Speeding Up Neural Machine Translation Decoding by Shrinking Run-time Vocabulary

Xing Shi and Kevin Knight
Information Sciences Institute & Computer Science Department
University of Southern California
{xingshi,knight}@isi.edu

Abstract

We speed up Neural Machine Translation (NMT) decoding by shrinking run-time target vocabulary. We experiment with two shrinking approaches: Locality Sensitive Hashing (LSH) and word alignments. Using the latter method, we get a 2x overall speed-up over a highly-optimized GPU implementation, without hurting BLEU. On certain low-resource language pairs, the same methods improve BLEU by 0.5 points. We also report a negative result for LSH on GPUs, due to relatively large overhead, though it was successful on CPUs. Compared with Locality Sensitive Hashing (LSH), decoding with word alignments is GPU-friendly, orthogonal to existing speedup methods and more robust across language pairs.

1 Introduction

Neural Machine Translation (NMT) has been demonstrated as an effective model and been put into large-scale production (Wu et al., 2016; He, 2015). For online translation services, decoding speed is a crucial factor to achieve a better user experience. Several recently proposed training methods (Shen et al., 2015; Wiseman and Rush, 2016) aim to solve the *exposure bias* problem, but require decoding the whole training set multiple times, which is extremely time-consuming for millions of sentences.

Slow decoding speed is partly due to the large target vocabulary size V, which is usually in the tens of thousands. The first two columns of Table 1 show the breakdown of the runtimes required by sub-modules to decode 1812 Japanese sentences to English using a sequence-to-sequence model with local attention (Luong et al., 2015).

Sub-module	Full vocab	WA50	Speedup
Total	1002.78 s	481.52 s	2.08
– Beam expansion	174.28 s	76.52 s	2.28
– Source-side	83.67 s	83.44 s	1
– Target-side	743.25 s	354.52 s	2.1
– – Softmax	402.77 s	20.68 s	19.48
– – Attention	123.05 s	123.12 s	1
– – 2nd layer	64.72 s	64.76 s	1
– – 1st layer	88.02 s	87.96 s	1
Shrink vocab	-	0.39 s	-
BLEU	25.16	25.13	-

Table 1: Time breakdown and BLEU score of full vocabulary decoding and our "WA50" decoding, both with beam size 12. WA50 means decoding informed by word alignments, where each source word can select at most 50 relevant target words. The model is a 2-layer, 1000-hidden dimension, 50,000-target vocabulary LSTM seq2seq model with local attention trained on the ASPEC Japanese-to-English corpus (Nakazawa et al., 2016). The time is measured on a single Nvidia Tesla K20 GPU.

Softmax is the most computationally intensive part, where each hidden vector $h_t \in \mathbb{R}^d$ needs to dot-product with V target embeddings $e_i \in \mathbb{R}^d$. It occupies 40% of the total decoding time. Another sub-module whose computation time is proportional to V is *Beam Expansion*, where we need to find the top B words among all V vocabulary according to their probability. It takes around 17% of the decoding time.

Several approaches have proposed to improve decoding speed:

1. Using special hardware, such as GPU and Tensor Processing Unit (TPU), and low-precision calculation (Wu et al., 2016).

2. Compressing deep neural models through knowledge distillation and weight pruning (See et al., 2016; Kim and Rush, 2016).

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 574–579
Vancouver, Canada, July 30 - August 4, 2017.
https://doi.org/10.18653/v1/P17-2091

3. Several variants of *Softmax* have been proposed to solve its poor scaling properties on large vocabularies. Morin and Bengio (2005) propose *hierarchical softmax*, where at each step $\log_2 V$ binary classifications are performed instead of a single classification on a large number of classes. Gutmann and Hyvärinen (2010) propose *noise-contrastive estimation* which discriminate between positive labels and k ($k << V$) negative labels sampled from a distribution, and is applied successfully on natural language processing tasks (Mnih and Teh, 2012; Vaswani et al., 2013; Williams et al., 2015; Zoph et al., 2016). Although these two approaches provide good speedups for training, they still suffer at test time. Chen et al. (2016) introduces *differentiated softmax*, where frequent words have more parameters in the embedding and rare words have less, offering speedups on both training and testing.

