Sparse Non-negative Matrix Language Modeling

Joris Pelemans
Google Inc.
ESAT, KU Leuven
joris@pelemans.be

Noam Shazeer
Google Inc.
noam@google.com

Ciprian Chelba
Google Inc.
ciprianchelba@google.com

Abstract

We present Sparse Non-negative Matrix (SNM) estimation, a novel probability estimation technique for language modeling that can efficiently incorporate arbitrary features. We evaluate SNM language models on two corpora: the One Billion Word Benchmark and a subset of the LDC English Gigaword corpus. Results show that SNM language models trained with n-gram features are a close match for the well-established Kneser-Ney models. The addition of skip-gram features yields a model that is in the same league as the state-of-the-art recurrent neural network language models, as well as complementary: combining the two modeling techniques yields the best known result on the One Billion Word Benchmark. On the Gigaword corpus further improvements are observed using features that cross sentence boundaries. The computational advantages of SNM estimation over both maximum entropy and neural network estimation are probably its main strength, promising an approach that has large flexibility in combining arbitrary features and yet scales gracefully to large amounts of data.

1 Introduction

A *statistical language model* estimates probability values $P(W)$ for strings of words W in a vocabulary $\mathcal{V}$ whose size can be in the tens or 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 often assumed to be conditionally independent; we will assume that W is such a segment, or sentence.

Estimating full sentence language models (Rosenfeld et al., 2001) is computationally hard if one seeks a properly normalized probability model[1] over strings of words of finite length in $\mathcal{V}^*$. 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 $W = w_1^N = w_1, \ldots, w_N$ we get:

$$P(w_1^N) = \prod_{k=1}^{N} P(w_k | w_1^{k-1}) \tag{1}$$

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

$$P(w_1^N) \cong \prod_{k=1}^{N} P(w_k | \Phi(w_1^{k-1})) \tag{2}$$

Research in language modeling consists of finding appropriate equivalence classifiers Φ and methods to estimate $P(w_k|\Phi(w_1^{k-1}))$. Arguably the most successful paradigm in language modeling uses the n-*gram* equivalence classification, that is, defines

$$\Phi_{\text{n-gram}}(w_1^{k-1}) \doteq w_{k-n+1}, w_{k-n+2}, \ldots, w_{k-1}$$

Once the form $\Phi(w_1^{k-1})$ is specified, only the problem of estimating $P(w_k|\Phi(w_1^{k-1}))$ from training data remains.

In order to outperform the n-gram equivalence class, one must find a way to leverage long-distance context. This can be done explicitly, e.g. by combining multiple arbitrary features (Rosenfeld, 1994), or implicitly as is the case for the current state of the art

[1] In some practical systems the constraint on using a properly normalized language model is side-stepped at a gain in modeling power and simplicity, see e.g. Chen et al. (1998).

Transactions of the Association for Computational Linguistics, vol. 4, pp. 329–342, 2016. Action Editor: Jason Eisner.
Submission batch: 12/2014; Revision batch: 7/2015; 11/2015; 4/2016; 6/2016; Published 7/2016.

recurrent neural network language models (Mikolov, 2012). Unfortunately, either method comes at a large computational cost which makes training and evaluation on a large corpus impractical.

In this paper we present a novel probability estimation technique, called Sparse Non-negative Matrix (SNM) estimation. Although SNM estimation is a general approach that can be applied to many problems, its efficient combination of arbitrary features makes it particularly interesting for language modeling. We demonstrate this by training models with variable-length n-gram features and skip-gram features to incorporate long-distance context.

The paper is organized as follows: Section 2 discusses work that is related to SNM which is described in Section 3. We then present a complexity analysis in Section 4 and experimental results on two English corpora in Sections 5 and 6. We end with conclusions and future work in Section 7.

2 Related Work

2.1 Neural networks

Recently, neural networks (NN) (Bengio et al., 2003; Emami, 2006; Schwenk, 2007), and in particular recurrent neural networks (RNN) (Mikolov, 2012; Sundermeyer et al., 2012) have shown excellent performance in language modeling (Chelba et al., 2014). RNNLMs have two main advantages over n-gram language models: 1) they learn a low-dimensional continuous vector representation for words which allows them to discover fine-grained similarities between words; 2) they are capable of modeling dependencies that span over longer distances, i.e. they can extend the context past the n-gram window. Their main disadvantage however is that they take a long time to train and evaluate.

2.2 Feature-based models

Another popular method to leverage long-distance context is Maximum Entropy (ME) (Rosenfeld, 1994). ME is interesting because it can mix different types of features extracted from large context windows, e.g. n-gram, skip-gram, bag-of-word and syntactic features. Unfortunately it suffers from the same drawback as neural networks, as we will see in Section 2.4.

The above-mentioned features can also be used in other ways, e.g. Chelba and Jelinek (2000) use a left-to-right syntactic parser to identify long-distance dependencies (at sentence level), whereas approaches such as Bellegarda (2000) leverage latent semantic information (at document level). Tan et al. (2012) integrate both syntactic and topic-based modeling with n-grams in a unified approach.

2.3 Skip-grams

The type of long-distance features that we incorporate into our SNMLMs are skip-grams (Huang et al., 1993; Ney et al., 1994; Rosenfeld, 1994), which can effectively capture dependencies across longer contexts. We are not the first to highlight this effectiveness; previous such results were reported in Singh and Klakow (2013). Recently, Pickhardt et al. (2014) also showed that a backoff generalization using single skips yields significant perplexity reductions. We note though that our SNMLMs are trained by mixing single as well as longer skips, combining both in one model. More fundamentally, the SNM model parameterization and method of estimation are completely original, as far as we know.

In our approach, a skip-gram feature extracted from the context w_1^{k-1} is characterized by the tuple (r, s, a) where:

- r denotes the 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. The window size of a feature extractor then corresponds to $r + s + a$. 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 both r and s.

We configure the skip-gram feature extractor to produce all features $\mathcal{F}$, defined by the equivalence class $\Phi(w_1^{k-1})$, that meet constraints on the minimum and maximum values for:

- the number of context words $r + a$
- 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. The resulting tied skip-gram features will look like:

```
[curiosity skip-* the cat]
```

In order to build a good probability estimate for the target word w_k in a context w_1^{k-1} we need a way of combining an arbitrary number of skip-gram features, which do not fall into a simple hierarchy like regular n-gram features. The standard way to combine such predictors is ME, but it is computationally hard. The proposed SNM estimation on the other hand is capable of 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.

