Improved Beam Search with Constrained Softmax for NMT

Xiaoguang Hu huxiaoguang@baidu.com
Wei Li liwei08@baidu.com
Xiang Lan lanxiang@baidu.com
Hua Wu wu_hua@baidu.com
Haifeng Wang wanghaifeng@baidu.com
Baidu Inc, Beijing, China

Abstract

We propose an improved beam search decoding algorithm with constrained softmax operations for neural machine translation (NMT). NMT is a newly emerging approach to predict the best translation by building a neural network instead of a log-linear model. It has achieved comparable translation quality to the existing phrase-based statistical machine translation systems. However, how to perform efficient decoding for NMT is still challenging, especially for commercial systems which provide real-time translation service. Unlike the standard beam search algorithm, we use a priority queue to choose the best hypothesis for the next search, which drastically reduces search space. Another time consuming factor is the softmax operation in the output layer because of the large target vocabulary size. To solve this problem, we introduce a limited word set of translation candidates to greatly reduce the computation complexity. Our experiments show that, under the GPU environment, our method achieves a speed about 3.5 times faster than the well optimized baseline system without sacrificing the translation quality. Our method translates about 117 words per second, beating the real-time translation requirements for practical MT systems.

1 Introduction

Neural network models have become increasingly popular in recent machine translation tasks. Initially, the neural networks were designed for language models, such as neural network language model (Bengio et al., 2003) and recurrent neural network language model (Mikolov et al., 2010). Devlin et al. (2014) proposed a neural network joint model to improve translation by considering the source contexts jointly. The above work integrated the neural network models into the traditional translation decoders as additional features in a log-linear framework (Och and Ney, 2002). Rather than applying a neural network as a part of the existing system, Cho et al. (2014) proposed a whole new RNN Encoder-Decoder approach which generates the target translation with a neural network directly. Bahdanau et al. (2014) extended the above approach by allowing an RNN model to automatically (soft-)search for parts of a source sentence to predict a target word, and achieved a translation performance comparable to the existing state-of-the-art phrase-based systems (Koehn et al., 2003).

Although NMT shows high potential, its decoding efficiency is still challenging. Cho et al. (2014) implemented a standard beam search decoding algorithm (Koehn, 2004) for an RNN

Encoder-Decoder system published as GroundHog [1]. The beam search algorithm successfully reduces the search space from exponential size to polynomial size, and it is able to translate about ten words per second. Even though the speed is acceptable for most research tasks, it is not yet efficient enough to meet the requirement of commercial systems for providing real-time translation service. Huang and Chiang (2007) proposed the forest rescoring algorithm which succeed to accelerate the decoding by using cube pruning. The main improvement comes from the reduction of language model calculation which is the most time-consuming part in both phrase-based and syntax-based system. However, there is no separable language model feature in NMT, which makes it difficult to use the cube pruning algorithm in NMT. Furthermore, we found that one of the most time-consuming part in NMT decoding is the softmax operation over the output layer because of the large vocabulary size. The hierarchical softmax (Mikolov et al., 2011) and Noise-contrastive estimation (Gutmann and Hyvarinen, 2010) succeed to accelerate softmax operation effectively both for training and predicting in the RNN language model. However, they don't work well for the decoding application. In the language model task, we only need to compute the probability of the certain next word which we have known in advance. But in the decoding task, we have to compute all probabilities of candidates in the target vocabulary to know which one is the best candidate.

In this work, in order to make the NMT system meet the real-time translation requirements, we propose an improved beam search decoding algorithm with constrained softmax operation over the output layer for the RNN Encoder-Decoder machine translation. We found that many hypotheses can be pruned safely without affecting the final translation result, so that we apply a priority queue to choose the best hypothesis to be extended. Furthermore, we build a limited word set of translation candidates which is used to effectively reduce the computation complexity in softmax operation. The experimental results show that, our optimized algorithm achieves a speed about 3.5 times faster than well optimized baseline system at the same level of translation quality. The improved beam search with priority queue helps to reduce about 37.6% hypothesis extension operations which results in 1.7 times speedup. The constrained softmax strategy succeeds to reduce the output layer size from 30,000 to about 300 which leads to another 2.1 times speedup.