In this work, we aim to speed up decoding by shrinking the run-time target vocabulary size, and this approach is **orthogonal** to the methods above. It is important to note that approaches 1 and 2 will maintain or even increase the ratio of target word embedding parameters to the total parameters, thus the *Beam Expansion* and *Softmax* will occupy the same or greater portion of the decoding time. A small run-time vocabulary will dramatically reduce the time spent on these two portions and gain a further speedup even after applying other speedup methods.

To shrink the run-time target vocabulary, our first method uses Locality Sensitive Hashing. Vijayanarasimhan et al. (2015) successfully applies it on CPUs and gains speedup on single step prediction tasks such as image classification and video identification. Our second method is to use word alignments to select a very small number of candidate target words given the source sentence. Recent works (Jean et al., 2015; Mi et al., 2016; L'Hostis et al., 2016) apply a similar strategy and report speedups for decoding on CPUs on rich-source language pairs.

Our major contributions are:

1. To our best of our knowledge, this work is the first attempt to apply LSH technique on sequence generation tasks on GPU other than single-step classification on CPU. We find current LSH algorithms have a poor performance/speed trade-off on GPU, due to the large overhead introduced by many hash table lookups and list-merging involved in LSH.

2. For our word alignment method, we find that only the candidate list derived from lexical translation table of IBM model 4 is adequate to achieve good BLEU/speedup trade-off for decoding on GPU. There is no need to combine the top frequent words or words from phrase table, as proposed in Mi et al. (2016).

3. We conduct our experiments on GPU and provide a detailed analysis of BLEU/speedup trade-off on both resource-rich/poor language pairs and both attention/non-attention NMT models. We achieve more than 2x speedup on 4 language pairs with only a tiny BLEU drop, demonstrating the robustness and efficiency of our methods.

2 Methods

At each step during decoding, the *softmax* function is calculated as:

$$P(y = j|h_i) = \frac{e^{h_i^T w_j + b_j}}{\sum_{k=1}^{V} e^{h_i^T w_k + b_k}} \quad (1)$$

where $P(y = j|h_i)$ is the probability of word $j = 1...V$ given the hidden vector $h_i \in \mathbb{R}^d, i = 1...B$. B represents the beam size. $w_j \in \mathbb{R}^d$ is output word embedding and $b_j \in \mathbb{R}$ is the corresponding bias. The complexity is $\mathcal{O}(dBV)$. To speed up *softmax*, we use *word frequency*, *locality sensitive hashing*, and *word alignments* respectively to select C ($C << V$) potential words and evaluate their probability only, reducing the complexity to $\mathcal{O}(dBC + overhead)$.

2.1 Word Frequency

A simple baseline to reduce target vocabulary is to select the top C words based on their frequency in the training corpus. There is no run-time overhead and the overall complexity is $\mathcal{O}(dBC)$.

2.2 Locality Sensitive Hashing

The word $j = \arg\max_k P(y = k|h_i)$ will have the largest value of $h_i^T w_j + b_j$. Thus the $\arg\max$ problem can be converted to finding the nearest neighbor of vector $[h_i; 1]$ among the vectors $[w_j; b_j]$ under the distance measure of dot-product.