2.4 Log-linear models

Neural networks and ME are related in the sense that for both models $P(w_k|\Phi(w_1^{k-1}))$ takes the following form:

$$P(w_k|\Phi(w_1^{k-1})) = \frac{\exp(\hat{y}_{w_k})}{\sum_{t' \in \mathcal{V}} \exp(\hat{y}_{t'})} \tag{3}$$

where the $\hat{y}_{t'}$ are the unnormalized log-probabilities for each potential target word t' and depend on the model in question. For a ME model with features $\mathcal{F}$, they can be represented as follows:

$$\hat{\mathbf{y}} = \mathbf{x^T M} \tag{4}$$

where $\mathbf{x}$ is the word feature activation vector and $\mathbf{M}$ is a $|\mathcal{F}| \times |\mathcal{V}|$ feature weight matrix. The $\hat{y}_i$ of neural networks on the other hand are computed as follows:

$$\hat{\mathbf{y}} = g(\mathbf{x^T H})\mathbf{W} \tag{5}$$

where $g(\cdot)$ is the activation function of the hidden layer (typically a $\tanh$ or sigmoid) and $\mathbf{W}$ and $\mathbf{H}$ are weight matrices for the output and hidden layer respectively. Feed-forward and recurrent neural networks differ only in their input vectors $\mathbf{x}$: in a feed-forward neural network, $\mathbf{x}$ is a concatenation of the input features whereas in a recurrent neural network, $\mathbf{x}$ is a concatenation of the input word with the previous hidden state. Because of their shared log-linearity, training and evaluating these models becomes computationally complex.

Although log-linear models have been shown to perform better than linear models (Klakow, 1998), their performance is also hampered by their complexity and we will show in the rest of the paper that a linear model can in fact compete with the state of the art when trained with variable-length n-gram and skip-gram features combined.

3 Sparse Non-negative Matrix Estimation

3.1 Linear model

Contrary to neural networks and ME, SNM language models do not estimate $P(w_k|\Phi(w_1^{k-1}))$ in a log-linear fashion, but are in fact linear models:

$$P(w_k|\Phi(w_1^{k-1})) = \frac{\hat{y}_{w_k}}{\sum_{t' \in \mathcal{V}} \hat{y}_{t'}} \tag{6}$$

where $\hat{\mathbf{y}}$ is defined as in Eq. (4).

Like ME however, SNM uses features $\mathcal{F}$ that are predefined and arbitrary, e.g. n-grams, skip-grams, bags of words, syntactic features, ... The features are extracted from the left context of w_k and stored in a feature activation vector $\mathbf{x} = \Phi(w_1^{k-1})$, which is binary-valued, i.e. x_f represents the presence or absence of the feature with index f.

In what follows, we represent the target word w_k by a vector $\mathbf{y}$, which is a one-hot encoding of the vocabulary $\mathcal{V}$: $y_t = 1$ for $t = w_k$, $y_t = 0$ otherwise. To further simplify notation, we will not make the distinction between a feature or target and its index, but rather denote both of them by f and t, respectively.

The $\hat{y}_{t'}$ in SNM are computed in the same way as ME, using Eq. (4), where $\mathbf{M}$ is a $|\mathcal{F}| \times |\mathcal{V}|$ feature weight matrix, which is sparse and non-negative. M_{ft} is indexed by feature f and target t and denotes the influence of feature f in the prediction of t. Plugging Eq. (4) into Eq. (6), we can derive the complete form of the conditional distribution $P(\mathbf{y}|\mathbf{x}) = P(w_k|\Phi(w_1^{k-1}))$ in SNMLMs:

$$\begin{aligned} P(\mathbf{y}|\mathbf{x}) &= \frac{(\mathbf{x^T M})_{w_k}}{\sum_{t' \in \mathcal{V}} (\mathbf{x^T M})_{t'}} \\ &= \frac{\sum_{f' \in \mathcal{F}} x_{f'} M_{f'w_k}}{\sum_{t' \in \mathcal{V}} \sum_{f' \in \mathcal{F}} x_{f'} M_{f't'}} \\ &= \frac{\sum_{f' \in \mathcal{F}} x_{f'} M_{f'w_k}}{\sum_{f' \in \mathcal{F}} x_{f'} \sum_{t' \in \mathcal{V}} M_{f't'}} \end{aligned} \tag{7}$$

As required by the denominator in Eq. (7), this computation also involves summing over all the present features for the entire vocabulary. However, because of the linearity of the model, we can precompute the row sums $\sum_{t' \in \mathcal{V}} M_{f't'}$ for each f' and store them together with the model. This means that the evaluation can be done very efficiently, since the remaining summation involves a limited number of terms: even though the amount of features $|\mathcal{F}|$ gathered over the entire training data is potentially huge, the amount of active, non-zero features for a given $\mathbf{x}$ is small. For example, for SNM models using variable-length n-gram features, the maximum number of active features is n; in our experiments with a large variety of skip-grams, it was around 100.

Notice that this precomputation is not possible for the log-linear ME which is otherwise similar, because the sum over all features does not distribute outside the sum over all targets in the denominator:

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp(\sum_{f' \in \mathcal{F}} x_{f'} M_{f'w_k})}{\sum_{t' \in \mathcal{V}} \exp(\sum_{f' \in \mathcal{F}} x_{f'} M_{f't'})} \quad (8)$$

This is a huge difference and essentially makes SNM a more efficient model at runtime.

3.2 Adjustment function and meta-features

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

$$M_{ft} = e^{A(f,t)} \frac{C_{ft}}{C_{f*}} \quad (9)$$

where $A(f,t)$ is a real-valued function, dubbed the *adjustment function* (to be defined below), and $\mathbf{C}$ is a feature-target count matrix, computed over the entire training corpus $\mathcal{T}$. C_{ft} denotes the co-occurrence count of feature f and target t, whereas C_{f*} denotes the total occurrence count of feature f, summed over all targets t'.