In section 2, we introduce the standard beam search algorithm of RNN Encoder-Decoder approach in detail. In section 3, we present our improved beam search algorithm with constrained softmax operation. We describe our experimental settings in section 4 and report the experimental results in section 5. Finally, we summarize our conclusions and future work in section 6.

2 Neural Machine Translation

The end-to-end neural machine translation has been newly proposed in recent years. We first introduce the general RNN Encoder-Decoder translation framework (Bahdanau et al., 2014), and then we elaborate the standard beam search decoding algorithm for NMT in detail.

2.1 Preliminary Definitions

In the neural machine translation framework, a source sentence is represented as a sequence of 1-of-K coded word vectors

$$\mathbf{x} = (x_1, \ldots, x_{T_x}), x_i \in \mathbb{R}^{K_x}$$

and a target sentence is also represented as a sequence of 1-of-K coded word vectors

$$\mathbf{y} = (y_1, \ldots, y_{T_y}), y_i \in \mathbb{R}^{K_y}$$

[1] https://github.com/lisa-groundhog/GroundHog

where K_x and K_y are the vocabulary sizes of source and target languages respectively. T_x and T_y are the lengths of source and target sentences respectively.

Given an input sentence $\mathbf{x}$ and $\mathbf{y}$, we define some important concepts and symbols in advance.

- *encoder*: RNN to convert the input sentence $\mathbf{x}$ into *a sequence of hidden states* $\mathbf{h}$.
- *decoder*: RNN to predict the probability distribution over $y_i \in \mathbb{R}^{K_y}$ from the given *context vector* and *hidden states*.
- *hidden state*: a vector from RNN hidden layer at a certain step t. Where t is defined as the number of accepted or generated words right now. We use h_t to represent the hidden state for the encoder, and s_t for the decoder.
- *context vector*: a vector c_t used to predict translation probability at step t generated from $\mathbf{h}$ and s_t.
- *hypothesis*: a data structure used to store search states includes s_{t-1}, c_t, y_t, and the translation probability at step t.
- *extend*: an operation to construct several new hypotheses from a hypothesis with translation candidates generated from the decoder.

2.2 Encoder-Decoder Framework

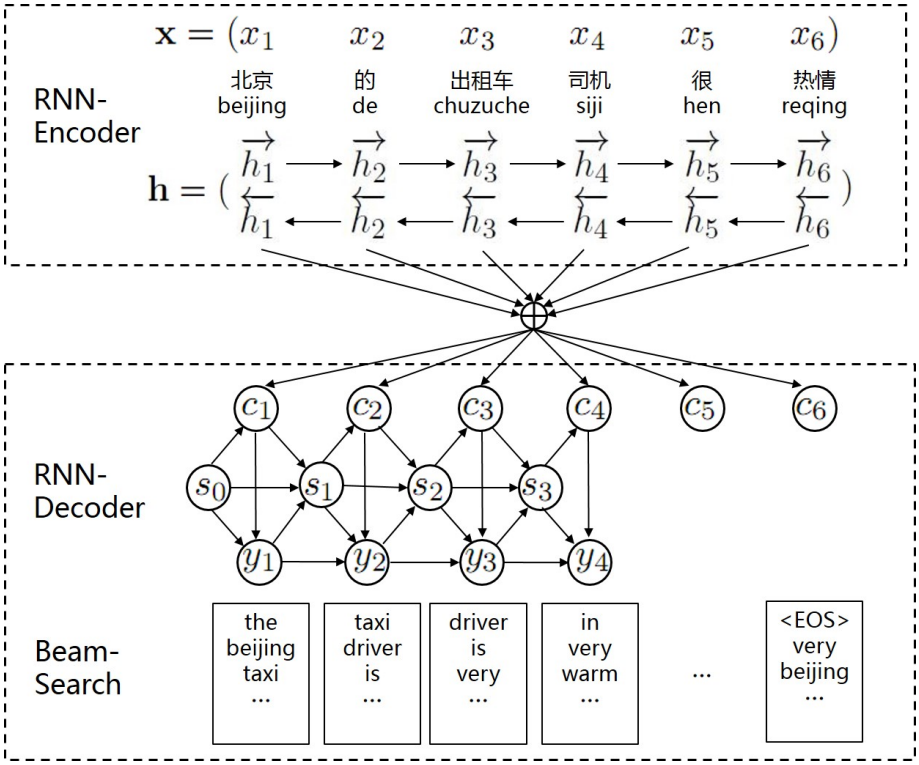


Figure 1: RNN Encoder-Decoder Framework

An RNN Encoder-Decoder translation system often contains an encoder and a decoder. Figure 1 shows how this system works to translate a Chinese sentence into an English sentence.