Locality Sensitive Hashing (LSH) is a powerful technique for the nearest neighbor problem. We employ the winner-take-all (WTA) hashing (Yagnik et al., 2011) defined as:

$$WTA(x \in \mathbb{R}^d) = [I_1; ...; I_p; ...; I_P] \quad (2)$$
$$I_p = \arg\max_{k=1}^{K} Permute_p(x)[k] \quad (3)$$
$$WTA_{band}(x) = [B_1; ...; B_w; ...; B_W] \quad (4)$$
$$B_w = [I_{(w-1)*u+1}; ...; I_{(w-1)*u+i}; ...; I_{w*u}] \quad (5)$$
$$u = P/W \quad (6)$$

where P distinct permutations are applied and the index of the maximum value of the first K elements of each permutations is recorded. To perform approximate nearest neighbor searching, we follow the scheme used in (Dean et al., 2013; Vijayanarasimhan et al., 2015):

1. Split the hash code $WTA(x)$ into W bands (as shown in equation 4), with each band $\frac{P}{W}\log_2(K)$ bits long.

2. Create W hash tables $[T_1, ..., T_w, ..., T_W]$, and hash every word index j into every table T_w using $WTA_{band}(w_j)[w]$ as the key.

3. Given the hidden vector h_i, extract a list of word indexes from each table T_w using the key $WTA_{band}(h_i)[w]$. Then we merge the W lists and count the number of the occurrences of each word index. Select the top C word indexes with the largest counts, and calculate their probability using equation 1.

The 4 hyper-parameters that define a WTA-LSH are $\{K, P, W, C\}$. The run-time overhead comprises hashing the hidden vector, W times hash table lookups and W lists merging. The overall complexity is $\mathcal{O}(B(dC+K*P+W+W*N_{avg})))$, where N_{avg} is the average number of the word indexes stored in a hash bin of T_w. Although the complexity is much smaller than $\mathcal{O}(dBV)$, the runtime in practice is not guaranteed to be shorter, especially on GPUs, as hash table lookups introduce too many small kernel launches and list merging is hard to parallelize.

2.3 Word Alignment

Intuitively, LSH shrinks the search space utilizing the spatial relationship between the query vector and database vectors in high dimension space. It is a task-independent technique. However, when focusing on our specific task (MT), we can employ translation-related heuristics to prune the run-time vocabulary precisely and efficiently.

One simple heuristic relies on the fact that each source word can only be translated to a small set of target words. The word alignment model, a foundation of phrase-base machine translation, also follows the same spirit in its generative story: each source word is translated to zero, one, or more target words and then reordered to form target sentences. Thus, we apply the following algorithm to reduce the run-time vocabulary size:

1. Apply IBM Model 4 and the *grow-diag-final* heuristic on the training data to get word alignments. Calculate the lexical translation table P($e|f$) based on word alignments.

2. For each word f in the source vocabulary of the neural machine translation model, store the top M target words according to P($e|f$) in a hash table $T_{f2e} = \{f : [e_1, ...e_M]\}$

3. Given a source sentence $s = [f_1, ..., f_N]$, extract the candidate target word list from T_{f2e} for each source word f_i. Merge the N lists to form the reduced target vocabulary V_{new};

4. Construct the new embedding matrix and bias vector according to V_{new}, then perform the normal beam search on target side.

The only hyper-parameter is $\{M\}$, the number of candidate target words for each source word. Given a source sentence of length L_s, the run-time overhead includes L_s times hash table lookups and L_s lists merging. The complexity for each decoding step is $\mathcal{O}(dB|V_{new}|+(L_s+L_sM)/L_t)$, where L_t is the maximum number of decoding steps. Unlike LSH, these table lookups and list mergings are performed once per sentence, and do not depend on the any hidden vectors. Thus, we can overlap the computation with source side forward propagation.

3 Experiments

To examine the robustness of these decoding methods, we vary experiment settings in different ways: 1) We train both attention (Luong et al., 2015) and non-attention (Sutskever et al., 2014) models; 2) We train models on

