Speeding Up Neural Machine Translation Decoding by Cube Pruning

Wen Zhang[1,2] **Liang Huang**[3,4] **Yang Feng**[1,2] **Lei Shen**[1,2] **Qun Liu**[5,1]

[1]Key Lab. of Intelligent Information Processing, CAS Institute of Computing Technology
[2]University of Chinese Academy of Sciences, Beijing, China
{zhangwen,fengyang,shenlei17z}@ict.ac.cn

[3]Oregon State University, Corvallis, OR, USA [4]Baidu Research, Sunnyvale, CA, USA
liang.huang.sh@gmail.com

[5]Huawei Noah's Ark Lab, Hong Kong, China
qun.liu@huawei.com

Abstract

Although neural machine translation has achieved promising results, it suffers from slow translation speed. The direct consequence is that a trade-off has to be made between translation quality and speed, thus its performance cannot come into full play. We apply cube pruning, a popular technique to speed up dynamic programming, into neural machine translation to speed up the translation. To construct the equivalence class, similar target hidden states are combined, leading to less RNN expansion operations on the target side and less $\mathrm{softmax}$ operations over the large target vocabulary. The experiments show that, at the same or even better translation quality, our method can translate faster compared with naive beam search by $3.3\times$ on GPUs and $3.5\times$ on CPUs.

1 Introduction

Neural machine translation (NMT) has shown promising results and drawn more attention recently (Kalchbrenner and Blunsom, 2013; Cho et al., 2014b; Bahdanau et al., 2015; Gehring et al., 2017a,b; Vaswani et al., 2017). A widely used architecture is the attention-based encoder-decoder framework (Cho et al., 2014b; Bahdanau et al., 2015) which assumes there is a common semantic space between the source and target language pairs. The encoder encodes the source sentence to a representation in the common space with the recurrent neural network (RNN) (Hochreiter and Schmidhuber, 1997) and the decoder decodes this representation to generate the target sentence word by word. To generate a target word, a probability distribution over the target vocabulary is drawn based on the attention over the entire source sequence and the target information rolled by another RNN. At the training time, the decoder is forced to generate the ground truth sentence, while at inference, it needs to employ the beam search algorithm to search through a constrained space due to the huge search space.

Even with beam search, NMT still suffers from slow translation speed, especially when it works not on GPUs, but on CPUs, which are more common practice. The first reason for the inefficiency is that the generation of each target word requires extensive computation to go through all the source words to calculate the attention. Worse still, due to the recurrence of RNNs, target words can only be generated *sequentially* rather than in parallel. The second reason is that large vocabulary on target side is employed to avoid unknown words (UNKs), which leads to a large number of normalization factors for the $\mathrm{softmax}$ operation when drawing the probability distribution. To accelerate the translation, the widely used method is to trade off between the translation quality and the decoding speed by reducing the size of vocabulary (Mi et al., 2016a) or/and the number of parameters, which can not realize the full potential of NMT.

In this paper, we borrow ideas from phrase-based and syntax-based machine translation where cube pruning has been successfully applied to speed up the decoding (Chiang, 2007; Huang and Chiang, 2007). Informally, cube pruning "coarsens" the search space by clustering similar states according to some equivalence relations. To apply this idea to NMT, however, is much more involved. Specifically, in the process of beam search, we cluster similar target hidden states to construct equivalence classes, the three dimensions of which are target words in the target vocabulary, part translations retained in the beam search and different combinations of similar target hidden states, respectively. The clustering operation

can directly decrease the number of target hidden states in the following calculations, together with cube pruning, resulting in less RNN expansion operations to generate the next hidden state (related to the first reason) and less softmax operations over the target vocabulary (related to the second reason). The experiment results show that, when receiving the same or even better translation quality, our method can speed up the decoding speed by 3.3× on GPUs and 3.5× on CPUs.

2 Background

The proposed strategy can be adapted to optimize the beam search algorithm in the decoder of various NMT models. Without loss of generality, we take the attention-based NMT (Bahdanau et al., 2015) as an example to introduce our method. In this section, we first introduce the attention-based NMT model and then the cube pruning algorithm.

2.1 The Attention-based NMT Model

The attention-based NMT model follows the encoder-decoder framework with an extra attention module. In the following parts, we will introduce each of the three components. Assume the source sequence and the observed translation are $\mathbf{x} = \{x_1, \cdots, x_{|\mathbf{x}|}\}$ and $\mathbf{y} = \{y_1^*, \cdots, y_{|\mathbf{y}|}^*\}$.

