Received November 18, 2018, accepted November 28, 2018, date of publication December 10, 2018, date of current version January 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2885698

Unlabeled Short Text Similarity With LSTM Encoder

LIN YAO[1], ZHENGYU PAN[1,2], AND HUANSHENG NING[1,2], (Senior Member, IEEE)

[1]School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China
[2]Beijing Engineering Research Center for Cyberspace Data Analysis and Applications, Beijing 100083, China

Corresponding author: Huansheng Ning (ninghuansheng@ustb.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61471035, Grant 61872038, and Grant 61811530335, and in part by the Fundamental Research Funds for the Central Universities under Grant FRF-BD-18-016A.

ABSTRACT Short texts play an important role in our daily communication. It has been applied in many fields. In this paper, we propose a novel short text similarity measurement algorithm-based long short-term memory (LSTM) encoder. It contains preprocessing, training, and evaluating stages. Our preprocessing algorithm can avoid gradient vanishing problems in the process of backward propagation faster after normalization. The training stage fully leverages the inception module to extract the features of different dimensions and improves the LSTM network to process the relationships of word sequences. The evaluating stage employs cosine distance to calculate the semantic similarity of two short texts. We do experiments on two short text dataset of different lengths and analyze the experiment result. The experiment result shows that our algorithm can fully employ semantic information and sequence information of short texts and have a higher accuracy and recall compared to other short text similarity measurement algorithms.

INDEX TERMS Unlabeled, short text, similarity measurement, LSTM encoder.

I. INTRODUCTION

Short texts play an important role on our daily communication. It has been applied in many fields, like search engine [1], question answer system [2], text classification [3], machine translation [4], information retrieval [5] and more. If two words have a similar character sequence, they are similar lexically. On the contrary, they are similar semantically if two words have similar meanings [6]. However, how to measure similarities between two short texts is still a tough issue. Many scholars have done researches on it.

Some approaches have been proposed for short text similarity. They contain a lexical match, knowledge base, bag of word model, and Neural network. Lexical match mainly compares character similarities between two short texts according to edit distance [7], largest common subsequence distance, Jaccard similarity coefficient [8], or lexical overlap [9]. However, since a lexical match doesn't consider the semantic information of word and between words in a short text, the effect of lexical match is limited. For example, the literals of synonyms are different, but their meanings are the same. WordNet [10]–[15] and Wikipedia[1] are the most commonly used knowledge bases in comparing short text similarity. But they're not available to all languages, and domain-specific terms. Bag of Word Model (BOW) was built based on distributional hypothesis in which words that occur in the same contexts tend to have similar meanings. The basic idea of BOW was that text was represented as a combination of a series of words. Taking semantic levels into considered, BOW mainly included vector space model [16], latent semantic analysis [17], and latent Dirichlet Allocation [18]. But these three algorithms don't consider the order of terms in text. For example, ‘‘people hit dog'' and ‘‘dog hit people'' are the same text from the view of BOW, but in the fact they have different meanings. Neural network is a new hot research direction in NLP. Word2Vec model [19] is a shallow Neural network. When short text was fed into Word2Vec model, word vectors were generated. The essence of word vectors was that low-dimensional real vectors are trained from unlabeled unstructured text. The distance of these word vectors represents two texts' similarity. However, the order of words is not taken into account either.

In this paper, we propose a novel short text similarity measurement algorithm based LSTM autoencoder. Only unlabeled text data is required in the algorithm. This algorithm is divided into three stages: preprocessing, training, and evaluating. We preprocess and normalize text data during the

[1]https://www.wikipedia.org/

preprocessing stage. In training stage, we leverage inception module [20] to extract different sizes of convolution feature. Besides, we also improve LSTM network in order to use sequence information. Lastly, we employ cosine distance to calculate vector distance between two short texts in evaluating stage. In conclusion, the main contributions of this paper is as follows:

. It proposes a novel normalization algorithm to preprocess short text.
. It improves LSTM cell to control whether choose to forget the information accumulated before.
. It proposes a new algorithm computing the short unlabeled text similarity based LSTM autoencoder.
. It only requires unlabeled short text data.
. It solves semantic similar problems for word sequences of text.

The rest of the paper is organized as follows. In Section 2, we describe related work. We introduce our algorithm based LSTM encoder in Section 3. The experiments are presented in Section 4. We analyze the experiment result in two datasets in Section 5. In Section 6, the conclusion and future work are given.

II. RELATED WORK

There are many scholars researching short text similarity. We can classify their work into four kinds.