	J2E			E2J			F2E			U2E		
	TC	BLEU	X	TC	BLEU	X	TC	BLEU	X	TC	BLEU	X
Full	0.87	25.16	1	0.95	33.87	1	0.84	**28.12**	1	0.9	11.67	1
TF1K	0.14	13.42	2.11	0.15	18.91	2.42	0.1	12.1	2.32	0.29	8.78	1.65
TF5K	0.49	21.31	1.93	0.56	29.77	2.23	0.38	21.98	2.04	0.67	11.54	1.51
TF10K	0.67	23.62	1.76	0.75	32.28	2.04	0.56	24.88	1.78	0.81	11.67	1.33
TF20K	0.78	24.61	1.48	0.87	33.41	1.74	0.72	26.95	1.42	0.89	11.66	1.09
LSH1K	-	19.45	0.026	-	22.23	0.027	-	3.43	0.036	-	9.41	0.025
LSH5K	-	23.43	0.023	-	30.63	0.025	-	12.81	0.031	-	11.41	0.022
LSH10K	-	24.82	0.022	-	32.63	0.024	-	18.45	0.028	-	11.63	0.020
LSH20K	-	**25.20**	0.020	-	33.78	0.022	-	24.31	0.025	-	11.73	0.018
WA10	0.75	24.74	2.12	0.77	33.24	2.46	0.72	27.9	2.37	0.66	**12.17**	1.7
WA50	0.82	25.13	2.08	0.85	33.79	2.43	0.77	27.94	2.34	0.71	12.01	1.67
WA250	0.84	25.13	1.89	0.88	**34.05**	2.27	0.8	27.95	2.1	0.73	11.94	1.62
WA1000	0.85	25.17	1.57	0.9	33.97	1.93	0.82	28.08	1.67	0.75	11.89	1.58

Table 2: Word type coverage (TC), BLEU score, and speedups (X) for full-vocabulary decoding (Full), top frequency vocabulary decoding (TF*), LSH decoding (LSH*), and decoding with word alignments(WA*). TF10K represents decoding with top 10,000 frequent target vocabulary ($C = 10,000$). WA10 means decoding with word alignments, where each source word can select at most 10 candidate target words ($M = 10$). For LSH decoding, we choose (32, 5000, 1000) for (K,P,W), and vary C.

both resource-rich language pairs, French to English (F2E) and Japanese to English (J2E), and a resource-poor language pair, Uzbek to English (U2E); 3) We translate both to English (F2E, J2E, and U2E) and from English (E2J). We use 2-layer LSTM seq2seq models with different attention settings, hidden dimension sizes, dropout rates, and initial learning rates, as shown in Table 3. We use the ASPEC Japanese-English Corpus (Nakazawa et al., 2016), French-English Corpus from WMT2014 (Bojar et al., 2014), and Uzbek-English Corpus (Linguistic Data Consortium, 2016).

Table 2 shows the decoding results of the three methods. Decoding with word alignments achieves the best performance/speedup trade-off across all four translation directions. It can halve the overall decoding time with less than 0.17 BLEU drop. Table 1 compares the detailed time breakdown of full-vocabulary decoding and WA50 decoding. WA50 can gain a speedup of 19.48x and 2.28x on *softmax* and *beam expansion* respectively, leading to an overall 2.08x speedup with only 0.03 BLEU drop. In contrast, decoding with top frequent words will hurt the BLEU rapidly as the speedup goes higher. We calculate the word type coverage (TC) for the test reference data as follows:

	J2E	E2J	F2E	U2E
Source Vocab	80K	88K	200K	50K
Target Vocab	50K	66K	40K	25K
#Tokens	70.4M	70.4M	652M	3.3M
#Sent pairs	1.4M	1.4M	12M	88.7K
Attention	Yes	Yes	No	Yes
Dimension	1000	1000	1000	500
Dropout	0.2	0.2	0.2	0.5
Learning rate	0.5	1	0.35	0.5

Table 3: Training configurations on different language pairs.

$$TC = \frac{|\{\text{run-time vocab}\} \cap \{\text{word types in test}\}|}{|\{\text{word types in test}\}|}$$

The top 1000 words only cover 14% word types of J2E test data, whereas WA10 covers 75%, whose run-time vocabulary is no more than 200 for a 20 words source sentence.

The speedup of English-to-Uzbek translation is relatively low (around 1.7x). This is because the original full vocabulary size is small (25k), leaving less room for shrinkage.