Encoder The encoder uses a bidirectional GRU to obtain two sequences of hidden states. The final hidden state of each source word is got by concatenating the corresponding pair of hidden states in those sequences. Note that e_{x_i} is employed to represent the embedding vector of the word x_i.

$$\overrightarrow{h}_i = \overrightarrow{\mathbf{GRU}}\left(e_{x_i}, \overrightarrow{h}_{i-1}\right) \quad (1)$$

$$\overleftarrow{h}_i = \overleftarrow{\mathbf{GRU}}\left(e_{x_i}, \overleftarrow{h}_{i+1}\right) \quad (2)$$

$$h_i = \left[\overrightarrow{h}_i; \overleftarrow{h}_i\right] \quad (3)$$

Attention The attention module is designed to extract source information (called context vector) which is highly related to the generation of the next target word. At the j-th step, to get the context vector, the relevance between the target word y_j^* and the i-th source word is firstly evaluated as

$$r_{ij} = \mathbf{v}_a^T \tanh\left(\mathbf{W}_a s_{j-1} + \mathbf{U}_a h_i\right) \quad (4)$$

Then, the relevance is normalized over the source sequence, and all source hidden states are added weightedly to produce the context vector.

$$\alpha_{ij} = \frac{\exp\left(r_{ij}\right)}{\sum_{i'=1}^{|\mathbf{x}|} \exp\left(r_{i'j}\right)}; \quad c_j = \sum_{i=1}^{|\mathbf{x}|} \alpha_{ij} h_i \quad (5)$$

Decoder The decoder also employs a GRU to unroll the target information. The details are described in Bahdanau et al. (2015). At the j-th decoding step, the target hidden state s_j is given by

$$s_j = f\left(e_{y_{j-1}^*}, s_{j-1}, c_j\right) \quad (6)$$

The probability distribution $\mathcal{D}_j$ over all the words in the target vocabulary is predicted conditioned on the previous ground truth words, the context vector c_j and the unrolled target information s_j.

$$t_j = g\left(e_{y_{j-1}^*}, c_j, s_j\right) \quad (7)$$

$$o_j = \mathbf{W}_o t_j \quad (8)$$

$$\mathcal{D}_j = \mathrm{softmax}\left(o_j\right) \quad (9)$$

where g stands for a linear transformation, $\mathbf{W}_o$ is used to map t_j to o_j so that each target word has one corresponding dimension in o_j.

2.2 Cube Pruning

The cube pruning algorithm, proposed by Chiang (2007) based on the k-best parsing algorithm of Huang and Chiang (2005), is actually an accelerated extension based on the naive beam search algorithm. Beam search, a heuristic dynamic programming searching algorithm, explores a graph by expanding the most promising nodes in a limited set and searches approximate optimal results from candidates. For the sequence-to-sequence learning task, given a pre-trained model, the beam search algorithm finds a sequence that approximately maximizes the conditional probability (Graves, 2012; Boulanger-Lewandowski et al., 2013). Both Sutskever et al. (2014) and Bahdanau et al. (2015) employed the beam search algorithm into the NMT decoding to produce translations with relatively larger conditional probability with respect to the optimized model parameters. Remarkably, Huang and Chiang (2007) successfully applied the cube pruning algorithm to the decoding of SMT. They found that the beam search algorithm in SMT can be extended, and they utilized the cube pruning and some variants to optimize the search process in the decoding phase of phrase-based (Och and Ney, 2004) and syntax-based (Chiang, 2005; Galley et al., 2006) systems,

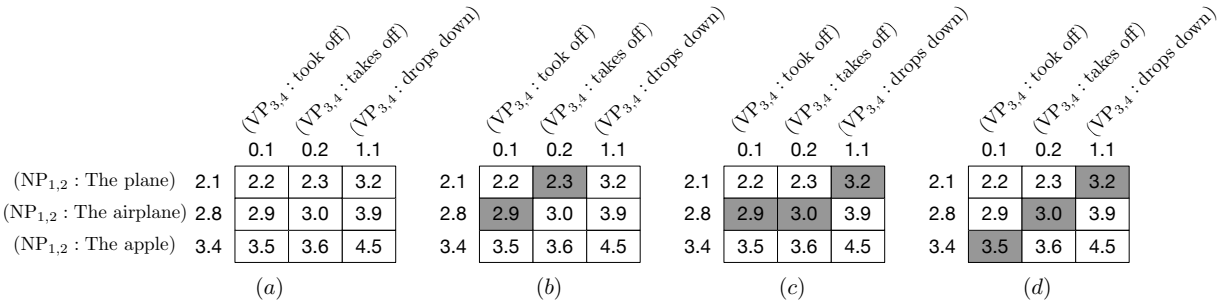


Figure 1: Cube pruning in SMT decoding. (a): the values in the grid denote the negative log-likelihood cost of the terminal combinations on both dimensions, and each dimension denotes a translation candidate in this example; (b)-(d): the process of popping the best candidate of top three items.

	GPU		CPU	
Calculation Units	**Time(s)**	**Percentage**	**Time(s)**	**Percentage**
Eq. (6): $s_j = f(e_{y^*_{j-1}}, s_{j-1}, c_j)$	551.07	75.73%	1370.92	19.42%
Eq. (7): $t_j = g(e_{y^*_{j-1}}, c_j, s_j)$	88.25	12.13%	277.76	3.93%
Eq. (8): $o_j = \mathbf{W}_o t_j$	25.33	3.48%	2342.53	33.18%
Eq. (9): $\mathcal{D}_j = \mathrm{softmax}(o_j)$	63.00	8.66%	3069.25	43.47%

Table 1: Time cost statistics for decoding the whole MT03 testset on GPUs and CPUs with beam size 10.

which decreased a mass of translation candidates and achieved a significant speed improvement by reducing the size of complicated search space, thereby making it possible to actualize the thought of improving the translation performance through increasing the beam size.

