Binarized LSTM Language Model

Xuan Liu*, Di Cao*, Kai Yu
Key Laboratory of Shanghai Education Commission for
Intelligent Interaction and Cognitive Engineering,
SpeechLab, Department of Computer Science and Engineering,
Brain Science and Technology Research Center,
Shanghai Jiao Tong University, Shanghai, China
{liuxuan0526, caodi0207, ky219.cam}@gmail.com

Abstract

The long short-term memory (LSTM) language model (LM) has been widely investigated for automatic speech recognition (ASR) and natural language processing (NLP). Although excellent performance is obtained for large vocabulary tasks, tremendous memory consumption prohibits the use of LSTM LMs in low-resource devices. The memory consumption mainly comes from the word embedding layer. In this paper, a novel binarized LSTM LM is proposed to address the problem. Words are encoded into binary vectors and other LSTM parameters are further binarized to achieve high memory compression. This is the first effort to investigate binary LSTMs for large vocabulary language modeling. Experiments on both English and Chinese LM and ASR tasks showed that binarization can achieve a compression ratio of 11.3 without any loss of LM and ASR performance and a compression ratio of 31.6 with acceptable minor performance degradation.

1 Introduction

Language models (LMs) play an important role in natural language processing (NLP) tasks. N-gram language models used to be the most popular language models. Considering the previous N-1 words, N-gram language models predict the next word. However, this leads to the loss of long-term dependencies. The sample space size increases exponentially as N grows, which induces data sparseness (Cao and Yu, 2017).

Neural network (NN) based models were first introduced into language modeling in 2003 (Bengio et al., 2003). Given contexts with a fixed size, the model can calculate the probability distribution of the next word. However, the problem of long-term dependencies still remained, because the context window is fixed. Currently, recurrent neural network (RNN) based models are widely used on natural language processing (NLP) tasks for excellent performance (Mikolov et al., 2010). Recurrent structures in neural networks can solve the problem of long-term dependencies to a great extent. Some gate based structures, such as long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) and gated recurrent unit (GRU) (Chung et al., 2014) improve the recurrent structures and achieve state-of-the-art performance on most NLP tasks.

However, neural network models occupy tremendous memory space so that it is almost impossible to put the models into low-resource devices. In practice, the vocabulary is usually very large. So the memory consumption mainly comes from the embedding layers. And, the word embedding parameters are floating point values, which adds to the memory consumption.

The first contribution in this paper is that a novel language model, the binarized embedding language model (BELM) is proposed to reduce the memory consumption. Words are represented in the form of binarized vectors. Thus, the consumption of memory space is significantly reduced. Another contribution in the paper is that we binarize the LSTM language model combined with the binarized embeddings to further compress the parameter space. All the parameters in the LSTM language model are binarized.

Experiments are conducted in language modeling and automatic speech recognition (ASR) rescoring tasks. Our model performs well without any loss of performance at a compression ratio of 11.3 and still has acceptable results with only a minor loss of performance even at a compression ratio of 31.6. Investigations are also made to evaluate whether the binarized embeddings lose information. Experiments are conducted on word

*Both authors contributed equally to this work.

similarity tasks. The results show the binarized embeddings generated by our models still perform well on the two datasets.

The rest of the paper is organized as follows, section 2 is the related work. Section 3 explains the proposed language model and section 4 shows the experimental setup and results. Finally, conclusions will be given in section 5 and we describe future work in section 6.

2 Related Work

Nowadays, with the development of deep learning, neural networks have yielded good results in many areas. However, neural networks may require tremendous memory space, making it difficult to run such models on low-resource devices. Thus, it is necessary to compress neural networks.

In recent years, many methods of compressing neural networks have been proposed. Pruning (Han et al., 2015) reduces the number of parameters of the neural network by removing all connections with the weights below a threshold. Quantization (Han et al., 2015) clusters weights to several clusters. A few bits are used to represent the neurons and to index a few float values.