A. LEXICAL MATCH

Lexical match refers to compare two short texts according to the extent to which two short texts contain similar characters. The more the same characters or words two short texts contain, the more similar they are. There have been several methods proposed currently. Edit distance is the first algorithm to calculate the similarity of short texts. The main idea of edit distance [7] is to calculate transform times from the first short text changing to the second short text. The transform includes inserting, deleting or replacing a character. However, the largest common subsequence distance is the edit distance with inserting or deleting only. And it also counts the transform times in two short texts. Jaccard similarity coefficient [8] is also a practical short text similarity algorithm. It calculates the ratio of the intersection and union of two short texts. The larger the value is, the more similar the two texts are. When it comes to massively parallel computing, the algorithm has certain advantages in efficiency. TF-IDF is the most commonly used algorithm that computes the term frequency of every word and inverse document frequency in the whole corpus. We count all the words in two short text as a vector. Then, we calculate TF-IDF value of every word in short text as every dimension the vector above. Lastly, we compute the consine distance of two short text vector. This consine distance represents two short texts' similarity. Due to lexical match doesn't take semantic information into account, we can't compute the similarity just depending on lexical match.

B. KNOWLEDGE BASE

The knowledge base is a system to store and query structured information. Currently, many knowledge bases have been proposed, such as Wikipedia, WordNet, DBpedia, YAGO. The common knowledge bases to calculate short text similarity are Wikipedia and WordNet. The earliest using Wikipedia to calculate semantic similarity is WikiRelate! proposed by Strube and Ponzetto [21]. They searched the pages related with terms and got category tree by extracting terms' category. Finally, computing semantic similarity was based on extracted pages and paths along the category taxonomy. Reference [22] proposed Explicit Semantic Analysis (ESA) method that represented the meaning of texts in a high-dimensional space of concepts derived from Wikipedia and represented term as a weighted vector of Wikipedia-based concepts. By comparing the corresponding vectors using conventional metrics (e.g., cosine), ESA computed semantic relatedness. The model was easy to use and robust. Witten and Milne [23] used the hyperlink structure rather than category hierarchy or textual content in Wikipedia in order to balance efficiency and accuracy.

C. BAG OF WORD MODEL

Bag of Word Model is a representation way of text. It regards texts as the set of words and doesn't the order of word.The next are a few models that have been proposed. Vector space modele [16] is that every text is represented as a real vector in the model. N documents form an n-dimensional real-valued vector space, in which every dimension corresponds to terms in a text, two texts' similarity is calculated by two dimensional distance (i.e., cosine distance). Vector space model is a simple linear algebra model, but it has some limitations. For example, term order occurring to the document can't be expressed by vector. The texts with similar context but different terms have a bad similarity. Latent semantic analysis [17] models term and document to the same concept space. It constructs a matrix in which row indicates the word and column represents document. The similarity between word and word is compared by calculating the cosine distance or the dot product between the two vectors. The latent Dirichlet Allocation (LDA) [18] topic model is a three-layer Bayesian probability model. It models the topic of text to get topic distribution of text, and computes text similarity according to the topic distribution. These three algorithms didn't consider the order of term in text.

D. NEURAL NETWORK

Neural network is made up of multiple layer neural units. In general, it includes one input layer, one or more hidden layers, and one output layer. The Word2Vec model [19] is a shallow Neural network that is used to generate word embedding. The Word2Vec model contains three layers: input layer, hidden layer, output layer. Word vector is just the value of middle hidden layer. It has two architectures: Continuous Bag-of-Words (CBOW) and Skip-gram. CBOW model predicts the current word from context words. However, Skip-gram predicts context words from the current word. Reference [24] combined different dimensional word vectors that were trained by different algorithms, different corpus, different parameter settings, and got a classifier by supervised learning algorithm, lastly got a unlabeled text similarity score. Reference [25] presented a novel method to compute short text similarity that calculated the Word Mover's Distance (WMD) from all words in a text "travel" to corresponding word in another text. And this method was straightforward implemented and didn't need any hyperparameter. Reference [26] improved WMD algorithm and proposed a Supervised-WMD (SWMD) algorithm that applied to supervised text.

III. SHORT TEXT SIMILARITY ALGORITHM BASED LSTM ENCODER

Recurrent Neural Networks (RNN) is a kind of neural network whose input is sequence data $(x_1, \ldots, x_T)$ and connections between nodes form a directed graph along a sequence. It is widely used in natural language processing (NLP), speech recognition, and generating image descriptions recently. The input of the hidden layer includes not only the output of the input layer but also the output of the hidden layer at the previous moment. The equation is $S_t = f(U{*}X_t + W{*}S_{t-1})$, in which X_t is the input at time t, U is the weight matrix from input to current hidden state, W is the weight matrix from previous hidden state to current hidden state, f is activation function. However, RNN has gradient vanishing problems in which the gradients grow vanishingly small over long sequences in backpropagated process to make gradient-based training learning algorithms difficult for long distance learning. To solve gradient vanishing, long short term memory (LSTM) [27] comes into being. LSTM is a variant of RNN that can learn long-term dependencies. Hidden unit is only a single neural network layer unit in RNN. However, LSTM hidden unit is composed of a memory cell, an input gate, an output gate and a forget gate. Memory cell is used for storing a value or multiple values, input gate decides how many values enter the unit, output gate decides how many values output from unit, forget gate decides whether value remains in the unit.