In the traditional SMT decoding, the cube pruning algorithm aims to prune a great number of partial translation hypotheses without computing and storing them. For each decoding step, those hypotheses with the same translation rule are grouped together, then the cube pruning algorithm is conducted over the hypotheses. We illustrate the detailed process in Figure 1.

3 NMT Decoder with Cube Pruning

3.1 Definitions

We define the related storage unit tuple of the i-th candidate word in the j-th beam as $n^i_j = (c^i_j, s^i_j, y^i_j, bp^i_j)$, where c^i_j is the negative log-likelihood (NLL) accumulation in the j-th beam, s^i_j is the decoder hidden state in the j-th beam, y^i_j is the index of the j-th target word in large vocabulary and bp^i_j is the backtracking pointer for the j-th decoding step. Note that, for each source sentence, we begin with calculating its encoded representation and the first hidden state s^0_0 in decoder, then searching from the initial tuple $(0.0, s^0_0, 0, 0)$ existing in the first beam[1].

It is a fact that Equation (9) produces the probability distribution of the predicted target words over the target vocabulary V. Cho et al. (2014b) indicated that whenever a target word is generated, the $\mathrm{softmax}$ function over V computes probabilities for all words in V, so the calculation is expensive when the target vocabulary is large. As such, Bahdanau et al. (2015) (and many others) only used the top-$30k$ frequent words as target vocabulary, and replaced others with UNK. However, the final normalization operation still brought high computation complexity for forward calculations.

3.2 Time Cost in Decoding

We conducted an experiment to explore how long each calculation unit in the decoder would take. We decoded the MT03 test dataset by using naive beam search with beam size of 10 and recorded the time consumed in the computation of Equation (6), (7), (8) and (9), respectively. The statistical results in Table 1 show that the recurrent calculation unit consumed the most time on GPUs, while

[1]The initial target word index y^0_0 equals to 0, which actually corresponds to the Beginning Of Sentence (BOS) token in target vocabulary.

the softmax computation also took lots of time. On CPUs, the most expensive computational time cost was caused by the softmax operation over the entire target vocabulary[2]. In order to avoid the time-consuming normalization operation in testing, we introduced *self-normalization* (denoted as **SN**) into the training.

3.3 Self-normalization

Self-normalization (Devlin et al., 2014) was designed to make the model scores which are produced by the output layer be approximated by the probability distribution over the target vocabulary without normalization operation. According to Equation (9), for an observed target sentence $\mathbf{y} = \{y_1^*, \cdots, y_{|\mathbf{y}|}^*\}$, the Cross-Entropy (CE) loss could be written as

$$\begin{aligned} \mathrm{L}_\theta &= -\sum_{j=1}^{|\mathbf{y}|} \log \mathcal{D}_j[y_j^*] \\ &= -\sum_{j=1}^{|\mathbf{y}|} \log \frac{\exp\left(o_j[y_j^*]\right)}{\sum_{y' \in V} \exp\left(o_j[y']\right)} \\ &= \sum_{j=1}^{|\mathbf{y}|} \log \sum_{y' \in V} \exp\left(o_j[y']\right) - o_j[y_j^*] \end{aligned} \tag{10}$$

where o_j is the model score generated by Equation (8) at the j-th step, we marked the softmax normalizer $\sum_{y' \in V} \exp\left(o_j[y']\right)$ as Z.

Following the work of Devlin et al. (2014), we modified the CE loss function into

$$\begin{aligned} \mathrm{L}_\theta &= -\sum_{j=1}^{|\mathbf{y}|} \left(\log \mathcal{D}_j[y_j^*] - \alpha(\log Z - 0)^2\right) \\ &= -\sum_{j=1}^{|\mathbf{y}|} \left(\log \mathcal{D}_j[y_j^*] - \alpha \log^2 Z\right) \end{aligned} \tag{11}$$

The objective function, shown in Equation (11), is optimized to make sure $\log Z$ is approximated to 0, equally, make Z close to 1 once it converges. We chose the value of α empirically. Because the softmax normalizer Z is converged to 1 in inference, we just need to ignore Z and predict the target word distribution at the j-th step only with o_j:

$$\mathcal{D}_j = o_j \tag{12}$$

3.4 Cube Pruning

Table 1 clearly shows that the equations in the NMT forward calculation take lots of time. Here, according to the idea behind the cube pruning algorithm, we tried to reduce the time of time-consuming calculations, e.g., Equation (6), and further decrease the search space by introducing the cube pruning algorithm.

3.4.1 Integrating into NMT Decoder

Extended from the naive beam search in the NMT decoder, cube pruning, treated as a pruning algorithm, attempts to reduce the search space and computation complexity by merging some similar items in a beam to accelerate the naive beam search, keeping the 1-best searching result almost unchanged or even better by increasing the beam size. Thus, it is a fast and effective algorithm to generate candidates.

