Simple, Fast Noise-Contrastive Estimation for Large RNN Vocabularies

Barret Zoph*, **Ashish Vaswani***, **Jonathan May, and Kevin Knight**
Information Sciences Institute
Department of Computer Science
University of Southern California
{zoph, avaswani, jonmay, knight}@isi.edu

Abstract

We present a simple algorithm to efficiently train language models with noise-contrastive estimation (NCE) on graphics processing units (GPUs). Our NCE-trained language models achieve significantly lower perplexity on the One Billion Word Benchmark language modeling challenge, and contain one sixth of the parameters in the best single model in Chelba et al. (2013). When incorporated into a strong Arabic-English machine translation system they give a strong boost in translation quality. We release a toolkit so that others may also train large-scale, large vocabulary LSTM language models with NCE, parallelizing computation across multiple GPUs.

1 Introduction

Language models are used to compute probabilities of sequences of words. They are crucial for good performance in tasks like machine translation, speech recognition, and spelling correction. They can be classified into two categories: count-based and continuous-space language models. The language modeling literature abounds with successful approaches for learning-count based language models: modified Kneser-Ney smoothing, Jelinek-Mercer smoothing, etc. In recent years, continuous-space language models such as feed-forward neural probabilistic language models (NPLMs) and recurrent neural network language models (RNNs)[1] have outperformed their count-based counterparts (Chelba et al., 2013; Zaremba et al., 2014; Mikolov, 2012). RNNs are more powerful than n-gram language models, as they can exploit longer word contexts to predict words. Long short-term memory language models (LSTMs) are a class of RNNs that have been designed to model long histories and are easier to train than standard RNNs. LSTMs are currently the best performing language models on the Penn Treebank (PTB) dataset (Zaremba et al., 2014).

The most common method for training LSTMs, maximum likelihood estimation (MLE), is prohibitively expensive for large vocabularies, as it involves time-intensive matrix-matrix multiplications. Noise-contrastive estimation (NCE) has been a successful alternative to train continuous space language models with large vocabularies (Mnih and Teh, 2012; Vaswani et al., 2013). However, NCE in its standard form is not suitable for GPUs, as the computations are not amenable to dense matrix operations. In this paper, we present a natural modification to the NCE objective function for language modeling that allows a very efficient GPU implementation. Using our new objective, we train large multi-layer LSTMs on the One Billion Word benchmark (Chelba et al., 2013), with its full 780k word vocabulary. We achieve significantly lower perplexities with a single model, while using only a sixth of the parameters of a very strong baseline model (Chelba et al., 2013). We release our toolkit[2] to allow researchers to train large-scale, large-vocabulary LSTMs with NCE. The contributions in this paper are the following:

*Equal contribution.

[1]Henceforth we will use terms like "RNN" and "LSTM" with the understanding that we are referring to language models that use these formalisms

[2]www.github.com/isi-nlp/Zoph_RNN

Proceedings of NAACL-HLT 2016, pages 1217–1222,
San Diego, California, June 12-17, 2016.

- A fast and simple approach for handling large vocabularies effectively on the GPU.
- Significantly improved perplexities (43.2) on the One Billion Word benchmark over Chelba et al. (2013)
- Extrinsic machine translation improvement over a strong baseline.
- Fast decoding times because in practice there is no need to normalize.

2 Long Short Term Memory Language Models

In recent years, LSTMs (Hochreiter and Schmidhuber, 1997) have achieved state-of-the-art performance in many natural language tasks such as language modeling (Zaremba et al., 2014) and statistical machine translation (Sutskever et al., 2014; Luong et al., 2015). LSTMs were designed to have longer memories than standard RNNs, allowing them to exploit more context to make predictions. To compute word probabilities, the LSTM reads words left-to-right, updating its memory after each word and producing a hidden state h, which summarizes all of the history. For details on the architecture and computations of the LSTM, the reader can refer to (Zaremba et al., 2014). In this model the probability of word w given history $\mathbf{u}$ is

$$P(w \mid \mathbf{u}) = \frac{p(w \mid \mathbf{u})}{\mathbf{Z}(\mathbf{u})}, \tag{1}$$

where $p(w \mid \mathbf{u}) = \exp D_w h^T + b_w$ is an unnormalized probability. D_w and b_w are the output word embedding and biases respectively, which are learned during training. The normalization constant is $\mathbf{Z}(u) = \sum_{w \in V} p(w \mid \mathbf{u})$, and V is the vocabulary.

3 Noise Contrastive Estimation For Training Neural Language Models

The standard approach for estimating neural language models is maximum liklelihood estimation (MLE), where we learn the parameters θ^* that maximize the likelihood of the training data,

$$\theta^* = arg\ max_\theta \sum_{w,\mathbf{u}} \log P(w \mid \mathbf{u}; \theta) \tag{2}$$

However, for each training instance, gradient-based approaches for MLE require a summation over all units in the output layer, one for each word in V. This makes MLE training infeasible for large vocabularies.