An unadjusted SNM model, where $A(f,t) = 0$, is a linear mixture of simple feature models $P(t|f)$ with uniform mixture weights. The adjustment function enables the models to be weighted by the relative importance of each input feature and, because it also parameterized by t, takes into account the current target. The function is computed by a linear model on binary *meta-features* (Lee et al., 2007):

$$A(f,t) = \theta \cdot h(f,t) \quad (10)$$

where $h(f,t)$ is the meta-feature vector extracted from the feature-target pair (f,t).

Estimating weights θ_k on the meta-feature level rather than the input feature level enables similar input features to share weights which improves generalization. We illustrate this by an example.

Given the word sequence `the quick brown fox`, we extract the following elementary meta-features from the 3-gram feature `the quick brown` and the target `fox`:

- feature identity: `[the quick brown]`
- feature type: 3-gram
- feature count: C_{f*}
- target identity: `fox`
- feature-target count: C_{ft}

We also allow conjunctions of (single or multiple) elementary meta-features to form more complex meta-features. This explains the absence of the feature-target identity (and others, see Appendix A) in the above list: it is represented by the conjunction of the feature and target identities. The resulting meta-features enable the model to share weights between, e.g. all 3-grams, all 3-grams that have target `fox`, etc. Although these conjunctions may in theory override C_{ft}/C_{f*} in Eq. (9), keeping the relative frequencies allows us to train the adjustment function on part of the data (see also Section 3.4).

We apply smoothing to all of the count meta-features: since count meta-features of the same order of magnitude carry similar information, we group them so they can share weights. We do this by bucketing the count meta-features 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 (ceilinged) bucket to assure smoother transitions. Both buckets are then weighted according to the $\log_2$ fraction lost by the corresponding rounding operation. Pseudocode for meta-feature extraction and count bucketing is presented in Appendix A.

To control memory usage, we employ a feature hashing technique (Langford et al., 2007; Ganchev and Dredze, 2008) where we store the meta-feature weights in a flat hash table θ of predefined size. Strings are fingerprinted (converted into a byte array, then hashed), counts are hashed, and the resulting integer is mapped to an index in θ by taking its

value modulo the pre-defined $size(\theta)$. We do not prevent collisions, which has the potentially undesirable effect of tying together the weights of different meta-features. However, as was previously observed by Mikolov et al. (2011), when this happens the most frequent meta-feature will dominate the final value after training, which essentially boils down to a form of pruning. Because of this, the model performance does not strongly depend on the size of the hash table. Note that we only apply hashing to the meta-feature weights: the adjusted and raw relative frequencies are stored as SSTables (Sorted String Table).

3.3 Model estimation

Although it is in principle possible to use regularized maximum likelihood to estimate the parameters of the model, a gradient-based approach would end up with parameter updates involving the gradient of the log of Eq. (7) which works out to:

$$\frac{\partial \log P(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} = x_f M_{ft}\left(\frac{y_t}{\hat{y}_{w_k}} - \frac{1}{\sum_{t' \in \mathcal{V}} \hat{y}_{t'}}\right) \quad (11)$$

For the complete derivation, see Appendix B. The problem with this gradient is that we need to sum over the entire vocabulary $\mathcal{V}$ in the denominator. In Eq. (7) we could get away with this by precomputing the row sums, but here the sums change after each update. Instead, we were inspired by Xu et al. (2011) and chose to use an independent binary predictor for each word in the vocabulary during estimation. Our approach however differs from Xu et al. (2011) in that we do not use $|\mathcal{V}|$ Bernoullis, but $|\mathcal{V}|$ Poissons[2], using the fact that for a large number of trials a Bernoulli with small p is well approximated by a Poisson with small λ.

If we consider each $y_{t'}$ in $\mathbf{y}$ to be Poisson distributed with parameter $\hat{y}_{t'}$, the conditional probability $P_{\text{Pois}}(\mathbf{y}|\mathbf{x})$ is given by:

$$P_{\text{Pois}}(\mathbf{y}|\mathbf{x}) = \prod_{t' \in \mathcal{V}} \frac{\hat{y}_{t'}^{y_{t'}} e^{-\hat{y}_{t'}}}{y_{t'}!} = \prod_{t' \in \mathcal{V}} \hat{y}_{t'}^{y_{t'}} e^{-\hat{y}_{t'}} \quad (12)$$

[2]We originally chose Poisson so we could apply the model to tasks with outputs $y_t > 1$. More recent experiments using a multinomial loss can be found in Chelba and Pereira (2016).

and the gradient of the log-probability works out to:

$$\frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} = x_f M_{ft}\left(\frac{y_t}{\hat{y}_{w_k}} - 1\right) \quad (13)$$

For the complete derivation, see Appendix C.

The parameters θ of the adjustment function are learned by maximizing the Poisson log-probability, using stochastic gradient ascent. That is, for each feature-target pair (f,t) we compute the gradient in Eq. (13) and propagate it to the meta-feature weights θ_k by multiplying it with $\partial A(f,t)/\partial\theta_k = h_k$. At the N^{th} occurrence of feature-target pair (f,t), each weight θ_k is updated using the propagated gradient, weighted by a learning rate η:

$$\theta_{k,N} \leftarrow \theta_{k,N-1} + \eta \partial_N(f,t) \quad (14)$$

where $\partial_N(f,t)$ is a short-hand notation for the N^{th} gradient with respect to θ_k.

Rather than using a single fixed learning rate, we use AdaGrad (Duchi, 2011) which uses a separate adaptive learning rate $\eta_{k,N}$ for each weight $\theta_{k,N}$:

$$\eta_{k,N} = \frac{\gamma}{\sqrt{\Delta_0 + \sum_{n=1}^{N} \partial_n(f,t)^2}} \quad (15)$$

where γ is a constant scaling factor for all learning rates and Δ_0 is an initial accumulator constant. Basing the learning rate on historical information tempers the effect of frequently occurring features which keeps the weights small and as such acts as a form of regularization.