Assume that T restores the set of the finished translations. For each step in naive beam search process, $\mathrm{beamsize}-|\mathrm{T}|$ times forward calculations are required to acquire $\mathrm{beamsize}-|\mathrm{T}|$ probability distributions corresponding to each item in the previous beam (Bahdanau et al., 2015). while for each step in cube pruning, in terms of some constraints, we merge all similar items in the previous beam into one *equivalence class* (called a sub-cube). The constraint we used here is that items being merged in the previous beam should have the same target words. Then, for the sub-cube, only one forward calculation is required to obtain the approximate predictions by using the loose hidden state. Elements in the sub-cube are sorted by previous accumulated NLL along the columns (the first dimension of beam size) and by the approximate predictions along the rows (the second dimension of vocabulary size). After merging, one beam may contain several sub-cubes (the third dimension), we start to search from item in the upper left corner of each sub-cube, which is the best one in the sub-cube, and continue to spread out until enough candidates are found. Once a item is selected, the exact hidden state will be used to calculate its exact NLL.

Through all above steps, the frequency of forward computations decreases. We give an example to dive into the details in Figure 2.

Assume that the beam size is 4. Given the 10^{th}

[2]Note that, identical to Bahdanau et al. (2015), we only used $30k$ as the vocabulary size.

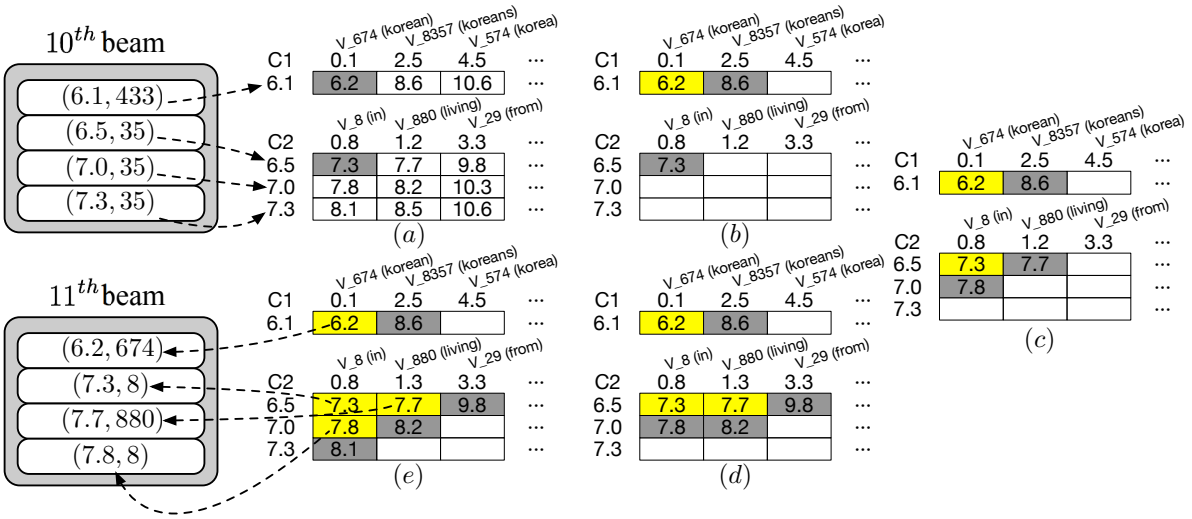


Figure 2: Cube pruning diagram in beam search process during NMT decoding. We only depict the accumulated NLL and the word-level candidate for each item in the beam (in the bracket). Assume the beam size is 4, we initialize a heap for the current step, elements in the 10^{th} beam are merged into two sub-cubes C1 and C2 according to the previous target words; (a) the two elements located in the upper-left corner of the two sub-cubes are pushed into the heap; (b) minimal element $(6.2, 674)$ is popped out, meanwhile, its neighbor $(8.6, 8357)$ is pushed into the heap; (c) minimal element $(7.3, 8)$ is popped out, its right-neighbor $(7.7, 880)$ and lower-neighbor $(7.8, 8)$ are pushed into the heap; (d) minimal element $(7.7, 880)$ is popped out, its right-neighbor $(9.8, 29)$ and down-neighbor $(8.2, 880)$ are pushed into the heap; (e) minimal element $(7.8, 8)$ is popped out, then its down-neighbor $(8.1, 8)$ is pushed into the heap. 4 elements have been popped out, we use them to construct the 11^{th} beam. Yellow boxes indicate the 4-best word-level candidates to be pushed into the 11^{th} beam.

beam, we generate the 11^{th} beam. Different from the naive beam search, we first group items in the previous beam into two sub-cubes C1 and C2 in term of the target word y_{j-1}. As shown in part (a) of Figure 2, $(6.1, 433)$ constructs the sub-cube C1; $(6.5, 35)$, $(7.0, 35)$ and $(7.3, 35)$ are put together to compose another sub-cube C2. Items in part (a) are ranked in ascending order along both row and column dimension according to the accumulated NLL. For each sub-cube, we use the first state vector in each sub-cube as the approximate one to produce the next probability distribution and the next state. At beginning, each upper-left corner element in each sub-cube is pushed into a minimum heap, after popping minimum element from the heap, we calculate and restore the exact NLL of the element, then push the right and lower ones alongside the minimum element into heap. At this rate, the searching continues just like the "diffusion" in the sub-cube until 4 elements are popped, which are ranked in terms of their exact NLLs to construct the 11^{th} beam. Note that once an element is popped, we calculate its exact NLL. From the step (e) in Figure 2, we can see that 4 elements have been popped from C1 and C2, and then ranked in terms of their exact NLLs to build the 11^{th} beam.