Binarization is also a method to compress neural networks. BNNs(Courbariaux et al., 2016) are binarized deep neural networks. The weights and activations are constrained to 1 or -1. BNNs can drastically reduce memory size and replace most arithmetic operations with bit-wise operations. Different from pruning and quantization, binarization does not necessarily require pre-training and can achieve a great compression ratio. Many binarization methods have been proposed (Courbariaux et al., 2015, 2016; Rastegari et al., 2016; Xiang et al., 2017). However, only a few (Hou et al., 2016; Edel and Köppe, 2016) are related to recurrent neural network. (Hou et al., 2016) implements a character level binarized language model with a vocabulary size of 87. However, they did not do a comprehensive study on binarized large vocabulary LSTM language models.

3 Binarized Language Model

3.1 LSTM Language Model

The RNN language model is proposed to deal with sequential data. Due to the vanishing and exploding gradients problem, it is difficult for a RNN language model to learn long-term dependencies. The LSTM, which strengthens the recurrent neural model with a gating mechanism, tackles this problem and is widely used in natural language processing tasks.

The goal of a language model is to compute the probability of a sentence $(x_1, \ldots, x_N)$. A typical method is to decompose this probability word by word.

$$P(x_1, ..., x_N) = \prod_{t=1}^{N} P(x_t|x_1, ..., x_{t-1}) \tag{1}$$

(Hochreiter and Schmidhuber, 1997) proposed a Long Short-Term Memory Network, which can be used for sequence processing tasks. Consider an one-layer LSTM network, where N is the length of the sentence, and x_t is the input at the t-th moment. y_t is the output at the t-th moment, which is equal to x_{t+1} in a language model. Denote $\boldsymbol{h_t}$ and $\boldsymbol{c_t}$ as the hidden vector and the cell vector at the t-th moment, which is used for representing the history of $(x_1, ..., x_{t-1})$. $\boldsymbol{h_0}$ and $\boldsymbol{c_0}$ are initialized with zero. Given x_t, $\boldsymbol{h_{t-1}}$ and $\boldsymbol{c_{t-1}}$, the model calculates the probability of outputting y_t.

The first step of an LSTM language model is to extract the representation $\boldsymbol{e_t}$ of the input x_t from the embeddings $\mathbf{W_e}$. Since x_t is a one-hot vector, this operation can be implemented by indexing rather than multiplication.

$$\boldsymbol{e_t} = \mathbf{W_e}\boldsymbol{x_t} \tag{2}$$

After that, $\boldsymbol{e_t}$, along with $\boldsymbol{h_{t-1}}$ and $\boldsymbol{c_{t-1}}$ are fed into the LSTM cell. The hidden vector $\boldsymbol{h_t}$ and the cell vector $\boldsymbol{c_t}$ can be computed according to:

$$\begin{aligned}
\boldsymbol{f_t} &= \text{sigmoid}\left(\mathbf{W_f}\left\{\boldsymbol{h_{t-1}}, \boldsymbol{e_t}\right\} + \boldsymbol{b_f}\right) \\
\boldsymbol{i_t} &= \text{sigmoid}\left(\mathbf{W_i}\left\{\boldsymbol{h_{t-1}}, \boldsymbol{e_t}\right\} + \boldsymbol{b_i}\right) \\
\boldsymbol{o_t} &= \text{sigmoid}\left(\mathbf{W_o}\left\{\boldsymbol{h_{t-1}}, \boldsymbol{e_t}\right\} + \boldsymbol{b_o}\right) \\
\boldsymbol{\hat{c}_t} &= \tanh\left(\mathbf{W_{\hat{c}}}\left\{\boldsymbol{h_{t-1}}, \boldsymbol{e_t}\right\} + \boldsymbol{b_{\hat{c}}}\right) \\
\boldsymbol{c_t} &= \boldsymbol{f_t} \cdot \boldsymbol{c_{t-1}} + \boldsymbol{i_t} \cdot \boldsymbol{\hat{c}_t} \\
\boldsymbol{h_t} &= \boldsymbol{o_t} \cdot \tanh\left(\boldsymbol{c_t}\right)
\end{aligned} \tag{3}$$

The word probability distribution at the t-th moment can be calculated by:

$$P(y_t|x_1, ..., x_t) = \boldsymbol{p_t} = \text{softmax}(\mathbf{W_y}\boldsymbol{h_t}) \tag{4}$$