Autoencoder is an unsupervised artificial learning neural network. It also has one input layer, one or more hidden layers, and one output layer. The output of output layer will fit the input of input layer as much as possible. It essentially compresses features and then decompresses them. It is mainly used for data dimensionality reduction and feature extraction. It is now also used in generative models and made up of encoder and decoder. In general, the output of encoder is used as the feature of data. Figure 1 is Schematic diagram of autoencoder.

To leverage semantic information and sequence information of short text, we propose a new algorithm based LSTM encoder to calculate short text similarity. It doesn't require labeling short text. Unlabeled short text can directly feed into our algorithm. Our algorithm includes preprocessing, training, and evaluation stage.

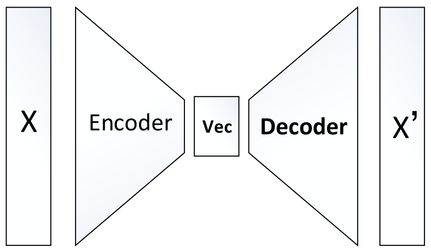

FIGURE 1. Autoencoder schematic diagram.

Algorithm 1 Preprocess Short Text

```
set model = word vector matrix after Word2ec
set vocab_vectors = the array of all word vectors
set max_x = maximum word vectors' norms of vocal_vectors
set predict= zero vector matrix whose dimension is same as model
for each word in model do
    predict[word]=(model[word]+max_x)/(2*max_x)
end for
for each short_text in dataset do
    train_list=[list(word) for word in short_text]
    train_list=[i+['']*(seqLen-len(i)) for i in train_list]
    train_list=[i[:seqLen] for i in train_list]
end for
```

A. PREPROCESSING SHORT TEXT

In preprocessing stage, we want to get vector representation of short texts. Firstly, we need to transform short text to vector format by Word2ec or GloVe [28]. Due to the value of every dimension after Word2ec doesn't have a fixed range, we pay attention to the direction rather than the length of the word vector. Besides, normalization can avoid gradient vanishing problems in the process of backward propagation. In order to make our autoencoder algorithm converge faster and unify the input size of autoencoder, we need to normalize the word vector. Algorithm 1 below is our preprocessing algorithm: We will assign a matrix variable 'predict' which its dimension is the same as the word vector in the first step. Secondly, we initialize the variables using zero. Then, we get the norms of every word vector in corpus and find the largest norm 'max_x'. Lastly, we calculate the normalized word vector by equation

$$predict[word] = (model[word] + max\_x)/(2 * max\_x). \quad (1)$$

Due to max_x is the maximum word vectors' norms of the array of all word vectors, it's a scalar. But model[word] is a vector. For every dimension of model[word] vector, we have adopted a broadcast algorithm to make every dimension add max_x. After normalizing, the representation of word vector

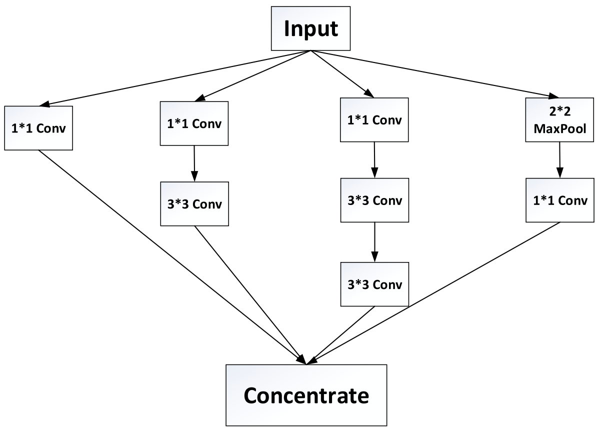

FIGURE 2. **Inception module.**

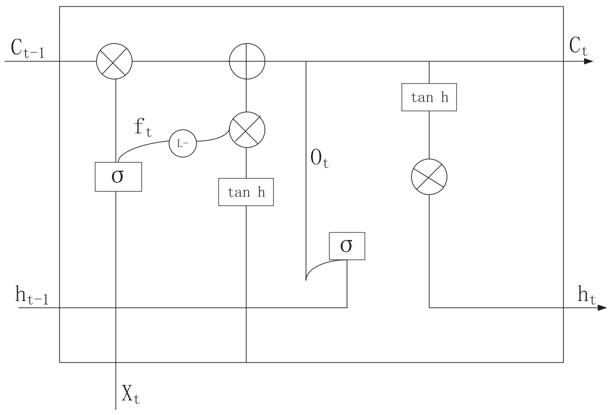

FIGURE 3. **Improved LSTM Cell.**

is that every dimension is between 0 and 1 and the mean value is 1/2. Moreover, we will truncate different size of short text into the same length for ease of calculation. When the length of short text is less than fixed length, we will add empty characters after the short text. The concrete equation is:

$$\text{train\_list} = [\text{i+['']*(seqLen-len(i)) for i in train\_list}]. \quad (2)$$

$$\text{train\_list} = [\text{i[:seqLen] for i in train\_list}]. \quad (3)$$

B. TRAINING THE WORD VECTORS OF SHORT TEXT

In training stage, our model employs autoencoder structure. We employ inception module [20] to extract more features in encoder part.

As shown in Figure 2, inception module leverages parallel convolution structures with different sizes of convolution kernels and concentrates the result of convolution and pooling of different branches. Inception module can extract short text features of different dimensions due to different sizes of convolution kernels. When convolution, we employ the "same" padding mode to make full use of edge text information. After convolution, we use the relu activation function to add nonlinear features. The formula is

$$f(x) = max(0, x) \quad (4)$$

Moreover, we improve LSTM network in order to use word sequence information fully in short text. The state of current input information can't affect the output information of output gate. We increase the connection from current input gate to output gate to control what every cell outputs. We combine forget gate and input gate to a new update gate. In this case, forget gate and input gate decide to forget and add new information together. Forget can be excuted when input. When forget certain information, new value can be added to the current state. LSTM network can choose to forget the information accumulated before. The improved LSTM cell is showed in Figure 3.

In t moment, the update of LSTM is as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f) \quad (5)$$

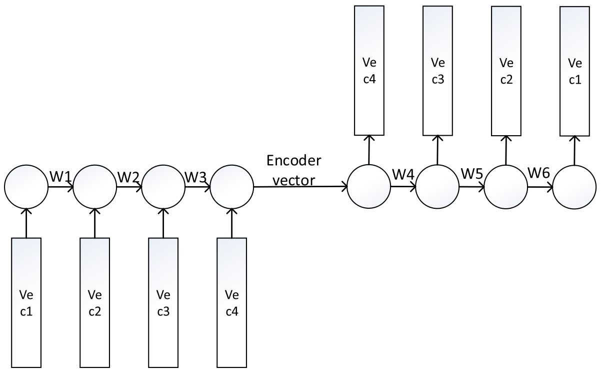

FIGURE 4. **LSTM autoencoder model.**

$$o_t = \sigma(W_o[C_t, h_{t-1}, X_t] + b_o) \quad (6)$$

$$\overline{C_t} = \tanh[W_c \cdot [h_{t-1}, X_t] + b_c] \quad (7)$$

$$C_t = f_t \cdot c_{t-1} + (1 - f_t) \cdot \overline{C_t} \quad (8)$$

$$h_t = o_t \circ \tanh(C_t) \quad (9)$$

The output of encoder is vector representation of short text. Decoder part use opposite structure, and is symmetrical with the encoder part. In our model, we employ adadelta optimizer that can adjust the learning rate automatically. The formula is:

$$g_t = (1 - \gamma) f'(\theta_t)^2 + \gamma g_{t-1} \quad (10)$$

And we use binary cross-entropy cost function. The formula is:

$$binaryCrossentropy(t, o) = -(t \cdot log(o) + (1-t) \cdot log(1-o)) \quad (11)$$

When the loss value of our model doesn't fall and has saturated, training stage ends. The output of encoder is the vector of short texts. Figure 4 is our LSTM autoencoder model.

C. EVALUATING THE SHORT TEXT SIMILARITY

To compare the similarity between two short texts, we will use cosine distance to calculate semantic similarity of two short text. The concrete steps is that we calculate the vector of every short text respectively firstly. Then, computing the cosine distance of two vectors. The distance lies in 0 and 1. And the mean value is 1/2. If the cosine distance of the two vectors is closer to 1, they are more similar. Otherwise, vice versa. The similarity formula is

$$similarity = \cos\varphi = \frac{vec1 \cdot vec2}{|vec1| \cdot |vec2|}. \quad (12)$$

vec1 represents the vector of the first short text and vec2 represents the vector of the second short text.

IV. EXPERIMENTS

A. GOAL

In this experiment, we want to compute the accuracy and recall of short text similarity algorithm for different sizes of short text and compare with short text similarity with LSTM encoder algorithm. Lastly, we analyze the reason of different algorithm performances.

