This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2601521, IEEE Transactions on Industrial Informatics

Understanding Subtitles by Character-Level Sequence-to-Sequence Learning

Haijun Zhang, *Member, IEEE*, Jingxuan Li, Yuzhu Ji, and Heng Yue

***Abstract*—This paper presents a character-level sequence-to-sequence learning method, RNNembed. This method allows the system to read raw characters, instead of words generated by pre-processing steps, into a pure, single neural network model under an end-to-end framework. Specifically, we embed a Recurrent Neural Network (RNN) into an encoder-decoder framework and generate character-level sequence representation as input. The dimension of input feature space can be significantly reduced as well as avoiding the need to handle unknown or rare words in sequences. In the language model, we improve the basic structure of a Gated Recurrent Unit (GRU) by adding an output gate, which is used for filtering out unimportant information involved in the attention scheme of the alignment model. Our proposed method was examined in a large-scale dataset on an English-to-Chinese translation task. Experimental results demonstrate that the proposed approach achieves a translation performance comparable, or close, to conventional word-based and phrase-based systems.**

***Index Terms*—Sequence learning, neural machine translation, recurrent neural network, character-level.**

I. Introduction

Recent advances in video-sharing technologies are changing the way that people view media and the media that people view. For example, if you enjoy watching Bleach, Pretty Little Liars, True Blood and many other television series, but are not living in the broadcast countries and have no television, you can try to view them online with a computer. Many video-sharing websites, such as Tudou, Hulu, CastTV, and 10StarMovies, are providing such services for free without requiring downloading. Popular television series, television shows, and films are currently being released in this way worldwide. However, most users do not possess the capability to watch these videos without translation. At present, high-quality subtitle translations are usually completed by self-organized groups from video-sharing forums. However, the translation process is both labor-intensive and error-prone. For example, a group with four members requires approximately 10 hours to translate a 90-minute film from English to Chinese. Therefore, building machine translation applications is critical.

Among the major problems in natural language processing, sequence-to-sequence learning, a fundamental problem, has several important applications, such as machine translation [1][2], part-of-speed tagging [3], and dependency parsing [4]. Without loss of generality, we take machine translation as an example of sequence-to-sequence learning in this paper. Machine translation has been extensively studied since the 1950s [5]. Dictionaries and rules for producing correct word-order were originally used for translation systems. Many models aided by knowledge of language were developed in the following years. In the 1990s, statistical methods relying on corpora of translation examples began to emerge [5]. Despite certain rule-based pieces remaining in machine translation systems, these statistical methods became dominant because of the availability of large corpora, toolkits for performing basic translation processes, and increased computational efficiency.

Initially, neural networks were utilized for natural language processing [6]. Bengio et al. then used neural networks to learn a statistical model of the distribution of word sequences, and trained the model on a large scale [7]. Recently, neural networks have attracted increased attention in machine translation [2][8][9]. These newly developed neural network-based methods are often called *neural machine translation*, which aims at building and training a single, large neural network that reads a sentence and outputs a correct translation. Regarding neural machine translation models, the encoder-decoder framework receives much focus in the extant literature [2][8][10]. In this framework, an encoder neural network reads a source sentence and encodes it into a fixed-length vector. A decoder then produces a translation by decoding the fixed-length vector into a sentence of variable length. The whole system, which includes the encoder and the decoder for a language pair, is trained to maximize the conditional probability of a correct translation given a source sentence. However, since the encoder-decoder framework needs to be able to compress all of the necessary information of a source sentence into a fixed-length vector, difficulty may arise for the neural network in handling long sentences [1]. In order to address this issue, Bahdanau et al. [1] recently introduced RNNsearch, an extension to the encoder-decoder model, which embeds an attention mechanism into the learning process. The proposed model generates a word in a translation by (soft-)searching for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all of the previously generated

Manuscript received June 1, 2016. Accepted for publication July 30, 2016. This work was supported in part by the Natural Science Foundation of China under Grant no. 61300209 and no. 61572156, and the Shenzhen Science and Technology Program under Grant no. JCYJ20150625142543464, no. JSGG20150512145714247 and no. JCYJ20160330163900579.

Haijun Zhang, Jingxuan Li, and Yuzhu Ji are with the Shenzhen Graduate School, Harbin Institute of Technology, Xili University Town, Shenzhen 518055, P.R.China. Heng Yue is with the National Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang, Liaoning 110004, P.R.China (e-mail: hjzhang@hitsz.edu.cn; jing_910307@163.com; Nero2074@outlook.com; yueheng@mail.neu.edu.cn).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2601521, IEEE Transactions on Industrial Informatics

target words. Empirical comparison has demonstrated that this dynamic alignment in RNNsearch is capable of producing superior performance over the traditional encoder-decoder approach [11].

Despite the encouraging results achieved by RNNsearch being representative, a common issue with these models is that they usually rely on a tokenization process before model training. A shortlist of the most frequently occurring words in each language is then used to train the models. Any word not included in the shortlist will be mapped to a special token (i.e., [UNK]), and thus the systems cannot output complete translations to some extent. In order to address these special tokens, Luong et al. attempted to employ a post-processing step that translates every out-of-vocabulary (OOV) word using an additional dictionary [12]. On the other hand, Jean et al. introduced a method based on importance sampling that allows the translation system to use a large target vocabulary without increasing training complexity [11]. However, these methods usually rely on tokenization as a pre-processing step to produce the vocabulary, and require extra computational effort to cope with the special tokens remaining in target sentences. In this paper, we develop a character-level sequence-to-sequence learning method for neural machine translation, in particular for subtitle translation. The translation space for short-length sequences, such as subtitles, is smaller than that for normal sentences, which makes our proposed character-level translation model possible. The proposed method, which we refer to as RNNembed, allows us to input quantized characters into the translation system, instead of using a vocabulary of common words tokenized in the pre-processing step. We show that subtitle understanding can be handled by a single neural network system without artificially embedding knowledge about words, phrases, or any other syntactic or semantic structures associated with a language. Specifically, for the task of English-to-Chinese translation, we embed a Recurrent Neural Network (RNN) [13] into the encoder-decoder approach, in which the inputs are quantized English characters and the outputs are Chinese characters. The most apparent feature of our proposed method, RNNembed, is that it does not require knowledge of words. This renders a usual tokenization unnecessary. All previous works start with words instead of raw characters, thus they usually require a large vocabulary and often make special tokens remain in target translation results. We trained our model in a large-scale subtitle dataset collected from the Internet. Experimental results suggest that the proposed approach achieves, with a single neural network model, a translation performance comparable, or close, to conventional word-based and phrase-based systems.