Given a Chinese sentence $\mathbf{x}$ = *"beijing de chuzuche siji hen reqing"* which means *"the taxi driver in beijing is enthusiastic"*, the encoder converts $\mathbf{x}$ into a series of hidden states $\mathbf{h} = (h_1, \ldots, h_6)$ sequentially via equation (1).

$$h_j = \begin{cases} f(x_j, h_{j-1}) & j > 0 \\ 0 & j = 0 \end{cases} \tag{1}$$

where $h_j \in \mathbf{R}^n$ is a hidden state of the encoder after reading input word x_j, n is the hidden layer size of the encoder, and f is a nonlinear function which is sometimes complicated such as in the long short term memory model (Hochreiter and Schmidhuber, 1997). Furthermore, a bidirectional RNN (Schuster and Paliwal, 1997), which consists of forward and backward RNNs, is used here.

Given the encoder hidden states $\mathbf{h}$ generated from $\mathbf{x}$, and all the previously predicted history words $(y_1, \ldots, y_{i-1})$, the decoder predicts the probability distribution over $y_i \in \mathbb{R}^{K_y}$. For a translation candidate $\mathbf{y}$, the translation probability is computed in equation (2).

$$p(\mathbf{y}) = \prod_{i=1}^{T} p(y_i | y_1, \ldots, y_{i-1}, \mathbf{h}) \tag{2}$$

The translation history $(y_1, \ldots, y_{i-1})$ is represented by hidden state s_{i-1} and y_{i-1}. The encoder hidden states $\mathbf{h}$ is represented as c_i. The output probability is computed in equation (3).

$$p(y_i | y_1, \ldots, y_{i-1}, \mathbf{h}) = g(s_{i-1}, y_{i-1}, c_i) \tag{3}$$

where g is a non-linear function, of which we will give an example later in section 3.2.

2.3 Standard Beam Search

We describe the beam search algorithm implemented in GroundHog as in Algorithm 1. Given the encoder, decoder and input sentence $\mathbf{x}$, we try to find the best translation $\widehat{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. A group of stacks are used to store hypotheses during searching. Beam size N is used to control the search space by extending only the top-N hypotheses in the current stack. With the above settings, the translation $\mathbf{y}$ is generated word by word from left to right. We define *complete hypothesis* as hypothesis which outputs *EOS*, where *EOS* is a special target word indicating the end of sentence.

As compared to the standard decoding algorithm (Koehn, 2004) in phrase-based machine translation, there are some differences. Firstly, a hypothesis is always of less score than the hypothesis from which it extended. Because after every extension in NMT, the total score will always be multiplied with a probability which is less than 1. But in phrase-based system, the score of a extension is not always less than 1 because the features such as word penalty score and phrase penalty score may be greater than 1 in the log-linear framework. Secondly, the stop conditions are different. The phrase-based system will stop searching when all the words in the source sentence are translated. In NMT, there is no exactly word alignment information to tell which word is translated or not. It will stop searching when it has generated N *complete hypotheses*.

3 Improved Beam Search

In this section, we introduce our improvement over the standard beam search algorithm for NMT. We observed that not all the hypothesis extensions are necessary, and some extensions

Algorithm 1 Standard Beam Search

```
1: function StandardSearch(enc, dec, x, y)
2:     define stacks s
3:     define set c
4:     create initial hypo and put it into s[0]
5:     i ← 0
6:     N ← beam size
7:     while s[i] ≠ ∅ do
8:         for all h ∈ s[i] do
9:             extend new hypos from h
10:            put new hypos into s[i + 1]
11:        end for
12:        prune s[i + 1] to keep N − c.size hypos
13:        move complete hypo in s[i + 1] to c
14:        i ← i + 1
15:    end while
16:    y ← trace back from best h ∈ c
17: end function
```

of inferior hypotheses could be skipped safely without affecting the translation results. Furthermore, we found that the operation to compute the softmax score over the output layer is very slow because the target vocabulary size K_y is often large. We can accelerate beam search by reducing the vocabulary size in advance with translation knowledge learned from the phrase-based model.