3.4 Optimization and leave-one-out training

Each feature-target pair (f,t) constitutes a training example where examples with $y_t = 0$ are called negative and others positive. Using the short-hand notations $T = |\mathcal{T}|$, $F = |\mathcal{F}|$ and $V = |\mathcal{V}|$, this means that the training data consists of approximately $TF(V-1)$ negative and only TF positive training examples. If we examine the two terms of Eq. (13) separately, we see that the first term $x_f M_{ft} \frac{y_t}{\hat{y}_{w_k}}$ depends on y_t which means it becomes zero for all the negative training examples. The second term $-x_f M_{ft}$ however does not depend on y_t and therefore never becomes zero. This also means that the total gradient is never zero and because of

this, the vast amount of updates required for the negative examples makes the update algorithm computationally too expensive.

To speed up the algorithm we use a heuristic that allows us to express the second term as a function of y_t, essentially redistributing the updates for the numerous negative examples to the fewer positive training examples. Appendix D shows that for batch training this has the same effect if run over the entire corpus. We note that for online training this is not strictly correct, since M_{ft} changes after each update. Nonetheless, we found this to yield good results as well as seriously reducing the computational cost. After applying the redistribution, the online gradient that is applied to each training example becomes:

$$\frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} = x_f y_t M_{ft} \left(\frac{1}{\hat{y}_{w_k}} - \frac{C_{f*}}{C_{ft}} \right) \quad (16)$$

which is non-zero only for positive training examples, hence making training independent of the size of the vocabulary.

One practical way to further prevent overfitting and adapt the model to a specific task is to use held-out data, i.e. compute the count matrix $\mathbf{C}$ on the training data and estimate the parameters θ on the held-out data. Unfortunately, since the aggregated gradients in Eq. (16) tie the updates to the counts C_{f*} and C_{ft} in the training data, they can't differentiate between held-out and training data, which means that the meta-feature weights can't be tuned specifically to the held-out data. Experiments in which we tried to use the held-out counts instead did not yield good results, presumably because we are violating the redistribution heuristic.

Rather than adding a regularizer on the meta-feature weights, we instead opted for leave-one-out training. With the notation $A(f,t,C_{f*},C_{ft})$ reflecting the dependence of the adjustment function on feature and feature-target counts, the gradient under leave-one-out training becomes:

$$x_f y_t \left((\frac{1}{\hat{y}^+_{w_k}} - 1) M^+_{ft} - \frac{C_{f*} - C_{ft}}{C_{ft}} M^-_{ft} \right) \quad (17)$$

where M^-_{ft}, M^+_{ft} and $\hat{y}^+_{w_k}$ are defined as follows:

$$M^-_{ft} = e^{A(f,t,C_{f*}-1,C_{ft})} \frac{C_{ft}}{C_{f*}-1}$$
$$M^+_{ft} = e^{A(f,t,C_{f*}-1,C_{ft}-1)} \frac{C_{ft}-1}{C_{f*}-1}$$
$$\hat{y}^+_{w_k} = (\mathbf{x}^T \mathbf{M}^+)_{w_k}$$

The full derivation can be found in Appendix E. We note that in practice, it often suffices to use only a subset of the training examples for leave-one-out training, which has the additional advantage of speeding up training even further.

4 Complexity analysis

Besides their excellent results, RNNs have also been shown to scale well with large amounts of data with regards to memory and accuracy (Williams et al., 2015). Compared to n-gram models which grow huge very quickly with only modest improvements, RNNs take up but a fraction of the memory and exhibit a near linear reduction in $\log$ perplexity with $\log$ training words. Moreover, a larger hidden layer can yield more improvements, whereas n-gram models quickly suffer from data sparsity. The problem with RNNs however is that they are computationally complex which makes training and evaluation slow. A standard Elman network (Elman, 1990) with hidden layer of size H trained on a corpus of size T with vocabulary of size V has complexity

$$IT(H^2 + HV) \quad (18)$$

where I indicates the number of iterations. Several attempts have been made to reduce training time, focusing mostly on reducing the large factors T or V:

- vocabulary shortlisting (Schwenk and Gauvain, 2004)
- subsampling (Schwenk and Gauvain, 2005; Xu et al., 2011)
- class-based (Goodman, 2001b; Morin and Bengio, 2005; Mikolov et al., 2011)
- noise-contrastive estimation (Gutmann and Hyvärinen, 2012; Chen et al., 2015)

However, these techniques either come with a serious performance degradation (Le et al., 2013) or

do not sufficiently speed up training. The class-based implementation for example, still has a training computational complexity of:

$$IT(H^2 + HC + CV_C) \tag{19}$$

where C indicates the number of classes and V_C the variable amount of words in a class. Although this is a significant reduction in complexity, the dominant term ITH^2 is still large. The same applies to noise-contrastive estimation.

As was shown in Mikolov et al. (2011), a Maximum Entropy model can be regarded as a neural network with direct connections for the features, i.e. it has no hidden layers. The model uses the same softmax activation at its output and its complexity therefore also depends on the size of the vocabulary:

$$IT(F_+V) \tag{20}$$

where $F_+ \ll F$ denotes the number of active features. To achieve state-of-the-art results this model is often combined with an RNN, which yields a total complexity of:

$$IT(H^2 + HV + F_+V) \tag{21}$$

The computational complexity for training SNM models on the other hand is independent of V:

$$TF_+ + IT'F_+\Theta_+ \tag{22}$$

where Θ_+ is the number of meta-features for each of the F_+ input features. The first term is related to counting features and feature-target pairs and the second term to training the adjustment model on a subset T' of the training data. If we compare an SNMLM with typical values of $F_+ \approx 100$ and $\Theta_+ < 40$, to the RNNLM configurations with $H = 1024$ in Chelba et al. (2014) and Williams et al. (2015), we find that training comes at a reduced complexity of at least two orders of magnitude.

A even more striking difference in complexity can be seen at test time. Whereas the complexity of a class-based RNN for a single test step is proportional to $H^2+HC+CV_C$, testing SNMLMs is linear in F_+ because of the reasons outlined in Section 3.1.

5 Experiment 1: 1B Word Benchmark

Our first experimental setup used the One Billion Word Benchmark[3] made available by Chelba et al. (2014). It consists of an English training and test set of about 0.8 billion and 159658 tokens, respectively. The vocabulary contains 793471 words and was constructed by discarding all words with count below 3. OOV words are mapped to an `<UNK>` token which is also part of the vocabulary. The OOV rate of the test set is 0.28%. Sentence order is randomized.

