - \frac{1}{y_j}) \qquad (14)$$

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

These aggregated gradients however do not allow us to use additional data to train the adjustment function, since they tie the update computation to the relative frequencies $\frac{C_{i*}}{C_{ij}}$. Instead, we have to resort to leave-one-out training to prevent the model from overfitting the training data. We do this by excluding the event, generating 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 f_i from C_{i*} and $f_i t_j$ from C_{ij}. For the gradients of the negative examples on the other hand we only exclude f_i from C_{i*} and we leave C_{ij} untouched, since here 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:

$$\begin{aligned}\frac{\partial(L_{Poisson}(\mathbf{Mf}, \mathbf{t}))}{\partial(A(i,j))} &= f_i t_j \frac{C_{i*} - C_{ij}}{C_{ij}} \frac{C_{ij}}{C_{i*} - 1} e^{\sum_k \theta(hash[\alpha_k(i,j,C_{i*}-1,C_{ij})])} \\ &+ f_i t_j \frac{C_{ij} - 1}{C_{i*} - 1} \frac{y'_j - 1}{y'_j} e^{\sum_k \theta(hash[\alpha_k(i,j,C_{i*}-1,C_{ij}-1)])} \quad (15)\end{aligned}$$

where y'_j is the product of leaving one out for all the relevant features i.e. $y'_j = (\mathbf{M}'\mathbf{f})_j$ and $\mathbf{M}'_{ij} = e^{\sum_k \theta(hash[\alpha_k(i,j,C_{i*}-1,C_{ij}-1)])} \frac{C_{ij}-1}{C_{i*}-1}$.

4 Experiments

4.1 Corpus: One Billion Benchmark

Our experimental setup used the One Billion Word Benchmark corpus[4] made available by [Chelba et al., 2013].

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

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

Model	5	6	7	8
KN	67.6	64.3	63.2	62.9
Katz	79.9	80.5	82.2	83.5
SNM	70.8	67.0	65.4	64.8
KN+SNM	66.5	63.0	61.7	61.4

Table 1: Perplexity results for Kneser-Ney, Katz and SNM, as well as for the linear interpolation of Kneser-Ney and SNM. Optimal interpolation weights are always around $0.6 - 0.7$ (KN) and $0.3 - 0.4$ (SNM).

- Words outside of the vocabulary were mapped to <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%.

4.2 SNM for n-gram LMs

When trained using solely n-gram features, SNM comes very close to the state-of-the-art Kneser-Ney [Kneser and Ney, 1995] (KN) models. Table 1 shows that Katz [Katz, 1995] performs considerably worse than both SNM and KN which only differ by about 5%. When we interpolate these two models linearly, the added gain is only about 1%, suggesting that they are approximately modeling the same things. The difference between KN and SNM becomes smaller when we increase the size of the context, going from 5% for 5-grams to 3% for 8-grams, which indicates that SNM is better suited to a large number of features.

4.3 Sparse Non-negative Modeling for Skip n-grams

When we incorporate skip-gram features, we can either build a 'pure' skip-gram SNM that contains no regular n-gram features, except for unigrams, and interpolate this model with KN, or we can build a single SNM that has both the regular n-gram features and the skip-gram features. We compared the two approaches by choosing skip-gram features that can be considered the skip-equivalent of 5-grams i.e. they contain at most 4 words. In particular, we used skip-gram features where the remote span is limited to at most 3 words for skips of length between 1 and 3 ($r = [1..3]$, $s = [1..3]$, $r + a = [1..4]$) and where all skips longer than 4 are tied

Model	Num. Params	PPL
SNM5-skip (no n-grams)	61 B	69.8
SNM5-skip	62 B	54.2
KN5+SNM5-skip (no n-grams)		56.5
KN5+SNM5-skip		53.6

Table 2: Number of parameters (in billions) and perplexity results for SNM5-skip models with and without n-grams, as well as perplexity results for the interpolation with KN5.

and limited by a remote span length of at most 2 words ($r = [1..2]$, $s = [4..*]$, $r + a = [1..4]$). We then built a model that uses both these features and regular 5-grams (SNM5-skip), as well as one that only uses the skip-gram features (SNM5-skip (no n-grams)).

