Compressing Neural Language Models by Sparse Word Representations

Yunchuan Chen,[1,2] **Lili Mou,**[1,3] **Yan Xu,**[1,3] **Ge Li,**[1,3] **Zhi Jin**[1,3,*]
[1]Key Laboratory of High Confidence Software Technologies (Peking University), MoE, China
[2]University of Chinese Academy of Sciences, chenyunchuan11@mails.ucas.ac.cn
[3]Institute of Software, Peking University, doublepower.mou@gmail.com,
{xuyan14,lige,zhijin}@pku.edu.cn *Corresponding author

Abstract

Neural networks are among the state-of-the-art techniques for language modeling. Existing neural language models typically map discrete words to distributed, dense vector representations. After information processing of the preceding context words by hidden layers, an output layer estimates the probability of the next word. Such approaches are time- and memory-intensive because of the large numbers of parameters for word embeddings and the output layer. In this paper, we propose to compress neural language models by sparse word representations. In the experiments, the number of parameters in our model increases very slowly with the growth of the vocabulary size, which is almost imperceptible. Moreover, our approach not only reduces the parameter space to a large extent, but also improves the performance in terms of the perplexity measure.[1]

1 Introduction

Language models (LMs) play an important role in a variety of applications in natural language processing (NLP), including speech recognition and document recognition. In recent years, neural network-based LMs have achieved significant breakthroughs: they can model language more precisely than traditional n-gram statistics (Mikolov et al., 2011); it is even possible to generate new sentences from a neural LM, benefiting various downstream tasks like machine translation, summarization, and dialogue systems (Devlin et al., 2014; Rush et al., 2015; Sordoni et al., 2015; Mou et al., 2015b).

[1]Code released on https://github.com/chenych11/lm

Existing neural LMs typically map a discrete word to a distributed, real-valued vector representation (called *embedding*) and use a neural model to predict the probability of each word in a sentence. Such approaches necessitate a large number of parameters to represent the embeddings and the output layer's weights, which is unfavorable in many scenarios. First, with a wider application of neural networks in resource-restricted systems (Hinton et al., 2015), such approach is too memory-consuming and may fail to be deployed in mobile phones or embedded systems. Second, as each word is assigned with a dense vector—which is tuned by gradient-based methods—neural LMs are unlikely to learn meaningful representations for infrequent words. The reason is that infrequent words' gradient is only occasionally computed during training; thus their vector representations can hardly been tuned adequately.

In this paper, we propose a compressed neural language model where we can reduce the number of parameters to a large extent. To accomplish this, we first represent infrequent words' embeddings with frequent words' by sparse linear combinations. This is inspired by the observation that, in a dictionary, an unfamiliar word is typically defined by common words. We therefore propose an optimization objective to compute the sparse codes of infrequent words. The property of sparseness (only 4–8 values for each word) ensures the efficiency of our model.

Based on the pre-computed sparse codes, we design our compressed language model as follows. A dense embedding is assigned to each common word; an infrequent word, on the other hand, computes its vector representation by a sparse combination of common words' embeddings. We use the long short term memory (LSTM)-based recurrent neural network (RNN) as the hidden layer of our model. The weights of the output layer are also compressed in a same way as embeddings. Consequently, the number of trainable neural parameters is a constant regardless of the vocabulary size if we ignore the biases of words. Even considering sparse codes (which are very small), we find the memory consumption grows imperceptibly with respect to the vocabulary.

We evaluate our LM on the Wikipedia corpus containing up to 1.6 billion words. During training, we adopt noise-contrastive estimation (NCE) (Gutmann and Hyvärinen, 2012) to estimate the parameters of our neural LMs. However, different from Mnih and Teh (2012), we tailor the NCE method by adding a regression layer (called `ZRegressoion`) to predict the normalization factor, which stabilizes the training process. Experimental results show that, our compressed LM not only reduces the memory consumption, but also improves the performance in terms of the perplexity measure.

To sum up, the main contributions of this paper are three-fold. (1) We propose an approach to represent uncommon words' embeddings by a sparse linear combination of common ones'. (2) We propose a compressed neural language model based on the pre-computed sparse codes. The memory increases very slowly with the vocabulary size (4–8 values for each word). (3) We further introduce a `ZRegression` mechanism to stabilize the NCE algorithm, which is potentially applicable to other LMs in general.

