Journal of Computer Science and Cybernetics, V.35, N.2 (2019), 147–166
DOI 10.15625/1813-9663/35/2/13233

NEURAL MACHINE TRANSLATION BETWEEN VIETNAMESE AND ENGLISH: AN EMPIRICAL STUDY

HONG-HAI PHAN-VU[1], VIET-TRUNG TRAN[1,*], VAN-NAM NGUYEN[2], HOANG-VU DANG[2], PHAN-THUAN DO[1]

[1]*Hanoi University of Science and Technology (HUST)*
[2]*FPT Technology Research Institute, FPT University*
[*]*trungtv@soict.hust.edu.vn*

Abstract. Machine translation is shifting to an end-to-end approach based on deep neural networks. The state of the art achieves impressive results for popular language pairs such as English - French or English - Chinese. However for English - Vietnamese the shortage of parallel corpora and expensive hyper-parameter search present practical challenges to neural-based approaches. This paper highlights our efforts on improving English-Vietnamese translations in two directions: (1) Building the largest open Vietnamese - English corpus to date, and (2) Extensive experiments with the latest neural models to achieve the highest BLEU scores. Our experiments provide practical examples of effectively employing different neural machine translation models with low-resource language pairs.

Keywords. Neural Machine Translation; Seq2seq; RNN; Attention Mechanism; ConvS2S; Transformer; ByteNet;

1. INTRODUCTION

Machine translation is shifting to an end-to-end approach based on deep neural networks. Recent studies in neural machine translation (NMT) such as [2, 14, 40, 41] have produced impressive advancements over phrase-based systems while eliminating the need for hand-engineered features. Most NMT systems are based on the encoder-decoder architecture which consists of two neural networks. The encoder compresses the source sequences into a real-valued vector, which is consumed by the decoder to generate the target sequences. The process is done in an end-to-end fashion, demonstrated the capability of learning representation directly from the training data.

The typical sequence-to-sequence machine translation model consists of two recurrent neural networks (RNNs) and an attention mechanism [2, 26]. Despite great improvements over traditional models [27, 34, 41] this architecture has certain shortcomings, namely that the recurrent networks are not easily parallelized and limited gradient flow while training deep models.

Recent designs such as ConvS2S [14] and Transformer [40] can be better parallelized while producing better results on WMT datasets. However, NMT models take a long time to train and include many hyper-parameters. There is a number of works that tackle the problem of hyper-parameter selection [5, 33] but they mostly focus on high-resource language pairs data, thus their findings may not translate well to low-resource translation tasks such as English-Vietnamese. Unlike in Computer Vision [17, 20], the task of adapting parameters spaces from one NMT model to other NMT models is nearly impossible [5]. This reason limits researchers and engineers to reach well-chose hyper-parameters and well-trained models.

To date there are several research works on English-Vietnamese machine translation such as [3, 13, 22, 32], using traditional methods with modest BLEU scores. Some newer works such as [18, 25] experimented on the IWSLT English-Vietnamese dataset [6] and showed great potential to improve English-Vietnamese translation tasks using more data and more complex models.

In [31] the authors introduced datasets for bilingual English-Vietnamese translation and attained state-of-the-art BLEU scores using sequence-to-sequence models and vanilla preprocessing. In this work we perform extensive experiments on large-scale English-Vietnamese datasets with the latest NMT architectures for further improvements in BLEU scores and report our empirical findings.

Our main contributions are as follows: (1) A brief survey of current state of the art in NMT. (2) The construction of a large parallel corpus for English-Vietnamese translation, which will be publicly available. (3) Implementation and experimentation of the newest models, and our source code will also be shared. (4) Empirical findings on tuning the aforementioned models.

2. LATEST NMT ARCHITECTURES

2.1. Sequence-to-sequence RNNs

Here we introduce the sequence-to-sequence model based on an encoder-decoder architecture with attention mechanism [26]. Let (X, Y) be the pair of source and target sentences, where $X = x_1, \ldots, x_m$ is a sequence of m symbols and $Y = y_1, \ldots, y_n$ a sequence of n symbols. The encoder function f_{enc} maps the input sequence X to a fixed size vector, which the decoder function f_{dec} uses to generate the output sequence Y.

While f_{dec} is usually a uni-directional RNN, f_{enc} can be a uni-directional, bi-directional or hybrid RNN. In this work we consider bi-directional encoders. Each state of f_{enc} has the form $\overline{h}_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$ where the components encode X in forward and backward directions. The auto-regressive decoder f_{dec} then predicts each output token y_i from the recurrent state s_i, the previous tokens $y_{<i}$ and a context vector c_i.

The context vector c_i is also called attention vector and depends on encoder states together with the current decoder state. Among known attention architectures, in this work we use the most efficient as described in [26]. At the decoding step t, an alignment vector a_t is derived from the current decoder hidden state h_t and each encoder hidden state $\overline{h}_s$. The context vector c_t is a weighted average over all encoder states with weights a_t.

$$a_t(s) = \text{align}(h_t, \overline{h}_s), \tag{1}$$

$$c_t = \sum a_t h_s. \tag{2}$$

The context vector c_t is concatenated with the current hidden decoder state h_t to produce an attentional state $\tilde{h}$, which is fed through a softmax layer to produce the predicted distribution.

$$\tilde{h} = \tanh(W_c[c_t; h_t]), \tag{3}$$

$$p(y_t|y_{<t}, x) = \text{softmax}(W_s\tilde{h}_s). \tag{4}$$

2.2. The convolutional sequence-to-sequence model

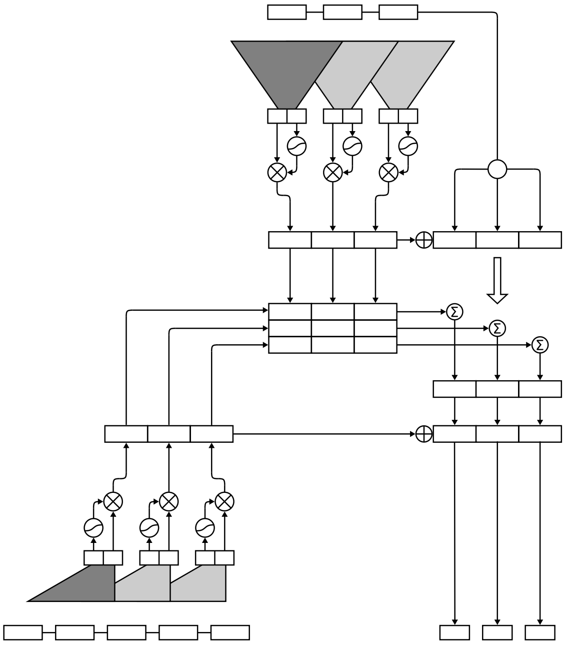


Figure 1. The convolution sequence-to-sequence model architecture, adapted from [14]