B. DATASET

In this experiment, we measure on short texts of different lengths. We use two datasets: MSR Paraphrase Corpus dataset [29], [30] and Quora QR (Question Pairs) dataset. Every short text in MSR about 30 to 40 words. However, every short text in Quora Question Pairs dataset just have about 10 to 20 words. MSR dataset contains 5801 pairs of sentences extracted from web. Training set contains 4076 pairs and testset contains 1725 pairs. The format of every pair of sentences is "Quality #1 ID #2 ID #1 String #2 String". Quality indicates whether two sentences are semantically similar. In our experiment, we employ training set of corpus as training set. Because our model just need unlabeled data. We divide every pair of sentences into two sentences. Therefore our training set contains 9152 pieces of sentences. Besides, our testset just contains test set that its quality is 1 in MSR Paraphrase Corpus test set. The number is 1147 pairs of dataset.

Quora Question Pairs[2] is a new public question pair dataset recently. Because Quora as a knowledge sharing website doesn't allow similar questions to appear twice. For example, the question "Which country have the largest population in the world" has been asked. When other user publish the question "What is the most populous country all over the world", the system will give warning message: "the question is duplicate". Quora QR (Question Pairs) dataset contains 404302 pairs of questions. Every pair's format is 'id qid1 qid2 question1 question2 is_duplicate', when is_duplicate is 0, it represents that two question not similar. When is_duplicate is 1, it represents that two question is similar. There are 149263 similar question pairs in total. We randomly divide the data set into a training set, test set according to a ratio of 4:1. Finally, we look for the most similar question for every question in testset.

[2]https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

C. PROCEDURE

We train our Word2ec model on an Ubuntu 16.04 server with CPU (Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz) and 64G memory. At the beginning, we will use gensim's Word2ec algorithm[3] to train word of short text. We set 300 dimensions for every word. The window size of Word2ec is 5. When frequency of word is less than 5, we will ignore it (i.e. min_count = 5). For those words that are not in the training set but in the testset, we map each such word to a random vector. In this case, some specific words such as people name, address name, have specific meanings. Next, we need to normalize word's vectors that were trained in last step. The result is that all vectors lie in between 0 and 1. Short texts have different lengths, most are between 30 and 40 words. In order to unify the length of the input, we only cut the first 35 words of each short text in MSR Paraphrase Corpus dataset (short text of 35 words or less, we fill in 0) and cut the first 15 words of each short text in Quora QR dataset.

After this point, preprocessing is complete. Then, we start training LSTM encoder model. In encoder part, we use inception convolution module that employs various sizes of convolution kernels to extract features. And LSTM network with 256 dimensions is applied. The output of LSTM is the result of encoder and is a vector of 256 dimensions. Figure 5 is the encoder structure of autoencoder. Encoder part contains inception module and LSTM network. Inception module employ four-way convolution or maxpool, then concatenates result of every way. Finally, pass the result vector to LSTM. In decoder part, the output of encoder is the input of decoder. So we repeat 32 times of the output of encoder. The dimension of input of decoder is two dimensions of matrix. The first dimension is the repeated times (i.e. 32). The second dimension is the length of short text that we cut. Next, we use the reverse inception module structure showed in Figure 6. We train 10000 batches. There are 512 short texts in every batch.

V. RESULT ANALYSIS

In this section, we will show the analysis of our experimental results.

A. MSR RESULT ANALYSIS

In our experiment, we compare our short text similarity algorithms with other algorithm like TF-IDF, edit distance. Apart from training word vectors on the MSR Paraphrase Corpus dataset, we also train word vector on the Wikipedia corpus that includes all Wikipedia article of English version whose size is about 8G. Moreover, some institutions or individuals also publish their own word vectors. For example, Google's word vector are available[4] and their dimensions are 300. Facebook's fastText[5] pre-trained word vectors are

[3]https://radimrehurek.com/gensim/models/word2vec.html
[4]https://code.google.com/archive/p/word2vec/
[5]https://github.com/facebookresearch/fastText

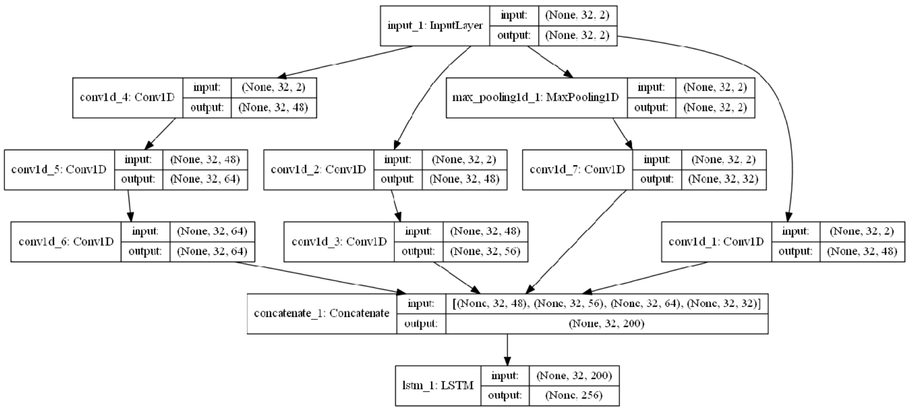

FIGURE 5. Encoder part of autoencoder.