We refer above algorithm as the naive cube pruning algorithm (called **NCP**)

3.4.2 Accelerated Cube Pruning

In each step of the cube pruning algorithm, after merging the items in the previous beam, some similar candidates are grouped together into one or more sub-cube(s). We also try to predict the approximate distribution for each sub-cube only according to the top-1 state vector (the first row in the sub-cube in Figure 2), and select next candidates after ranking. The predicted probability distribution will be very similar to that of the naive beam search. Besides, Each sub-cube only requires one forward calculation. Thus, it could save more search space and further reduce the computation complexity for the decoder. Unlike the naive cube pruning algorithm, accelerated cube pruning pops each item, then still use the approximate NLL instead of the exact one. We denote this kind of accelerated cube pruning algorithm as **ACP**.

4 Experiments

We verified the effectiveness of proposed cube pruning algorithm on the Chinese-to-English (Zh-En) translation task.

4.1 Data Preparation

The Chinese-English training dataset consists of 1.25M sentence pairs[3]. We used the NIST 2002 (MT02) dataset as the validation set with 878 sentences, and the NIST 2003 (MT03) dataset as the test dataset, which contains 919 sentences.

The lengths of the sentences on both sides were limited up to 50 tokens, then actually 1.11M sentence pairs were left with 25.0M Chinese words and 27.0M English words. We extracted $30k$ most frequent words as the source and target vocabularies for both sides.

In all the experiments, case-insensitive 4-gram BLEU (Papineni et al., 2002) was employed for the automatic evaluation, we used the script *mteval-v11b.pl*[4] to calculate the BLEU score.

4.2 System

The system is an improved version of attention-based NMT system named RNNsearch (Bahdanau et al., 2015) where the decoder employs a conditional GRU layer with attention, consisting of two GRUs and an attention module for each step[5]. Specifically, Equation (6) is replaced with the following two equations:

$$\tilde{s}_j = \mathbf{GRU}_1(e_{y^*_{j-1}}, s_{j-1}) \quad (13)$$

$$s_j = \mathbf{GRU}_2(c_j, \tilde{s}_j) \quad (14)$$

Besides, for the calculation of attention in Equation (4), s_{j-1} is replaced with $\tilde{s}_{j-1}$. Other components of the system keep the same as RNNsearch. Also, we re-implemented the beam search algorithm as the naive decoding method, and naive searching on the GPU and CPU server were conducted as two baselines.

4.3 Training Details

Specially, we employed a little different settings from Bahdanau et al. (2015): Word embedding sizes on both sides were set to 512, all hidden sizes in the GRUs of both encoder and decoder were also set to 512. All parameter matrices, including bias matrices, were initialized with the uniform distribution over $[-0.1, 0.1]$. Parameters were updated by using mini-batch Stochastic Gradient Descent (SGD) with batch size of 80 and the learning rate was adjusted by AdaDelta (Zeiler, 2012) with decay constant ρ=0.95 and denominator constant ϵ=$1e$-6. The gradients of all variables whose L2-norm are larger than a pre-defined threshold 1.0 were normalized to the threshold to avoid gradient explosion (Pascanu et al., 2013). Dropout was applied to the output layer with dropout rate of 0.5. We exploited length normalization (Cho et al., 2014a) strategy on candidate translations in beam search decoding.

The model whose BLEU score was the highest on the validation set was used to do testing. Maximal epoch number was set to 20. Training was conducted on a single Tesla K80 GPU, it took about 2 days to train a single NMT model on the Zh-En training data. For *self-normalization*, we empirically set α as 0.5 in Equation (11)[6].

4.4 Search Strategies

We conducted experiments to decode the MT03 test dataset on the GPU and CPU server respectively, then compared search quality and efficiency among following six search strategies under different beam sizes.

NBS-SN: Naive Beam Search without **SN**

NBS+SN: Naive Beam Search with **SN**

NCP-SN: Cube Pruning without **SN**

NCP+SN: Cube Pruning with **SN**

ACP-SN: Accelerated Cube Pruning without **SN**

ACP+SN: Accelerated Cube Pruning with **SN**

4.5 Comparison of Average Merging Rate

We first give the definition of the Average Merging Rate (denoted as AMR). Given a test dataset, we counted the total word-level candidates (noted as N_w) and the total sub-cubes (noted as N_c) during the whole decoding process, then the AMR can be simply computed as

$$\overline{m} = \mathrm{N}_w / \mathrm{N}_c \quad (15)$$

The MT03 test dataset was utilized to compare the trends of the AMR values under all

[3]These sentence pairs are mainly extracted from LDC2002E18, LDC2003E07, LDC2003E14, Hansards portion of LDC2004T07, LDC2004T08 and LDC2005T06

[4]https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/mteval-v11b.pl

[5]https://github.com/nyu-dl/dl4mt-tutorial/blob/master/docs/cgru.pdf

[6]Following Devlin et al. (2014), we had tried 0.01, 0.1, 0.5 1.0 and 10.0 for the value of α, we found that 0.5 produced the best result.