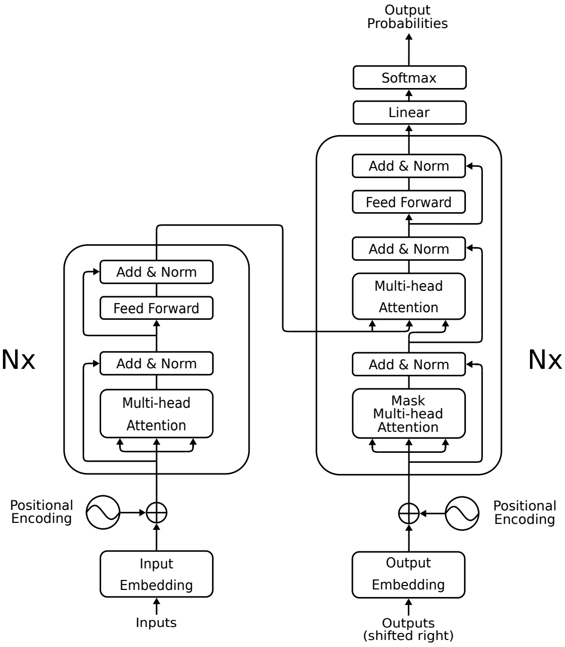


Figure 2. Overall architecture of the transformer

The Convolutional Sequence-to-Sequence Model (ConvS2S) [14] is a sequence-to-sequence model that uses a fully convolutional architecture. The model is equipped with gated linear units [9] and residual connections [15].

2.2.1. Position embeddings

Because the CNN itself can not convey positional information, ConvS2S uses position embeddings to tackle this problem. The input element $x = (x_1, \dots, x_m)$ is represented as a vector $z = w + p$ where $w = (w_1, \dots, w_m)$ embeds the symbols x_i into an Euclidean space $\mathbb{R}^f$ and $p = (p_1, \dots, p_m)$ embeds the positions of the x_i into $\mathbb{R}^f$. The same process is applied to the output elements generated by the decoder network, and the resulting representations are fed back into the decoder.

2.2.2. Convolutional layer structure

We denote the output of the i^{th} layer by $e^i = (e^i_1, \dots, e^i_n)$ for the encoder network and $d^i = (d^i_1, \dots, d^i_o)$ for the decoder network. In the model, each layer contains a one dimensional convolution followed by a non-linearity.

Each convolution kernel is parameterized as a weight $W \in \mathbb{R}^{2s \times ks}$ and a bias $b_w \in \mathbb{R}^{2s}$. The kernel's input is a matrix $X \in \mathbb{R}^{k \times s}$ which is a concatenation of k input elements embedded in s dimensions, the kernel's output is a vector $Y \in \mathbb{R}^{2s}$ that has twice the dimensionality of the input elements. Each group of k output elements of the previous layer are operated by a subsequence layer. The non-linearity is the gated linear unit (GLU:[9]) which implements a gating mechanism over the output of the convolution $Y = [A\ B] \in \mathbb{R}^{2s}$

$$v([A\ B]) = A \otimes \sigma(B), \tag{5}$$

where $A, B \in \mathbb{R}^s$ are the non-linearity input, $\otimes$ is the point-wise multiplication and the output $v([A\ B]) \in \mathbb{R}^s$ has half size of Y. The gates $\sigma(B)$ control which inputs A of the current context are relevant [14].

Residual connections from the input of each convolution to the output are applied, similar to [15]

$$d_j^i = v(W^i[d_{j-k/2}^{i-1}, \dots, d_{j+k/2}^{i-1}] + b_w^i) + d_j^{i-1}. \tag{6}$$

The convolution outputs that are of size $2s$ are mapped to the embedding of size f by linear projections. These linear mappings are applied to w while feeding embeddings to the encoder network, to the encoder output e_j^i, to the final layer of the decoder just before the softmax d^L and to all decoder layers d^i before computing the scores of the attentions.

Finally, a distribution over the T possible next target elements y_{j+1} is computed by transforming the top decoder output d_j^L via a linear layer with weights W_o and bias b_o

$$p(y_{j+1}|y_1, \dots, y_j, x) = \text{softmax}(W_o d_j^L + b_o) \in \mathbb{R}^T. \tag{7}$$

2.2.3. Multi-step attention

In ConvS2S, the attention mechanism is applied separately for each encoder layer. The attention mechanism works as multiple "hops" [36] compared to single step attention [2, 26, 41, 42]. At the decoder layer i, the attention a_{kj}^i of state k and the source element j are computed as a dot-product between the decoder state summary v_k^i and each output e_j^u of the last encoder layer u

$$a_{kj}^i = \frac{\exp(v_k^i \cdot e_j^u)}{\sum_{t=1}^m \exp(v_k^i \cdot e_j^u)} \tag{8}$$

where v_k^i is combined of the current decoder state d_k^i and the embedding of the previous target element g_k

$$v_k^i = W_v^i d_k^i + b_v^i + g_k. \tag{9}$$

The conditional input vector c_k^i to the current decoder layer is a weighted sum of the encoder output as well as the input embeddings z_j

$$c_k^i = \sum_{j=1}^m a_{kj}^i (e_j^u + z_j). \tag{10}$$

After that, c_k^i is added to the output of the corresponding decoder layer d_k^i. This attention mechanism can be seen as determining useful information from the current layer to feed to the subsequent layer. The decoder can easily access the attention history of $k-1$ previous time steps. Therefore, the model can take into account which previous inputs have been attended more easily than recurrent networks [14].

2.3. The transformer model

Unlike other transduction models, Transformer does not use RNNs or CNNs for modeling sequences. It has been claimed by authors to be the first transduction model to rely entirely on self-attention to compute representations of its input and output [40]. Like other competitive sequence transduction models, Transformer has an encoder and a decoder. The model is auto-regressive, consuming at each step the previous generated symbols as additional input to emit the next symbol. Compared to RNNs the proposed self-attention mechanism allows for a high degree of parallelization in training, while relying on positional embeddings to capture global dependencies within each sequence.