The probability of taking y_t as the output at the t-th moment is:

$$p_{y_t} = \boldsymbol{p_t} \times \boldsymbol{y_t} \tag{5}$$

3.2 Binarized Embedding Language Model

The binarized embedding language model (BELM) is a novel LSTM language model with binarized input embeddings and output embeddings. For a one-layer LSTM language model with a vocabulary size of V, embedding and hidden layer size of H. The size in bytes of the input embeddings, the output embeddings, and the LSTM cells are $4VH$, $4VH$ and $32H^2 + 16H$. When V is much larger than H, which is often the case for language models, the parameters of the input embeddings and the output embeddings occupy most of the space. If the embeddings of the input layer and the output layer are binarized, the input layer and the output layer will only take $1/32$ of the original memory consumption, which can greatly reduce the memory consumption of running neural language model.

It is important to find good binary embeddings. Directly binarizing well-trained word embeddings cannot yield good binarized representations. Instead, we train good binary embeddings from scratch. The training approach is similar to the methods proposed in (Courbariaux et al., 2016; Rastegari et al., 2016). At run-time, the input embedding and the output embedding are binarized matrices. However, at train-time, float versions of the embeddings, which are used for calculating the binarized version of embeddings, are still maintained. In the propagation step, a deterministic function sign is used to binarize the float versions of the embeddings. In the back-propagation step, the float versions of the embeddings are updated according to the gradient of the binarized embedding.

$$w^b = \mathrm{sign}\left(w\right) = \begin{cases} +1 & if\ w > 0, \\ -1 & otherwise. \end{cases} \tag{6}$$

The derivative of the sign function is zero almost everywhere, and it is impossible to back-propagate through this function. As introduced in (Hubara et al., 2016), a straight-through estimator is used to get the gradient. Assume the gradient of the binarized weight $\frac{\partial C}{\partial \mathbf{W^b}}$ has been obtained, the gradient of the float version of the weight is:

$$\frac{\partial C}{\partial \mathbf{W}} = \frac{\partial C}{\partial \mathbf{W^b}} \tag{7}$$

A typical weight initialization method initializes each neuron's weights randomly from the Gaussian distribution $N(0, \sqrt{1/H})$. This initialization approach can maximize the gradients and mitigate the vanishing gradients problem. From this perspective, 1 or -1 is too large. So, in practice, we binarize the embeddings to a smaller scale. Although the weight is binarized to a floating point number, the matrix can also be saved one bit per neuron, as long as the fixed float value is memorized separately.

$$\mathrm{binarize}\left(w\right) = \begin{cases} +\sqrt{1/H} & if\ w > 0, \\ -\sqrt{1/H} & otherwise. \end{cases} \tag{8}$$

Since directly binarizing the input embeddings $\mathbf{W_e}$ and the output embeddings $\mathbf{W_y}$ will limit the scale of the embeddings, additional linear layers (without activation) are added behind the input embedding layer and in front of the output embedding layer to enhance the model. Denote $\mathbf{W_e^b}$ and $\mathbf{W_y^b}$ as the binarized weights corresponding to $\mathbf{W_e}$ and $\mathbf{W_y}$. Denote $\mathbf{W_{T_e}}$ and $\boldsymbol{b_{T_e}}$, $\mathbf{W_{T_y}}$ and $\boldsymbol{b_{T_y}}$ as the weights and the biases of the first and the second linear layer. The input of the LSTM $\boldsymbol{e_t}$ and the word probability $\boldsymbol{p_t}$ of the binarized embedding language model are calculated according to:

$$\begin{aligned} \boldsymbol{e_t} &= \mathbf{W_{T_e}}\left(\mathbf{W_e^b}\boldsymbol{x_t}\right) + \boldsymbol{b_{T_e}} \\ \boldsymbol{p_t} &= \mathrm{softmax}\left(\mathbf{W_y^b}\left(\mathbf{W_{T_y}}\boldsymbol{h_t} + \boldsymbol{b_{T_y}}\right)\right) \end{aligned} \tag{9}$$

The additional linear layer before the output embedding layer is very important for the binarized embedding language model, especially for low dimensional models. Removing this layer will result in an obvious decrease in performance.