2 Background

2.1 Standard Neural LMs

Language modeling aims to minimize the joint probability of a corpus (Jurafsky and Martin, 2014). Traditional n-gram models impose a Markov assumption that a word is only dependent on previous $n-1$ words and independent of its position. When estimating the parameters, researchers have proposed various smoothing techniques including back-off models to alleviate the problem of data sparsity.

Bengio et al. (2003) propose to use a feed-forward neural network (FFNN) to replace the multinomial parameter estimation in n-gram models. Recurrent neural networks (RNNs) can also be used for language modeling; they are especially capable of capturing long range dependencies in sentences (Mikolov et al., 2010; Sundermeyer et al., 2015).

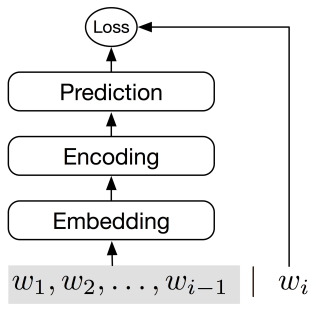


Figure 1: The architecture of a neural network-based language model.

In the above models, we can view that a neural LM is composed of three main parts, namely the `Embedding`, `Encoding`, and `Prediction` subnets, as shown in Figure 1.

The `Embedding` subnet maps a word to a dense vector, representing some abstract features of the word (Mikolov et al., 2013). Note that this subnet usually accepts a list of words (known as history or context words) and outputs a sequence of word embeddings.

The `Encoding` subnet encodes the history of a target word into a dense vector (known as *context* or *history* representation). We may either leverage FFNNs (Bengio et al., 2003) or RNNs (Mikolov et al., 2010) as the `Encoding` subnet, but RNNs typically yield a better performance (Sundermeyer et al., 2015).

The `Prediction` subnet outputs a distribution of target words as

$$p(w = w_i | h) = \frac{\exp(s(h, w_i))}{\sum_j \exp(s(h, w_j))}, \quad (1)$$

$$s(h, w_i) = \boldsymbol{W}_i^\top \boldsymbol{h} + b_i, \quad (2)$$

where $\boldsymbol{h}$ is the vector representation of context/history h, obtained by the `Encoding` subnet. $\boldsymbol{W} = (\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_V) \in \mathbb{R}^{C\times V}$ is the *output weights* of `Prediction`; $\boldsymbol{b} = (b_1, b_2, \ldots, b_V) \in \mathbb{R}^C$ is the bias (the prior). $s(h, w_i)$ is a scoring function indicating the degree to which the context h matches a target word w_i. (V is the size of vocabulary $\mathcal{V}$; C is the dimension of context/history, given by the `Encoding` subnet.)

2.2 Complexity Concerns of Neural LMs

Neural network-based LMs can capture more precise semantics of natural language than n-gram models because the regularity of the `Embedding` subnet extracts meaningful semantics of a word and the high capacity of `Encoding` subnet enables complicated information processing.

Despite these, neural LMs also suffer from several disadvantages mainly out of complexity concerns.

Time complexity. Training neural LMs is typically time-consuming especially when the vocabulary size is large. The normalization factor in Equation (1) contributes most to time complexity. Morin and Bengio (2005) propose hierarchical softmax by using a Bayesian network so that the probability is self-normalized. Sampling techniques—for example, importance sampling (Bengio and Senécal, 2003), noise-contrastive estimation (Gutmann and Hyvärinen, 2012), and target sampling (Jean et al., 2014)—are applied to avoid computation over the entire vocabulary. Infrequent normalization maximizes the unnormalized likelihood with a penalty term that favors normalized predictions (Andreas and Klein, 2014).

Memory complexity and model complexity. The number of parameters in the `Embedding` and `Prediction` subnets in neural LMs increases linearly with respect to the vocabulary size, which is large (Table 1). As said in Section 1, this is sometimes unfavorable in memory-restricted systems. Even with sufficient hardware resources, it is problematic because we are unlikely to fully tune these parameters. Chen et al. (2015) propose the differentiated softmax model by assigning fewer parameters to rare words than to frequent words. However, their approach only handles the output weights, i.e., $\boldsymbol{W}$ in Equation (2); the input embeddings remain uncompressed in their approach.