2.3.1. Overall structure

Like ByteNet [19] or ConvS2S [14], the decoder is stacked directly on top of the encoder. Without the recurrence or the convolution, Transformer encodes the positional information of each input token by a *position encoding* function. Thus the input of the bottom layer for each network can be expressed as $Input = Embedding + PositionalEncoding$.

The encoder has several identical layers stacked together. Each layer consists of a *multi-head self-attention mechanism* and a *position-wise fully connected feed-forward network*. Each of these sub-layers has a residual connection around itself, followed by layer normalization [23] (Figure 2). The output of each sub-layer is $LayerNorm(x+Sublayer(x))$ where x is the sub-layer input and $Sublayer$ is the function implemented by the sub-layer itself. The outputs of all sub-layers and the embedding layers in the model are vectors of dimension d_{model}.

The decoder is also a stack of identical layers, each layer comprising three sub-layers. At the bottom is a masked multi-head self-attention, which ensures that the predictions for position i depend only on the known outputs at the positions less than i. In the middle is another multi-head attention which performs the attention over the the encoder output. The top of the stack is a position-wise fully connected feed-forward sub-layer.

The decoder output finally goes through a linear transform with softmax activation to produce the output probabilities. The final linear transform shares the same weight matrix with the embedding layers of the encoder and decoder networks, except that the embedding weights are multiplied by $\sqrt{d_{model}}$.

2.3.2. Attention

The attention is crucial in NMT. It maps a *query* and a set of *key-value* pairs to an output. The output of the attention is a weighted sum of the *values* whose weights show the correlation between each *key* and *query*. The novelty is that the Transformer's attention is a *multi-head self-attention*. In the Transformer's architecture, the *query* is the decoder's hidden state, the *key* is the encoder's hidden state and the *value* is the normalized weight measuring the "attention" that each *key* is given. It is assumed that the queries and the keys are of dimension d_k and the values are of dimension d_v.

- Scaled dot-product attention: Let Q be the matrix of queries, K be the matrix of keys and V be the matrix of values. The attention is calculated as follows

$$Attention(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V. \tag{11}$$

Instead of using a single attention function, Transformer uses multi-head attentions. The multi-head attention consists of h layers (heads). The queries, keys and values are linearly projected to d_k and d_v dimensions. Each head receives a set of projections and performs a separate attention function yielding d_v-dimensional output values. The heads' outputs are concatenated and projected, resulting in the final multi-head attention output.

$$MultiHead(Q, K, V) = Concat(head_1, head_2, \ldots, head_h)W^O \tag{12}$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$. The projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W_i^O \in \mathbb{R}^{hd_v \times d_{model}}$.

If we set $d_k = d_v = d_{model}/h$, the multi-head attention then has the same computational cost as a single full-dimensionality attention. The Transformer's attention mechanism imitates the classical attention mechanism where the attention queries are previous decoder layer outputs, the keys and the values (memory) are the encoder layer outputs.

2.3.3. Position-wise feed-forward networks

The fully connected feed-forward network (FFN) at the top of each layer is applied to each input position separately and identically. Each FFN here consists of two linear transformations with a ReLU activation in between, acting like a stack two convolutions with kernel size 1

$$FFN(x) = ReLU(xW_1 + b_1)W_x + b_2. \tag{13}$$

2.3.4. Positional encoding

There are many types of positional encodings, including both learned and fixed variants [14]. Here the positional encodings are chosen as follows

$$PE(pos, 2i) = \sin(\frac{pos}{10000^{2i/d_{model}}}), \tag{14}$$

$$PE(pos, 2i + 1) = \cos(\frac{pos}{10000^{2i/d_{model}}}), \tag{15}$$

where pos is the position and i is the dimension. The authors hypothesized this function would allow the model to easily learn to attend by relative positions [40]. Their experiments showed that these encodings have the same performance as learned positional embedding. Furthermore they allow the model to extrapolate to sequences longer than the training sequences.

2.4. Theoretical analysis

Considering a sequence transduction task, where an input sequence $X = (x_1, x_2, \ldots, x_n)$ is mapped to another sequence of equal length $Y = (y_1, y_2, \ldots, y_n)$ with $x_i, y_i \in \mathbb{R}^d$. There are some essential aspects of transduction models that need to be marked: the total computational complexity per layer, the parallel ability (that is inversely proportional to the amount of computation that needs to be performed sequentially) and the distance of long-range dependencies in the network.

Table 1. Complexity per layer, minimum number sequential operations and maximum distance from the start to the end of signal traveling path in each type of models; n is the length of the sequence, d is the representation dimension and k is the kernel size of convolutions

	complexity per layer	sequential operations	maximum distance
recurrent	$O(nd^2)$	$O(n)$	$O(n)$
convolution	$O(knd^2)$	$O(1)$	$O(\log_k n)$
self-attention	$O(n^2d)$	$O(1)$	$O(1)$

In terms of computation amount, as described in Table 1, self-attention layers are faster than the recurrent layers when the representation dimension d is larger than the sequence length n, which is the most often cases in neural machine translation. The convolution layers have the highest complexity and are more expensive than the recurrent layers by factor of k in general.

The convolution layers and self-attention layers connect all positions with a constant number of operations, whereas a recurrent layer required at least n operations over a sequence of length n.

The long-range dependencies are a key challenge in many translation task, the longer the signal travel, the harder to learn the long-range dependencies [16]. The maximum length of the traveling path of a signal can go up to $2n$ in recurrent layers. In convolution model, it takes $O(\log_k n)$ of stacking dilated convolutions to represent the whole sequence, therefore the maximum distance is $O(\log_k n)$. Because of self-attention mechanism, where all pairs of input and output are connected, self-attention layers have a constant length of traveling path of forward and backward signals.

3. PARALLEL CORPUS CONSTRUCTION FROM PUBLIC SOURCES

3.1. Data sources

An essential component of any machine translation system is the parallel corpus. A good system requires a parallel corpus with a substantial number of qualified sentence pairs. There are various projects building English-Vietnamese corpora for specific tasks such as word-sense disambiguation [12, 10], VLSP project [1], web mining [8], etc. EVBCorpus [29] is a multi-layered English-Vietnamese Bilingual Corpus (EVBCorpus) containing over 10,000,000 words.