The remaining sections of this paper are organized as follows. A character-level encoder-decoder framework is described in Section II, where we present each module under the framework in detail. The model training procedures and implementation details are presented in Section III. Experimental results are demonstrated in Section IV. The paper ends with conclusions and future work propositions in Section V.

II. Character-Level Sequence-to-Sequence Learning

A. Character-Level Encoder-Decoder Framework

Neural machine translation has shown great success in recent years [7]-[12]. In the field of neural machine translation, most previous studies worked under the encoder-decoder framework, which learns to encode a variable-length source sentence into a fixed-length vector representation c and to decode a vector into a variable-length target sentence [8]. Once the network structures of the encoder and the decoder in this typical end-to-end framework are designed, given sentence pairs an optimal translation model can be learned by a training process. In order to maintain the word order in the source and target sentence, the input or output sequence needs to be processed in time order. The notation "<eol>" is usually used for representing the end of a sentence, and will determine when we stop predicting the next word in a sequence. Therefore, the entire translation model constitutes a typical end-to-end learning process. The model is capable of encoding all of the information of an input sentence into a fixed-length vector c and to predict each word in an output sentence in time order.

For clarity, similar to [1], we briefly describe the underlying framework, called RNN Encoder-Decoder [2][8], upon which we build a new architecture that allows character-level sequence input without requiring knowledge of words.

In the encoder-decoder framework, given a source sentence, a sequence of vectors $\mathrm{x} = (x_1, x_2, ..., x_{T_x})$, an encoder learns to encode the input sequence into a vector c. An RNN is used in the form of

$$h_t = f(x_t, h_{t-1}) \tag{1}$$

and

$$c = q(\{h_1, h_2, ..., h_{T_x}\}), \tag{2}$$

where $h_t \in \Re^n$ is a hidden state at time t; c is a vector generated from the sequence of the hidden states; T_x is the length of the input sentence; f and q are nonlinear functions that can be represented by long short-term memory (LSTM) units [2]. The decoder is trained to predict the next word y_t considering the context vector c and all of the previously predicted words $\{y_1, y_2, ..., y_{t-1}\}$. The final translation can be obtained by decomposing the joint probability into the ordered conditionals in the form of

$$p(\mathrm{y}) = \sum_{t=1}^{T_y} p(y_t | \{y_1, y_2, ..., y_{t-1}\}, c), \tag{3}$$

where $\mathrm{y} = (y_1, y_2, ..., y_{T_y})$, and T_y denotes the length of the output sentence. Moreover, each conditional probability using an RNN can be modeled as

$$p(y_t | \{y_1, y_2, ..., y_{t-1}\}, c) = g(y_{t-1}, s_t, c), \tag{4}$$

where g is a nonlinear, potentially multi-layered, function, and s_t is the hidden state of the RNN. It is worth noting that other architectures, such as a hybrid of an RNN and a deconvolutional neural network, can also be used [9].

However, this basic framework needs to be specified with the network structure on a real-world translation task. For

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2601521, IEEE Transactions on Industrial Informatics

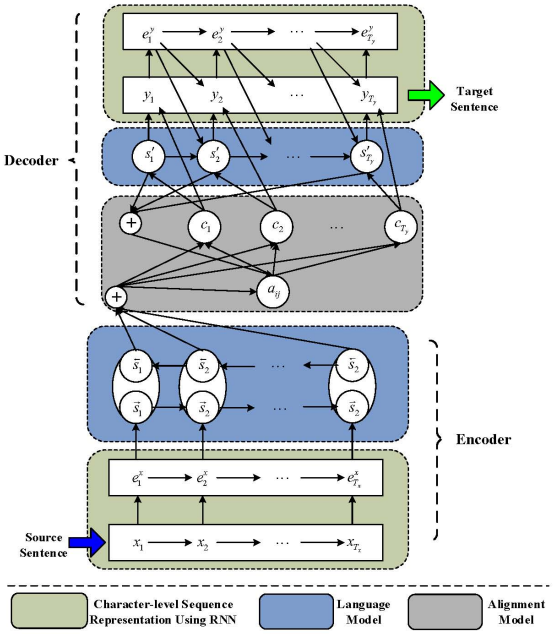


Fig. 1. Character-level sequence-to-sequence encoder-decoder framework.

example, this simple encoder-decoder framework encodes all of the necessary information of a source sentence into a fixed-length vector c. However, this may cause the difficulty of handling long sequences [1]. In order to overcome this challenge, Bahdanau et al. [1] recently introduced an extension model, RNNsearch, which embeds an attention mechanism into the learning process. Empirical comparison has demonstrated that RNNsearch is able to produce better performance than the traditional encoder-decoder approach [11]. From a systematic perspective, the whole network can be roughly divided into four sections according to their functions, i.e., word vector representation, language model, word alignment, and target sentence generation. In line with the encoder-decoder framework, in this paper we propose a new network model that embeds two RNNs into the word vector representations of source sentences and target sentences, respectively. An overall architecture of our model is shown in Fig.1. Our model enables us to input character-level sequences and to avoid the use of special tokens for the representation of OOV words. Specifically, we used an RNN to project the input sequence, $\mathrm{x} = (x_1, x_2, ..., x_{T_x})$, and the output sequence, $\mathrm{y} = (y_1, y_2, ..., y_{T_y})$, into feature space $(e^{x_1}, e^{x_2}, ..., e^{x_{T_x}})$ and $(e^{y_1}, e^{y_2}, ..., e^{y_{T_y}})$, respectively. It is worth noting that the input/output feature space generated from character-level sequences is of low dimension in comparison to the word-level feature space used in traditional models. Here, similar to RNNsearch [1], we used a bidirectional RNN to learn an encoder and employed the attention scheme for dynamic word alignment.