As it turns out and as can be seen from Table 2, it is better to incorporate all the features into one single SNM model than to interpolate with a KN 5-gram model (KN5). Interpolating the all-in-one SNM5-skip with KN5 yields almost no additional gain.

The best SNM results so far (SNM10-skip) were achieved using 10-grams, together with untied skip features of at most 5 words with a skip of exactly 1 word ($s = 1$, $r + a = [1..5]$) as well as tied skip features of at most 4 words where only 1 word is remote, but up to 10 words can be skipped ($r = 1$, $s = [1..10]$, $r + a = [1..4]$).

This mixture of rich short-distance and shallow long-distance features enables the model to achieve state-of-the-art results, as can be seen in Table 3. When we compare the perplexity of this model with the state-of-the art RNN results in [Chelba et al., 2013], the difference is only 3%. Moreover, although our model has more parameters than the RNN (33 vs 20 billion), training takes about a tenth of the time (24 hours vs 240 hours). Interestingly, when we interpolate the two models, we have an additional gain of 20%, and as far as we know, the perplexity of 41.3 is already the best ever reported on this database, beating the previous best by 6% [Chelba et al., 2013].

Finally, when we optimize interpolation weights over all models in [Chelba et al., 2013], including SNM5-skip and SNM10-skip, the contribution of the other models as well as the perplexity reduction is negligible, as can be seen in Table 3, which also summarizes the perplexity results for each of the individual models.

Model	Num. Params	PPL	interpolation weights		
KN5	1.76 B	67.6		0.06	0.00
HSME	6 B	101.3		0.00	0.00
SBO	1.13 B	87.9		0.20	0.04
SNM5-skip	62 B	54.2			0.10
SNM10-skip	33 B	52.9	0.4		0.27
RNN256	20 B	58.2		0.00	0.00
RNN512	20 B	54.6		0.13	0.07
RNN1024	20 B	51.3	0.6	0.61	0.53
SNM10-skip+RNN1024			41.3		
Previous best				43.8	
ALL					41.0

Table 3: Number of parameters (in billions) and perplexity results for each of the models in [Chelba et al., 2013], and SNM5-skip and SNM10-skip, as well as interpolation results and weights.

4.4 Ablation Experiments

To find out how much, if anything at all, each metafeature contributes to the adjustment function, we ran a series of ablation experiments in which we ablated one metafeature at a time. When we experimented on SNM5, we found, unsurprisingly, that the most important metafeature is the feature-target count. At first glance, it does not seem to matter much whether the counts are stored in 1 or 2 buckets, but the second bucket really starts to pay off for models with a large number of singleton features e.g. SNM10-skip[5]. This is not the case for the feature counts, where having a single bucket is always better, although in general the feature counts do not contribute much. In any case, feature counts are definitely the least important for the model. The remaining metafeatures all contribute more or less equally, all of which can be seen in Table 4.

5 Related Work

SNM estimation is closely related to all n-gram LM smoothing techniques that rely on mixing relative frequencies at various orders. Unlike most of those, it combines the predictors at various orders without relying on a hierarchical nesting of the contexts, setting it closer to the family of maximum entropy (ME) [Rosenfeld, 1994],

[5]Ideally we want to have the SNM10-skip ablation results as well, but this takes up a lot of time, during which other development is hindered.

Ablated feature	PPL
No ablation	70.8
Feature	71.9
Feature type	71.4
Feature count	70.6
Feature count: second bucket	70.3
Link count	73.2
Link count: second bucket	70.6

Table 4: Metafeature ablation experiments on SNM5

or exponential models.

We are not the first ones to highlight the effectiveness of skip n-grams at capturing dependencies across longer contexts, similar to RNN LMs; previous such results were reported in [Singh and Klakow, 2013].