All of the described SNM models are initialized with meta-feature weights $\theta_k = 0$ which are updated using AdaGrad with accumulator $\Delta_0 = 1$ and scaling factor $\gamma = 0.02$ over a single epoch of 30M training examples. The hash table for the meta-features was limited to 200M entries as increasing it yielded no significant improvements.

5.1 N-gram experiments

In the first set of experiments, we used all variable-length n-gram features that appeared at least once in the training data up to a given length. This yields at most n active features: one for each m-gram of length $0 \leq m < n$ where $m = 0$ corresponds to an *empty feature* which is always present and produces the unigram distribution. The number of features is smaller than n when the context is shorter than $n-1$ words (near sentence boundaries) and during evaluation where an n-gram that did not occur in the training data is discarded.

When trained using these features, SNMLMs come very close to n-gram models with interpolated Kneser-Ney (KN) smoothing (Kneser and Ney, 1995), where no count cut-off was applied and the discount does not change with the order of the model. Table 1 shows that Katz smoothing (Katz, 1987) performs considerably worse than both SNM and KN. KN and SNM are not very complementary as linear interpolation with weights optimized on the test data only yields an additional perplexity reduction of about 1%. 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 SNMLMs might be better suited to a large number of features.

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

	n-gram order			
Model	5	6	7	8
KN	**67.6**	**64.3**	**63.2**	**62.9**
Katz	79.9	80.5	82.2	83.5
SNM (proposed)	70.8	67.0	65.4	64.8
KN+SNM	66.5	63.0	61.7	61.4

Table 1: Perplexity results on the 1B Word Benchmark for Kneser-Ney (KN), Katz and SNM n-gram models of different order.

Model	PPL
SNM5-skip (no n-grams)	69.8
+ n-grams = SNM5-skip	**54.2**
+ KN5	56.5
SNM5-skip + KN5	**53.6**

Table 2: Perplexity (PPL) results comparing two ways of adding n-grams to a 'pure' skip-gram SNM model (no n-grams): joint modeling (SNM5-skip) and linear interpolation with KN5.

5.2 Integrating skip-gram features

To incorporate skip-gram features, we can either build a 'pure' skip-gram SNMLM that contains no regular n-gram features (except for unigrams) and interpolate this model with KN, or we can build a single SNMLM 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 context words. In particular, we configured the following feature extractors:

- $1 \leq r \leq 3$; $1 \leq s \leq 3$; $1 \leq r + a \leq 4$
- $1 \leq r \leq 2$; $s \geq 4$ (tied); $1 \leq r + a \leq 4$

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)). In addition, both models were interpolated with a KN 5-gram model (KN5).

As can be seen from Table 2, it is better to incorporate all features into one single SNM model than to interpolate with a KN 5-gram model (KN5). This is not surprising as linear interpolation uses a fixed weight for the evaluation of every word sequence, whereas the SNM model applies a variable weight that is dependent both on the context and the target word. Finally, interpolating the all-in-one SNM5-skip with KN5 yields almost no additional gain.

5.3 Skip-gram experiments

The best SNMLM results so far (SNM10-skip) were achieved using 10-grams, together with skip-grams defined by the following feature extractors:

- $s = 1$; $1 \leq r + a \leq 5$
- $r = 1$; $1 \leq s \leq 10$ (tied); $1 \leq r + a \leq 4$

This mixture of rich (large context) short-distance and shallow long-distance features enables the model to achieve state-of-the-art results. Table 3 compares its perplexity to KN5 as well as to the following language models:

- Stupid Backoff LM (SBO) (Brants et al., 2007)
- Hierarchical Softmax Maximum Entropy LM (HSME) (Goodman, 2001b; Morin and Bengio, 2005)
- Recurrent Neural Network LM with Maximum Entropy (RNNME) (Mikolov, 2012)

Describing these models however is beyond the scope of this paper. Instead we refer the reader to Chelba et al. (2014) for a detailed description. The table also lists the number of model parameters, which in the case of SNMLMs consist of the non-zero entries and precomputed row sums of $\mathbf{M}$.

When we compare the perplexity of SNM10-skip with the state-of-the-art RNNLM with 1024 hidden neurons (RNNME-1024), the difference is only 3%. Moreover, this small advantage comes at the cost of increased training and evaluation complexity. Interestingly, when we interpolate the two models, we have an additional gain of 20%, which shows that SNM10-skip and RNNME-1024 are also complementary. As far as we know, the resulting perplexity of 41.3 is already the best ever reported on this corpus, beating the optimized combination of several models, reported in Chelba et al. (2014) by 6%. Finally, interpolation over all models shows that the contribution of other models as well as the additional perplexity reduction of 0.3 is negligible.

5.4 Runtime experiments

In this Section we present actual runtimes to give some idea of how the theoretical complexity analysis of Section 4 translates to a practical application.

Model	Params	PPL
KN5	1.76 B	67.6
SNM5 (proposed)	1.74 B	70.8
SBO	1.13 B	87.9
HSME	6 B	101.3
SNM5-skip (proposed)	62 B	54.2
SNM10-skip (proposed)	33 B	**52.9**
RNNME-256	20 B	58.2
RNNME-512	20 B	54.6
RNNME-1024	20 B	**51.3**
SNM10-skip + RNNME-1024		**41.3**
KN5 + SBO + RNNME-512 + RNNME-1024		43.8
ALL		**41.0**

Table 3: Number of parameters and perplexity (PPL) results on the 1B Word Benchmark for the proposed models, compared to the models in Chelba et al. (2014).

More specifically, we compare the training runtime (in machine hours) of the best SNM model to the best RNN and n-gram models:

- KN5: 28 machine hours
- SNM5: 115 machine hours
- SNM10-skip: 487 machine hours
- RNNME-1024: 5760 machine hours

As these models were trained using different architectures (number of CPUs, type of distributed computing, etc.), a runtime comparison is inherently hard and we would therefore like to stress that these numbers should be taken with a grain of salt. However, based on the order of magnitude we can clearly conclude that SNM's reduced training complexity shown in Section 4 translates to a substantial reduction in training time compared to RNNs. Moreover, the large difference between KN5 and SNM5 suggests that our vanilla implementation can be further improved to achieve even larger speed-ups.