3.1 Hypotheses Extension with Priority Queue

The standard beam search algorithm extends the hypotheses from left to right stack by stack as shown in Figure 2 part (a). At certain step t, all the hypotheses in the stack before t have to be extended. Actually, not all the extended hypotheses will succeed to generate a *complete hypothesis* with EOS, because plenty of extended hypotheses will be pruned in the future.

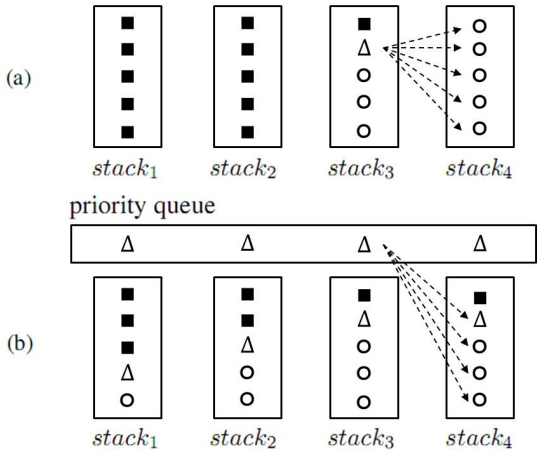


Figure 2: (a) standard beam search (b) improved beam search; ■ extended hypothesis, △ best hypothesis to be extended, ○ unextended hypothesis

In our improved beam search algorithm, we use a priority queue q to record the best hypotheses of each stack as shown in Figure 2 part (b). In a stack, there are at most N hypotheses, which are ranked in descending order by their translation probabilities. If there is any unextended hypothesis in a stack, the one with the highest translation probability will be inserted into queue q. With above settings, the improved beam search works as shown in Algorithm 2.

- Firstly, we define the stacks s and priority queue q to manage hypotheses, and we create the initial hypothesis and insert it into $s[0]$ and q.
- Secondly, we select the best hypothesis in the queue q which is the initial hypothesis by now, and extend at most N new hypotheses from the initial hypothesis. We insert new hypotheses into $s[1]$ in descending order, then insert the best hypothesis in $s[1]$ into queue q. We update the queue q by eliminating the initial hypothesis.
- After that, we select the best hypothesis h in the queue q which is of stack index i, and extend at most N new hypotheses from h. We insert new hypotheses into stack $s[i+1]$ in descending order, and prune the inferior hypotheses to make sure the number of hypotheses is less than N. For stack $s[i]$ and $s[i+1]$, we update the hypotheses in queue q to make sure that q contains only the best unextended hypothesis of each stack. We adjust the queue q to be in the priority order.
- We repeat the process until the best hypothesis h to be extended in the queue is a *complete hypothesis*. It's safe to stop searching and output the best translation $\widehat{\mathbf{y}}$ by tracing back from h, because h is associated with the highest translation probability among any other unextended hypotheses by now, and no hypothesis in the searching space with beam size N will get higher translation probability than h in the future since $p(y_i|y_1,\ldots,y_{i-1},\mathbf{h}) \leq 1$.

Instead of extending the hypothesis stack by stack sequentially, our improved beam search algorithm only extend the best unextended hypothesis in all stacks. The standard algorithm stops searching after generating N *complete hypotheses*, while our algorithm stops when a *complete hypothesis* h is selected to be extended. Because h is better than any other hypotheses in q, and the hypothesis in q is better than any other unextended hypotheses in the same stack, and the unextended hypotheses in the stack is better than any other hypotheses extended from it in the future. Therefore our algorithm extends less hypotheses without changing translation result. All the unextended hypotheses inferior to h will not be extended. Considering other breadth-first searching algorithms with a queue such as the classical Dijksta's algorithm, the size of the queue grows exponentially. In our algorithm, there is at most one hypothesis in the priority queue for each stack, and it costs less memory and more efficient to manage the queue.

Furthermore, we find that it's unfair to compare the translation probabilities of hypotheses in the priority queue directly. Because the hypotheses with longer translation often tend to get lower probability. We design a parameter α to alleviate the influence of translation length, which is similar to the weight of word penalty feature in the phrase based system.

$$\widetilde{p}(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}|\mathbf{x})^{\frac{1}{1+\alpha\times(i-1)}} \tag{4}$$

Where $i \geq 1$ is the stack index of current hypothesis. If $\alpha = 0$, then $\widetilde{p} = p$; if $\alpha = 1$ the $\widetilde{p}$ is the geometry average of p over i. The lower the α is, the less hypothesis will be pruned. After normalization, the translation result will be changed.