3.3 Binarized LSTM Language Model

Subsection 3.2 explains how to binarize the embedding layer, but the LSTM network can also be binarized. In a binarized LSTM language model, all the matrices in the parameters are binarized, which can save much more memory space. Implementing the binarized linear layer is important for designing a binarized LSTM language model (BLLM). In a binarized linear layer, there are three parameters, $\mathbf{W}$, γ and $\boldsymbol{b}$. $\mathbf{W}$ is a matrix, γ and $\boldsymbol{b}$ are vectors. The matrix $\mathbf{W}$, which takes up most of the space in a linear layer, is binarized. γ and $\boldsymbol{b}$ remain floating point values. b is the bias of the linear layer, and γ is introduced to fix the scale problem of the binary matrix.

The forward- and back-propagation algorithms are shown in Algorithm 1 and Algorithm 2. The structure of this linear layer is very similar to the structure of batch normalization (Ioffe and Szegedy, 2015), except the output of each dimension over the mini-batches is not normalized. Batch normalization is hard to apply to a recurrent neural network, due to the dependency over entire sequences. However, the structure of the batch normalization is quite useful. Since binarizing W would fix the scale of the weight, additional freedom is needed to overcome this issue. The shift operation can rescale the output to a reasonable range.

Algorithm 1 The propagation of linear layer

```
Input: input $\boldsymbol{x}$, weights $\mathbf{W}$, $\gamma$ and $\boldsymbol{b}$
Output: output $\boldsymbol{y}$
1: $\mathbf{W^b} = \text{binarize}\left(\mathbf{W}\right)$
2: $\boldsymbol{s} = \mathbf{W^b}\boldsymbol{x}$
3: $\boldsymbol{y} = \boldsymbol{s} \cdot \exp\left(\gamma\right) + \boldsymbol{b}$
```

Algorithm 2 The back-propagation of linear layer

```
Input: input $\boldsymbol{x}$, weights $\mathbf{W}$, $\gamma$ and $\boldsymbol{b}$, binarized weight $\mathbf{W^b}$, temporary value $\boldsymbol{s}$ (calculated in the propagation period), the gradient of the output $\frac{\partial C}{\partial \boldsymbol{y}}$, learning rate $\eta$, binary weight range $\alpha$
Output: the gradient of the input $\frac{\partial C}{\partial \boldsymbol{x}}$, the gradient of the weight $\frac{\partial C}{\partial \mathbf{W}}$, $\frac{\partial C}{\partial \gamma}$, $\frac{\partial C}{\partial \boldsymbol{b}}$, update the weights
1: $\frac{\partial C}{\partial \boldsymbol{b}} = \frac{\partial C}{\partial \boldsymbol{y}}$
2: $\frac{\partial C}{\partial \gamma} = \frac{\partial C}{\partial \boldsymbol{y}} \cdot \boldsymbol{s} \cdot \exp\left(\gamma\right)$
3: $\frac{\partial C}{\partial \boldsymbol{s}} = \frac{\partial C}{\partial \boldsymbol{y}} \cdot \exp\left(\gamma\right)$, $\frac{\partial C}{\partial \mathbf{W^b}} = \frac{\partial C}{\partial \boldsymbol{s}}\boldsymbol{x}$, $\frac{\partial C}{\partial \mathbf{W}} = \frac{\partial C}{\partial \mathbf{W^b}}$
4: $\frac{\partial C}{\partial \boldsymbol{x}} = \frac{\partial C}{\partial \boldsymbol{s}}\mathbf{W^b}$
5: update $\mathbf{W}$, $\gamma$, $\boldsymbol{b}$ according to $\frac{\partial C}{\partial \mathbf{W}}$, $\frac{\partial C}{\partial \gamma}$, $\frac{\partial C}{\partial \boldsymbol{b}}$ with learning rate $\eta$.
6: clamp$(\mathbf{W}, -\alpha, \alpha)$
7: return $\frac{\partial C}{\partial x}$
```

The structure of the input embeddings and the output embeddings of the binarized LSTM language model is similar to the binarized embedding language model. The embeddings are binarized and additional linear layers are added after the input embedding layer and in front of the output embedding layer. However, the additional linear layers are also binarized according to Algorithm 1 and Algorithm 2.