6 Experiment 2: 44M Word Corpus

In addition to the experiments on the One Billion Word Benchmark, we also conducted experiments on a small subset of the LDC English Gigaword corpus. This has the advantage that the experiments are more easily reproducible and, since this corpus preserves the original sentence order, it also allows us to investigate SNM's capabilities of modeling phenomena that cross sentence boundaries.

The corpus is the one used in Tan et al. (2012), which we acquired with the help of the authors and is now available at `http://www.esat.kuleuven.be/psi/spraak/downloads/`[4]. It consists of a training set of 44M tokens, a check set of 1.7M tokens and a test set of 13.7M tokens. The vocabulary contains 56k words which corresponds to an OOV rate of 0.89% and 1.98% for the check and test set, respectively. OOV words are mapped to an `<UNK>` token. The large difference in OOV rate between the check and test set is explained by the fact that the training data and check data are from the same source (Agence France-Presse), whereas the test data is drawn from CNA (Central News Agency of Taiwan) which seems to be out of domain relative to the training data. This discrepancy also shows in the perplexity results, presented in Table 4.

All of the described SNM models are initialized with meta-feature weights $\theta_k = 0$ which are updated using AdaGrad with accumulator $\Delta_0 = 1$ and scaling factor $\gamma = 0.02$ over a single epoch of 10M training examples. The hash table for the meta-features was limited to 10M entries as increasing it yielded no significant improvements.

With regards to n-gram modeling, the results are analogous to the 1B word experiment: SNM5 is close to KN5; both outperform Katz5 by a large mar-

[4]In order to comply with the LDC license, the data was encrypted using a key derived from the original data.

gin. This is the case for the check set and the test set.

Tan et al. (2012) showed that by crossing sentence boundaries, perplexities can be drastically reduced. Although they did not publish any results on the check set, their mixture of n-gram, syntactic language models and topic models achieved a perplexity of 176 on the test set, a 23% relative reduction compared to KN5. A similar observation was made for the SNM models by adding a feature extractor (r, s, a) analogous to regular skip-grams, but with s now denoting the number of skipped sentence boundaries </S> instead of words. Adding skip-</S> features with $r + a = 4$, $1 \leq r \leq 2$ and $1 \leq s \leq 10$, yielded an even larger reduction of 26% than the one reported by Tan et al. (2012). On the check set we observed a 25% reduction.

The RNNME results are achieved with a setup similar to the one in Chelba et al. (2014). The main differences are related to the ME features (3-grams only instead of 10-grams and bag-of-words features) and the number of iterations over the training data (20 epochs instead of 10). These choices are related to the size of the training data. It can be seen from Table 4 that the best RNNME model outperforms the best SNM model by 13% on the check set. The out-of-domain test set shows that due to its compactness, RNNME is better suited for LM adaptation.

	PPL	
Model	check	test
KN5	104.7	229.0
Katz5	124.1	292.6
SNM5 (proposed)	108.3	232.3
SLM	-	279
n-gram/SLM	-	243.0
n-gram/PLSA	-	196.0
n-gram/SLM/PLSA	-	176.0
SNM5-skip (proposed)	89.5	198.4
SNM10-skip (proposed)	87.5	195.3
SNM5-skip-</S> (proposed)	79.5	176.0
SNM10-skip-</S> (proposed)	78.4	174.0
RNNME-512	70.8	136.7
RNNME-1024	**68.0**	**133.3**

Table 4: Perplexity (PPL) results on the 44M corpus. On the small check set, SNM outperforms a mixture of n-gram, syntactic language models (SLM) and topic models (PLSA), but RNNME performs best. The out-of-domain test set shows that due to its compactness, RNNME is better suited for LM adaptation.

7 Conclusions and Future Work

We have presented SNM, a novel probability estimation technique for language modeling that can efficiently incorporate arbitrary features. A first set of empirical evaluations on two data sets shows that SNM n-gram LMs perform almost as well as the well-established KN models. When we add skip-gram features, the models are able to match the state-of-the-art RNNLMs on the One Billion Word Benchmark (Chelba et al., 2014). Combining the two modeling techniques yields the best known result on the benchmark which shows that the two models are complementary.

On a smaller subset of the LDC English Gigaword corpus, SNMLMs are able to exploit cross-sentence dependencies and outperform a mixture of n-gram models, syntactic language models and topic models. Although RNNLMs still outperform SNM by 13% on this corpus, a complexity analysis and measured runtimes show that the RNN comes at an increased training and evaluation time.

We conclude that the computational advantages of SNMLMs over both Maximum Entropy and RNN estimation promise an approach that has large flexibility in combining arbitrary features effectively and yet scales gracefully to large amounts of data.

Future work includes exploring richer features similar to Goodman (2001a), as well as richer meta-features in the adjustment model. A comparison of SNM models with Maximum Entropy at feature parity is also planned. One additional idea was pointed out to us by action editor Jason Eisner. Rather than using one-hot target vectors which emphasizes fit, it is possible to use low-dimensional word embeddings. This would most likely yield a smaller model with improved generalization.

8 Acknowledgments

We would like to thank Mike Schuster for his help with training and evaluating the RNN models. Thanks also go to Tan et al. who provided us with the 44M word corpus and to action editor Jason Eisner and the anonymous reviewers whose helpful comments certainly improved the quality of the paper.