3.2 Constrained Softmax Over Output Layer

For a hypothesis extension, the decoder predicts the next word probability $p(y_i|s_{i-1},y_{i-1},c_i)$ by executing RNN feed forward operation. This procedure comprises some matrix multiplication operations and a final softmax operation over the output layer as shown in Figure 3. Here,

Algorithm 2 Improved Beam Search

```
function IMPROVEDSEARCH(enc, dec, x, y)
    define stacks s
    define priority queue q
    create initial hypo and put it into s[0], q
    N ← beam size
    h ← find the best hypo in q
    while h is not complete hypo do
        i ← get stack index of h
        extend new hypos from h
        put new hypos into s[i + 1]
        prune s[i + 1] to keep N hypos
        UPDATEQUEUE(q, s[i])
        UPDATEQUEUE(q, s[i + 1])
        h ← find the best hypo in q
    end while
    y ← trace back from h
end function
function UPDATEQUEUE(q,s)
    h ← q ∩ s                                   ▷ hypo both in q and s
    if ∃ best unextended hypo uh ∈ s then
        replace h with uh in q
    else                                        ▷ no unextended hypo in s
        delete h from q
    end if
    adjust q to be a priority queue
end function
```

we compute word probability using the same g function proposed by Bahdanau et al. (2014) as follows.

$$p(y_i|s_{i-1}, y_{i-1}, c_i) \propto \exp\left(y_i^{\intercal} W_o u_i\right) \tag{5}$$

$$u_i = \left[\max\left(\widetilde{u}_{i,2j-1}, \widetilde{u}_{i,2j}\right)\right]_{j=1,\ldots,l}^{\intercal} \tag{6}$$

$$\widetilde{u}_i = U_o s_{i-1} + V_o E y_{i-1} + C_o c_i \tag{7}$$

Where $W_o \in \mathbb{R}^{K_y \times l}, U_o \in \mathbb{R}^{2l \times n}, V_o \in \mathbb{R}^{2l \times m}$ and $C_o \in \mathbb{R}^{2l \times 2n}$ are weight matrices. And l is the size of maxout hidden layer t_i; n is the size of hidden layer s_i; m is the word embedding dimensionality of Ey_{i-1}.

Actually, we find that the matrix multiplication to compute the softmax over the output layer is often the most time-consuming operation. For most translation tasks, the size of output layer ($K_y \geq 10,000$) is usually over ten times larger than sizes of other layers ($l, n, m \leq 1,000$). The computation time complexity is close to $O(l \times K_y)$. If we can reduce K_y, the hypothesis extension will become faster. RNN with output layer factorized by class layer (Mikolov et al., 2011) and Noise-contrastive estimation (Gutmann and Hyvarinen, 2010) is able to accelerate softmax operation effectively both for training and predicting in RNNLM application. However, unlike estimating the translation probability for only one certain target word y_i, we have to estimate the probability distribution for all target words to get the best candidate.

Given an input source sentence $\mathbf{x}$, we note that the number of possible translation candidates T_x is much smaller compared to the target vocabulary size K_y. Instead of calculating the

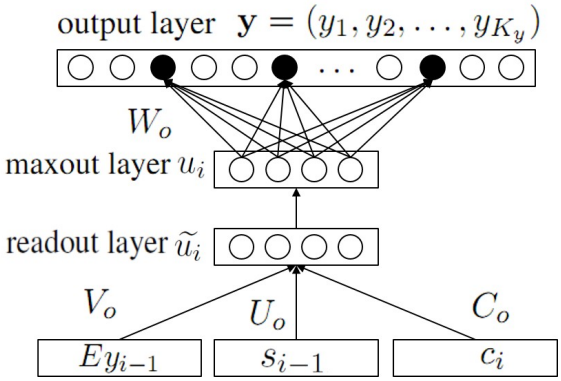


Figure 3: Deep Output Layer of RNN Decoder

translation probability for every word in the target vocabulary, we only calculate the probability for a limited set of target candidates which are relevant to the given $\mathbf{x}$. The problem is how to get such a candidate set. Intuitively, we can make use of the word alignment information generated with IBM model (Brown et al., 1993). And we found that there are often too many candidates especially for the common words, which makes the improvement limited.