LSH achieves better BLEU than decoding with top frequent words of the same run-time vocabulary size C on attention models. However, it in-

troduces too large an overhead (50 times slower), especially when *softmax* is highly optimized on GPU. When doing sequential beam search, search error accumulates rapidly. To reach reasonable performance, we have to apply an adequately large number of permutations ($P = 5000$).

We also find that decoding with word alignments can even improve BLEU on resource-poor languages (12.17 vs. 11.67). Our conjecture is that rare words are not trained enough, so neural models confuse them, and word alignments can provide a hard constraint to rule out the unreasonable word choices.

4 Conclusion

We apply word alignments to shrink run-time vocabulary to speed up neural machine translation decoding on GPUs, and achieve more than 2x speedup on 4 translation directions without hurting BLEU. We also compare with two other speedup methods: decoding with top frequent words and decoding with LSH. Experiments and analyses demonstrate that word alignments provides accurate candidate target words and introduces only a tiny overhead over a highly-optimized GPU implementation.

References

Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Matous Machacek, Christof Monz, Pavel Pecina, Matt Post, Herv Saint-Amand, Radu Soricut, and Lucia Specia, editors. 2014. *Proc. Ninth Workshop on Statistical Machine Translation*.

Welin Chen, David Grangier, and Michael Auli. 2016. Strategies for training large vocabulary neural language models. In *Proc. ACL*.

Thomas Dean, Mark A Ruzon, Mark Segal, Jonathon Shlens, Sudheendra Vijayanarasimhan, and Jay Yagnik. 2013. Fast, accurate detection of 100,000 object classes on a single machine. In *Proc. CVPR*.

Michael Gutmann and Aapo Hyvärinen. 2010. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proc. AISTATS*.

Zhongjun He. 2015. Baidu translate: research and products. In *Proc. ACL-IJCNLP*.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *Proc. ACL*.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In *Proc. EMNLP*.

Gurvan L'Hostis, David Grangier, and Michael Auli. 2016. Vocabulary Selection Strategies for Neural Machine Translation. *Arxiv preprint* arXiv:1610.00072.

Linguistic Data Consortium. 2016. (bolt lrl uzbek representative language pack v1.0. ldc2016e29.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proc. EMNLP*.

Haitao Mi, Zhiguo Wang, and Abraham Ittycheriah. 2016. Vocabulary Manipulation for Neural Machine Translation. In *Proc. ACL*.

Andriy Mnih and Yee Whye Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. *Proc. ICML* .

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proc. AISTATS*.

Toshiaki Nakazawa, Manabu Yaguchi, Kiyotaka Uchimoto, Masao Utiyama, Eiichiro Sumita, Sadao Kurohashi, and Hitoshi Isahara. 2016. ASPEC: Asian Scientific Paper Excerpt Corpus. In *Proc. LREC*.

Abigail See, Minh-Thang Luong, and Christopher D Manning. 2016. Compression of neural machine translation models via pruning.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2015. Minimum risk training for neural machine translation. In *Proc. ACL*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proc. NIPS*.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. 2013. Decoding with large-scale neural language models improves translation. In *Proc. EMNLP*.

Sudheendra Vijayanarasimhan, Jonathon Shlens, Rajat Monga, and Jay Yagnik. 2015. Deep networks with large output spaces. In *Proc. ICLR*.

Will Williams, Niranjani Prasad, David Mrva, Tom Ash, and Tony Robinson. 2015. Scaling recurrent neural network language models. In *Proc. ICASSP*.

Sam Wiseman and Alexander M Rush. 2016. Sequence-to-sequence learning as beam-search optimization. In *Proc. EMNLP*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus

Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* .

Jay Yagnik, Dennis Strelow, David A Ross, and Ruei-sung Lin. 2011. The power of comparative reasoning. In *Proc. ICCV*.

Barret Zoph, Ashish Vaswani, Jonathan May, and Kevin Knight. 2016. Simple, fast noise-contrastive estimation for large rnn vocabularies. In *Proc. NAACL-HLT*.