B. Character-Level Sequence Representation Using RNN

Previous studies usually use a preprocessing step to extract words from source sequences and to build a large vocabulary that helps to transform an input sequence into fixed-length vectors using the one-hot scheme which refers to a vector where the component values are only those with a single high (1) bit and all the others low (0). However, due to the sparsity and high dimension of this word-level representation, the used vocabulary cannot comprise all of the words occurring in the data set. Thus, these OOV words are often replaced by a special token ([UNK]). In order to obtain the final translation results, a post-processing step is required to handle those UNKs in the output sequence [12]. In this paper, we develop a method, RNNembed, for character-level sequence representation using RNN. This method offers a number of desirable features. First, it allows the system to read raw characters. In other words, we do not require any prior knowledge about words, phrases, or other syntactic structures associated with a language. Second, since the number of characters in a language is usually limited, the feature space of the input/output sequence in our method is of low dimension. Third, RNNembed avoids using a post-processing step to handle special tokens in output sequences. Finally, with RNNembed, the sequence-to-sequence learning system can be trained and implemented by a unified neural network framework without using any pre-processing or post-processing procedures.

Given an input vector x_t at time t, a hidden state h_t in an RNN can be calculated by Eq.1. An output vector with a one-hot representation at time t is modeled as

$$p(x_{t,j} = 1 | x_{t-1}, ..., x_t) = \frac{\exp(w_j h_{t-1})}{\sum_{j'=1}^{K} \exp(w_{j'} h_{t-1})}, \quad (5)$$

where w_j $(j = 1, 2, ..., K)$ is a row of the weight matrix W. Thus, the probability that a sequence of vectors x occurs can be obtained by the following joint probability

$$p(\mathrm{x}) = \prod_{t=1}^{T} p(x_t | x_{t-1}, ..., x_1). \quad (6)$$

In this process, the hidden unit h_T at the last time step T can be seen as a vector representation of the whole input sentence, rather than a word vector representation. However, if we keep each hidden unit h_t at time t and reset the hidden state to zero before inputting it to the next hidden unit h_{t+1}, a word vector sequence $(h_1, h_2, ..., h_T)$ can be achieved. In fact, this is equivalent to the case in which we set $h_t = f(x_t)$. In other words, it is a non-linear transformation on the basis of a one-hot representation of a sequence. However, in line with this concept, an RNN can be used for character-level sequence representation if we choose the correct time t to reset the hidden state of h_t to zero.

From the natural language processing perspective, "reset to zero" can be regarded as word segmentation, which is exactly required by the character-level input sequence. Actually, this functionality can be implemented by adding a gate into an RNN. The gate is used for outputting the segmented word-vector and for controlling the context information of the last word not to be considered at the current time (i.e., "reset to zero"). Motivated by this, we design the RNN with a gate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2601521, IEEE Transactions on Industrial Informatics

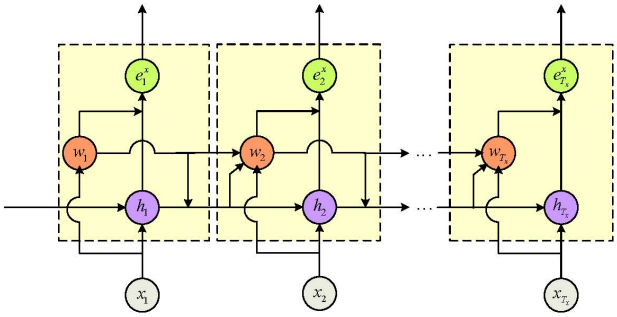


Fig. 2. The RNN structure for character-Level sequence representation.

TABLE I
AN EXAMPLE OF CHARACTER-LEVEL REPRESENTATION WITH RNN (<> REPRESENTS THE BLANK CHARACTER)

x_i	w	h	y	<>	n	o	t	?
h_i	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8
w_i	0	0	1	1	0	0	1	1
e_i^x	0	0	h_3	h_4	0	0	h_7	h_8
Updated h_i	h_1	h_2	0	0	h_5	h_6	0	0

layer in the form of

$$h_i = \tanh(W_h x_i + U_h h_{i-1}), \tag{7}$$

$$\tilde{w}_i = \sigma(W_{\tilde{w}} x_i + U_{\tilde{w}} h_{i-1} + V_{\tilde{w}} w_{i-1}), \tag{8}$$

$$w_i = \begin{cases} 0, \text{if} \tilde{w}_{i,1} \geq \tilde{w}_{i,2} \\ 1, \text{if} \tilde{w}_{i,1} < \tilde{w}_{i,2} \end{cases}, \tag{9}$$

$$e_i^x = w_i h_i, \tag{10}$$

$$h_i \leftarrow (1 - w_i) h_i, \tag{11}$$

where, $x_i \in \Re^{K_x}$ (K_x is the vocabulary size generated from the source language, i.e., the number of characters in English in this paper); σ is the sigmoid activation function; $W_h \in \Re^{m \times K_x}$, $U_h \in \Re^{m \times m}$, $W_{\tilde{w}} \in \Re^{2 \times K_x}$, and $U_{\tilde{w}} \in \Re^{2 \times m}$ are the weight matrices; and m is the dimension after word embedding. Here, h_i is the output of the hidden unit of an RNN, and $h_0 = 0$. w_i can be regarded as a gate, which is used to determine whether or not we output h_i at time i or not. Thus, given a source sequence, we can segment the characters dynamically, such that a certain number of "words" are formulated in a principled manner. h_i is updated by Eq.11 in order to ensure that e_i^x will not include the information of previous characters. Fig.2 shows the architecture of our proposed character-level sequence representation using an RNN. For clarity, we give an example to illustrate the process of our method, as shown in Table I. Here, the source sequence x is "why not?". The length of this sequence at the character level is $T_x = 8$. The desirable segmentation result should be "why| |not|?". Then, the output produced by RNN will be represented by $e^x = (0, 0, h_3, h_4, 0, 0, h_7, h_8)$ (here, the space is treated as a word). It is worth noting that the gate output w_i of the last character is always set to 1. Moreover, in order to maintain the length T_x of the source sequence, we did not choose to delete the components with value 0 in e^x.

Similar to the above process, we calculate the word vector e^y given a target sequence in the decoder as shown in Fig.1. The target sequence is represented by $\mathrm{y} = (y_1, y_2, ..., y_{T_y})$, $y_i \in \Re^{K_y}$, where K_y is the vocabulary size generated from the target language (i.e., the number of characters in Chinese in this paper). In comparison to the distributed representation [14], even though the training process of this word vector generation method based on an RNN is slightly complicated, the automated word segmentation featured by our proposed method is very desirable. Distributed representation relies on projecting a high-dimensional word vector formulated by the bag-of-words model onto a low-dimensional feature space. The projection does not consider input sequences in time order at all. By contrast, our method reads a sequence in time order and produces a low-dimensional word vector by segmenting characters dynamically.