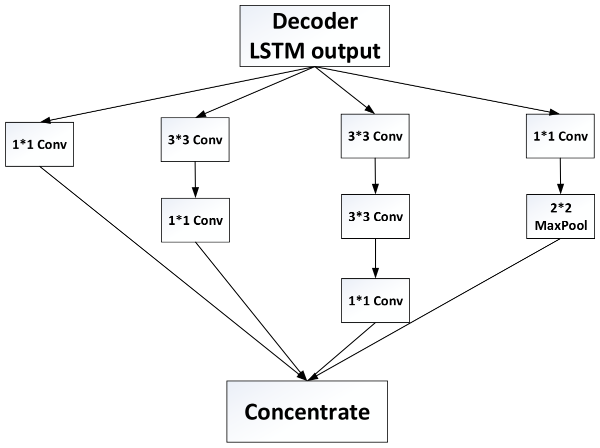

FIGURE 6. Reverse Inception Module.

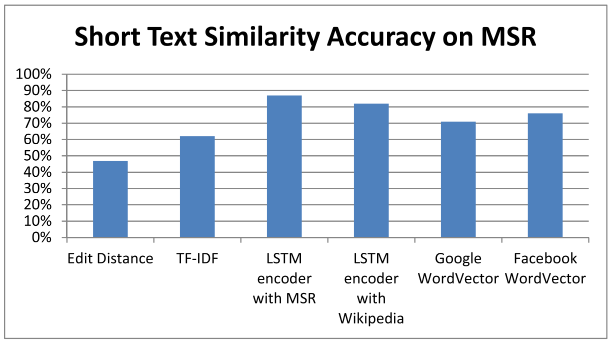

FIGURE 7. Short Text Similarity Accuracy on MSR.

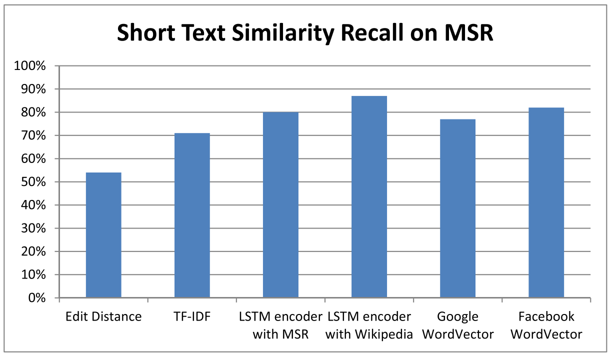

FIGURE 8. Short Text Similarity Recall on MSR.

also public and their dimensions are 400, word context window is 5, negative samples are 10. We use these four kinds of word vectors as our preprocess word vectors respectively. TF-IDF computes word frequency using MSR Paraphrase Corpus dataset. Edit distance algorithm doesn't need any corpus.

From the view of accuracy, we can know from Figure 7, the effect of edit distance is the worst. It only pays attention to text similarity literally. TF-IDF leverages word frequency as similarity standard. However, word frequencies can't stand for semantic similarity. For example, Beijing and Chinese capital have a big difference in literal. But they are similar in semantics. For our short text similarity algorithm with LSTM encoder, four different preprocessing word vectors lead to different performances. The effect of word vectors based on MSR Paraphrase Corpus dataset exceeds other three training word vectors. This shows that the corpus has a great influence on the effect of the word vector. We use MSR Paraphrase Corpus test dataset that is similar with train set. Wikipedia corpus is made up of Wikipedia articles. So words appearing in Wikipedia may not be common in test dataset. This maybe a reason that the result based MSR corpus is better than based on Wikipedia.

From the view of recall of Figure 8, the effect of our short text similarity with LSTM encoder using Wikipedia corpus is better than other algorithms. And the Facebook's WordVector is also better than MSR Paraphrase Corpus dataset. The effect of edit distance is also the worst. It may be related to the test dataset. Every short text in MSR test dataset represents the knowledge of different fields on the web.