However since corpora such as EVBCorpus or VLSP are not openly published, we first needed to build a high-quality large-scale English - Vietnamese parallel corpus. We developed a web crawler to collect English - Vietnamese sentences from 1,500 clip subtitles with variety

Table 2. Details of input data sources

dataset	sentences	data sources
subtitles.en	1103456	1,500 clip subtitles
subtitles.vi	1103456	1,500 clip subtitles
IWSLT15.en	133,317	Web Inventory of Transcribed and Translated Talks
IWSLT15.vi	133,317	Web Inventory of Transcribed and Translated Talks

Table 3. Details of experiment dataset

dataset	sentences	tokens	vocabulary
train.en	886,224	10151378	75059
train.vi	886,224	11454886	39061
tst2012.en	1553	28723	3412
tst2012.vi	1553	34345	2056
tst2013.en	1268	27317	3563
tst2013.vi	1268	33764	2204
tst2015.en	1080	21332	3056
tst2015.vi	1080	25341	2098

of genres from the Internet. Moreover, we also include the well-known IWSLT'15 English-Vietnamese data [7] to the corpus (Table 2).

3.2. Data cleaning and preprocessing

The following steps were conducted to clean the dataset:

- Detecting and removing incomplete translations: A big part of our dataset is clip subtitles, .where we found many partially translated examples. In order to detect and remove such subtitles, we use Princeton WordNet [28] to extract an English vocabulary. We then scan each subtitle for tokens found in the vocabulary. If a half of all tokens match this criteria, the subtitle is considered untranslated. We also use *langdetect* package[1] to filter out sentences which are not in Vietnamese. Manual observation on a random subset of removed subtitles shows that this heuristic filter works sufficiently for our purpose.

- Removing low quality translations: There are many low quality translations in our collected data, which we had to remove manually.

After filtering we obtained 886,224 sentences pairs for training. We use tst2012 for validation; tst2013, tst2015 for testing; all the three are from IWSLT as provided in [7]. The sizes of the datasets are shown in Table 3.

Following[31] we only use subword for our experiments. In particular we created a shared subword code file using Byte Pair Encoding (BPE) [34] using 32,000 merge operations. This shared subword code file was then used to transform the training, validation and test datasets to sub-words with a vocabulary size of approximately 20,000.

[1] https://pypi.python.org/pypi/langdetect

4. EXPERIMENTS AND DISCUSSIONS

4.1. Overview of training configurations

For authenticity the experiments with each model are performed on original software provided by the authors. Specifically sequence to sequence RNN experiments are performed using [24], the Transformer experiments are performed using Tensor2Tensor (T2T) software [39] and the experiments on ConvS2S are performed using Facebook AI Research Sequence-to-Sequence Toolkit [14].

Training is performed on a single Nvidia Geforce Titan X. We run each experiment 3 times with random initializations and save one model checkpoint every 1000 steps. The checkpoint for reporting results is selected based on BLEU score for the validation set. We train and report the model's performance at the maximum of 64th epoch due to our computing resource constraints. For the sake of brevity, we only report mean BLEU on our result tables.

In all our experiments, there are some common terms in all the models, which are specified as follows:

- **Maximum input length** (`max_length`). Specifies the maximum length of a sentence in tokens (sub-words in our case). Sentences longer than `max_length` are either excluded from the training (T2T) or cut to match the `max_length` (RNN). Lowering `max_length` allows us to use a higher batch size and/or bigger model but biases the translation towards shorter sentences. Since 99% of the training sentences are not longer than 70, we set `max_length` to 70.

- **Batch size** (`batch_size`). For T2T `batch_size` is the approximate number of tokens (subwords) consumed in one training step, while for ConvS2S and RNN `batch_size` is the number of sentence pairs consumed in one training step. Hence for consistency we define `batch_size` as the approximate average number of tokens consumed in one training step. In fact the number of tokens in a sentence is the maximum of source and target subwords from the pair of training sentences. During training this allows us to put as many training tokens per batch as possible while ensuring that batches with long sentences still fit in GPU memory. In contrast, if we fixed the number of sentence pairs in a training batch, the model can run out of memory if a batch has many long sentences.

- **Training epoch** is one complete pass through the whole training set. The number of training steps can be converted to epochs by multiplying by the batch size and dividing by the number of subwords in the training data.

- **Model size** is number of trainable parameters of each model. Because of the difference in model structures, it is almost certain that two models with the same model size will not have the same training time.

Human judgment is always the best evaluation of machine translation systems; but in practice it is prohibitively expensive in time and resources. Therefore automatic scoring systems that evaluate machine translations against standard human translations are more commonly used. The most popular automatic metric in use is undoubtedly the BLEU score

[30]. BLEU has a high correlation with human judgments of quality and is easy to compute. Even though there are some acknowledged problems with BLEU and other better-performing metrics [4], we still stick to BLEU for its simplicity. In this work, we use the case-insensitive sacrBLEU [2] version which uses a fixed tokenization.

4.2. Sequence-to-Sequence RNN

Based on previous literature in [5, 31], we build a baseline which is set reasonably large for our dataset. We use LSTM [16] for the two models as suggested in [5]. The embedding dimension in the baseline model is set to be equal to the number of cells in each layer. We use two layers of 1024-unit LSTMs for both the encoder and the decoder, whereas the encoder's first layer is a bi-directional LSTM network and each layer is equipped with a residual connection and a dropout of 0.15 is applied to the input of each cell. We use Stochastic Gradient Descent (SGD) as the optimization algorithm with the batch size set to approximately 1280 tokens per step. The learning rate is set to 1.0; after 10 epochs we begin to halve the learning rate every single epoch. To prevent gradient explosion we enforce a hard constraint on the norm of the gradient by scaling it when its norm exceeds a threshold. In our two models the threshold is set to 5.0. For each training batch we compute $s = ||g||_2$ where g is the gradient divided by the batch size. If $s > 5.0$, we set $g = \frac{5g}{s}$.

Table 4. The *baseline* system's performance with approximately 98 millions parameters

task	batch size	training epochs	training time (days)	tst2012	tst2013	tst2015
En-Vi	1300	32	3.5	34.77	37.00	31.03
Vi-En	1300	20	2.5	35.21	38.83	30.29

4.3. Convolution Sequence to Sequence

We introduce four different models for each direction of translation. The hyper-parameters for each experiment are shown in Table 5:

- We used Nesterov's accelerate gradient (NAG) [37] with a fixed learning rate: 0.25 for the B_{base} model and 0.5 for the rests. After a certain number of epochs, we force-anneal the learning rate (`lr`) by a `lr_shrink` factor: `lr_new = lr * lr_shrink`. We start annealing the learning rate at the 24th epoch with a width `lr_shrink` of 0.1 for B_{base} model and at the 50th epoch with the width `lr_shrink` set to 0.2 for the rest. Once the learning rate falls below 10^{-5} we stop the training process.