C. Language Model

The earliest work on language models is the Neural Network Language Model (NNLM) introduced by Bengio et al. [7]. This model aims at predicting the current word by using the information of previous words. In addition to NNLM, an RNN can also be used for building a language model, because the hidden units in an RNN can be utilized for learning historical data. One of the popular used RNN structures is LSTM [15], which is capable of learning long-term dependencies. It usually outperforms the traditional Hidden Markov Model (HMM) in the fields of speech recognition [16], machine translation [2][8], etc. There are many popular LSTM variants, showing different performances on certain tasks [17]. In this paper, we employed a slightly more dramatic variation on the LSTM, the Gated Recurrent Unit (GRU), introduced by Cho et al. [8]. The resulting model is simpler than standard LSTM models, and has been increasing in popularity. On the basis of GRU, we improved the basic structure of GRU by adding an output gate.

All RNNs have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module has a very simple structure, such as a single $tanh$ layer. LSTMs also possess this chain-like structure, but the repeating module has a different structure. GRU merges the cell state and hidden state by integrating the "forget gate" and "input gate" into a single "update gate". However, this combination may affect the attention scheme adopted in the alignment model. Hence, we design an improved GRU with an additional output gate without overshadowing the basic structure of GRU. Our improved GRU includes two control gates and an output gate. Specifically, the language model is generated by

$$\vec{z}_i = \sigma(\vec{W}_z e_i^x + \vec{U}_z \vec{g}_{i-1}), \tag{12}$$

$$\vec{r}_i = \sigma(\vec{W}_r e_i^x + \vec{U}_r \vec{g}_{i-1}), \tag{13}$$

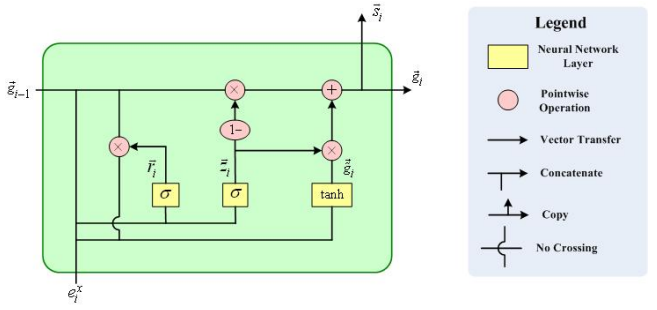
$$\vec{\tilde{g}}_i = \tanh(\vec{W}_g e_i^x + \vec{U}_g [\vec{r}_i \circ \vec{g}_{i-1}]), \tag{14}$$

$$\vec{g}_i = (1 - \vec{z}_i) \circ \vec{g}_{i-1} + \vec{z}_i \circ \vec{\tilde{g}}_i, \tag{15}$$

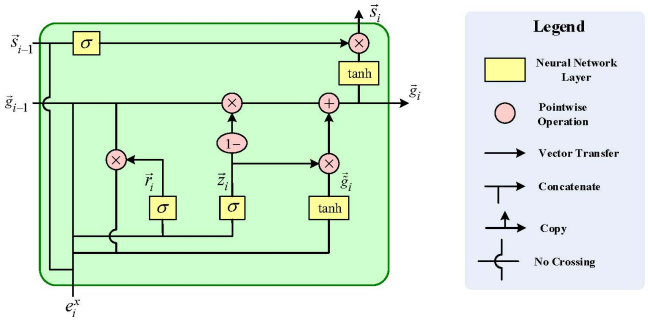
$$\vec{s}_i = \sigma(\vec{W}_s e_i^x + \vec{U}_s \vec{s}_{i-1}) \circ \tanh(\vec{g}_i), \tag{16}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2601521, IEEE Transactions on Industrial Informatics

where, $e_i^x \in \Re^m$ is the output vector for sequence representation using an RNN (see Section II-B); $\vec{W}_z, \vec{W}_r, \vec{W}_g \in Re^{n\times m}$, $\vec{U}_z, \vec{U}_r, \vec{U}_g$, and $\vec{U}_s \in Re^{n\times n}$ are the weight matrices; $\vec{g}_0 = \vec{0}$; $\vec{s}_0 = \vec{0}$; $\vec{z}_i$ represents the update gate, which allows each hidden state to maintain the activated state in the previous steps; $\vec{r}_i$ is a reset gate, which controls what and how much information needs to be reset from the previous state; and $\vec{s}_i$ is an output gate, which is used to filter out unimportant information involved in the attention scheme of the alignment model. Thus, each hidden state has an update gate, reset gate and output gate, which capture the memory dependencies in different time scales. For clarity, Fig.3(b) describes the LSTM diagram that we used in this paper in comparison to the original GRU shown in Fig.3(a).



(a)



(b)

Fig. 3. Network structure for language model: (a) the original GRU; (b) our improve GRU.

Similar to RNNsearch [1], we used a bidirectional RNN (BiRNN), which has been successfully applied in speech recognition. A BiRNN includes forward and backward RNNs. The forward RNN reads the input sequence $(x_1, x_2, ..., x_{T_x})$ and calculates a sequence of forward hidden states $(\vec{s}_1, \vec{s}_2, ..., \vec{s}_{T_x})$. The backward RNN reads the sequence in the reverse order, resulting in a sequence of backward hidden states $(\overleftarrow{s}_1, \overleftarrow{s}_2, ..., \overleftarrow{s}_{T_x})$. By concatenating the forward hidden state $\vec{s}_i$ and the backward one $\overleftarrow{s}_i$, i.e. $s_i = [\vec{s}_i^{\mathrm{T}}; \overleftarrow{s}_i^{\mathrm{T}}]^{\mathrm{T}}$, we obtain the hidden layer annotating the source sequence.