In this work, we mainly focus on memory and model complexity, i.e., we propose a novel method to compress the `Embedding` and `Prediction` subnets in neural language models.

2.3 Related Work

Existing work on model compression for neural networks. Buciluǎ et al. (2006) and Hinton et al. (2015) use a well-trained large network to guide the training of a small network for model compression. Jaderberg et al. (2014) compress neural models by matrix factorization, Gong et al. (2014) by quantization. In NLP, Mou et al. (2015a) learn an embedding subspace by supervised training. Our work resembles little, if any, to the above methods as we compress embeddings and output weights using sparse word representations. Existing model

Sub-nets	RNN-LSTM	FFNN
`Embedding`	VE	VE
`Encoding`	$4(CE + C^2 + C)$	$nCE + C$
`Prediction`	$V(C+1)$	$V(C+1)$
TOTAL†	$\mathcal{O}((C+E)V)$	$\mathcal{O}((E+C)V)$

Table 1: Number of parameters in different neural network-based LMs. E: embedding dimension; C: context dimension; V: vocabulary size. †Note that $V \gg C$ (or E).

compression typically works with a compromise of performance. On the contrary, our model improves the perplexity measure after compression.

Sparse word representations. We leverage sparse codes of words to compress neural LMs. Faruqui et al. (2015) propose a sparse coding method to represent each word with a sparse vector. They solve an optimization problem to obtain the sparse vectors of words as well as a dictionary matrix simultaneously. By contrast, we do not estimate any dictionary matrix when learning sparse codes, which results in a simple and easy-to-optimize model.

3 Our Proposed Model

In this section, we describe our compressed language model in detail. Subsection 3.1 formalizes the sparse representation of words, serving as the premise of our model. On such a basis, we compress the `Embedding` and `Prediction` subnets in Subsections 3.2 and 3.3, respectively. Finally, Subsection 3.4 introduces NCE for parameter estimation where we further propose the `ZRegression` mechanism to stabilize our model.

3.1 Sparse Representations of Words

We split the vocabulary $\mathcal{V}$ into two disjoint subsets ($\mathcal{B}$ and $\mathcal{C}$). The first subset $\mathcal{B}$ is a base set, containing a fixed number of common words (8k in our experiments). $\mathcal{C} = \mathcal{V}\backslash\mathcal{B}$ is a set of uncommon words. We would like to use $\mathcal{B}$'s word embeddings to encode $\mathcal{C}$'s.

Our intuition is that oftentimes a word can be defined by a few other words, and that rare words should be defined by common ones. Therefore, it is reasonable to use a few common words' embeddings to represent that of a rare word. Following most work in the literature (Lee et al., 2006; Yang et al., 2011), we represent each uncommon word with a sparse, linear combination of common ones' embeddings. The sparse coefficients are called a *sparse code* for a given word.

We first train a word representation model like SkipGram (Mikolov et al., 2013) to obtain a set of embeddings for each word in the vocabulary, including both common words and rare words. Suppose $\boldsymbol{U} = (\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_B) \in \mathbb{R}^{E\times B}$ is the (learned) embedding matrix of common words, i.e., $\boldsymbol{U}_i$ is the embedding of i-th word in $\mathcal{B}$. (Here, $B = |\mathcal{B}|$.)

Each word in $\mathcal{B}$ has a natural sparse code (denoted as $\boldsymbol{x}$): it is a one-hot vector with B elements, the i-th dimension being on for the i-th word in $\mathcal{B}$.

For a word $w \in \mathcal{C}$, we shall learn a sparse vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_B)$ as the sparse code of the word. Provided that $\boldsymbol{x}$ has been learned (which will be introduced shortly), the embedding of w is

$$\hat{\boldsymbol{w}} = \sum_{j=1}^{B} x_j \boldsymbol{U}_j = \boldsymbol{U}\boldsymbol{x}, \tag{3}$$

To learn the sparse representation of a certain word w, we propose the following optimization objective

$$\min_{\boldsymbol{x}} \; \|\boldsymbol{U}\boldsymbol{x} - \boldsymbol{w}\|_2^2 + \alpha\|\boldsymbol{x}\|_1 + \beta|\mathbf{1}^\top \boldsymbol{x} - 1| + \gamma \mathbf{1}^\top \max\{\mathbf{0}, -\boldsymbol{x}\}, \tag{4}$$

where max denotes the component-wise maximum; $\boldsymbol{w}$ is the embedding for a rare word $w \in \mathcal{C}$.