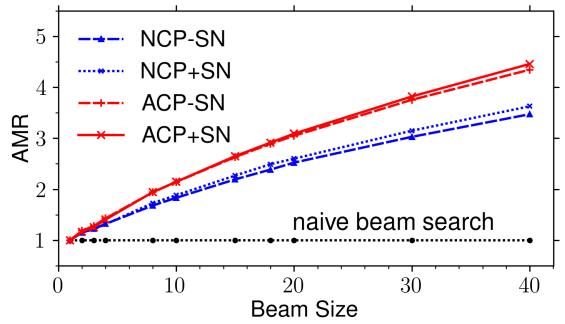


Figure 3: AMR comparison on the MT03 test dataset. Decoding the MT03 test dataset on a single GeForce GTX TITAN X GPU server under the different searching settings. y-axis represents the AMR on the test dataset in the whole searching process and x-axis indicates beam size. Unsurprisingly, we got exactly the same results on the CPU server, not shown here.

six methods. We used the pre-trained model to translate the test dataset on a single GeForce GTX TITAN X GPU server. Beam size varies from 1 to 40, values are included in the set $\{1, 2, 3, 4, 8, 10, 15, 18, 20, 30, 40\}$. For each beam size, six different searching settings were applied to translate the test dataset respectively. The curves of the AMRs during the decoding on the MT03 test dataset under the proposed methods are shown in Figure 3. Note that the AMR values of **NBS** are always 1 whether there is **SN** or not.

Comparing the curves in the Figure 3, we could observe that the naive beam search does not conduct any merging operation in the whole searching process, while the average merging rate in the cube pruning almost grows as the beam size increases. Comparing the red curves to the blue ones, we can conclude that, in any case of beam size, the AMR of the accelerated cube pruning surpasses the basic cube pruning by a large margin. Besides, self-normalization could produces the higher average merging rate comparing to the counterpart without self-normalization.

4.6 Comparison on the GPU Server

Intuitively, as the value of the AMR increases, the search space will be reduced and computation efficiency improves. We compare the two proposed searching strategies and the naive beam search in two conditions (with self-normalization and without self-normalization). Figure 4 demonstrates the results of comparison between the proposed searching methods and the naive beam search baseline in terms of search quality and search efficiency under different beam sizes.

By fixing the beam size and the dataset, we compared the changing trend of BLEU scores for the three distinct searching strategies under two conditions. Without self-normalization, Figure 4a shows the significant improvement of the search speed, however the BLEU score drops about 0.5 points. We then equipped the search algorithm with self-normalization. Figure 4b shows that the accelerated cube pruning search algorithm only spend about one-third of the time of the naive beam search to achieve the best BLEU score with beam size 30. Concretely, when the beam size is set to be 30, **ACP+SN** is 3.3 times faster than the baseline on the MT03 test dataset, and both performances are almost the same.

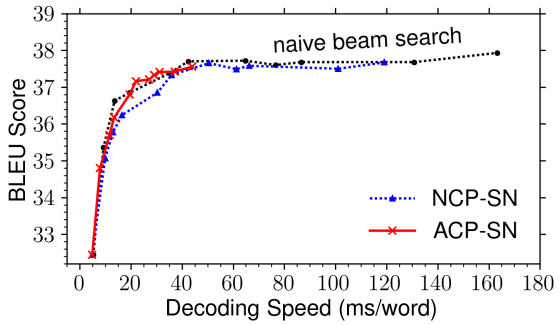
4.7 Comparison on the CPU Server

Similar to the experiments conducted on GPUs, we also translated the whole MT03 test dataset on the CPU server by using all six search strategies under different beam sizes. The trends of the BLEU scores over those strategies are shown in Figure 5.

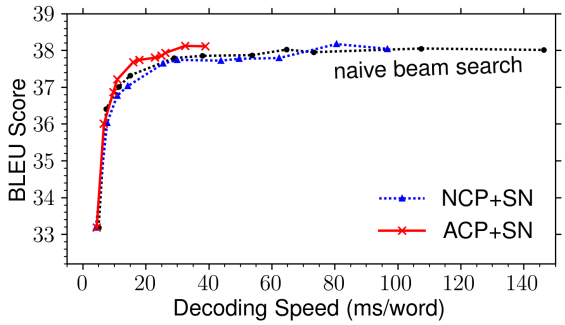
The proposed search methods gain the similar superiority on CPUs to that on GPUs, and the decoding speed is obviously slower than that on GPUs. From the Figure 5a, we can also clearly see that, compared with the **NBS-SN**, **NCP-SN** only speeds up the decoder a little, **ACP-SN** produces much more acceleration. However, when we did not introduce self-normalization, the proposed search methods will also result in a loss of about 0.5 BLEU score. The self-normalization made the **ACP** strategy faster than the baseline by about $3.5\times$, in which condition the **NBS+SN** got the best BLEU score 38.05 with beam size 30 while the **ACP+SN** achieved the highest score 38.12 with beam size 30. The results could be observed in Figure 5b. Because our method is on the algorithmic level and platform-independent, it is reasonable that the proposed method can not only perform well on GPUs, but also accelerate the decoding significantly on CPUs. Thus, the accelerated cube pruning with self-normalization could improve the search quality and efficiency stably.