3.4 Memory Reduction

Denote the size of the vocabulary as V, and the size of the embedding and hidden layer as H. The memory consumptions of a one-layer LSTM language model, BELM and BLLM are listed in Table 1.

Model	Memory (bytes)
LSTM	$8VH + 32H^2 + 16H$
BELM	$0.25VH + 40H^2 + 24H$
BLLM	$0.25VH + 1.25H^2 + 48H$

Table 1: Memory Requirements

For a language model, the vocabulary size is usually much larger than the hidden layer size. The main memory consumption comes from the embedding layers, which require $8VH$ bytes for an LSTM language model. Binarized embeddings can reduce this term to $0.25VH$ bytes. Further compression of the LSTM can drop the coefficient of H^2 from 32 to 1.25.

4 Experiments

4.1 Experimental Setup

Our model is evaluated on the English Penn TreeBank (PTB) (Marcus et al., 1993), Chinese short message (SMS) and SWB-Fisher (SWB). The Penn TreeBank corpus is a famous English dataset, with a vocabulary size of 10K and 4.8% words out of vocabulary (OOV), which is widely used to evaluate the performance of a language model. The training set contains approximately 42K sentences with 887K words. The Chinese SMS corpus is collected from short messages. The corpus has a vocabulary size of about 40K. The training set contains 380K sentences with 1931K words. The SWB-Fisher corpus is an English corpus containing approximately 2.5M sentences with 24.9M words. The corpus has a vocabulary size of about 30K. hub5e is the dataset for the SWB ASR task.

We also evaluate the word embeddings produced by our models on two word similarity datasets. The models are trained on the Text8 corpus to extract the word embeddings. The Text8 corpus is published by Google and collected from Wikipedia. Text8 contains about 17M words with a vocabulary size of about 47k. The WordSimilarity-353(WS-353) Test Collection contains two sets of English word pairs along with human-assigned similarity judgments. The collection can be used to train and test computer algorithms implementing semantic similarity measures. A combined set (combined) is provided that contains a list of all 353 words, along with their mean similarity scores. (Finkelstein et al., 2001) The MEN dataset consists of 3,000 word pairs, randomly selected from words that occur at least 700 times in the freely available ukWaC and Wackypedia corpora combined (size: 1.9B and 820M tokens, respectively) and at least 50 times (as tags) in the open-sourced subset of the ESP game dataset. In order to avoid picking unrelated pairs only, the pairs are sampled so that they represent a balanced range of relatedness levels according to a text-based semantic score (Bruni et al., 2014).

First, we conduct experiments on the PTB, SWB and Text8 corpora respectively to evaluate language modeling performance. We use perplexity (PPL) as the metric to evaluate models of different sizes. Then, the models are evaluated on ASR rescoring tasks. Rescoring the 100-best sentences generated by the weighted finite state transducer (WFST), the model is evaluated by word error rate (WER). Finally, we conduct experiments on word similarity tasks to evaluate whether the word embeddings produced by our models lose any information.

4.2 Experiments in Language Modeling

For traditional RNN based language models, the memory consumption mainly comes from the embedding layers (both input and output layers). However, when the hidden layer size grows, the memory consumption of the RNN module also becomes larger. So the total memory usage relates to both the vocabulary size and hidden layer size, as mentioned in section 3.4.

Experiments are conducted in language modeling to evaluate the model on the PTB, SWB, and SMS corpora respectively. In language modeling tasks, we regularize the networks using dropout(Zaremba et al., 2014). We use stochastic gradient descent (SGD) for optimization. The batch size is set to 64. For the PTB corpus, the dropout rate is tuned for different training settings. For the SWB corpus, we do not use dropout technique. For the SMS corpus, the dropout rate is set to 0.25. We train models of different sizes on the three corpora and record the memory usage of the trained models. The initial learning rate is set to 1.0 for all settings. Since PTB is a relatively small dataset and the convergence rates of the BELM and the BLLM are slower than LSTM language model, we reduce the learning rate by half every three epochs if the perplexity on the validation set is not reduced. For the other experiments, the learning rate is always reduced by half every epoch if the perplexity on the validation set is not reduced. As introduced in section 3, the bias of the output embedding layer is omitted. Adding bias term in the output embedding layer leads to small performance degradation in the BELM and the BLLM model, although it leads to a small improvement in the LSTM model. This phenomenon may be related to optimization problems.

