A Proposed Language Model Based on LSTM

Yumeng Zhang(✉), Xuanmin Lu, Bei Quan, and Yuanyuan Wei

Northwestern Polytechnical University, Xi'an, China
zhangyumeng@mail.nwpu.edu.cn, luxuanmin@nwpu.edu.cn

Abstract. In view of the shortcomings of language model N-gram, this paper presents a Long Short-Term Memory (LSTM)-based language model based on the advantage that LSTM can theoretically utilize any long sequence of information. It's an improved RNN model. Experimental results show that the perplexity of the LSTM language model in the PBT corpus is only one-half that of the N-gram language model.

Keywords: Language model · N-gram · RNN · LSTM · Perplexity

1 Introduction

The language model is widely used in natural language processing, machine translation and speech recognition, and it can directly affect the performance of the corresponding system. Language models are usually divided into two categories: one is a statistical language model based on large-scale corpus. In actual use, the model can only reflect the adjacent constraint relationship of the language due to the limitation of space and time of the system, and cannot solve the recursion problem when the processing language distance is long. The other is a rule-based language model. The model is suitable for the processing of closed corpus, which can effectively reflect the long-distance constraint relationship and recursion phenomenon in language, but the stability of this model is poor, and it is open in processing. Sexual corpus does not have a good consistency in knowledge expression. The commonly used language model is the statistical model N-gram, but it has two disadvantages. First, the semantics of word vectors are not satisfactory in the continuous space. When the system adjusts the parameters of a word or phrase, the word or phrase with similar meanings will also be changed accordingly. Second, the appearing probability of a word depends only on the first few words, and it does not perform well in a particularly long context. In view of the above drawbacks, based on the theory that LSTM can theoretically take advantage of arbitrary long sequence information, a LSTM-based language model is proposed. The model converts each word in the sentence into a word vector and trains it through the LSTM network. Then, it obtains the probability distribution of the word. The experimental results show that the perplexity of the LSTM-based language model in the PBT corpus is only half of that of the N-gram language model.

B. Li et al. (Eds.): IoTaaS 2018, LNICST 271, pp. 348–355, 2019.
https://doi.org/10.1007/978-3-030-14657-3_35

2 N-Gram Language Model

N-gram refers to a sequence of N items in a given piece of text or speech. Items can be syllables, letters, words, or base pairs. The N-gram model is a statistical language model. Usually N-grams are taken from texts or corpus [1]. When N = 1, it is called unigram. When N = 2, it is called bigram. When N = 3, it is called trigram, and so on.

For a sequence of m words (sentences), the probability $P(\omega_1, \ldots, \omega_m)$ can be calculated according to the chain rule:

$$P(\omega_1, \ldots, \omega_m) = P(\omega_1)P(\omega_2|\omega_1)P(\omega_3|\omega_1, \omega_2)\ldots P(\omega_m|\omega_1, \ldots, \omega_{m-1}) \tag{1}$$

In order to reduce the computational difficulty, Markov approximation is used for the above formula, i.e., the probability that the current word appears is only related to the appears the current word. In this way, it is unnecessary to trace back to the initial word of the sentence at the time of calculation, thereby it can greatly reduce the length of the above formula (1):

$$P(\omega_i|\omega_1, \omega_2, \ldots, \omega_{i-1}) = P(\omega_i|\omega_{i-n+1}, \ldots, \omega_{i-1}) \tag{2}$$

When N = 1, a unigram model is:

$$P(\omega_1,\ \omega_2,\ \ldots,\ \omega_m) = \prod_{i=1}^{m} P(\omega_i) \tag{3}$$

When N = 2, a bigram model is:

$$P(w_1, w_2, \ldots, w_m) = \prod_{i=1}^{m} P(w_i|w_{i-1}) \tag{4}$$

When N = 3, a trigram model is:

$$P(w_1, w_2, \ldots, w_m) = \prod_{i=1}^{m} P(w_i|w_{i-2}w_{i-1}) \tag{5}$$

A set of parameters can be calculated by the maximum likelihood method to obtain the maximum value of the training sample probability.

For the unigram model, where $C(w_1, \ldots, w_n)$ represents the frequency of occurrence of N-grams $(w_1, \ldots, w_n)$ in the training corpus data, M represents all the words in the corpus:

$$P(w_i) = \frac{C(w_i)}{M} \tag{6}$$

For the bigram model:

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i)}{C(w_{i-1})} \tag{7}$$

For the N-gram model:

$$\hat{y}_t = softmax(W^{(s)}h_t) \tag{8}$$

In actual use, because the training corpus is very limited, when the test corpus from the same source is used to verify the training results, some N-grams do not appear in the training corpus. This means that the probabilities calculated by the above formula may be zero, which leads to data sparseness. For languages, the existence of data sparseness makes it not satisfy the law of large numbers, and the calculated probability is distorted. The solution called smoothing or discounting is needed.