The speed-ups to ME, and RNN LM training provided by hierarchically predicting words at the output layer [Goodman, 2001b], and subsampling [Xu et al., 2011] still require updates that are linear in the vocabulary size times the number of words in the training data, whereas the SNM updates in Eq. (15) for the much smaller adjustment function eliminate the dependency on the vocabulary size.

The computational advantages of SNM over both Maximum Entropy and RNN LM estimation are probably its main strength, promising 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.

6 Conclusions and Future Work

We have presented SNM, a new family of LM estimation techniques. A first empirical evaluation on the One Billion Word Benchmark [Chelba et al., 2013] shows that SNM n-gram LMs perform almost as well as the well-established KN models.

When using skip-gram features the models are able to match the stat-of-the-art RNN LMs; combining the two modeling techniques yields the best known result on the benchmark.

Future work items include model pruning, exploring richer features similar to [Goodman, 2001a], 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.

References

[Bengio et al., 2003] Y. Bengio, R. Ducharme, and P. Vincent. 2003. *A neural probabilistic language model*. Journal of Machine Learning Research, 3:1137-1155.

[Brown et al., 1992] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai. 1992. *Class-Based N-gram Models of Natural Language*. Computational Linguistics, 18, 467-479.

[Emami, 2006] A. Emami. 2006. *A Neural Syntactic Language Model*. Ph.D. thesis, Johns Hopkins University.

[Goodman, 2001a] J. T. Goodman. 2001a. *A bit of progress in language modeling, extended version*. Technical report MSR-TR-2001-72.

[Goodman, 2001b] J. T. Goodman. 2001b. *Classes for fast maximum entropy training*. In Proceedings of ICASSP.

[Katz, 1995] S. Katz. 1987. *Estimation of probabilities from sparse data for the language model component of a speech recognizer*. In IEEE Transactions on Acoustics, Speech and Signal Processing.

[Kneser and Ney, 1995] R. Kneser and H. Ney. 1995. *Improved Backing-Off For M-Gram Language Modeling*. In Proceedings of ICASSP.

[Mikolov, 2012] T. Mikolov. 2012. *Statistical Language Models based on Neural Networks*. Ph.D. thesis, Brno University of Technology.

[Morin and Bengio, 2005] F. Morin and Y. Bengio. 2005. *Hierarchical Probabilistic Neural Network Language Model*. In Proceedings of AISTATS.

[Rosenfeld, 1994] R. Rosenfeld. 1994. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. Ph.D. thesis, Carnegie Mellon University.

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

[Sundermeyer et al., 2012] M. Sundermeyer, R. Schluter, and H. Ney. 2012. *LSTM Neural Networks for Language Modeling*. In Proceedings of Interspeech.

[Teh, 2006] Y. W. Teh. 2006. *A hierarchical Bayesian language model based on PitmanYor processes*. In Proceedings of Coling/ACL.

[Chelba et al., 2013] Ciprian Chelba and Tomas Mikolov and Mike Schuster and Qi Ge and Thorsten Brants and Phillipp Koehn and Tony Robinson. 2013. *One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling*. Google Tech Report 41880.

[Jelinek and Mercer, 1980] Frederick Jelinek and Robert Mercer 1980. *Interpolated estimation of Markov source parameters from sparse data*. In Pattern Recognition in Practice, (381–397), Gelsema and Kanal (eds.).

[Jelinek, 1997] Frederick Jelinek 1997. *Information Extraction From Speech And Text*. MIT Press.

[Singh and Klakow, 2013] M. Singh, D. Klakow. 2013. *Comparing RNNs and log-linear interpolation of improved skip-model on four Babel languages: Cantonese, Pashto, Tagalog, Turkish*. In Proceedings of ICASSP.

[Xu et al., 2011] Puyang Xu, A. Gunawardana, and S. Khudanpur. 2011. *Efficient Subsampling for Training Complex Language Models*. In Proceedings of EMNLP.