Sparse Non-negative Matrix Language Modeling For Skip-grams

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

[1]Google, Inc., 1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
[2]Dept. ESAT, KU Leuven, Kasteelpark Arenberg 10
B-3001 Leuven, Belgium
`{noam,jpeleman,ciprianchelba}@google.com`
`joris.pelemans@esat.kuleuven.be`

Abstract

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 these techniques on the One Billion Word Benchmark [3] shows that with skip-gram features SNMLMs 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 RNNLM 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.

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

1. Introduction

Recently, neural network (NN) smoothing [1], [5], [18], and in particular recurrent neural networks (RNNs) [12], [20] have shown excellent performance in language modeling [3]. Their excellent performance is attributed to a combination of leveraging long-distance context, and training a vector representation for words. Although these models are currently the state of the art, they do not scale well to very large amounts of data with training times in the order of weeks.

Another way of leveraging long distance context is to use skip-grams [8], [14], [17]. Skip-grams are a generalization of regular n-grams where in addition to allowing adjacent word sequences, words are also allowed to be skipped, thus covering a longer context without being hampered as much by data sparsity. Previous work has revealed that training a model with skip-gram features is able to compete with neural network-based models [19].

In order to build a good probability estimate for a target word using skip-gram features we need a way of combining an arbitrary number of these features, which do not fall into a simple hierarchy like regular n-gram features. In this paper we propose a simple, yet novel approach called Sparse Non-negative Matrix (SNM) estimation 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. We evaluate the approach by comparing perplexity results with several other popular language models on the One Billion Word Benchmark [3] and show that SNMLMs using skip-gram features are able to match the state-of-the-art RNNLMs; combining the two modeling techniques yields the best known result on the benchmark.

In the remainder of this paper we introduce skip-gram language modeling (Section 2), describe the SNMLM paradigm (Section 3), evaluate it experimentally (Section 4) and discuss some related work (Section 5). We end with conclusions and future work in Section 6.

2. Skip-gram Language Modeling

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 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. 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 Language Modeling

In this section we describe our new paradigm without working out all the derivations. The interested reader can find these in [15].

3.1. Model definition

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

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

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

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

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

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

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

3.2. Adjustment function and metafeatures

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

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

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

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

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

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

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

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

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

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

3.3. Model estimation

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

In [15] we suggested a loss function based on the Poisson distribution: we consider each t_j in $\mathbf{t}$ to be Poisson distributed with parameter y_j. The conditional probability of $P_{Poisson}(\mathbf{t}|\mathbf{f})$ then is:

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

and the corresponding Poisson loss function is:

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

where we dropped the last term, since t_j is binary-valued[1]. Although this choice is not obvious in the context of language modeling, it is well suited to gradient-based optimization and, as we will see, the experimental results are in fact excellent. Moreover, the Poisson loss also lends itself nicely for multiple target prediction which might be useful in e.g. subword modeling.

---

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

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

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

For the complete derivation we refer to [15].

We then use the Adagrad [4] adaptive learning rate procedure to update the metafeature weights. Rather than using a single fixed learning rate, Adagrad uses a separate adaptive learning rate $\eta_{k,N}(i,j)$ for each weight $\theta_k(i,j)$ at the Nth occurrence of (f_i, t_j):

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

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

3.4. Optimization

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

However, since the first term does not depend on y_j, we are able to distribute the updates for the negative examples over the positive ones by adding in gradients for a fraction of the events where $f_i = 1$, but $t_j = 0$. In particular, instead of adding the term $f_i M_{ij}$, we add $f_i t_j \frac{C_{i*}}{C_{ij}} M_{ij}$ which lets us update the gradient only on positive examples. This is based on the observation that, over the entire training set, it amounts to the same thing. For the complete derivation we refer to [15].

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

$$\frac{\partial(L_{Poisson}(\mathbf{Mf}, \mathbf{t}))}{\partial(A(i,j))} = f_i t_j \frac{C_{i*} - C_{ij}}{C_{ij}} M_{ij} + f_i t_j (1 - \frac{1}{y_j}) M_{ij} \tag{9}$$

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

3.5. Leave-one-out training

A model with a huge amount of parameters is prone to overfitting the training data. The preferred way to deal with this issue is to use held-out data to estimate the parameters. Unfortunately the aggregated gradients in Eq. (9) do not allow us to use additional data to train the adjustment function, since they tie the update computation to the relative frequencies $\frac{C_{i*}}{C_{ij}}$ in the training data. Instead, we have to resort to leave-one-out training to prevent the model from overfitting. We do this by excluding the event that generates the gradients from the counts used to compute those gradients. So, for each positive example (f_i, t_j) of each event $e = (\mathbf{f}, \mathbf{t})$, we compute the gradient, excluding 1 from C_{i*} and C_{ij}. For the gradients of the negative examples on the other hand we only exclude 1 from C_{i*}, because we did not observe t_j. In order to keep the aggregate computation of the gradients for the negative examples, we distribute them uniformly over all the positive examples with the same feature; each of the C_{ij} positive examples will then compute the gradient of $\frac{C_{i*} - C_{ij}}{C_{ij}}$ negative examples.