- In the B_2 model, we use embedding size of 768 for all internal embeddings except the decoder output embedding (pre-softmax layer) which is set to 512.

- The effective context size of B_{base}, B_1, B_2 and B_3 are 9, 13, 27 and 25, respectively.

[2] https://github.com/awslabs/sockeye/tree/master/contrib/sacrebleu

Table 5. The hyper-parameters set of ConvS2S model

	encoder	decoder	emb size	lr	p_{drop}	training times (hours)	params $\times 10^6$
B_{base}	$4\times[256\times(3\times3)]$	$3\times[256\times(3\times3)]$	256	0.5	0.1	4	10
B_1	$4\times[512\times(3\times3)]$ $2\times[1024\times(3\times3)]$ $1\times[2048\times(1\times1)]$	$4\times[512\times(3\times3)]$ $2\times[1024\times(3\times3)]$ $1\times[2048\times(1\times1)]$	384	0.5	0.15	20	62
B_2	$9\times[512\times(3\times3)]$ $4\times[1024\times(3\times3)]$ $2\times[2048\times(1\times1)]$	$9\times[512\times(3\times3)]$ $4\times[1024\times(3\times3)]$ $2\times[2048\times(1\times1)]$	768 (512)	0.5	0.15	48	144
B_3	$8\times[512\times(3\times3)]$ $4\times[1024\times(3\times3)]$ $2\times[2048\times(1\times1)]$ $1\times[4096\times(1\times1)]$	$8\times[512\times(3\times3)]$ $4\times[1024\times(3\times3)]$ $2\times[2048\times(1\times1)]$ $1\times[4096\times(1\times1)]$	768	0.5	0.15	78	199

- We apply label smoothing of $\epsilon_{ls} = 0.1$ for all 4 models. This makes training perplexity fluctuate in a small interval but improves accuracy and BLEU score [40].

We use cross-validation's BLEU score to decide which checkpoint to select for evaluation: the B_{base} model is evaluated after 32 training epochs, the rest are evaluated after 64 training epochs. We found that the best perplexity of the validation dataset does not correspond to the best BLEU score on the test set but the BLEU score on the validation dataset does.

We did not observe over-fitting with the large number of parameters from the results in Table 6, that suggests the training data is fairly good and the model's dropout probability is suitable. With the hypothesis that the models' beam size and length penalty parameters are independence, we found that all the model's BLEU scores are improved a lot when beam size is increased from $b = 1$ to $b = 10$ and is only improved by a small margin (or even worse) when we keep increasing the beam size further. Because the decoding speed would slow down when we increase the beam size, we can conclude that the beam size of the ConvS2S models should be set to 10.

4.4. Transformer

In the Transformer architecture there are many hyper-parameters to be configured such as the number of layers in the encoder and the decoder, the number of attention heads or the size of the FFN weight matrix etc. In this work, we introduce three models based on their number of parameters. Each model's hyper-parameters are shown Table 7:

- We used the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. The learning rate is varied over the training processes according to the following formula: $lr = d_{model}^{-0.5} \cdot \min(step_{num}^{-0.5}, step_{num}^{-0.5} \cdot warmup_{steps}^{-1.5})$, where `warmup_steps` is set to 4000 from C_{base} and `warmup_steps` is set to 16000 for the rest.

- In the course of training we found that if `batch_size` is too big, the model can sometimes run out of GPU memory after a long training time. Therefore while C_2 can be trained with a `batch_size` of 3000 and the rest can be trained with a `batch_size`

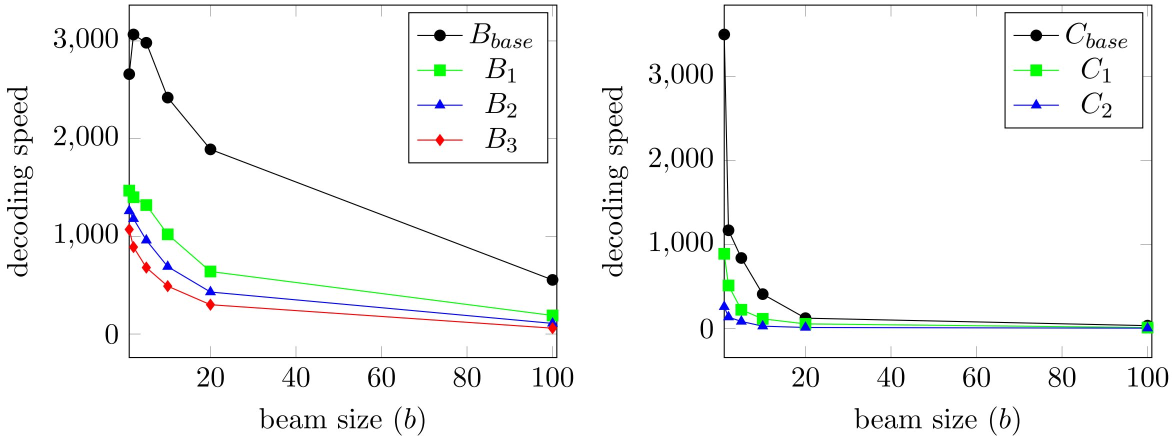
Table 6. BLEU score of English-Vietnamese and Vietnamese-English translation task on tst2012, tst2013, tst2015 of B_{base}, B_1, B_2 and B_3 model using beam-search with length penalty set to 1, b is the beam size