4.8 Decoding Time

In this section, we only focus on the consuming time of translating the entire MT03 test dataset.

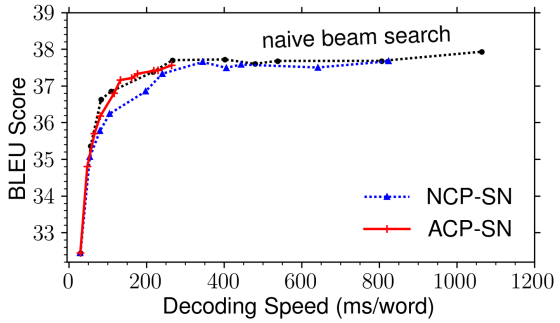


(a) BLEU vs. decoding speed, without self-normalization

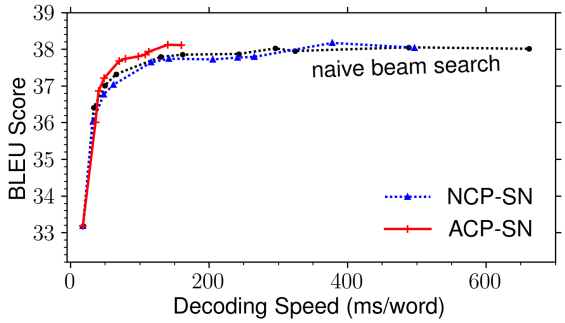


(b) BLEU vs. decoding speed, with self-normalization

Figure 4: Comparison among the decoding results of the MT03 test dataset on the single GeForce GTX TITAN X GPU server under the three different searching settings. y-axis represents the BLEU score of translations, x-axis indicates that how long it will take for translating one word on average.



(a) BLEU vs. decoding speed, without self-normalization



(b) BLEU vs. decoding speed, with self-normalization

Figure 5: Comparison among the decoding results of the MT03 test dataset on the single AMD Opteron(tm) Processor under the three different searching settings. y-axis represents the BLEU score of translations, x-axis indicates that how long it will take for translating one word on average.

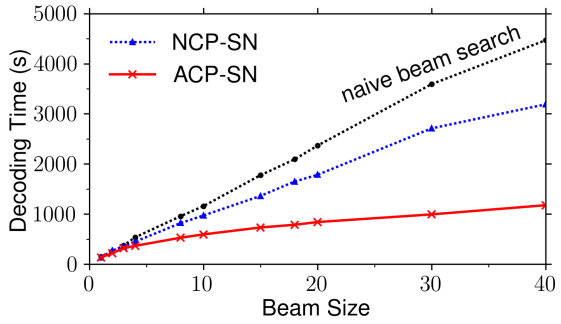
Under the two conditions, we calculated the times spent on translating the entire test dataset for different beam sizes, then draw the curves in Figure 6 and 7. From the Figure 6a and 6b, we could observe that accelerated cube pruning algorithm speeds up the decoding by about $3.8\times$ on GPUs when the beam size is set to 40. Figure 7a and 7b show that the accelerated cube pruning algorithm speeds up the decoding by about $4.2\times$ on CPU server with the beam size 40.

5 Related Work

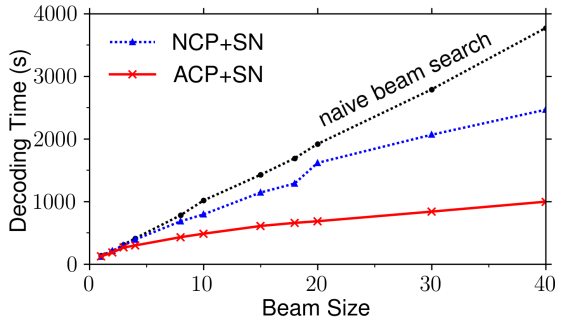
Recently, lots of works devoted to improve the efficiency of the NMT decoder. Some researchers employed the way of decreasing the target vocabulary size. Jean et al. (2015) improved the decoding efficiency even with the model using a very large target vocabulary but selecting only a small subset of the whole target vocabulary. Based on the work of Jean et al. (2015), Mi et al. (2016b) introduced sentence-level and batch-level vocabularies as a very small subset of the full output vocabulary, then predicted target words only on this small vocabulary, in this way, they only lost 0.1 BLEU points, but reduced target vocabulary substantially.

Some other researchers tried to raise the efficiency of decoding from other perspectives. Wu et al. (2016) introduced a coverage penalty α and length normalization β into beam search decoder to prune hypotheses and sped up the search process by $30\%\sim40\%$ when running on CPUs. Hu et al. (2015) used a priority queue to choose the best hypothesis for the next search step, which drastically reduced search space.