Actually, we can get more accurate translation candidates from the phrase pairs of the phrase-based translation model (Koehn et al., 2003). Firstly, we train a phrase-based translation model with the word aligned bilingual sentence pairs. Then we segment the input sentence $\mathbf{x}$ into a number of continuous source phrases. Next, we search the corresponding target phrases for all source phrases from the phrase-based translation model, namely phrase table. After that, we build a candidate word set by enumerating all the words in the target phrases. Finally, we constrain the softmax operation with the candidate word set for all the hypothesis extension of $\mathbf{x}$.

One disadvantage of above approach is that it is too costly to load the whole phrase table into memory. Fortunately, we can reduce the memory cost by pruning phrase table carefully. The phrase table filtering techniques have been widely used in machine translation. Instead of considering both the phrase coverage and translation features in the filtering technique, our system only considers the word coverage, which is much easier to implement. We investigated the histogram pruning (Zens et al., 2012) and length-based pruning method to reduce the memory used in our system. For histogram pruning, we preserves the X target phrases with highest probability for each source phrase. For length-based pruning, we prune the phrase pairs which contain more than X words in source or target side. Where X is the threshold. We found that even the basic length-based pruning method works well enough in our task.

4 Experimental Settings

In this section, we evaluate the improved beam search algorithm on the task of Chinese-to-English translation.

4.1 Dataset

In this paper, we use Chinese-to-English bilingual corpora from LDC which is a subset of corpus for NIST08 task, containing 67M Chinese words and 74M English words, to train our models. No monolingual corpus are used to help the model training. The NIST MT03 Chinese-to-English test set is used as the development set, and NIST MT08 Chinese-to-English test set is used to evaluate our translation results. The development set is used to choose the best RNN

model in history, because the performance of RNN model will fluctuate during training.

4.2 Toolkits

The open source SMT system Moses [2] (Koehn et al., 2007) is used to train a phrase-based machine translation system. The phrase table trained with Moses will be used to constrain the softmax translation. We use the open source RNN Encoder-Decoder toolkits GroundHog which is implemented with Theano (Bergstra et al., 2010; Bastien et al., 2012) to train a neural machine translation model. It is re-implemented with C++ in the GPU Environment and named NetTrans, which is well optimized and faster than GroundHog which is implemented in Python. Given the same model, there is no translation differences between GroundHog and NetTrans. We will report our experimental results on NetTrans instead of GroundHog.

4.3 Settings

We train a phrase-based model from the bilingual corpus with Moses using "grow-diag-final" word alignments. We train two RNN encoder-decoder models (set K_x=K_y=30,000 and K_x=K_y=60,000 separately) with GroundHog toolkit written with Theano by using the default settings proposed by Bahdanau et al. (2014). A multilayer network with a single maxout (Goodfellow et al., 2013) hidden layer is used to compute the conditional probability of each target word. The size of a hidden layer n is 1000, the word embedding dimensionality m is 620 and the size of maxout hidden layer in the deep output l is 500. We use a minibatch stochastic gradient descent (SGD) algorithm together with Adadelta (Zeiler, 2012) to train all the RNN models. Each SGD update direction is computed using a mini-batch of 50 sentences. The training time for each model is approximately 22 days. We set beam size $N = 10$ for beam search decoding. We run both the training and decoding programs on a machine with GPU cards (NVIDIA Tesla K10).

5 Results and Analysis

5.1 Baseline

In this section, we report the translation results of Moses, GroundHog and NetTrans. Our experimental results in table 1 shows that the NetTrans runs about 3 times faster than GroundHog with the same translation results. GroundHog is implemented with Theano, which is a very flexible toolkit for general deep learning tasks. NetTrans is well designed especially for the RNN Encoder-Decoder framework on the GPU architecture with pure C++. In NetTrans, we execute the operations between the same matrices continuously to avoid additional GPU initialization and we use memory pool to avoid allocating memory frequently and store the matrix in memory continuously.