model	English-Vietnamese			Vietnamese-English		
	tst2012	tst2013	tst2015	tst2012	tst2013	tst2015
B_{base}, $b = 1$	24.31	25.34	23.98	25.18	27.76	23.89
B_{base}, $b = 2$	26.40	28.02	26.56	26.09	27.42	24.89
B_{base}, $b = 5$	26.92	**28.75**	27.86	26.74	28.60	25.36
B_{base}, $b = 10$	27.09	28.64	27.87	**26.97**	**29.21**	25.59
B_{base}, $b = 20$	27.08	28.66	28.09	26.86	29.46	**25.61**
B_{base}, $b = 100$	**27.22**	28.55	**28.18**	26.83	29.31	25.60
B_1 , $b = 1$	25.29	27.01	24.99	26.08	28.91	24.57
B_1 , $b = 2$	27.97	29.01	27.15	27.67	30.07	26.38
B_1 , $b = 5$	29.39	31.77	28.88	28.35	31.24	27.16
B_1 , $b = 10$	29.86	**32.26**	29.31	28.40	31.63	27.32
B_1 , $b = 20$	**29.94**	32.25	29.41	28.44	31.86	27.24
B_1 , $b = 100$	29.84	32.15	**29.75**	**28.58**	**31.87**	**27.39**
B_2 , $b = 1$	34.87	36.57	29.13	37.90	39.11	28.33
B_2 , $b = 2$	36.36	37.61	30.42	39.85	41.78	29.99
B_2 , $b = 5$	37.20	**38.53**	31.10	41.07	42.85	30.52
B_2 , $b = 10$	37.19	38.48	31.23	41.19	43.03	30.60
B_2 , $b = 20$	37.36	38.36	31.25	41.44	43.32	**30.74**
B_2 , $b = 100$	**37.49**	38.42	**31.42**	**41.49**	**43.36**	30.71
B_3 , $b = 1$	40.38	40.81	31.40	42.63	44.17	32.68
B_3 , $b = 2$	41.87	42.62	32.04	43.09	45.61	33.41
B_3 , $b = 5$	42.32	42.49	33.56	44.48	46.13	34.01
B_3 , $b = 10$	42.40	43.51	**33.50**	44.32	46.31	**34.11**
B_3 , $b = 20$	**42.51**	43.56	33.39	44.31	46.42	34.10
B_3 , $b = 100$	42.26	**43.60**	33.48	**44.52**	**46.45**	33.99

larger than 6500, we recommend a batch size of 2048 for the C_2 and 4096 for the rest in order to keep the training stable. Another reason is our observation that the time to convergence does not change significantly once the batch size gets sufficiently large.

- The learning rate are chosen based on `batch_size`. Specifically, we set the learning rate to 0.0001 for the largest model with beam size of 2048 and scale it by $\sqrt{k}$ when multiplying the `batch_size` by k.

- We also observed that when the `batch_size` is too small (i.e. < 512 for the biggest model), the model can only converge when the learning rate is smaller than 0.00005. Even then the model's BLEU is much lower than with a large `batch_size`. This is due to the fact that the *gradient noise scale* is proportional to the learning rate divided by the batch size. Thus, lowering the batch size increases the noise scale [35]. Therefore, we would rather reduce the model's complexity than reduce the batch size.



(a) **(b)**

Figure 3. The impact of beam size to the decoding speed of ConvS2S models (a) and Transformer models (b). The decoding speed of ConvS2S models is often higher and becomes slower when increasing the beamsize than the decoding speed of Transformer models with the same model size

- The performance of the C_2 model kept improving epoch by epoch and could potentially be better than reported. However due to resource constraints we report the model's performance at the 64^{th} checkpoint.

Table 7. Transformer hyper-parameters

	N	d_{model}	h	d_{ff}	p_{drop}	training time (hours)	params $\times 10^6$
C_{base}	2	256	4	1024	0.1	5	9M
C_1	6	512	16	2048	0.15	36	54M
C_2	8	1024	16	4096	0.15	72	197M

4.5. Length normalization for beam search

Beam search is a widespread technique in NMT, which finds the target sequence that maximizes some scoring function by a tree search. In the simplest case the score is the log probability of the target sequence. This simple scoring favors shorter sequences over longer ones on average since a negative log-probability is added at each decoding step.

Recently, length normalization [41] have been shown to improve decoding results for RNN based models. However, there is no guarantee that this strategy works well for other models. In this work, we experiment on two normalization functions described below

$$f_1 = \frac{(5+|Y|)^\alpha}{6^\alpha}, \tag{16}$$

$$f_2 = (1+|Y|)^\alpha. \tag{17}$$

Table 8. BLEU score of English-Vietnamese translation task on tst2012, tst2013, tst2015 of C_{base}, C_1 and C_2 model, b is the beam size with a default length penalty function

model	English-Vietnamese			Vietnamese-English		
C_{base}, $b = 1$	31.88	33.70	31.16	27.71	29.32	25.35
C_{base}, $b = 2$	31.99	34.44	31.54	28.63	30.02	25.94
C_{base}, $b = 5$	**31.19**	**34.61**	**32.06**	**28.86**	**30.48**	26.17
C_{base}, $b = 10$	31.93	34.44	31.79	28.74	30.01	**26.28**
C_{base}, $b = 20$	31.86	34.18	30.79	28.55	30.95	26.14
C_{base}, $b = 100$	30.96	33.85	30.21	25.11	27.42	26.09
C_1, $b = 1$	36.85	39.88	33.62	33.31	35.71	29.58
C_1, $b = 2$	37.46	**40.99**	30.88	33.27	35.77	30.28
C_1, $b = 5$	37.61	40.88	33.48	**33.32**	**36.06**	30.33
C_1, $b = 10$	37.51	40.81	33.59	33.28	35.88	**30.36**
C_1, $b = 20$	**37.65**	40.66	**33.73**	33.18	35.83	30.30
C_1, $b = 100$	37.43	40.02	33.31	32.62	34.99	30.17
C_2, $b = 1$	52.37	54.70	38.01	41.61	44.31	33.19
C_2, $b = 2$	52.89	55.31	38.81	43.16	45.41	33.83
C_2, $b = 5$	53.32	**55.89**	**39.14**	**43.41**	46.26	33.94
C_2, $b = 10$	**53.64**	55.85	39.01	43.32	46.27	34.05
C_2, $b = 20$	53.50	55.76	39.05	43.10	**46.49**	34.20
C_2, $b = 100$	53.36	55.31	38.92	42.86	45.71	**34.24**

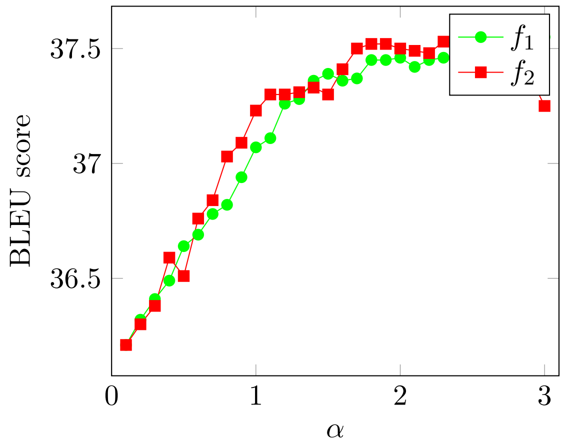


Figure 4. The effect of length penalty factor α on BLEU of B_2 on tst2012 with beam size fixed to 10

We found that the length penalty can help improve the model's performance up to 2 in BLEU scale. The length penalty should be chosen between 2.0 to 3.0.