	Hidden size	**LSTM**	**BELM**	**BLLM**
Memory	500	48.0M	11.3M	1.6M
PPL		91.8	**88.0**	95.2
Memory	1000	112.0M	42.5M	3.8M
PPL		89.4	**85.7**	94.9

Table 2: Performances on the English PTB corpus

	Hidden size	**LSTM**	**BELM**	**BLLM**
Memory	500	129.1M	13.8M	4.1M
PPL		**57.6**	58.4	60.4
Memory	1000	274.2M	47.6M	8.9M
PPL		56.1	**55.6**	56.2

Table 3: Performance on the English SWB corpus

	Hidden size	**LSTM**	**BELM**	**BLLM**
Memory	500	170.8M	15.1M	5.4M
PPL		90.0	**89.8**	96.8
Memory	1000	357.6M	50.2M	11.5M
PPL		89.5	**87.8**	94.3

Table 4: Performance on the Chinese SMS corpus

Because the total memory usage relates to both the vocabulary size and hidden layer size, the memory reduction on various corpora is quite different. For our BELM model, the floating point embedding parameters are replaced by single bits, which could significantly reduce the memory usage. On the PTB corpus, the BELM models even outperform the baseline LSTM LM. The small model (500 LSTM units) has a relative PPL improvement of 4.1% and achieves a compression ratio of 4.3 and the large model (1000 LSTM units) also has a relative PPL improvement of 4.1% and achieves a compression ratio of 2.6. On the SWB corpus, the BELM models still perform well compared with the baseline model and achieve compression ratios of 9.4 and 5.8 respectively for the small and large models. On the SMS corpus, the BELMs model also gains relative PPL improvements of 0.2% and 1.9%, and achieves compression ratios of 11.3 and 7.1 respectively. In summary, the BELM model performs as well as the baseline model both on English and Chinese corpora, and reduces the memory consumption to a large extent.

The BLLM model, however, does not outperform the baseline model, but still has acceptable results with a minor loss of performance. Since both the LSTM model and the embeddings are binarized, the total compression ratio is quite significant. The average compression ratio is about 32.0, so the memory consumption of the language model is significantly reduced.

We also study the performance of pruning the LSTM language model. We prune each parameter matrix and the embedding layers with various pruning rates respectively, and fine-tune the model with various dropout rates. In our experiments, pruning 75% parameter nodes hardly affects the performance. However, if we try pruning more parameter nodes, the perplexity increases rapidly. For example, for the English PTB dataset, when we prune 95% parameter nodes of the embedding layers of an LSTM language model (500 LSTM units), the perpexity will increase from 91.8 to 112.3. When we prune 95% parameter nodes of an LSTM language model (500 LSTM units), the perplexity will increase from 91.8 to 132.3. Therefore, the effect of pruning is not as good as binarization for the language modeling task.

Binarization can be considered as a special case of quantization, which quantizes the parameters to pairs of opposite numbers. So, compared to normal quantization, binarization can achieve a better compression ratio. In addition, for binarization, we do not need to determine the position of each unique values in advance. Therefore, binarization is more flexible than quantization.

We then study the effect of extra binary linear layers in the BLLM. The additional binary linear layer after the input embedding layer and the additional binary linear layer in front of the output embedding layer are removed respectively in this experiment. We use well-trained embeddings to initialize the corresponding embedding layers and do the binarization using the method proposed in (Rastegari et al., 2016) when the additional binary linear layer is removed. The perplexities are listed in Table 5. No-i means no additional binary linear layer after the input embedding layer. No-o means no additional binary linear layer in front of the output embedding layer. No-io means no additional binary linear layers. The experiment is conducted on the PTB corpus.