The first term (called *fitting loss* afterwards) evaluates the closeness between a word's coded vector representation and its "true" representation $\boldsymbol{w}$, which is the general goal of sparse coding.

The second term is an ℓ_1 regularizer, which encourages a sparse solution. The last two regularization terms favor a solution that sums to 1 and that is nonnegative, respectively. The nonnegative regularizer is applied as in He et al. (2012) due to psychological interpretation concerns.

It is difficult to determine the hyperparameters α, β, and γ. Therefore we perform several tricks. First, we drop the last term in the problem (4), but clip each element in $\boldsymbol{x}$ so that all the sparse codes are nonnegative during each update of training.

Second, we re-parametrize α and β by balancing the fitting loss and regularization terms dynamically during training. Concretely, we solve the following optimization problem, which is slightly different but closely related to the conceptual objective (4):

$$\min_{\boldsymbol{x}} \; L(\boldsymbol{x}) + \alpha_t R_1(\boldsymbol{x}) + \beta_t R_2(\boldsymbol{x}), \tag{5}$$

where $L(\boldsymbol{x}) = \|\boldsymbol{U}\boldsymbol{x} - \boldsymbol{w}\|_2^2$, $R_1(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$, and $R_2(\boldsymbol{x}) = |\mathbf{1}^\top \boldsymbol{x} - 1|$. α_t and β_t are adaptive parameters that are resolved during training time. Suppose $\boldsymbol{x}_t$ is the value we obtain after the update of the t-th step, we expect the importance of fitness and regularization remain unchanged during training. This is equivalent to

$$\frac{\alpha_t R_1(\boldsymbol{x}_t)}{L(\boldsymbol{x}_t)} = w_\alpha \equiv \text{const}, \tag{6}$$

$$\frac{\beta_t R_2(\boldsymbol{x}_t)}{L(\boldsymbol{x}_t)} = w_\beta \equiv \text{const}. \tag{7}$$

or

$$\alpha_t = \frac{L(\boldsymbol{x}_t)}{R_1(\boldsymbol{x}_t)} w_\alpha \text{ and } \beta_t = \frac{L(\boldsymbol{x}_t)}{R_2(\boldsymbol{x}_t)} w_\beta,$$

where w_α and w_β are the ratios between the regularization loss and the fitting loss. They are much easier to specify than α or β in the problem (4).

We have two remarks as follows.

- To learn the sparse codes, we first train the "true" embeddings by `word2vec`[2] for both common words and rare words. However, these true embeddings are slacked during our language modeling.
- As the codes are pre-computed and remain unchanged during language modeling, they are not tunable parameters of our neural model. Considering the learned sparse codes, we need only 4–8 values for each word on average, as the codes contain 0.05–0.1% nonzero values, which are almost negligible.

3.2 Parameter Compression for the `Embedding` Subnet

One main source of LM parameters is the `Embedding` subnet, which takes a list of words (history/context) as input, and outputs dense, low-dimensional vector representations of the words.

We leverage the sparse representation of words mentioned above to construct a compressed `Embedding` subnet, where the number of parameters is independent of the vocabulary size.

By solving the optimization problem (5) for each word, we obtain a non-negative sparse code $\boldsymbol{x} \in \mathbb{R}^B$ for each word, indicating the degree to which the word is related to common words in $\mathcal{B}$. Then the embedding of a word is given by $\hat{\boldsymbol{w}} = \boldsymbol{U}\boldsymbol{x}$.

[2]https://code.google.com/archive/p/word2vec

We would like to point out that the embedding of a word $\hat{\boldsymbol{w}}$ is not sparse because $\boldsymbol{U}$ is a dense matrix, which serves as a shared parameter of learning all words' vector representations.