Appendix A Meta-feature Extraction Pseudocode

```
// Meta-features are represented as tuples (hash_value, weight).
// New meta-features are either added (metafeatures.Add(mf_new)) or
// joint (metafeatures.Join(mf_new)) with the existing meta-features.
// Strings are fingerprinted, counts are hashed.
function COMPUTE_METAFEATURES(FeatureTargetPair pair)
    // feature-related meta-features
    metafeatures = {}
    metafeatures.Add(Fingerprint(pair.feature_identity), 1.0)
    metafeatures.Add(Fingerprint(pair.feature_type), 1.0)
    log_count = log(pair.feature_count) / log(2)
    bucket1 = floor(log_count)
    bucket2 = ceil(log_count)
    weight1 = bucket2 - log_count
    weight2 = log_count - bucket1
    metafeatures.Add(Hash(bucket1), weight1)
    metafeatures.Add(Hash(bucket2), weight2)

    // target-related meta-features
    metafeatures.Join(Fingerprint(pair.target_identity), 1.0)

    // feature-target-related meta-features
    log_count = log(pair.feature_target_count) / log(2)
    bucket1 = floor(log_count)
    bucket2 = ceil(log_count)
    weight1 = bucket2 - log_count
    weight2 = log_count - bucket1
    metafeatures.Join(Hash(bucket1), weight1)
    metafeatures.Join(Hash(bucket2), weight2)

    return metafeatures
```

Appendix B Multinomial Gradient

$$
\begin{aligned}
\frac{\partial \log P_{\text{multi}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} &= \left(\frac{\partial \log(\mathbf{x^T M})_{w_k}}{\partial M_{ft}} - \frac{\partial \log \sum_{t' \in \mathcal{V}} (\mathbf{x^T M})_{t'}}{\partial M_{ft}} \right) \frac{\partial M_{ft}}{\partial A_{ft}} \\
&= \left(\frac{1}{(\mathbf{x^T M})_{w_k}} \frac{\partial (\mathbf{x^T M})_{w_k}}{\partial M_{ft}} - \frac{1}{\sum_{t' \in \mathcal{V}} (\mathbf{x^T M})_{t'}} \frac{\partial \sum_{t' \in \mathcal{V}} (\mathbf{x^T M})_{t'}}{\partial M_{ft}} \right) M_{ft} \\
&= \left(\frac{x_f y_t}{\hat{y}_{w_k}} - \frac{x_f}{\sum_{t' \in \mathcal{V}} (\mathbf{x^T M})_{t'}} \right) M_{ft} \\
&= x_f M_{ft} \left(\frac{y_t}{\hat{y}_{w_k}} - \frac{1}{\sum_{t' \in \mathcal{V}} \hat{y}_{t'}} \right)
\end{aligned}
$$

Appendix C Poisson Gradient

$$\begin{aligned}\frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} &= \left(\frac{\partial \sum_{t' \in \mathcal{V}} y_{t'} \log(\mathbf{x^T M})_{t'}}{\partial M_{ft}} - \frac{\partial \sum_{t' \in \mathcal{V}} (\mathbf{x^T M})_{t'}}{\partial M_{ft}}\right) \frac{\partial M_{ft}}{\partial A(f,t)} \\ &= \left(\frac{1}{(\mathbf{x^T M})_{w_k}} \frac{\partial (\mathbf{x^T M})_{w_k}}{\partial M_{ft}} - x_f\right) M_{ft} \\ &= x_f M_{ft} \left(\frac{y_t}{\hat{y}_{w_k}} - 1\right)\end{aligned}$$

Appendix D Distributing Negative Updates

Over the entire training set, adding $\frac{C_{f*}}{C_{ft}} M_{ft}$ once on the target t that occurs with feature f amounts to the same as traversing all targets t' that co-occur with f in the training set and adding the term M_{ft} to each:

$$M_{ft} \sum_{(f,t') \in \mathcal{T}} x_f = \frac{C_{f*}}{C_{ft}} M_{ft} C_{ft} = \frac{C_{f*}}{C_{ft}} M_{ft} \sum_{(f,t') \in \mathcal{T}} x_f y_{t'}$$

Applying this to the second term of the Poisson gradient, we get:

$$\frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} = x_f M_{ft} \frac{y_t}{\hat{y}_{w_k}} - x_f M_{ft} = x_f M_{ft} \frac{y_t}{\hat{y}_{w_k}} - x_f y_t M_{ft} \frac{C_{f*}}{C_{ft}} = x_f y_t M_{ft} \left(\frac{1}{\hat{y}_{w_k}} - \frac{C_{f*}}{C_{ft}}\right)$$

Appendix E Leave-one-out Training

In leave-one-out training we exclude the event that generates the gradients from the counts used to compute those gradients. More specifically, for each training example (f,t) we let:

$$\begin{aligned} C_{f*} &\leftarrow C_{f*} - 1 && \text{if } x_f = 1 \\ C_{ft} &\leftarrow C_{ft} - 1 && \text{if } x_f = 1, y_t = 1 \end{aligned}$$

which means that the gradients for the positive and the negative examples are changed in a different way. Since Eq. (16) expresses the general update rule for both type of examples, we first have to separate it into updates for negative and positive examples and then adapt accordingly.

In particular, the second term of Eq. (16), i.e. $-x_f y_t M_{ft} \frac{C_{f*}}{C_{ft}}$ is a distribution of $C_{f*} - C_{ft}$ negative and C_{ft} positive updates over C_{ft} positive examples:

$$-x_f y_t M_{ft} \frac{C_{f*}}{C_{ft}} = -x_f y_t M_{ft} \left(\frac{C_{f*} - C_{ft}}{C_{ft}} + \frac{C_{ft}}{C_{ft}}\right) = -x_f y_t M_{ft} \frac{C_{f*} - C_{ft}}{C_{ft}} - x_f y_t M_{ft}$$

Furthermore, recall that the first term of Eq. (16), i.e. $\frac{x_f y_t M_{ft}}{\hat{y}_{w_k}}$ is non-zero only for positive examples, so it can be added to the positive updates. We can then apply leave-one-out to positive and negative updates separately, ending up with:

$$\frac{\partial \log P_{\text{Pois}}(\mathbf{y}|\mathbf{x})}{\partial A(f,t)} = x_f y_t \left((\frac{1}{\hat{y}^+_{w_k}} - 1) M^+_{ft} - \frac{C_{f*} - C_{ft}}{C_{ft}} M^-_{ft}\right)$$

where M^-_{ft}, M^+_{ft} and $\hat{y}^+_{w_k}$ are defined as follows:

$$\begin{aligned} M^-_{ft} &= e^{A(f,t,C_{f*}-1,C_{ft})} \frac{C_{ft}}{C_{f*} - 1} \\ M^+_{ft} &= e^{A(f,t,C_{f*}-1,C_{ft}-1)} \frac{C_{ft} - 1}{C_{f*} - 1} \\ \hat{y}^+_{w_k} &= (\mathbf{x}^T \mathbf{M}^+)_{w_k} \end{aligned}$$

References

Jerome Bellegarda. 2000. Exploiting Latent Semantic Information in Statistical Language Modeling. *Proceedings of the IEEE*, 88(8), 1279–1296.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3, 1137–1155.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large Language Models in Machine Translation. *Proceedings of EMNLP*, 858–867.