In the decoder, we can use a similar GRU to obtain s'_i, which contains the annotation information of the target sequence. However, it is not required to add an output gate (see Eq.16), as this function will be implemented in the target sequence generation process. In order to connect the hidden states s_i and s'_i, a context vector c_i is used to indicate the attention scheme in the encoder-decoder mapping process. We explain in detail how the context vectors are computed in the alignment model. The language model in the decoder is given by

$$z'_i = \sigma(W'_z e^y_{i-1} + U'_z s'_{i-1} + C'_z c_i), \tag{17}$$

$$r'_i = \sigma(W'_r e^y_{i-1} + U'_r s'_{i-1} + C'_r c_i), \tag{18}$$

$$\tilde{s}'_i = \tanh(W'_s e^y_{i-1} + U'_s [r'_i \circ s'_{i-1}] + C'_s c_i), \tag{19}$$

$$s'_i = (1 - z'_i) \circ s'_{i-1} + z'_i \circ \tilde{s}'_i, \tag{20}$$

where $e^y_i \in \Re^m$ is the word embedding vector for the target language; $W'_z, W'_r, W'_s \in \Re^{n\times m}$, $U'_z, U'_r, U'_s \in \Re^{n\times n}$, and $C'_z, C'_r, C'_s \in \Re^{n\times n'}$ are weights; and $s'_0 = \tanh(V_s \overleftarrow{s}_1)$.

D. Alignment Model

In a typical RNN encoder-decoder framework, the encoder is responsible to encode all of the information in the source sentence into a fixed-length vector c and to decoder the vector into a variable-length target sentence. Recent studies, e.g., RNNsearch [1], have shown that a soft alignment is capable of achieving significantly improved translation performance over the basic encoder-decoder approach. The alignment model introduced in RNNsearch encodes the input sentence into a sequence of vectors, c_i, and chooses a subset of these vectors adaptively while decoding the translation. In line with RNNsearch, we used a similar method to define the context vector c_i in the form of

$$a_{ij} = V_a^{\mathrm{T}} \tanh(W_a s'_{i-1} + U_a s_j), \tag{21}$$

$$\alpha_{ij} = \frac{\exp(a_{ij})}{\sum_{k=1}^{T_x} \exp(a_{ik})}, \tag{22}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} s_j, \tag{23}$$

where, $s_j \in \Re^{2n}$ is a hidden state generated by the language model for the source sequence; $s'_{i-1} \in \Re^n$ is a hidden state for the target sequence; and $W_a \in \Re^{n'\times n}$, $U_a \in \Re^{n'\times 2n}$, and $V_a \in \Re^{n'}$ are weight matrices. It is worth noting that, unlike in traditional machine translation, the alignment variable a_{ij} is not considered to be a latent variable. Instead, it is an energy function, which reflects the importance of the annotation s_j with respect to the previous hidden state s'_{i-1} in deciding the next s'_i and generating y_i. This implements a scheme of attention in the decoder. The decoder decides which parts of the source sentence to pay attention to [1]. Here, the design of the alignment model requires considering the lengths of sentence pairs, i.e., T_x and T_y. We usually need $T_x \times T_y$ operations. In order to minimize the computational burden, we can calculate $U_a s_j$ in advance, as it does not rely on time i.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2601521, IEEE Transactions on Industrial Informatics

Algorithm 1 Character-level learning model

Input: $\mathrm{x} = (x_1, x_2, ..., x_{T_x})$.
//Encoder
$\tilde{\mathrm{h}}_0 \leftarrow 0, \vec{w}_0 \leftarrow 0, \vec{s}_0 \leftarrow 0$
for $\vec{x}_t$ **in** $\mathrm{x} = (x_1, x_2, ..., x_{T_x})$ **do**
$\vec{e}_t^{\,x} = embed(\vec{h}_{t-1}, \vec{w}_{t-1}, \vec{x}_t)$, $\vec{s}_t = language(\vec{s}_{t-1}, \vec{e}_t^{\,x})$
endfor
$\overleftarrow{\mathrm{h}}_0 \leftarrow 0, \overleftarrow{w}_0 \leftarrow 0, \overleftarrow{s}_0 \leftarrow 0$
for $\overleftarrow{x}_t$ **in** $(x_{T_x}, x_{T_x-1}, ..., x_1)$ **do**
$\overleftarrow{e}_t^{\,x} = embed(\overleftarrow{h}_{t-1}, \overleftarrow{w}_{t-1}, \overleftarrow{x}_t)$, $\overleftarrow{s}_t = language(\overleftarrow{s}_{t-1}, \overleftarrow{e}_t^{\,x})$
endfor
$s = concatenate(\vec{s}, \overleftarrow{s})$
//Decoder
$s'_0 \leftarrow \tanh(V_s \overleftarrow{s}_1), e_0^y = 0$
while $y_{i-1} \neq' < eol >'$ **do**
$c_t = alignment(s'_{t-1}, s)$, $s_t = language(s_{t-1}, c_t, e_{t-1}^y)$
$y_t = output(s'_t, c_t, e_{t-1}^y)$, $e_t^y = embed(h_{t-1}, w_{t-1}, y_t)$
endwhile
Output: $y = (y_1, y_2, ..., y_{T_y})$

E. Target Sequence Generation

We have described three important components in our character-level sequence-to-sequence learning, i.e. character-level sequence representation, language model, and alignment model, in previous sections. The whole process of translating a source sequence $\mathrm{x} = (x_1, x_2, ..., x_{T_x})$ into a target sequence $y = (y_1, y_2, ..., y_{T_y})$ has been shown in Fig.1. However, this neural translation model is only an unsupervised, forward learning process. In order to learn a well-established translation model, we need a target sequence generation module to train language pairs in a large-scale data set. This module will be embedded into the decoder and help to generate a target sequence y. Given the word vector e_{i-1}^y generated in the previous step, the hidden state s'_i in the decoder, and the context vector c_i, at time i, the decoder calculates the probability that generates y_i in the form of

$$\tilde{t}_i = \sigma(W_t e_{i-1}^y + U_t s'_i + C_t c_i), \tag{24}$$

$$t_i = \max\{\tilde{t}_{i,2j-1}, \tilde{t}_{i,2j}\}, (j = 1, ..., l), \tag{25}$$

$$p(y_i | e_{i-1}^y, s'_i, c_i) = y_i^{\mathrm{T}} soft\max(W_p t_i), \tag{26}$$

where $W_t \in \Re^{2l \times m}$, $U_t \in \Re^{2l \times n}$, $C_t \in \Re^{2l \times n'}$, and $W_p \in \Re^{K_y \times l}$ are weight matrices. In Eq.25, we used a maxout [18] unit, which can be seen as a nonlinear activation function. A softmax function [19] is used to generate the probability that y_i is the target output character.

For clarity, we summarize the forward process of our proposed character-level sequence-to-sequence learning algorithm in **Algorithm 1**. The *embed(.)* function denotes the character-level sequence representation using RNN as shown in Section II-B; *the language(.)* function denotes the calculation process in the language model as described in Section II-C; the *concatenate(.)* function indicates that we concatenate two vectors together; the *alignment(.)* represents the process of alignment model in Section II-D; and the *output(.)* function denotes the target sequence generation module as introduced in Section II-E.