3.3 Parameter Compression for the `Prediction` Subnet

Another main source of parameters is the `Prediction` subnet. As Table 1 shows, the output layer contains V target-word weight vectors and biases; the number increases with the vocabulary size. To compress this part of a neural LM, we propose a weight-sharing method that uses words' sparse representations again. Similar to the compression of word embeddings, we define a base set of weight vectors, and use them to represent the rest weights by sparse linear combinations.

Without loss of generality, we let $\boldsymbol{D} = \boldsymbol{W}_{:,1:B}$ be the output weights of B base target words, and $\boldsymbol{c} = \boldsymbol{b}_{1:B}$ be bias of the B target words.[3] The goal is to use $\boldsymbol{D}$ and $\boldsymbol{c}$ to represent $\boldsymbol{W}$ and $\boldsymbol{b}$. However, as the values of $\boldsymbol{W}$ and $\boldsymbol{b}$ are unknown before the training of LM, we cannot obtain their sparse codes in advance.

We claim that it is reasonable to share the same set of sparse codes to represent word vectors in `Embedding` and the output weights in the `Prediction` subnet. In a given corpus, an occurrence of a word is always companied by its context. The co-occurrence statistics about a word or corresponding context are the same. As both word embedding and context vectors capture these co-occurrence statistics (Levy and Goldberg, 2014), we can expect that context vectors share the same internal structure as embeddings. Moreover, for a fine-trained network, given any word w and its context h, the output layer's weight vector corresponding to w should specify a large inner-product score for the context h; thus these context vectors should approximate the weight vector of w. Therefore, word embeddings and the output weight vectors should share the same internal structures and it is plausible to use a same set of sparse representations for both words and target-word weight vectors. As we shall show in Section 4, our treatment of compressing the `Prediction` subnet does make sense and achieves high performance.

Formally, the i-th output weight vector is estimated by

$$\hat{\boldsymbol{W}}_i = \boldsymbol{D}\boldsymbol{x}_i, \tag{8}$$

[3] $\boldsymbol{W}_{:,1:B}$ is the first B columns of $\boldsymbol{W}$.

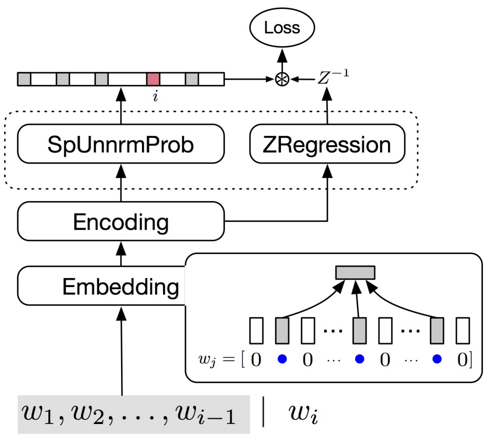

Loss
Z^{-1}
i
SpUnnrmProb
ZRegression
Encoding
Embedding
$w_j = [\,0 \;\bullet\; 0 \cdots \bullet\; 0 \cdots \bullet\; 0]$
$w_1, w_2, \ldots, w_{i-1}$
w_i

Figure 2: Compressing the output of neural LM. We apply NCE to estimate the parameters of the `Prediction` sub-network (dashed round rectangle). The SpUnnrmProb layer outputs a **sp**arse, **unnorm**alized **prob**ability of the next word. By "sparsity," we mean that, in NCE, the probability is computed for only the "true" next word (red) and a few generated negative samples.

The biases can also be compressed as

$$\hat{b}_i = \boldsymbol{c}\boldsymbol{x}_i. \tag{9}$$

where $\boldsymbol{x}_i$ is the sparse representation of the i-th word. (It is shared in the compression of weights and biases.)

In the above model, we have managed to compressed a language model whose number of parameters is irrelevant to the vocabulary size.

To better estimate a "prior" distribution of words, we may alternatively assign an independent bias to each word, i.e., $\boldsymbol{b}$ is not compressed. In this variant, the number of model parameters grows very slowly and is also negligible because each word needs only one extra parameter. Experimental results show that by not compressing the bias vector, we can even improve the performance while compressing LMs.

3.4 Noise-Contrastive Estimation with `ZRegression`

We adopt the noise-contrastive estimation (NCE) method to train our model. Compared with the maximum likelihood estimation of softmax, NCE reduces computational complexity to a large degree. We further propose the `ZRegression` mechanism to stablize training.