4.6. Ensembling

Ensemble methods combine multiple individual methods to create a learning algorithm that is better than any of its individual parts [11]. They are widely used to boost machine learning models' performance [11, 21]. In neural machine translation, the most popular ensemble method is checkpoint ensemble., in which the ensembled models are created by combining (averaging) multiple model checkpoints together [40, 41]. This method does not require training multiple model and the ensembled model has the size as same as the constituent models.

In many experiments the authors suggest to average checkpoints based on training time [38, 40], which depends on hardware and is hard to reproduce. In this work we experiment with checkpoint ensembling based on training epoch, which can be easily adapted to different platforms.

Table 9. Effect of checkpoint ensembling (n is number of checkpoint to be averaged) on the B_2 (a) and C_1 (b) model for English-Vietnamese translation task on tst2015

(a)

n	interval (% of a epoch)			
	1.5	3	4.5	6
8	32.26	32.74	**32.82**	32.68
16	31.58	31.75	31.55	31.68

(b)

n	interval (% of a epoch)			
	1.5	3	4.5	6
8	30.63	**30.90**	30.77	30.81
16	30.61	30.80	30.85	30.89

According to our experiments checkpoint ensembling always improves the model's performance. ConvS2S benefits the most (up to 7% on the B_2 model) while ensembling has a smaller effect on the Transformer models (at most 5% on the C_3 model). We observed that taking 8 checkpoints for ensembling often yielded better results than 16 checkpoints. This also has the advantage that less time is spent on checkpoint saving.

Ensembling can also be applied by training several new models starting form the same checkpoint. Each model is trained at a random position in the training data. In this setup, these models are semi-independent because they are rooted in the same source checkpoint. These semi-independent models can be averaged as described above, resulting in a boost in the result, but in a smaller margin.

5. RESULT AND EMPIRICAL STUDIES

From the above experiments we observed that the training data is well correlated with the test data and training does not suffer from overfitting. However this makes it hard to tell if the model is general enough.

For new experiments we can always choose RNNs as a reliable base line model, that does not take much effort to achieve good results. The Transformer model has the highest convergence speed while RNNs have the lowest convergence speed. The Transformer model has showed its superiority in terms of achieving state-of-the-art results when given a suitable batch size and learning rate. Interestingly, even a very simple Transformer model with only 5 training hours can achieve a comparable score.

In our experiments we showed that a well-tuned beam search with length penalty is crucial, which can help to boost the model's score by 1.5 to 3 BLEU point (Table 6, 8, Figure 4). The most effective beam-size is 10 for ConvS2S and 5 for Transformer. The length penalty has a high impact on the final result, which should be set from 2 to 3.

Finally from our experiment results we compared our best performing hyper-parameter sets across all models and combined to a final model with the state-of-the-art results (Table 10). This shows that careful hyper-parameter tuning can greatly improve performance.

Table 10. Hyper-parameter settings for our final combined model for bidirectional English-Vietnamese translation

Hyper-parameters	Value
N	8
d_{model}	1024
h	16
d_{ff}	4096
p_{drop}	0.15
`batch_size`	2048
`length_penalty`	1.5
`beam_size`	5
checkpoint to ensemble	8
ensemble interval	3% epoch

Table 11. BLEU results for our final combined model for bidirectional English-Vietnamese translation

	English-Vietnamese			Vietnamese-English		
	tst2012	tst2013	tst2015	tst2012	tst2013	tst2015
Combined model	55.04	56.88	40.01	46.36	49.23	35.81
C_2	52.37	54.70	38.01	42.63	44.17	32.68

6. CONCLUSION

We conducted a broad range of experiments with the RNN sequence-to-sequence, ConvS2S and Transformer models for English-Vietnamese and Vietnamese-English translation, pointing out key factors to achieving state-of-the-art results. In particular we performed extensive exploration of hyper-parameters settings, which can be useful for other research works. In sum, our experiments took about 2,000 GPU hours.

We highlighted several important points: efficient use of batch size, the importance of beam search and length penalty, the importance of initial learning rate, the effectiveness of checkpoint ensembling, and the model's complexity. Along with these contribution we also make our dataset publicly available at (location withheld for review).

We hope our findings can help accelerate the pace of research on and application of English-Vietnamese and Vietnamese-English translation.

7. ACKNOWLEDGEMENT

This research is funded by the Hanoi University of Science and Technology (HUST) under project number T2017-PC-078.

REFERENCES

[1] "Building basic resources and tools for vietnamese language and speech processing," in *VLSP Projects*, 2010. [Online]. Available: http://vlsp.vietlp.org:8080/demo/?page=resources

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014. [Online]. Available: http://arxiv.org/abs/1409.0473

[3] H. T. Bao, P. N. Khanh, H. T. Le, and N. T. P. Thao, "Issues and first development phase of the english-vietnamese translation system evsmt1.0," in *Proceedings of the third Hanoi Forum on Information Communication Technology.* Ha Noi, 2009. [Online]. Available: https://www.researchgate.net/publication/228966483_Issues_and_First_Development_Phase_of_the_English-Vietnamese_Translation_System_EVSMT1_0

[4] O. Bojar, Y. Graham, and A. Kamran, "Results of the WMT17 metrics shared task," in *Proceedings of the Second Conference on Machine Translation (WMT)*, September 7–8, 2017, pp. 489–513.

[5] D. Britz, A. Goldie, T. Luong, and Q. Le, "Massive Exploration of Neural Machine Translation Architectures," *ArXiv e-prints*, Mar. 2017.

[6] M. Cettolo, J. Niehues, S. Stuker, L. Bentivogli, R. Cattoni, and M. Federico, "The iwslt 2015 evaluation campaign," in *Proceeding of the 12th International Workshop on Spoken Language Translation*, 2015. [Online]. Available: http://workshop2015.iwslt.org

[7] M. Cettolo, C. Girardi, and M. Federico, "Wit^3: Web inventory of transcribed and translated talks," in *Proceedings of the 16th Conference of the European Association for Machine Translation (EAMT)*, Trento, Italy, May 2012, pp. 261–268.