TABLE II
STATISTICS OF WORD-BASED SEQUENCE LENGTH

Length of Sequence	Training Set (EN)	Training Set (ZH)	Test Set (EN)	Test Set (ZH)
1∼9	1270522	1366950	4226	2433
10∼19	520850	432225	774	2421
20∼29	4921	820	-	145
30∼39	3137	5	-	1
40∼49	541	-	-	-
50∼59	29	-	-	-

III. MODEL LEARNING AND EXPERIMENTAL SETTINGS

A. Data Set and Preprocessing

In order to build a well-established neural sequence-to-sequence translation model, it is essential to compile a large-scale data set for training. Regarding the task of English-to-Chinese subtitle translation, there is no public data set at present. Fortunately, there are several online websites, which provide high-quality subtitle translations offered by many volunteers. To train our proposed model, we collected the subtitles of 16,987 movies or television series from the Internet. Each subtitle file is in the format of SRT (Subripper). We extracted 13,717,380 English-Chinese sequence pairs in total from these subtitle files. Due to computational capacity, we randomly selected 1,800,000 language pairs for training and 5,000 ones for testing. Intuitively, sequence length constitutes an important factor that affects translation performance. The data set statistics with respect to sequence length for training and test are summarized in Table II and Table III. Table II shows the statistics of sequence length based on word-level sequence representation. We performed word segmentation for Chinese with the Stanford NLP toolkit, and English tokenization with the tokenizer from Moses[1]. In training of other models compared with our method, we limited the source and target vocabulary to the most frequent 30,000 words in English and Chinese. All of the OOV words were mapped to a special token UNK. Table III summarizes the statistics of sequence length based on the character-level sequence representation used for our proposed model. Our model accepts a sequence of encoded characters as input. For English/Chinese sequences, they are segmented by English letters/Chinese characters, punctuation characters, or other characters. The vocabulary used for English and Chinese sequences consists of 137 English characters and 5,133 Chinese characters, covering all of the occurred characters of the two corpora, respectively. In comparison to the traditional word-level sequence representation, the dimension of input sequence based on our character-level representation has been reduced in 219 times for English sequence and 5.8 times for Chinese sequence, respectively. Moreover, there is no demand for post-processing steps, as done in [12], to handle the OOV words.

B. Model Training

1) Model Size: For all of the models used in this paper, the word embedding dimensionality m is 1,000, the length

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2601521, IEEE Transactions on Industrial Informatics

TABLE III
STATISTICS OF CHARACTER-BASED SEQUENCE LENGTH

Length of Sequence	Training Set (EN)	Training Set (ZH)	Test Set (EN)	Test Set (ZH)
1∼9	97169	137409	219	2349
10∼19	256970	356670	664	2625
20∼29	417398	569065	1193	26
30∼39	456448	615481	1361	-
40∼49	339007	454852	1004	-
50∼59	181520	242359	462	-
60∼69	48107	63980	91	-
70∼79	3052	4424	5	-
80∼89	298	417	1	-
90∼99	26	31	-	-
100∼109	5	7	-	-

of hidden states s or s' in the language model is 2,000, the number of hidden units in the alignment model n' is 2,000, and the size of the maxout hidden layer in the target sequence generation module l is set at 1,000.

2) Parameter Initialization: We initialized the recurrent weight matrices U_h, U_w, V_w, $\vec{U}_z$, $\vec{U}_r$, $\vec{U}_s$, $\vec{U}_w$, $\overleftarrow{U}_z$, $\overleftarrow{U}_r$, $\overleftarrow{U}_s$, $\overleftarrow{U}_w$, U'_z, U'_r, and U'_s as random orthogonal matrices. Here, we firstly generated associated random matrices, and then used the SVD (Singular Value Decomposition) to produce random orthogonal matrices. For the weight matrices in the alignment model, W_a and U_a, we initialized them by sampling each element from the Gaussian distribution of mean 0 and variance 0.001^2. All of the elements of V_a and all of the bias vectors were initialized to zero. Any other weight matrices were initialized by sampling from the Gaussian distribution of mean 0 and variance 0.01^2.

3) Parameter Learning: The Stochastic Gradient Descent (SGD) algorithm was used for model training. We used Adadelta [1] to automatically adapt the learning rate of each parameter ($\varepsilon = 10^{-6}$ and $\rho = 0.95$). We explicitly normalized the L_2-norm of the gradient of the cost function each time to be at most a predefined threshold of 1, when the norm was larger than the threshold [20]. Each SGD update direction was computed with a minibatch of 20 sequences.

Our model was developed in an open-source toolkit, Theano, and we used a GPU (Graphics Processing Unit) to speed up the training process. All of the simulations were performed on a PC with Intel Core i7-4790K, 4.00GHz, and 4 GB memory.

IV. EXPERIMENTAL RESULTS

A. Evaluation Measure

In order to evaluate the performance of our method, we adopted a widely used evaluation metric, case-insensitive four-gram NIST BLEU (Bilingual Evaluation Understudy) score [21], which is defined in the form of

$$BLEU = BP \cdot \exp(\sum_{n=1}^{N} w_n \log(p_n)), \tag{27}$$

where w_n is the weights of co-occurred n-grams, p_n represents the n-gram precisions, and BP denotes the length-based brevity penalty. The detailed calculation of BLEU can be found in [21]. It is important to note that the more reference translations per sentence there are, the higher the BLEU score is.

B. Compared Models

We compare our method, RNNembed, with two competitive models: Moses [22] and RNNsearch [1]. The open source phrase-based translation system Moses (with default configuration) is a well-established conventional statistical machine translator. The word alignment is developed with GIZA++ [23]. We used the SRI Language Modeling Toolkit [24] to train a four-gram language model with modified Kneser-Ney smoothing on the target portion of the training data. RNNsearch is used as the neural machine translation baseline, because it represents the state-of-the-art neural machine translation method. We used the same parameter settings and the training procedures for RNNsearch with our model. In fact, the traditional RNNsearch can read characters as input. Therefore, we implemented two versions of RNNsearch, a word-level RNNsearch (denoted as RNNsearch (word)) and a character-level RNNsearch (denoted as RNNsearch (character)). For comparison, we also evaluated another two versions based on our proposed model, RNNembed-1 and RNNembed-2. RNNembed-1 represents the model that we used with the same structure as RNNsearch, except that an output gate is newly added into the GRU (see Eq.16). RNNembed-2 indicates that we only use the character-level sequence representation with RNN without adding the output gate in the GRU. We take RNNembed as the final system that integrates character-level sequence representation with RNN and the improved GRU into a unified framework.