Noise-contrastive estimation (NCE) (Gutmann and Hyvärinen, 2010) has been successfully adopted for training neural language models with large vocabularies (Mnih and Teh, 2012; Vaswani et al., 2013; Baltescu and Blunsom, 2014; Williams et al., 2015). NCE avoids repeated summations by training the model to correctly classify between generated noise samples and words observed in the training data. For each training pair $\mathbf{u}_i, w_i$, we generate k noise samples, $\bar{w}_{i1} \ldots, \bar{w}_{ik}$ from a noise distribution $q(w)$, which is known. We label $\mathbf{u}_i, w_i$ as true ($C = 1$), and all $\mathbf{u}_i, \bar{w}_{ik}$ as false ($C = 0$). We then train the parameters to maximize the binary classification log likelihood,

$$\mathbf{L} = \sum_i \Big(\log P(C = 1 \mid \mathbf{u}_i, w_i) + \sum_k \log P(C = 0 \mid \mathbf{u}, \bar{w}_{ik}) \Big), \tag{3}$$

where

$$P(C = 1 \mid w, \mathbf{u}) = \frac{\frac{p(w|\mathbf{u})}{\mathbf{Z}(\mathbf{u})}}{\frac{p(w|\mathbf{u})}{\mathbf{Z}(\mathbf{u})} + k \cdot q(w)}, \tag{4}$$

and $P(C = 0 \mid w, \mathbf{u}) = 1 - P(C = 1 \mid w, \mathbf{u})$.

We learn parameters to maximize this objective with gradient ascent. In NCE, we treat $\mathbf{Z}(\mathbf{u})$ as another parameter and learn its estimate along with the rest of the parameters. Following Mnih and Teh (2012) and Vaswani et al. (2013), we freeze $\mathbf{Z}(\mathbf{u})$ to 1 and the model learns to approximately satisfy the constraint. In practice, we find that our mean for $\mathbf{Z}(\mathbf{u})$ is very close to 1 and the variance is quite small (Section 6). For each training instance, we compute gradients for the observed word and each noise word, reducing the time complexity from $\mathcal{O}(|V|)$ for MLE to $\mathcal{O}(k)$. However, unlike MLE, since we typically sample different noise samples for each training instance, our gradient computations for the NCE objective are not amenable to dense matrix operations, making it difficult to benefit from fast dense matrix arithmetic on GPUs. In this paper, we present

a simple solution to this problem: sharing the noise samples across all the training instances in a minibatch. Sharing noise samples allows us to describe NCE gradients with dense matrix operations, and implement them easily on the GPU. In the next section, we describe our NCE implementation on the GPU with shared noise samples.

4 Our NCE Modification

In typical Noise-Contrastive Estimation, the objective function requires noise samples coming from some distribution (in our case, the uniform distribution). The objective function for NCE is shown above in Equation 3, where we must calculate $P(C = 1 \mid w, \mathbf{u})$ for the true word and the noise samples generated. There are three components to this calculation: $p(w \mid \mathbf{u})$, $\mathbf{Z}(\mathbf{u})$, and $k \cdot q(w)$. In NCE we fix $\mathbf{Z}(\mathbf{u})$ to be one, so we only need to calculate $p(w \mid \mathbf{u})$ and $k \cdot q(w)$. The term $k \cdot q(w)$ is simply an $O(1)$ lookup, so the only costly operation is computing $p(w \mid \mathbf{u})$ for all k noise samples and the true word. The operation to compute $p(w \mid \mathbf{u})$ for a single word w is $\exp D_w h^T + b_w$ where D_w and b_w represent the word embedding and bias corresponding to the word we are computing it for.

The main cost in calculating the NCE objective function is computing $\exp D_w h^T + b_w$, where in normal NCE a dense matrix multiplication cannot be done. The reason is that the noise samples generated per training example will be different. Therefore, when we parallelize our algorithm by running multiple training examples in parallel (a minibatch), the D_w we need are different per h^T that we are running in parallel. If a constraint is set such that the noise samples must be the same across all training examples in the minibatch, then a matrix multiplication can be done to compute $\exp D_w h^T + b_w$ for all words across that minibatch. This matrix multiplication is Dh_M^T, where h_M^T is now a matrix of all the hidden states being used in parallel, whereas before h_T was just a vector. Additionally, D is the matrix of word embedding for the samples that are being shared across a minibatch. Before, this was not possible as the D matrix would have to be much larger, being (minibatch size) $\cdot$ (k + 1) in size, making the algorithm run much slower. An alternative is to not restrict the noise samples to be the same, but then we must use a sparse matrix multiplication as in Williams et al. (2015), which is neither as fast nor as easy to implement. A comparison between these two approaches is shown in Figure 1. We find that sharing noise samples across the minibatch gives us significant speedups over a baseline of using different noise samples for each word in the minibatch. These timings were calculated for a single layer LSTM with 1000 hidden units, a vocabulary of $350k$, and a minibatch of 128. Not surprisingly, MLE is quite expensive, limiting it's use for large vocabularies. Additionally, the memory requirements for NCE are much lower than MLE, since we do not need to store the gradient which has the same size as the output embedding matrix. For this MLE run, we had to distribute the computation across two GPUs because of memory limitations.