The N-gram model predicts the follow-up words based on discrete unit words (for example, n prefix words) that do not have a genetic property in the text. In a continuous context, using only n prefix words as the window range may not be sufficient to accurately describe the context information, and thus does not have the semantic advantage that the word vectors in the continuous space satisfy. Similar-meaning words have similar word vectors. When the model adjusts the parameters of a particular word or phrase, similar words or phrases will change accordingly. At the same time, the size of n prefix words' memory information, used for current word prediction during the operation of the system, will increase exponentially, resulting in an inestimable probability. Therefore, in the face of a large continuous space, it is very difficult to capture context information between vocabularies without processing after the memory information is extracted.

3 LSTM-Based Language Model

Long Short Term Memory Networks, also known as LSTM [2], is a recursive time-recurrent neural network that can effectively handle long-term dependency problems in time series. LSTM network structure was proposed by Hochreater and Schmidhuber in 1997 and further improved and promoted by Alex Graves. In the fields of image analysis, machine translation, handwriting recognition, speech recognition, etc., LSTM has a very outstanding performance and has gotten a flourishing development [3].

LSTM solves long-term dependency problems by transforming RNNs. The long-term memory of information is a self-distribution for LSTM. It does not need to be achieved through deliberate learning. The advantage of LSTM is that it can take into account all the predicative vocabulary in corpus when training the language model.

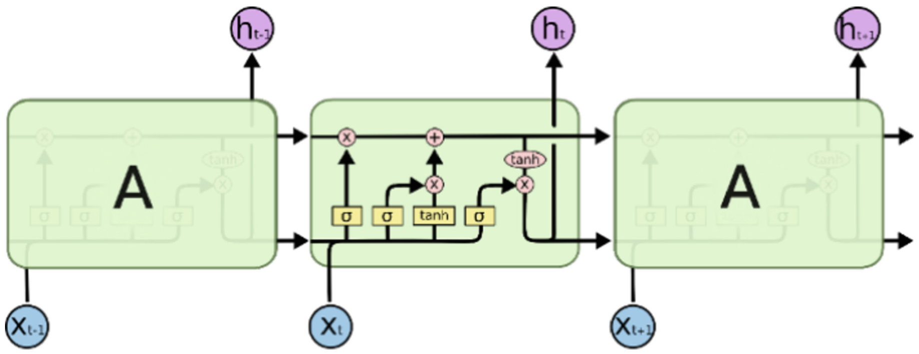

Fig. 1. LSTM structure

Figure 1 shows the structure of the LSTM network, where repeated blocks represent hidden layers in each iteration. Each hidden layer contains several neurons. Each neuron first performs a linear matrix calculation on the input vector and then outputs the corresponding result [4] after the nonlinear function of the activation function. In each iteration, the output of the last iteration interacts with the next word vector of the text to determine whether the information is saved or discarded and the current state is updated. x_t is the input of the hidden layer in this round of iterations. According to the current state information, the predicted output value of the hidden layer y is obtained and the output feature vector h_t is provided for the next hidden layer. Whenever there is a new word vector input in the network, the word vector is combined with the output of the hidden layer at the previous moment to calculate the output of the hidden layer at the next moment, and the hidden layer is used cyclically to maintain the latest state.

Each hidden layer is finally connected with a layer of traditional feedforward network as the output layer. Each node y_i in the output layer corresponds to the unnormalized logarithmic probability of the word i at the next moment, and then the output value y is normalized by the softmax function:

$$\hat{y}_t = softmax(W^{(s)}h_t) \tag{9}$$

$\hat{y}_t$ is the probability distribution calculated by the hidden layer based on all the vocabularies in each round of iterations, that is, when the weights of the entire preorder vocabulary of the document and the observed word vector x(t) are all determined, the subsequent words can be predicted by the model. The variable |V| of $W^{(s)} \in \mathbb{R}^{|v| \times D_h}$ and $\hat{y} \in \mathbb{R}^{|v|}$ represents the dictionary size of the entire corpus.