	Hidden size	**BLLM**	**BLLM no-i**	**BLLM no-o**	**BLLM no-io**
PPL	500	95.2	95.2	101.7	100.3
PPL	1000	94.9	94.5	96.7	96.3

Table 5: Performances on the English PTB corpus

If the additional binary linear layer after the input embedding layer is removed, the performance does not drop, and even becomes better when the hidden layer size is 1000. Although the additional binary layer after the input embedding layer is removed, the float version of the input embeddings of BLLM no-i is initialized with well-trained embeddings, while the BLLM is not initialized with the well-trained embeddings. We think initialization is the reason why the BLLM no-i performs comparatively to the BLLM. We also observe a PPL increase of 1-2 points for BLLM no-i if the input embeddings are not pre-trained (not listed in the table). This phenomenon prompts us to pretrain embeddings, which we leave to future work. Once the additional binary linear layer in front of the output embedding layer is removed, the performance degradation is serious. This shows that the output embeddings of the language model should not be directly binarized; the additional binary linear layer should be inserted to enhance the model's capacity, especially for low dimensional models.

4.3 Experiments on ASR Rescoring Tasks

Experiments are conducted on the ASR rescoring task to evaluate the model on the hub5e and SMS corpora. Hub5e is a test dataset of the SWB corpus which we use for ASR rescoring tasks. For the hub5e dateset, A VDCNN (Qian et al., 2016) (very deep CNN) model on the 300-hour task is applied as the acoustic model. For the Chinese SMS dataset, the acoustic model is a CD-DNN-HMM model. The weighted finite state transducer (WFST) is produced with a 4-gram language model. Then our language models are utilized to rescore the 100-best candidates. The models are evaluated by the metric of word error rate (WER).

Model	Hidden size	hub5e	SMS
LSTM	500	8.7	10.5
BELM		**8.5**	**10.3**
BLLM		8.7	10.8
LSTM	1000	8.5	10.4
BELM		8.5	**10.2**
BLLM		**8.4**	10.3

Table 6: Performances on ASR rescoring tasks

Table 6 shows the results on ASR rescoring tasks. The BELM model and BLLM model perform well both on the English and Chinese datasets. The BELM model achieves an absolute 0.2% WER improvement compared with the baseline model in three of the experiments. The BLLM model also has good results, even though it performs not so well in language modeling. The results show that our language models work well on ASR rescoring tasks even with much less memory consumption.

4.4 Investigation of Binarized Embeddings

The experiments above show the good performances of our models. We also want to investigate whether the binarized embeddings lose any information. So, the embeddings are evaluated on two word similarity tasks. Experiments are conducted on the WS-353 and MEN tasks. We have trained the baseline LSTM model, the BELM model and BLLM model of a medium size on the Text8 corpus. We binarize the embeddings of the trained baseline LSTM model to investigate whether there is any loss of information by the simple binarization method (labeled LSTM-bin in the table below). For each dimension, we calculate the mean and set the value to 1 if it is bigger than the mean, otherwise, we set it to -1.

The embedding size and the hidden layer size are set to 500. We use stochastic gradient descent (SGD) to optimize our models. We use cosine distance to evaluate the similarity of the word pairs. Spearman's rank correlation coefficient is calculated to evaluate the correlation between the two scores given by our models and domain experts.

Model	PPL
LSTM	166.0
BELM	164.7
BLLM	172.3

Table 7: Language modeling performance on the Text8 corpus

Model	WS-353	MEN
LSTM	53.1	46.3
LSTM-bin	25.5	19.4
BELM	49.1	47.0
BLLM	**56.0**	**52.2**

Table 8: Performances on the word similarity tasks

Table 7 shows our models perform well in language modeling on the Text8 corpus. Table 8 summarizes the performance of the word embeddings in the similarity tasks. The embeddings generated by the simple binarization method perform obviously worse than the other embeddings, which indicates that much information is lost. The BELM model outperforms the baseline model on the MEN task, although it doesnt perform as well as the baseline model on the WS-353 task. However, the MEN dataset contains many more word pairs, which makes the results on this dataset more convincing. The BLLM model significantly outperforms the baseline model on the two tasks. The results indicate that the binarized embeddings of the BLLM do not lose any semantic information although the parameters are represented only by -1 and 1.

We suspect that binarization plays a role in regularization and produces more robust vectors. We also give an example visualization of some word vectors. The dimension of the embeddings of the BLLM is reduced by TSNE (Maaten and Hinton, 2008). The words which are the closest to **father** (according to the cosine distance of word vectors) are shown in Figure 1.