System	BLEU	Speed (words/sec)
Moses	24.19	-
GroundHog	24.50	11.2
NetTrans	24.50	33.4

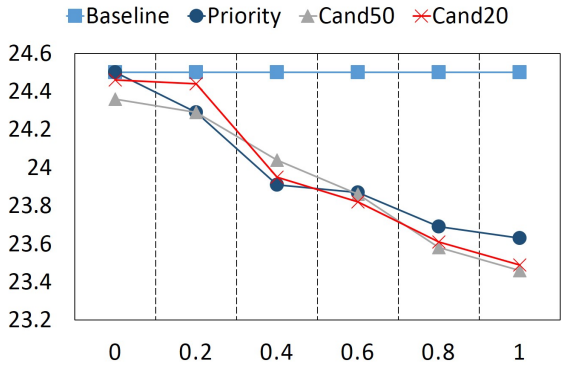
Table 1: Baseline Systems

5.2 Improved Beam Search with Constrained Softmax

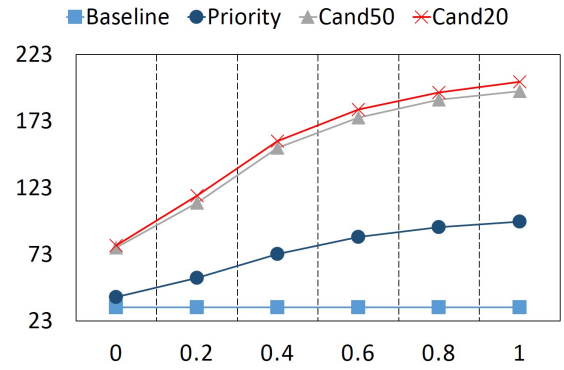
We describe our experimental settings as follows. *Baseline* is the standard beam search algorithm implemented in NetTrans. *Priority* represents the improved beam search algorithm with

[2] http://www.statmt.org/moses/

priority queue, and the parameter α ($0 \leq \alpha \leq 1$) is set as in Equation (4). *CandX* represents the histogram pruning with threshold X. Our experimental results under different *Alpha* ($0 \leq \alpha \leq 1$) are shown in Figure 4. Curve *Priority* tells the results of improved beam search with the priority queue. Curves *Cand50* [3], *Cand20* [4] report the results of constrained softmax with different *CandX*. As shown in Figure 4, both improved beam search and constrained softmax can accelerate the decoding speed effectively.



(a) BLEU Score with Different α



(b) Translation Speed (words/second) with Different α

Figure 4: Results of Improved Beam Search

As shown in Table 2, our method achieves a speed more than 3.5 times faster than the baseline at the same level of translation quality with α=0.2. The improved beam search with priority queue achieves in 1.7 times speedup by reducing 37.6% hypothesis extension operations. The hypothesis extension times both for our method and the baseline method (ext_o and ext_b) are counted respectively. The *pruning ratio* is computed with equation $(ext_b - ext_o)/ext_b$. The results show that the speed acceleration with different α is roughly consistent with the *pruning ratio*. The constrained softmax strategy leads to another 2.1 times speedup by reducing the output layer size from 30,000 to about 300. Theoretically, the speedup should be $K_y \times n/max\{n \times n, K_y \times n\}$ and thus 30 times faster with hidden layer dimension $n = 1000$. Actually, the standard softmax costs about half of the decoding time under the GPU environment. After constraining it with a candidate word set, it makes the constrained softmax no

[3] We use the default settings in Moses that performs well for phrase-based MT with *CandX*=50.

[4] We use a reduced candidate set to examine how different candidate sizes affects translation quality

longer the bottleneck of translation. At the same level of translation speed, our method performs more than 1 BLEU scores better than the method to reduce the beam size N from 10 to 3 directly.

System	BLEU	Speed	Ratio
Baseline	24.50	33.4	-
α=0	24.50	41.11	14.6%
α=0.2	**24.29**	**55.4**	**37.6%**
α=0.2,Cand50	24.29	111.6	42.8%
α=0.2,Cand20	**24.44**	**117.2**	**45.1%**
Beam Size=3	23.37	104.4	66.6%

Table 2: Optimized Search with Pruning Ratio

5.3 Constrained Candidates Selection

We define *PhraseX* as the length-based pruning with threshold X. *Cands* is the average word size of translation candidates set for a given sentence. *Coverage* represents the recall ratio of the real candidates generated by the baseline RNN system (α=0.2). *Memory* means the additional memory used to store the whole phrase table for candidates selection.