In a recurrent neural network, cross-entropy is usually used as a loss function [5]. Equation (10) represents the superposition of cross entropy over the entire corpus when the t-th iteration is performed:

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \times \log(\hat{y}_{t,j}) \tag{10}$$

For a corpus of size T, the cross entropy is calculated as follows:

$$J = -\frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j} \times \log(\hat{y}_{t,j}) \tag{11}$$

4 Analysis of Experimental Results

4.1 N-Gram Language Model Experiment

The dataset used in the experiment is the Penn Treebank (PTB) corpus. The language model is built by SRILM, and the used statistical data of the corpus file is: file testfile. txt: 3761 sentences, 78669 words and 4794 OOVs. The 1-gram (unigram), 2-gram (bigram) and 3-gram (trigram) models are constructed respectively and the corresponding complexity is calculated. The results are shown in above Table 1, where logprob represents logP(T), ppl represents perplexity of all words, and ppl1 represents perplexity in addition to </s > :

$$\begin{aligned} \mathrm{ppl} &= 10 \wedge (-\mathrm{logP(T)}/(\mathrm{sen} + \mathrm{word})) \\ \mathrm{ppl1} &= 10 \wedge (-\mathrm{logP(T)}/\mathrm{Word}) \end{aligned} \tag{12}$$

where Sen and Word represent sentences and words respectively.

Table 1. Three N-gram language models and their perplexity

	LogProb	PPL	PPL1
1-gram	logprob = − 233580.5	ppl = 1020.147	ppl1 = 1451.559
2-gram	logprob = − 185313.2	ppl = 243.7525	ppl1 = 322.4552
3-gram	logprob = − 173834.8	ppl = 173.4205	ppl1 = 225.4726

4.2 LSTM Language Model Experiment

The training model parameters are as follows: the initial value of the relevant parameters is randomly and uniformly distributed, the range is [-init_scale, +init_scale], and init_scale is 0.1; the learning rate is set to 1. When the number of iterations of text training exceeds the maximum value max epoch, it gradually decreases, and the learning rate decay also gradually decreases when the maximum number of iterations is exceeded. The number of LSTM layers is 2 layers, the number of nodes in each hidden layer is 200, the sequence length of a single data is 20, and the dictionary size is 10 K words. At the same time, a batch strategy is adopted, the scale of each batch of data is set to 20, and dropout method is used in order to prevent overfitting.

Firstly, a multi-layer LSTM unit is set up to preprocess the training corpora, and the dictionary id in the training text is converted into a vector format. Text data is sequentially input to the LSTM unit according to the sequence, and the stored state of the previous time period of the LSTM and the current input collectively calculate the output and state of the current unit (the probability of occurrence of the next word). Gradient descent is used to update weights, and gradients can be appropriately pruned (scaled) in order to control gradient explosions. The training corpus is divided into multiple batches, and records such as the cost and state of the network output are recorded, and the results are output according to the number of iterations. Each time a 10% iteration is completed, the current epoch progress, perplexity (natural constant index of the cross-entropy loss function), and training speed (words per second) are displayed. Return the result of the perplexity function at the end of the training.

The operating system is Windows 10, 1709 version, the CPU is Intel i5-6300hq, and the GPU is trained in the Nvidia GTX 960 m environment. Figure 2 shows the decrease curve of Learning Rate. In Fig. 3, the Train Perplexity is the perplexity of the training set, and the Valid Perplexity is the perplexity of the verification set. It can be seen that with the decrease of the Learning Rate, the decrease of perplexity finally tends to be stable. After 13 epochs, the Train Perplexity value was 40.692, the Valid Perplexity value was 119.441, and the Test Set's Test Perplexity value was 115.702.