5 Experiments

We conducted two series of experiments to validate the efficiency of our approach and the quality of the models we learned using it: An *intrinsic* study of language model perplexity using the standard One Billion Word benchmark (Chelba et al., 2013) and an *extrinsic* end-to-end statistical machine translation task that uses an LSTM as one of several feature functions in re-ranking. Both experiments achieve excellent results.

5.1 Language Modeling

For our language modeling experiment we use the One Billion Word benchmark proposed by Chelba et al. (2013). In this task there are roughly 0.8 billion words of training data. We use perplexity to evaluate the quality of language models we train on this data. We train an LSTM with 4 layers, where each layer has 2048 hidden units, with a target vocabulary size of 793,471. For training, we also use dropout to prevent overfitting. We follow Zaremba et al. (2014) for dropout locations, and we use a dropout rate of 0.2. The training is parallelized across 4 GPUs, such that each layer lies on its own GPU and communicates its activations to the next layer once it finishes its computation.

5.2 Statistical Machine Translation

We incorporate our LSTM as a rescoring feature on top of the output of a strong Arabic-English

syntax-based string-to-tree statistical machine translation system (Galley et al., 2006; Galley et al., 2004). That system is trained on 208 million words of parallel Arabic-English data from a variety of sources, much of which is Egyptian discussion forum content. The Arabic side is morphologically segmented with MADA-ARZ (Habash et al., 2013). We use a standard set of features, including two conventional count-based language models, as well as thousands of sparsely-occurring, lexicalized syntactic features (Chiang et al., 2009). The system additionally uses a source-to-target feed-forward neural network translation model during decoding, analogous to the model of (Devlin et al., 2014). These features are tuned with MIRA (Chiang et al., 2009) on approximately 63,000 words of Arabic discussion forum text, along with three references. We evaluate this baseline on two test sets, each with approximately 34,000 words from the same genre used in tuning.

We generate n-best lists ($n = 1000$) of unique translations for each sentence in the tuning set and re-rank the translations using an approach based on MERT (Och, 2003). We collapse all features other than language models, text length, and derivation size into a single feature, formed by taking the dot product of the previously learned feature and weight vectors. We then run a single iteration of MERT on the n-best lists to determine optimal weights for the collapsed feature, the uncollapsed features, and an LSTM feature formed by taking the score of the hypothesis according to the LSTM described in Section 5.1. We use the weights to rerank hypotheses from the n-best lists of the two test sets. We repeated this experiment, substituting instead a two-layer LSTM trained on the English side of the training data.

6 Results

Our two experiments with LSTMs trained with our modification of NCE show strong results in their corresponding tasks.

Our perplexity results are shown in Table 1, where we get significantly lower perplexities than the best single model from Chelba et al. (2013), while having almost 6 times fewer parameters. We also compute the partition function values, $\log\big(Z(\mathbf{u})\big)$, for our development set and we find that the mean is 0.058 and the variance is 0.139, indicating that training has encouraged self-normalization.

Model	Parameters	Perplexity
Chelba et al. (2013)	20m	51.3
NCE (ours)	3.4m	43.2

Table 1: For the One Billion Word Benchmark, our NCE method performs significantly better than the best single model in the baseline, Chelba et al. (2013), while using many fewer parameters.

Recently, (Józefowicz et al., 2016) achieved state-of-the-art language modeling perplexities (30.0) on the billion word dataset with a single model, using importance sampling to approximate the normalization constant, $\mathbf{Z}(\mathbf{u})$.

Independent of our work, they also share noise samples across the minibatch. However, they use 8192 noise samples, while we achieve strong perplexities with 100 noise samples. We also show significant improvements in machine translation, exploiting self-normalization for fast decoding, in addition to releasing a efficient toolkit that practitioners can use.