C. Quantitative Results

The comparative results of different models are given in Table IV. Unsurprisingly, the system Moses produces the highest BLEU score over other methods, as it is a well-established system by human experts. At present, all of the neural machine translation methods only achieve comparable performance over Moses, which has been observed by many researchers on different machine translation tasks [1][11]. RNNsearch based on word-level input is around 1.4 points behind Moses in BLEU, on average. It is worth noting that word-based RNNsearch delivered slightly better performance than Moses when only the sentences having no unknown words (UNK tokens) were evaluated on the task of English-to-French translation [1]. However, in a real-world translation task, UNK tokens must appear in many sequences if we use a small vocabulary without post-processing the rare words. As shown in Table IV, character-based RNNsearch performs the worst in comparison to the other methods, which indicates that the traditional RNNsearch is not suitable for reading characters as input. However, our proposed model, RNNembed, is designed especially for inputting character-level sequences. RNNembed can already achieve slightly better performance than word-based RNNsearch and deliver results comparable to Moses. It is also observed that RNNembed is able to produce better

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2601521, IEEE Transactions on Industrial Informatics

TABLE IV
COMPARATIVE RESULTS OF DIFFERENT MODELS

System	$BLEU$	BP	P_1	P_2	P_3	P_4
Moses (word)	26.59	0.980	55.6	32.7	21.4	13.9
RNN-search (word)	25.21	0.814	58.4	36.5	25.6	16.8
RNNembed-1 (word)	22.65	0.763	58.3	34.9	23.7	16.1
RNN-search (character)	21.21	1	38.7	25.0	17.2	12.1
RNNembed-1 (character)	22.2	1	39.8	26.0	18.2	13.0
RNNembed-2 (character)	22.25	1	40.0	26.1	18.2	12.9
RNN-embed (character)	25.29	1	42.4	29.0	21.2	15.7

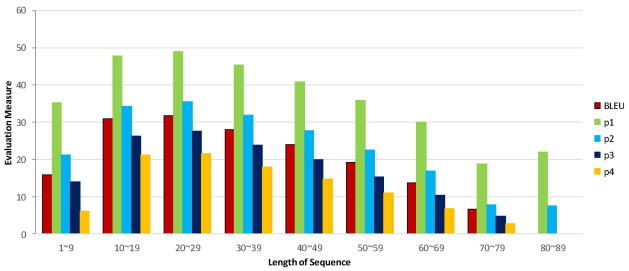


Fig. 4. Performance of RNNembed against length of sequences.

results in comparison to its two variants, i.e., RNNembed-1 and RNNembed-2.

D. Study on Sequence Length

On various translation tasks, the sequence length is critical for affecting the performance of different learning methods. In order to further investigate the performance of RNNembed with respect to the length of a given sequence, we divided the test set into nine subsets according to the length of sentences with 1∼9, 10∼19, ..., 80∼89. The results of RNNembed on these 10 subsets are summarized in Fig.4. The average length of sentences based on word-level representation is 7, while the average length of character-level sentences is 32. It is observed that RNNembed achieves the best performance over sequences with lengths from 20 to 29. Along with the increase of sentence length, the BLEU score drops quickly. This indicates the difficulty for RNNembed to handle lengthy sequences, which can constitute an interesting problem for future work.

E. Examples

Apart from the objective evaluation based on quantitative results, we selected some translation examples to further investigate the performance of our method. Here, RNNsearch and RNNembed were trained by character-level sequence representation. We also compared RNNembed with two popular used translate softwares in industry[1], i.e. Google Translate and Youdao Translate[2]. Table V shows the translation results of two short sentences. Basically, RNNembed, RNNsearch and the two industrial softwares are able to translate these sentences correctly. But RNNembed is capable of adding more "emotions" into the translation in comparison to other

[1]All the results were retrieved on 25 Nov. 2015.

[2]A popular used machine translation software in China, http://fanyi.youdao.com/

TABLE V
TRANSLATION EXAMPLES OF SHORT SENTENCES

Source sequence	Wow. I like it.	I love you.
Reference	我喜欢	我爱你
RNN-search	我喜欢这样的样子	我很喜欢你们
RNN-embed	我真喜欢这东西	我爱死你们了
Google Translate	哇。我喜欢。	我爱你
Youdao Translate	哇。我喜欢它。	我爱你

TABLE VI
TRANSLATION EXAMPLES OF RELATIVELY LONG SENTENCES

Source sequence	Look, man, you don't get to do anything	This one means a lot to me.
Reference	兄弟 你什么都不需要做	这个奖杯意义重大
RNN-search	听着 你不必要做任何事情的事情 你做不了	这意味这我的意思是我的意见了
RNN-embed	听着 老兄 你什么都不用做	这意味着 对我很重要
Google Translate	你看，男人，你不应该尽。	这其中意味着很多给我。
Youdao Translate	男人，你不需要做任何事。	正这个对我来说意味着很多。

TABLE VII
FAILURE EXAMPLE

Source sequence	Until our mission is complete you will have no names.
Reference	直到任务结束前 你们都没有名字
RNN-search	直到我们的家人都不知道你们的名字是什么意义的时候我们的
RNN-embed	直到我们的任务结束 你们完全不知道名字没有告诉你
Google Translate	直到我们的任务完成后，你不会有任何名称。
Youdao Translate	直到我们的任务是完成你将没有名字。

methods. Moreover, we summarized the translation results of three relatively long sentences in Table VI and Table VII. It is worth noting that long sentences usually will not appear in subtitles. This can be observed in dataset statistics as shown in Table II and Table III. For the first two sentences shown in VI, the result using RNNembed is closer to the reference in the test set in comparison to other methods. For the third sentence shown in VII, all the methods can not translate it well. But, obviously, the result delivered by RNNembed is more readable than that of RNNsearch.