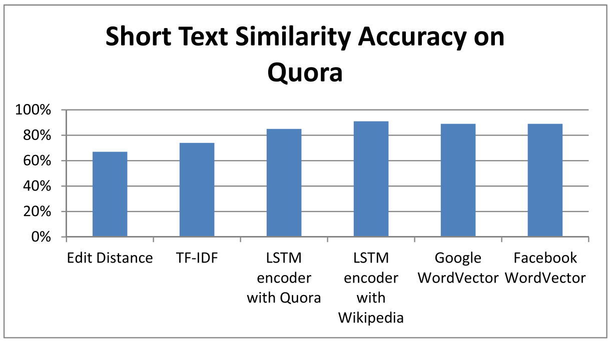

FIGURE 9. Short Text Similarity Accuracy on Quora.

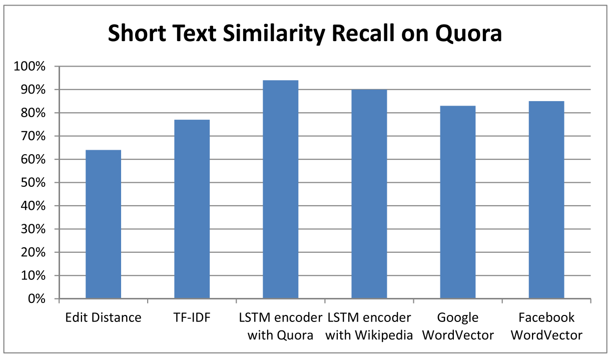

FIGURE 10. Short Text Similarity Recall on Quora.

B. QUORA QR RESULT ANALYSIS

In Quora QR dataset, we have the same experiment condition except we remove the common stop words in preprocessing stage. Figure 9 is the accuracy chart. From the view of accuracy, the effect of LSTM encoder with Wikipedia is the optimum. The result is different from MSR dataset. Maybe the reason is that the Quora QR dataset is shorter than the MSR dataset and the stop words are removed. And Short text consists of only a few phrases. At the same time, the results of Google Word vector and Facebook are better than LSTM encoder with Quora. It is still due to the length of short texts. The effect of edit distance and TF-IDF are still superior to MSR Paraphrase Corpus dataset.

From the view of recall, as shown in Figure 10, due to word vectors are trained by Quora QR dataset, the effect of LSTM encoder with Quora is the best. Besides, the results of other algorithms are all superior to MSR corpus. The reason is the same as the above: short text only comprise several word phrases.

VI. CONCLUSIONS

In this paper, we propose a novel algorithm computing unlabeled short text similarity based LSTM encoder. It can fully employ semantic information and sequence information of short texts. Our algorithm includes preprocessing, training and evaluating stage. Preprocessing stage uses Word2ec method to vector every word in short texts. Besides, we propose a new normalization algorithm to avoid gradient vanishing problems in the process of backward propagation. In training stage, our autoencoder model leverages inception module to extract more features from multiple dimensions and improves LSTM cell to learn the word sequence information of short texts and choose to forget the information accumulated before. The output of encoder is the vectors of short texts. Lastly, cosine distance is used for calculating short text similarity in evaluating stage. We conducted the experiment on two dataset: MSR Paraphrase Corpus dataset and Quora QR dataset. They represent different lengths of short text. Experiment shows that our short text similarity measure algorithm has a higher accuracy and recall on two different datasets.

For future work, we will investigate whether attention mechanism can optimize our model. Moreover, we can combine external knowledge into our model.

REFERENCES

[1] S. Chien and N. Immorlica, "Semantic similarity between search engine queries using temporal correlation," in *Proc. 14th Int. Conf. World Wide Web*, 2005, pp. 2–11.

[2] M. Mohler, R. Bunescu, and R. Mihalcea, "Learning to grade short answer questions using semantic similarity measures and dependency graph alignments," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1, 2011, pp. 752–762.

[3] M. Chen, X. Jin, and D. Shen, "Short text classification improved by learning multi-granularity topics," in *Proc. IJCAI*, 2011, pp. 1776–1781.

[4] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proc. 40th Annu. Meeting Assoc. Comput. Linguistics*, 2002, pp. 311–318.

[5] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas, "Short text classification in Twitter to improve information filtering," in *Proc. 33rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, 2010, pp. 841–842.

[6] W. H. Gomaa and A. A. Fahmy, "A survey of text similarity approaches," *Int. J. Comput. Appl.*, vol. 68, no. 13, Pp. 13–18, 2013.

[7] D. Buttler, "A short survey of document structure similarity algorithms," in *Proc. Int. Conf. Internet Comput.*, vol. 7, 2004, pp. 3–9.

[8] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of Jaccard coefficient for keywords similarity," in *Proc. Int. MultiConf. Eng. Comput. Sci.*, vol. 1, no. 6, 2013, pp. 1–5.