As shown in Table 3, the *Coverage* decreases when reducing *CandX*, while *Coverage* almost remains the same when reducing *PhraseX*. For example, when we only select those phrase pairs whose length is less than or equal to 2 (*PhraseX*=2), the coverage is almost the same as that when *PhraseX*=7. We find that our method works well with very limited memory (170MB) in setting *Cand50* and *Phrase2*. This result is consistent with that for phrase-based SMT system, where phrase pairs with 2 or 3 words are frequently used and the target candidates of the source phrase is set to 50 by default. Moreover, with such setting, the translation quality is not degraded in BLEU score metrics. This result shows that our method for constrained candidate selection is effective to speed up the decoder without degrading of translation quality and with limited memory.

Selection	Cands	Coverage	Memory
Cand5,Phrase7	83.8	87.3%	2.26GB
Cand10,Phrase7	126.6	91.4%	2.29GB
Cand20,Phrase7	195.4	94.0%	2.31GB
Cand50,Phrase7	**352.5**	**96.2%**	**2.33GB**
Cand50,Phrase3	350.9	96.2%	571MB
Cand50,Phrase2	**337.6**	**96.1%**	**170MB**
Cand50,Phrase1	276.3	95.3%	12MB

Table 3: Constrained Candidates Selection

5.4 The Effect of Vocabulary Size

We train a RNN Encoder-Decoder model with the target word size set to K_y=60,000, to examine how our method affects the decoding speed under different settings. As shown in Table 4, as the vocabulary size becomes larger, our approach performs even better, achieving a speed about 5.6 times faster than the baseline. The result is consistent with our intuition.

System	BLEU	Speed (words/sec)
Baseline	24.35	21.3
α=0	24.35	27.1
α=0.2	24.19	38.6
Cand50	24.05	113.8
Cand20	24.41	119.8

Table 4: Results of Vocabulary Size = 60,000

6 Conclusion and Future Work

In this paper, we propose an improved beam search decoding algorithm with constrained softmax operation for the neural network translation. It accelerates the translation speed more than 3.5 times compared to the standard beam search algorithm as in our well optimized baseline while keeping the same translation quality. Our experiments show that the improved beam search algorithm and the constrained softmax operation over the output layer are very effective for neural network translation.

In the future, we will try to accelerate the training for NMT using constrained softmax strategy. Furthermore, we will try to reduce the searching errors by selecting more accurate bilingual translation candidates.

Acknowledgments

This work is supported by the National Basic Research Program of China (973 program No. 2014CB340505).

References

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. In *arXiv:1409.0473 [cs.CL]*.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I., Bergeron, A., Bouchard, N., Varde-Farley, D., and Bengio, Y. (2012). Theano: new features and speed improvements. In *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop*.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research 3*, pages 1137–1155.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a cpu and gpu math expression compiler in python. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*.

Brown, P. F., Pietra, S. A. D., Pietra, V. J. D., and Mercer, R. L. (1993). The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoderdecoder approaches. In *Proceedings of Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8)*, pages 103–111, Doha, Qatar.

Devlin, J., Zbib, R., Huang, Z., Lamar, T., Schwartz, R., and Makhoul, J. (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*, pages 1370–1380, Baltimore, Maryland, USA.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1319–1327.

Gutmann, M. and Hyvarinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*.

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

Huang, L. and Chiang, D. (2007). Forest rescoring: Faster decoding with integrated language models. In *Proceedings of ACL*.

Koehn, P. (2004). Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Proceedings of the Sixth Conference of the Association for Machine Translation in the Americas*, pages 115–124.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open source toolkit for statistical machine translation. In *ACL 2007 demonstration session*.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of HLT-NAACL 2003*, pages 127–133.

Mikolov, T., Karafiat, M., Burget, L., Cernocky, J. H., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proceedings of Interspeech*.

Mikolov, T., Kombrink, S., Burget, L., Cernocky, J. H., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *Proceedings of ICASSP*.

Och, F. J. and Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of ACL*.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, pages 45(11), 2673–2681.

Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. In *arXiv:1212.5701 [cs.LG]*.

Zens, R., Stanton, D., and Xu, P. (2012). A systematic comparison of phrase table pruning techniques. page 972983.