V. CONCLUSION

In this paper, we proposed a character-level sequence-to-sequence learning method, RNNembed. This method reads quantized characters into the translation system, instead of using a predefined vocabulary with a limited number of words. On the task of English-to-Chinese subtitle translation, we embedded an RNN into the encoder-decoder approach for generating character-level sequence representation. We also improved the GRU in the language model of the encoder. The proposed model was examined in a large-scale subtitle dataset compiled by ourselves. Experimental results show that RNNembed is able to achieve a translation performance comparable to the most competitive word-based model, RNNsearch [1], and can deliver a result close to the well-established phrase-based system, Moses. The benefit of RNNembed is rendering a usual tokenization unnecessary, resulting in there being no need to handle unknown or rare words. To the best of our knowledge, the present study constitutes a pioneering work on building a pure neural network model for sequence-to-sequence learning by reading raw characters as input. One limitation of our method lies in sequence length, that is, it favors short sequences. One future challenge will be identifying ways to better cope with lengthy sequences. A possibility

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2016.2601521, IEEE Transactions on Industrial Informatics

for improving our model is to develop a deep-structured RNN for generating word vectors. It is potential to visualize the word segmentation or character clustering results from our model, and is worthy of further investigation. It would also be interesting to extend our work to automated answering systems [25].

REFERENCES

[1] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of International Conference on Learning Representations*, 2015, pp. 1-15.

[2] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, pp. 3104-3112, 2014.

[3] R. Collobert, *et al.*, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, pp. 2493-2537, 2011.

[4] D. Chen and C. D. Manning, "A fast and accurate dependency parser using neural networks," in *Proceedings of International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 740-750.

[5] M. R. Costa-Jussá and M. Farrús, "Statistical machine translation enhancements through linguistic levels: A survey," *ACM Computing Surveys*, vol. 46, no. 3, Article 42, pp. 1-28, 2014.

[6] R. Miikkulainen and M.G. Dyer, "Natural language processing with modular neural networks and distributed lexicon," *Cognitive Science*, vol. 15, pp. 343-399, 1991.

[7] Y. Bengio, *et al.*, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137-1155, 2003.

[8] K. Cho, *et al.*, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proceedings of International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724-1734.

[9] Kalchbrenner, Nal, and Phil Blunsom. Recurrent Continuous Translation Models. In *Proceedings of International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013, vol. 3, no. 39, pp. 1700-1709.

[10] M. Auli, *et al.*, "Joint language and translation modeling with recurrent neural networks," In *Proceedings of International Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013, vol.3, no. 8, pp. 1044-1054.

[11] Sébastien Jean, *et al.*, "On using very large target vocabulary for neural machine translation," in *the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015, pp. 1-10.

[12] M. Luong, *et al.*, "Addressing the rare word problem in neural machine translation," in *the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015, pp. 1-9.

[13] Z. Yan and J. Wang, "Model predictive control of nonlinear systems with unmodeled dynamics based on feedforward and recurrent neural networks," *IEEE Transactions on Industrial Informatics*, vol. 8, no. 4, pp. 746-756, Nov. 2012.

[14] A. Paccanaro, G. E. Hinton, "Learning distributed representations of concepts using linear relational embedding," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 2, pp. 232-244, Apr. 2001.

[15] Jürgen Schmidhuber, "Deep learning in neural networks: an overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015.

[16] G. Hinton, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82-97, 2012.

[17] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 2342-2350.

[18] I. Goodfellow, *et al.*, "Maxout Networks," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 1319-1327.

[19] G. E. Hinton and R. R. Salakhutdinov, "Replicated Softmax: an Undirected Topic Model," *Advances in Neural Information Processing Systems*, pp. 1607-1614, 2009.

[20] L. Deng and D. Yu, "Deep learning: methods and applications," *Journal of Foundations and Trends in Signal Processing*, vol. 7, no. 3-4, pp. 197-387, 2014.

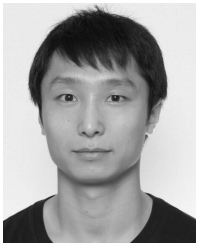
[21] K. Papineni, *et al.*, "BLEU: a Method for Automatic Evaluation of Machine Translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002, pp. 311-318.

[22] P. Koehn, *et al.*, "Moses: open source toolkit for statistical machine translation," in *Proceedings of the 45th Annual Meeting on Association for Computational Linguistics on Interactive Poster and Demonstration Sessions*, 2007, pp. 177-180.

[23] D. Chiang, "Hierarchical phrase-based translation," *Computational Linguistics*, vol. 33, no. 2, pp. 201-228, 2007.

[24] I. Androutsopoulos and P. Malakasiotis, "A survey of paraphrasing and textual entailment methods," *Journal of Artificial Intelligence Research*, vol. 38, pp. 135-187, 2010.

[25] T. Hao and Y. Qu. "QSem: A novel question representation framework for question matching over accumulated question – answer data," *Journal of Information Science*, Aug. 2015, Article ID: 0165551515602457.



Haijun Zhang (M'13) received the B.Eng. and Master's degrees from Northeastern University, Shenyang, China, and the Ph.D. degree from the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, in 2004, 2007, and 2010, respectively. He was a Post-Doctoral Research Fellow with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON, Canada, from 2010 to 2011. Since 2012, he has been with the Shenzhen Graduate School, Harbin Institute of Technology, China, where he is currently an Associate Professor of Computer Science. His current research interests include neural networks, multimedia data mining, machine learning, computational advertising, and evolutionary computing.

Jingxuan Li received her BS and MS degrees from the Harbin Institute of Technology in 2013 and 2016, respectively. She was a Master candidate in Computer Engineering of the Harbin Institute of Technology Shenzhen Graduate School, under this research performed. She is currently working on big data analysis at HUAWEI TECHNOLOGIES CO., LTD. Her research interests include data mining, natural language processing, and deep learning.

Yuzhu Ji received the MS degree in Computer Engineering from the Harbin Institute of Technology Shenzhen Graduate School in 2015. Before that, he received his BS degree from the PLA Information Engineering University. He is interested in data mining, computer vision, image processing, and deep learning. He is currently a PhD student in the Harbin Institute of Technology Shenzhen Graduate School.

Heng Yue received the PhD degree in Control Theory & Control Engineering from Northeastern University, China, in 1999. He is a Professor in the National Key Lab of Integrated Automation of Process industries, Northeastern University, China. He was the recipients of two National Second-Grade Awards of Technology Invention. He has been supported by the Program for New Century Excellent Talents in University in China. His research interests include: neural network, intelligent control, modeling and optimizing control of complex industrial processes.