Inspired by the works of Mi et al. (2016b) and Huang and Chiang (2007), we consider pruning hypothesis in NMT decoding by using cube pruning algorithm, but unlike traditional SMT decoding where dynamic programming was used to merge equivalent states (e.g., if we use phrase-

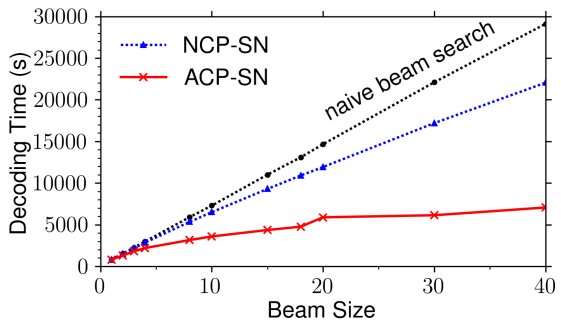


(a) Time spent on translating MT03 test dataset for different beam sizes without self-normalization

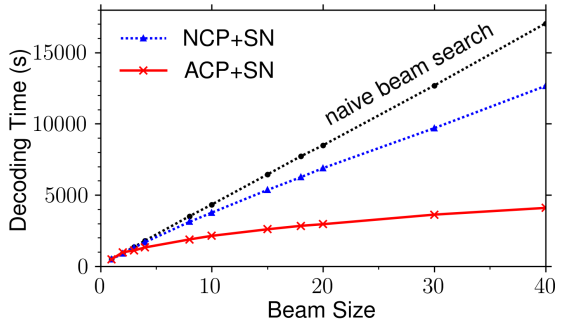


(b) Time spent on translating MT03 test dataset for different beam sizes with self-normalization

Figure 6: Comparison among the decoding results of the MT03 test dataset on the single GeForce GTX TITAN X GPU server under the three different searching settings. y-axis represents the BLEU score of translations, x-axis indicates that how long it will take for translating one word on average.



(a) Time spent on translating MT03 test dataset for different beam sizes without self-normalization



(b) Time spent on translating MT03 test dataset for different beam sizes with self-normalization

Figure 7: Comparison among the decoding results of the MT03 test dataset on the single AMD Opteron(tm) Processor under the three different searching settings. y-axis represents the BLEU score of translations, x-axis indicates that how long it will take for translating one word on average.

based decoding with trigram language model, we can merge states with same source-side coverage vector and same previous two target words). However, this is not appropriate for current NMT decoding, since the embedding of the previous target word is used as one input of the calculation unit of each step in the decoding process, we could group equivalence classes containing the same previous target word together.

6 Conclusions

We extended cube pruning algorithm into the decoder of the attention-based NMT. For each step in beam search, we grouped similar candidates in previous beam into one or more equivalence class(es), and bad hypotheses were pruned out. We started searching from the upper-left corner in each equivalence class and spread out until enough candidates were generated. Evaluations show that, compared with naive beam search, our method could improve the search quality and efficiency to a large extent, accelerating the NMT decoder by 3.3× and 3.5× on GPUs and CPUs, respectively. Also, the translation precision could be the same or even better in both situations. Besides, self-normalization is verified to be helpful to accelerate cube pruning even further.

Acknowledgements

We thank the three anonymous reviewers for their comments, Kai Zhao and Haitao Mi for ugges-tions. This work is supported in part by NSF IIS-1817231 & IIS-1656051, and is also supported in part by National Natural Science Foundation of China (Nos. 61472428 & 61662077).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *ICLR 2015*.

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. 2013. Audio chord recognition with recurrent neural networks. In *ISMIR*, pages 335–340. Citeseer.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics.

David Chiang. 2007. Hierarchical phrase-based translation. *computational linguistics*, 33(2):201–228.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1370–1380, Baltimore, Maryland. Association for Computational Linguistics.

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 961–968, Sydney, Australia. Association for Computational Linguistics.

Jonas Gehring, Michael Auli, David Grangier, and Yann Dauphin. 2017a. A convolutional encoder model for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 123–135, Vancouver, Canada. Association for Computational Linguistics.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017b. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252, International Convention Centre, Sydney, Australia. PMLR.

Alex Graves. 2012. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.

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

Xiaoguang Hu, Wei Li, Xiang Lan, Hua Wu, and Haifeng Wang. 2015. Improved beam search with constrained softmax for nmt. *Proceedings of MT Summit XV*, page 297.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pages 53–64, Stroudsburg, PA, USA. Association for Computational Linguistics.

Liang Huang and David Chiang. 2007. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic. Association for Computational Linguistics.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1–10, Beijing, China. Association for Computational Linguistics.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent convolutional neural networks for discourse compositionality. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, pages 119–126, Sofia, Bulgaria. Association for Computational Linguistics.

Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. 2016a. Coverage embedding models for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 955–960, Austin, Texas. Association for Computational Linguistics.

Haitao Mi, Zhiguo Wang, and Abe Ittycheriah. 2016b. Vocabulary manipulation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 124–129, Berlin, Germany. Association for Computational Linguistics.

Franz Josef Och and Hermann Ney. 2004. The alignment template approach to statistical machine translation. *Computational Linguistics*, 30(4):417–449.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA. PMLR.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

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*.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.