Ciprian Chelba and Frederick Jelinek. 2000. Structured Language Modeling. *Computer Speech and Language*, 14(4), 283–332.

Ciprian Chelba, Tomáš Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. *Proceedings of Interspeech*, 2635–2639.

Ciprian Chelba and Fernando Pereira. 2016. Multinomial Loss on Held-out Data for the Sparse Nonnegative Matrix Language Model. arXiv:1511.01574 [cs.CL].

Stanley F. Chen, Kristie Seymore, and Ronald Rosenfeld. 1998. Topic Adaptation for Language Modeling Using Unnormalized Exponential Models. *Proceedings of ICASSP*, 681–684.

Xie Chen, Xunying Liu, Mark Gales and Phil Woodland. 2015. Recurrent Neural Network Language Model Training with Noise Contrastive Estimation for Speech Recognition. *Proceedings of ICASSP*, 5411–5415.

John Duchi, Elad Hazan and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12, 2121–2159.

Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive Science*, 14(2), 179–211.

Ahmad Emami. 2006. A Neural Syntactic Language Model. *Ph.D. Thesis, Johns Hopkins University*.

Joshua T. Goodman. 2001a. A Bit of Progress in Language Modeling, Extended Version. *Technical Report MSR-TR-2001-72*.

Joshua T. Goodman. 2001b. Classes for Fast Maximum Entropy Training. *Proceedings of ICASSP*, 561–564.

Michael Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive Estimation of Unnormalized Statistical Models, with Applications to Natural Image Statistics. *Journal of Machine Learning Research*, 13(1), 307–361.

Xuedong Huang, Fileno Alleva, Mei-Yuh Hwang, and Ronald Rosenfeld. 1993. An Overview of the SPHINX-II Speech Recognition System. *Computer Speech and Language*, 2, 137–148.

Slava M. Katz. 1987. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3), 400–401.

Dietrich Klakow. 1998. Log-linear Interpolation of Language Models. *Proceedings of ICSLP*, 1695–1698.

Reinhard Kneser and Hermann Ney. 1995. Improved Backing-Off for M-Gram Language Modeling. *Proceedings of ICASSP*, 181–184.

John Langford, Lihong Li and Alex Strehl. 2007. Vowpal Wabbit Online Learning Project. `http://hunch.net/?p=309`.

Kuzman Ganchev and Mark Dredze. 2008. Small Statistical Models by Random Feature Mixing. *Proceedings of the ACL-2008 Workshop on Mobile Language Processing*, 19–20.

Hai-Son Le, Ilya Oparin, Alexandre Allauzen, Jean-Luc Gauvain and François Yvon. 2013. Structured Output Layer Neural Network Language Models for Speech Recognition. *IEEE Transactions on Audio, Speech & Language Processing*, 21, 195–204.

Su-in Lee, Vassil Chatalbashev, David Vickrey and Daphne Koller. 2007 Learning a Meta-level Prior for Feature Relevance from Multiple Related Tasks. *Proceedings of ICML*, 489–496.

Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukás Burget and Jan Cernocký. 2011. Strategies for Training Large Scale Neural Network Language Models. *Proceedings of ASRU*, 196–201.

Tomáš Mikolov. 2012. Statistical Language Models Based on Neural Networks. *Ph.D. Thesis, Brno University of Technology*.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical Probabilistic Neural Network Language Model. *Proceedings of AISTATS*, 246–252.

Hermann Ney, Ute Essen, and Reinhard Kneser. 1994. On Structuring Probabilistic Dependences in Stochastic Language Modeling. *Computer Speech and Language*, 8, 1–38.

Rene Pickhardt, Thomas Gottron, Martin Körner, Paul G. Wagner, Till Speicher, and Steffen Staab. 2014. A Generalized Language Model as the Combination of Skipped n-grams and Modified Kneser-Ney Smoothing. *Proceedings of ACL*, 1145–1154.

Ronald Rosenfeld. 1994. Adaptive Statistical Language Modeling: A Maximum Entropy Approach. *Ph.D. Thesis, Carnegie Mellon University*.

Ronald Rosenfeld, Stanley F. Chen, and Xiaojin Zhu. 2001. Whole-sentence Exponential Language Models: a Vehicle for Linguistic-Statistical Integration. *Computer Speech and Language*, 15, 55–73.

Holger Schwenk and Jean-Luc Gauvain. 2004. Neural Network Language Models for Conversational Speech Recognition. *Proceedings of ICSLP*, 1215-1218.

Holger Schwenk and Jean-Luc Gauvain. 2005. Training Neural Network Language Models On Very Large Corpora. *Proceedings of EMNLP*, 201–208.

Holger Schwenk. 2007. Continuous Space Language Models. *Computer Speech and Language*, 21, 492–518.

Mittul Singh and Dietrich Klakow. 2013. Comparing RNNs and Log-linear Interpolation of Improved Skip-model on Four Babel Languages: Cantonese, Pashto, Tagalog, Turkish. *Proceedings of ICASSP*, 8416–8420.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM Neural Networks for Language Modeling. *Proceedings of Interspeech*, 194–197.

Ming Tan, Wenli Zhou, Lei Zheng and Shaojun Wang. 2012. A Scalable Distributed Syntactic, Semantic, and Lexical Language Model. *Computational Linguistics*, 38(3), 631–671.

Puyang Xu, Asela Gunawardana, and Sanjeev Khudanpur. 2011. Efficient Subsampling for Training Complex Language Models. *Proceedings of EMNLP*, 1128–1136.

Will Williams, Niranjani Prasad, David Mrva, Tom Ash and Tony Robinson. 2015. Scaling Recurrent Neural Network Language Models. *Proceedings of ICASSP*, 5391–5395.