Model	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 1: 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.

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

$$\begin{aligned} &\frac{\partial(L_{Poisson}(\mathbf{Mf}, \mathbf{t}))}{\partial(A(i,j))} \\ &= f_i t_j \frac{C_{i*} - C_{ij}}{C_{ij}} e^{A(i,j,C_{i*}-1,C_{ij})} \frac{C_{ij}}{C_{i*} - 1} \\ &+ f_i t_j (1 - \frac{1}{y'_j}) e^{A(i,j,C_{i*}-1,C_{ij}-1)} \frac{C_{ij} - 1}{C_{i*} - 1} \end{aligned} \tag{10}$$

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

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

4. Experiments

Our experimental setup used the One Billion Word Benchmark corpus[2] made available by [3].

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

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

In a first set of experiments, we investigate how to best combine skip-gram features with regular n-gram features. All of the mentioned n-gram models are trained using interpolated Kneser-Ney (KN) smoothing [11] without count cut-off where the discount does not change with the order of the model.

To 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

---

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

Model	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
RNNME-256	20 B	58.2		0.00	0.00
RNNME-512	20 B	54.6		0.13	0.07
RNNME-1024	20 B	51.3	0.6	0.61	0.53
SNM10-skip+RNNME-1024			41.3		
Previous best				43.8	
ALL					41.0

Table 2: Number of parameters (in billions) and perplexity results for all the investigated models, as well as interpolation results and weights.

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 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 1, 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. 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.

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. Table 2 compares its perplexity to KN5 as well as to the following language models:

- Hierarchical Softmax Maximum Entropy LM (HSME) [7], [13]
- Stupid Backoff LM (SBO) [2]
- Recurrent Neural Network LM with Maximum Entropy (RNNME) [12]

Describing these models however is beyond the scope of this paper. Instead we refer the reader to [3] which contains a detailed description of all the models in Table 2.

When we compare the perplexity of SNM10-skip with the state-of-the-art RNNLM with 1024 neurons in the hidden layer (RNNME-1024), 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%.

Finally, when we optimize interpolation weights over all models, the contribution of the extra models as well as the perplexity reduction is negligible.

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) [17], or exponential models.

We are not the first ones to highlight the effectiveness of skip-grams at capturing dependencies across longer contexts, similar to RNNLMs; previous such results were reported in [19]. Recently, [16] also showed that a backoff generalization using single skips yields significant perplexity reductions. We note that our SNM models are trained using both single and longer skips and that our method of estimating the feature weights is, as far as we know, completely original.

The speed-ups to ME, and RNN LM training provided by hierarchically predicting words at the output layer [7], and subsampling [21] 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. (10) for the much smaller adjustment function eliminate the dependency on the vocabulary size.

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 [3] shows that with skip-gram features SNMLMs are able to match the state-of-the-art RNN LMs; combining the two modeling techniques yields the best known result on the benchmark.

The computational advantages of SNMLMs over both Maximum Entropy and RNNLM estimation promise an approach that has the same flexibility in combining arbitrary features effectively and yet should scale to very large amounts of data as gracefully as n-gram LMs do.

Future work items include model pruning, exploring richer features similar to [6], 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.

7. References

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

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

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

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

[5] Ahmad Emami. “A Neural Syntactic Language Model,” *Ph.D. Thesis, Johns Hopkins University*, 2006.

[6] Joshua T. Goodman. “A Bit of Progress in Language Modeling, Extended Version,” *Technical Report MSR-TR-2001-72*, 2001.

[7] Joshua T. Goodman. “Classes for Fast Maximum Entropy Training,” *Proceedings of ICASSP*, pp. 561–564, 2001.

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

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

[10] Slava M. Katz. “Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer,” *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-35, 3, pp. 400–401, 1987.

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

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

[13] Frederic Morin and Yoshua Bengio. “Hierarchical Probabilistic Neural Network Language Model,” *Proceedings of AISTATS*, pp. 246–252, 2005.

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

[15] Noam Shazeer, Joris Pelemans and Ciprian Chelba. “Skip-gram Language Modeling Using Sparse Non-negative Matrix Probability Estimation,” *CoRR*, abs/1412.1454, 2014. [Online]. Available: http://arxiv.org/abs/1412.1454.

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

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

[18] Holger Schwenk. “Continuous Space Language Models,” *Computer Speech and Language*, 21, pp. 492–518, 2007.

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

[20] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. “LSTM Neural Networks for Language Modeling,” *Proceedings of Interspeech*, pp. 194–197, 2012.

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