Table 2 shows our re-scoring experiments. When we incorporate only the LSTM trained on the BOLT dataset we get a +1.1 BLEU improvement on Tune, +1.4 on Test 1, and +1.1 on Test 2. When we also incorporate the LSTM trained on the One Billion Word dataset as a feature, we see another +0.2 BLEU increase on Tune and Test 2. In these re-scoring experiments we simply use the un-normalized numerator $p(w \mid \mathbf{u})$ as our word score, which means we never have to compute the costly partition function, $\mathbf{Z}(u)$. This is because the partition function is so close to 1 that the un-normalized

System	Tune	Test 1	Test 2
Baseline SMT	38.7	38.9	39.7
LSTM (BOLT)	39.8	40.3	40.8
LSTM (1b+BOLT)	40.0	40.3	41.0

Table 2: Our NCE-based language model successfully re-ranks Arabic-to-English n-best lists. The baseline is a state-of-the-art, statistical string-to-tree system. BOLT is a 208m-word, in-domain English corpus; "1b" refers to the One Billion Word benchmark corpus.

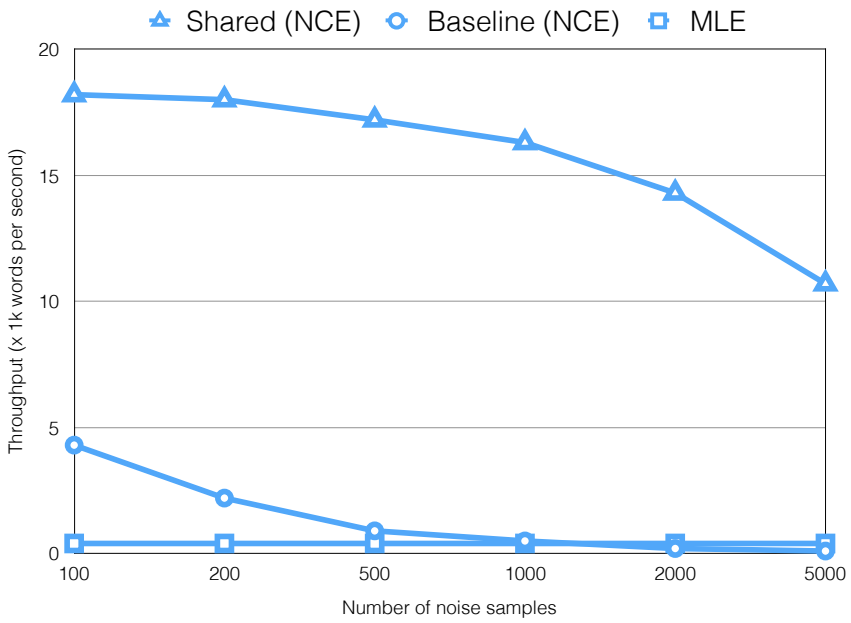

Figure 1: Sharing noise samples across the minibatch is at least 4 times faster than a baseline of not sharing them.

scores are very close to the normalized ones.

7 Conclusion

We describe a natural extension to NCE that achieves a large speedup on the GPU while also achieving good empirical results. LSTM models trained with our NCE modification achieve strong results on the One Billion Word dataset. Additionally, we get good BLEU gains when re-ranking n-best lists from a strong string-to-tree machine translation system. We also release an efficient toolkit for training LSTM language models with NCE.

8 Acknowledgments

This work was carried out with funding from DARPA (HR0011-15-C-0115) and ARL/ARO (W911NF-10-1-0533).

References

P. Baltescu and P. Blunsom. 2014. Pragmatic neural language modelling in machine translation. *arXiv preprint arXiv:1412.7119*.

C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, and P. Koehn. 2013. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.

D. Chiang, K. Knight, and W. Wang. 2009. 11,001 new features for statistical machine translation. In *Proc. NAACL*.

J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. Schwartz, and J. Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proc. ACL*.

M. Galley, M. Hopkins, K. Knight, and D. Marcu. 2004. What's in a translation rule? In *Proc. HLT-NAACL*.

M. Galley, J. Graehl, K. Knight, D. Marcu, S. DeNeefe, W. Wang, and I. Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proc. ACL-COLING*.

M. Gutmann and A. Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proc. AI Stats*.

N. Habash, R. Roth, O. Rambow, R. Eskander, and N. Tomeh. 2013. Morphological analysis and disambiguation for dialectal arabic. In *Proc. NAACL*.

S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *CoRR*, abs/1602.02410.

M. Luong, H. Pham, and C. Manning. 2015. Effective

approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Tomáš Mikolov. 2012. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*.

A. Mnih and Y. W. Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*.

F. J. Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. ACL*.

I. Sutskever, O. Vinyals, and Q. V. Le. 2014. Sequence to sequence learning with neural networks. In *Proc. NIPS*, pages 3104–3112.

A. Vaswani, Y. Zhao, V. Fossum, and D. Chiang. 2013. Decoding with large-scale neural language models improves translation. In *Proc. EMNLP*.

W. Williams, N. Prasad, D. Mrva, T. Ash, and T. Robinson. 2015. Scaling recurrent neural network language models. *CoRR*, abs/1502.00512.

W. Zaremba, I. Sutskever, and O. Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.