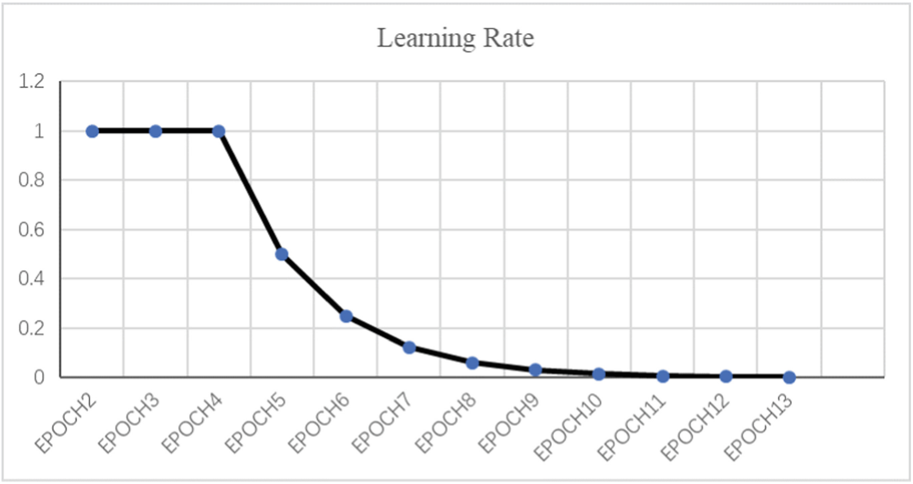

Fig. 2. Decrease curve of learning rate

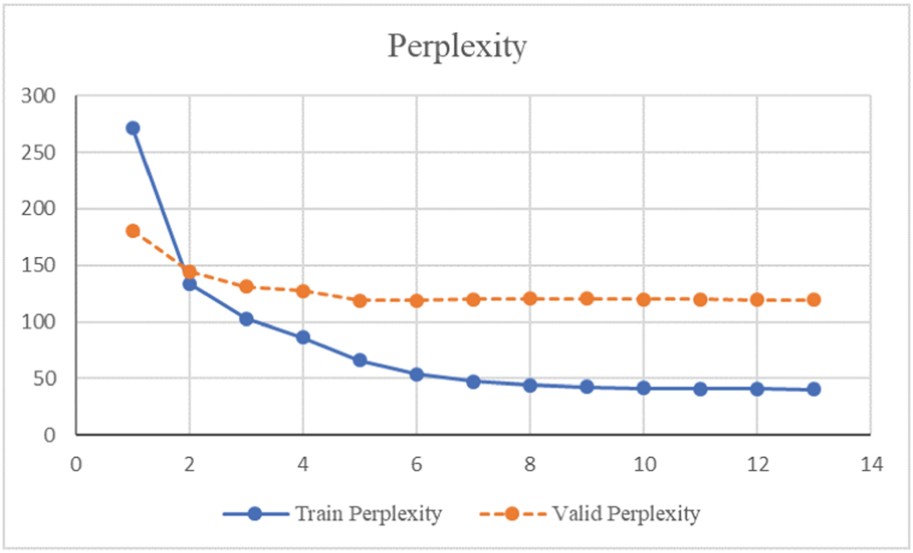

Fig. 3. Train perplexity and valid perplexity trends

4.3 Results and Analysis

The comparison of perplexity can be obtained between the two models from the previous experimental results (Table 2):

Table 2. Perplexity of LSTM model and N-gram model

Model name	Perplexity	
Unigram	ppl = 1020.147	ppl1 = 1451.559
Bigram	ppl = 243.7525	ppl1 = 322.4552
Trigram	ppl = 173.4205	ppl1 = 225.4726
LSTM	115.702	

From the above table, in the N-gram model, when the model is unigram, that is, n = 1, the perplexity degree is the highest; when the model is the bigram, that is, n = 2, the perplexity degree is the second, and when the model is trigram, that is, n = 3, the perplexity degree is the lowest. This is because, for an n-gram model, the larger the value of n, the more constraint information for the result, and the more accurate the model, that is, the trigram has more constraint information on the occurrence of subsequent words, and the reliability is also higher. However, the increase of n causes the model to become more complex, which increases the amount of computations when the model is built. The perplexity of the language model built by LSTM is only 115.702, which is equivalent to half of the trigram. Therefore, this experiment proves that the performance of language model based on LSTM is obviously better than that of N-gram model.

5 Conclusion

At present, the traditional speech recognition technology has not been able to adapt well to the needs of the big data era, and no deeper information can be excavated from the massive data. Deep learning has excellent modeling ability of big data and excellent fitting ability. Using deep learning to study speech recognition has important theoretical significance and practical value. This paper discusses the commonly used traditional statistical language model N-gram, and proposes a language model, based on the improved RNN model—LSTM model, to solve the problem that the N-gram model does not meet the semantic deficiency of word vectors in continuous space, and the LSTM language model has obvious advantages in performance compared with the N-gram.

References

Lin, C.Y., Hovy, E.: Automatic evaluation of summaries using N-gram co-occurrence statistics. In: Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology. Association for Computational Linguistics, pp. 71–78 (2003)

Xiong, W., Droppo, J., Huang, X., et al.: Achieving human parity in conversational speech recognition. IEEE/ACM Trans. Audio Speech Lang. Process. PP(99) (2016)

Li, J., Zhang, H., Cai, X.Y., et al.: Towards end-to-end speech recognition for Chinese Mandarin using long short-term memory recurrent neural networks (2015)

Tai, K.S., Socher, R., Manning, C.D.: Improved semantic representations from tree-structured long short-term memory networks. Comput. Sci. **5**(1), 36 (2015)

Mikolov, T.A.: Statistical language models based on neural networks (2012)