[9] V. Jijkoun and M. de Rijke, "Recognizing textual entailment using lexical similarity," in *Proc. PASCAL Challenges Workshop Recognising Textual Entailment*, 2005, pp. 73–76.

[10] T. Pedersen, S. Patwardhan, and J. Michelizzi, "WordNet: Similarity: Measuring the relatedness of concepts," in *Proc. Demonstration HLT-NAACL*, 2004, pp. 38–41.

[11] F. Šarić, G. Glavaš, M. Karan, J. Šnajder, and B. D. Bašić, "TakeLab: Systems for measuring semantic text similarity," in *Proc. 1st Joint Conf. Lexical Comput. Semantics, Conf. Shared Task, 6th Int. Workshop Semantic Eval.*, vol. 2, 2012, pp. 441–448.

[12] R. Mihalcea, C. Corley, and C. Strapparava, "Corpus-based and knowledge-based measures of text semantic similarity," in *Proc. AAAI*, vol. 6, 2006, pp. 775–780.

[13] R. Ferreira, R. D. Lins, F. Freitas, S. J. Simske, and M. Riss, "A new sentence similarity assessment measure based on a three-layer sentence representation," in *Proc. ACM Symp. Document Eng.*, 2014, pp. 25–34.

[14] S. Fernando and M. Stevenson, "A semantic similarity approach to paraphrase detection," in *Proc. 11th Annu. Res. Colloq. UK Special Interest Group Comput. Linguistics*, 2008, pp. 45–52.

[15] D. Bar, C. Biemann, I. Gurevych, and T. Zesch, "Ukp: Computing semantic textual similarity by combining multiple content similarity measures," in *Proc. 1st Joint Conf. Lexical Comput. Semantics, Conf. Shared Task, 6th Int. Workshop Semantic Eval.*, 2012, pp. 435–440.

[16] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[17] T. Hofmann, "Probabilistic latent semantic analysis," in *Proc. 15th Conf. Uncertainty Artif. Intell.*, 1999, pp. 289–296.

[18] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.
[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean. (2013). "Efficient estimation of word representations in vector space." [Online]. Available: https://arxiv.org/abs/1301.3781
[20] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. CVPR*, Apr. 2015, pp. 1–5.
[21] M. Strube and S. P. Ponzetto, "WikiRelate! Computing semantic relatedness using Wikipedia," in *Proc. AAAI*, vol. 6, 2006, pp. 1419–1424.
[22] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using Wikipedia-based explicit semantic analysis," in *Proc. IJcAI*, 2007, pp. 1606–1611.
[23] I. H. Witten and D. N. Milne, "An effective, low-cost measure of semantic relatedness obtained from Wikipedia links," in *Proc. AAIA*, 2008, pp. 1–6.
[24] T. Kenter and M. de Rijke, "Short text similarity with word embeddings," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, 2015, pp. 1411–1420.
[25] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, "From word embeddings to document distances," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 957–966.
[26] G. Huang, C. Guo, M. J. Kusner, Y. Sun, F. Sha, and K. Q. Weinberger, "Supervised word mover's distance," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4862–4870.
[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[28] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.
[29] B. Dolan, C. Quirk, and C. Brockett, "Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources," in *Proc. 20th Int. Conf. Comput. Linguistics*, 2004, p. 350.
[30] C. Quirk, C. Brockett, and W. Dolan, "Monolingual machine translation for paraphrase generation," in *Proc. EMNLP*, 2014, pp. 142–149.

LIN YAO is currently an Associate Professor with the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. His current research interests are natural language processing, the Internet of Things application, wireless indoor localization, and computer network security.

ZHENGYU PAN received the B.E. degree from the School of Microelectronics, Xihua University, China, in 2016. He is currently pursuing the M.S. degree with the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. His research interests include the Internet of Things, big data, and artificial intelligence.

HUANSHENG NING (SM'13) received the B.S. degree from Anhui University, in 1996, and the Ph.D. degree from Beihang University, in 2001. He is currently a Professor and the Vice Dean with the School of Computer and Communication Engineering, University of Science and Technology Beijing, China, and the Founder and Principal of the Cybermatics and Cyberspace International Science and Technology Cooperation Base. He has authored several books and over 70 papers in journals and at international conferences/workshops. His current research interests include the Internet of Things, cyber physical social systems, and electromagnetic sensing and computing. He has been the Associate Editor of IEEE Systems Journal and IEEE Internet of Things Journal, the Chairman and the Executive Chairman of the program committee at the IEEE International Internet of Things Conference, in 2012 and 2013, and the Co-Executive Chairman of the 2013 International Cyber Technology Conference and the 2015 Smart World Congress. His awards include the IEEE Computer Society Meritorious Service Award and the IEEE Computer Society Golden Core Member Award.