[8] V. B. Dang and B.-Q. Ho, "Automatic construction of english-vietnamese parallel corpus through web mining." in *RIVF*. IEEE, 2007, pp. 261–266. [Online]. Available: http://dblp.uni-trier.de/db/conf/rivf/rivf2007.html#DangH07

[9] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," *CoRR*, vol. abs/1612.08083, 2016. [Online]. Available: http://arxiv.org/abs/1612.08083

[10] D. Dien and H. Kiem, "Pos-tagger for english-vietnamese bilingual corpus," in *Proceedings of the HLT-NAACL 2003 Workshop on Building and Using Parallel Texts: Data Driven Machine Translation and Beyond - Volume 3*, ser. HLT-NAACL-PARALLEL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 88–95. [Online]. Available: http://dx.doi.org/10.3115/1118905.1118921

[11] T. G. Dietterich, "Ensemble methods in machine learning," in *Proceedings of the First International Workshop on Multiple Classifier Systems*, ser. MCS '00. London, UK, UK: Springer-Verlag, 2000, pp. 1–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=648054.743935

[12] D. Dinh, "Building a training corpus for word sense disambiguation in english-to-vietnamese machine translation," in *Proceedings of the 2002 COLING Workshop on Machine Translation in Asia - Volume 16*, ser. COLING-MTIA '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 1–7. [Online]. Available: http://dx.doi.org/10.3115/1118794.1118801

[13] D. Dinh, H. Kiem, and E. Hovy, "Btl: a hybrid model for english-vietnamese machine translation," in *Proceedings of the Machine Translation Summit IX*, 2003.

[14] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," *CoRR*, vol. abs/1705.03122, 2017. [Online]. Available: http://arxiv.org/abs/1705.03122

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[16] S. Hochreiter and J. Schmidhuber, "Lstm can solve hard long time lag problems," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, ser. NIPS'96. Cambridge, MA, USA: MIT Press, 1996, pp. 473–479. [Online]. Available: http://dl.acm.org/citation.cfm?id=2998981.2999048

[17] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," *CoRR*, vol. abs/1611.10012, 2016.

[18] P. Huang, C. Wang, D. Zhou, and L. Deng, "Neural phrase-based machine translation," *CoRR*, vol. abs/1706.05565, 2017.

[19] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu, "Neural machine translation in linear time," *CoRR*, vol. abs/1610.10099, 2016.

[20] C. Kandaswamy, L. M. Silva, L. A. Alexandre, J. M. Santos, and J. M. de Sá, "Improving deep neural network performance by reusing features trained with transductive transference," in *International Conference on Artificial Neural Networks*. Springer, 2014, pp. 265–272.

[21] A. Krogh and J. Vedelsby, "Neural network ensembles, cross validation and active learning," in *Proceedings of the 7th International Conference on Neural Information Processing Systems*, ser. NIPS'94. Cambridge, MA, USA: MIT Press, 1994, pp. 231–238. [Online]. Available: http://dl.acm.org/citation.cfm?id=2998687.2998716

[22] K. H. Le, "One method of interlingual translation," in *Proceedings of National Conference on IT Research, Development and Applications*, 2003.

[23] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *ArXiv e-prints*, Jul. 2016.

[24] M. Luong, E. Brevdo, and R. Zhao, "Neural machine translation (seq2seq) tutorial," 2017. [Online]. Available: https://github.com/tensorflow/nmt

[25] M.-T. Luong and C. D. Manning, "Stanford neural machine translation systems for spoken language domains," in *Proceeding of the 12th International Workshop on Spoken Language Translation*, 2015. [Online]. Available: http://workshop2015.iwslt.org

[26] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015. [Online]. Available: arXiv.org⟩cs⟩arXiv:1508.04025

[27] M.-T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba, “Addressing the rare word problem in neural machine translation,” May 2015. [Online]. Available: http://arxiv.org/abs/1410.8206

[28] G. A. Miller, “Wordnet: A lexical database for english,” *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995. [Online]. Available: http://doi.acm.org/10.1145/219717.219748

[29] Q. H. Ngo, W. Winiwarter, and B. Wloka, “Evbcorpus-a multi-layer english-vietnamese bilingual corpus for studying tasks in comparative linguistics,” in *Proceedings of the 11th Workshop on Asian Language Resources*, 2013, pp. 1–9. [Online]. Available: https://www.aclweb.org/anthology/W13-4301

[30] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: A method for automatic evaluation of machine translation,” in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL ’02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 311–318.

[31] H. Phan-Vu, V. Nguyen, V. Tran, and P. Do, “Towards state-of-the-art english-vietnamese neural machine translation,” in *Proceeding SoICT 2017 Proceedings of the Eighth International Symposium on Information and Communication Technology.* Nha Trang City, Viet Nam, December 07–08, 2017, pp. 120–126.

[32] N. Q. Phuoc, Y. Quan, and C.-Y. Ock, “Building a bidirectional english-vietnamese statistical machine translation system by using moses,” *International Journal of Computer and Electrical Engineering*, vol. 8, no. 2, pp. 161–168, 2016.

[33] M. Popel and O. Bojar, “Training tips for the transformer model,” *The Prague Bulletin of Mathemetical Linguistics*, vol. 110, no. 1, 2018. [Online]. Available: https://doi.org/10.2478/pralin-2018-0002

[34] R. Sennrich, B. Haddow, and A. Birch, “Edinburgh neural machine translation systems for WMT 16,” 2016. [Online]. Available: arXiv.org⟩cs⟩arXiv:1606.02891

[35] S. L. Smith, P. Kindermans, and Q. V. Le, “Don’t decay the learning rate, increase the batch size,” 2017. [Online]. Available: arXiv.org⟩cs⟩arXiv:1711.00489

[36] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, “End-to-end memory networks,” 2015. [Online]. Available: http://arxiv.org/abs/1503.08895

[37] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proceeding ICML’13 Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28.* Atlanta, GA, USA, June 16–21, 2013, pp. III–1139–III–1147. [Online]. Available: http://dl.acm.org/citation.cfm?id=3042817.3043064

[38] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, ser. NIPS’14. Cambridge, MA, USA: MIT Press, 2014, pp. 3104–3112. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969033.2969173

[39] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, L. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit, “Tensor2tensor for neural machine translation,” 2018. [Online]. Available: http://arxiv.org/abs/1803.07416

[40] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[41] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” *CoRR*, vol. abs/1609.08144, 2016. [Online]. Available: http://arxiv.org/abs/1609.08144

[42] J. Zhou, Y. Cao, X. Wang, P. Li, and W. Xu, “Deep recurrent models with fast-forward connections for neural machine translation,” *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 371–383, 2016. [Online]. Available: https://transacl.org/ojs/index.php/tacl/article/view/863

Received on October 29, 2018
Revised on April 09, 2019