In this figure, **mother** and **parents** are the closest words to **father**, which is quite understandable. The words **husband**, **wife**, **grandfather** and **grandmother** also gather together and most words in the figure are related to **father**, indicat-

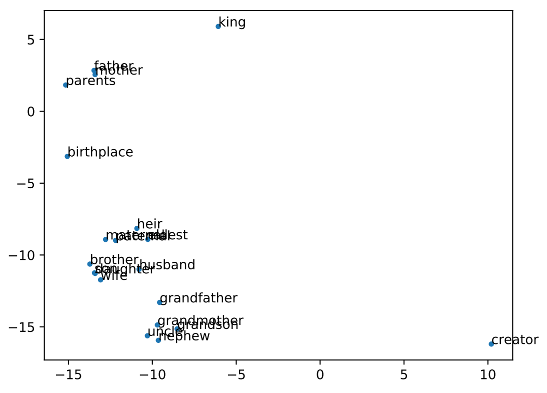

Figure 1: Visualization of the Binarized Embeddings

ing the embeddings indeed carry semantic information.

5 Conclusion

In this paper, a novel language model, the binarized embedding language model (BELM) is proposed to solve the problem that NN based language models occupy tremendous space. For traditional RNN based language models, the memory consumption mainly comes from the embedding layers (both input and output layers). However, when the hidden layer size grows, the memory consumption of the RNN module also becomes larger. So, the total memory usage relates to both the vocabulary size and hidden layer size. In the BELM model, words are represented in the form of binarized vectors, which only contain parameters of -1 or 1. For further compression, we binarize the long short-term memory language model combined with the binarized embeddings. Thus, the total memory usage can be significantly reduced. Experiments are conducted on language modeling and ASR rescoring tasks on various corpora. The results show that the BELM model performs well without any loss of performances at compression ratios of 2.6 to 11.3, depending on the hidden and vocabulary size. The BLLM model compresses the model parameters almost thirty-two times with a slight loss of performance. We also evaluate the embeddings on word similarity tasks. The results show the binarized embeddings even perform much better than the baseline embeddings.

6 Future Work

In the future, we will study how to improve the performance of the BLLM model. And, we will research methods to accelerate the training and reduce the memory consumption during training.

Acknowledgments

The corresponding author is Kai Yu. This work has been supported by the National Key Research and Development Program of China under Grant No.2017YFB1002102, and the China NSFC projects (No. 61573241). Experiments have been carried out on the PI supercomputer at Shanghai Jiao Tong University.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3(Feb):1137–1155.

Elia Bruni, Nam-Khanh Tran, and Marco Baroni. 2014. Multimodal distributional semantics. *J. Artif. Intell. Res.(JAIR)* 49(2014):1–47.

Di Cao and Kai Yu. 2017. Deep attentive structured language model based on lstm. In *International Conference on Intelligent Science and Big Data Engineering*. Springer, pages 169–180.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* .

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*. pages 3123–3131.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830* .

Marcus Edel and Enrico Köppe. 2016. Binarized-blstm-rnn based human activity recognition. In *Indoor Positioning and Indoor Navigation (IPIN), 2016 International Conference on*. IEEE, pages 1–7.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*. ACM, pages 406–414.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* .

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Lu Hou, Quanming Yao, and James T Kwok. 2016. Loss-aware binarization of deep networks. *arXiv preprint arXiv:1611.01600* .

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in neural information processing systems*. pages 4107–4115.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*. pages 448–456.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research* 9(Nov):2579–2605.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. *Building a large annotated corpus of English: the penn treebank*. MIT Press.

Tomas Mikolov, Martin Karafit, Lukas Burget, Jan Cernock, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September*. pages 1045–1048.

Yanmin Qian, Mengxiao Bi, Tian Tan, Kai Yu, Yanmin Qian, Mengxiao Bi, Tian Tan, and Kai Yu. 2016. Very deep convolutional neural networks for noise robust speech recognition. *IEEE/ACM Transactions on Audio Speech & Language Processing* 24(12):2263–2276.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, pages 525–542.

Xu Xiang, Yanmin Qian, and Kai Yu. 2017. Binary deep neural networks for speech recognition. *Proc. Interspeech 2017* pages 533–537.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* .