NCE generates a few negative samples for each positive data sample. During training, we only need to compute the unnormalized probability of these positive and negative samples. Interested readers are referred to (Gutmann and Hyvärinen, 2012) for more information.

Formally, the estimated probability of the word w_i with history/context h is

$$\begin{aligned} P(w|h;\boldsymbol{\theta}) &= \frac{1}{Z_h} P^0(w_i|h;\boldsymbol{\theta}) \\ &= \frac{1}{Z_h} \exp(s(w_i,h;\boldsymbol{\theta})), \end{aligned} \tag{10}$$

where $\boldsymbol{\theta}$ is the parameters and Z_h is a context-dependent normalization factor. $P^0(w_i|h;\boldsymbol{\theta})$ is the unnormalized probability of the w (given by the SpUnnrmProb layer in Figure 2).

The NCE algorithm suggests to take Z_h as parameters to optimize along with $\boldsymbol{\theta}$, but it is intractable for context with variable lengths or large sizes in language modeling. Following Mnih and Teh (2012), we set $Z_h = 1$ for all h in the base model (without `ZRegression`).

The objective for each occurrence of context/history h is

$$\begin{aligned} J(\boldsymbol{\theta}|h) = &\log \frac{P(w_i|h;\boldsymbol{\theta})}{P(w_i|h;\boldsymbol{\theta}) + kP_n(w_i)} + \\ &\sum_{j=1}^{k} \log \frac{kP_n(w_j)}{P(w_j|h;\boldsymbol{\theta}) + kP_n(w_j)}, \end{aligned}$$

where $P_n(w)$ is the probability of drawing a negative sample w; k is the number of negative samples that we draw for each positive sample.

The overall objective of NCE is

$$J(\boldsymbol{\theta}) = \mathbb{E}_h[J(\boldsymbol{\theta}|h)] \approx \frac{1}{M}\sum_{i=1}^{M} J(\boldsymbol{\theta}|h_i),$$

where h_i is an occurrence of the context and M is the total number of context occurrences.

Although setting Z_h to 1 generally works well in our experiment, we find that in certain scenarios, the model is unstable. Experiments show that when the true normalization factor is far away from 1, the cost function may vibrate. To comply with NCE in general, we therefore propose a `ZRegression` layer to predict the normalization constant Z_h dependent on h, instead of treating it as a constant.

The regression layer is computed by

$$Z_h^{-1} = \exp(\boldsymbol{W}_Z^\top \boldsymbol{h} + b_Z),$$

Partitions	Running words
Train (n-gram)	1.6 B
Train (neural LMs)	100 M
Dev	100 K
Test	5 M

Table 2: Statistics of our corpus.

where $\boldsymbol{W}_Z \in \mathbb{R}^C$ and $b_Z \in \mathbb{R}$ are weights and bias for `ZRegression`. Hence, the estimated probability by NCE with `ZRegression` is given by

$$P(w|h) = \exp(s(h,w)) \cdot \exp(\boldsymbol{W}_Z^\top \boldsymbol{h} + b_Z).$$

Note that the `ZRegression` layer does not guarantee normalized probabilities. During validation and testing, we explicitly normalize the probabilities by Equation (1).

4 Evaluation

In this part, we first describe our dataset in Subsection 4.1. We evaluate our learned sparse codes of rare words in Subsection 4.2 and the compressed language model in Subsection 4.3. Subsection 4.4 provides in-depth analysis of the `ZRegression` mechanism.

4.1 Dataset

We used the freely available Wikipedia[4] dump (2014) as our dataset. We extracted plain sentences from the dump and removed all markups. We further performed several steps of preprocessing such as text normalization, sentence splitting, and tokenization. Sentences were randomly shuffled, so that no information across sentences could be used, i.e., we did not consider cached language models. The resulting corpus contains about 1.6 billion running words.

The corpus was split into three parts for training, validation, and testing. As it is typically time-consuming to train neural networks, we sampled a subset of 100 million running words to train neural LMs, but the full training set was used to train the backoff n-gram models. We chose hyperparameters by the validation set and reported model performance on the test set. Table 2 presents some statistics of our dataset.

4.2 Qualitative Analysis of Sparse Codes

To obtain words' sparse codes, we chose 8k common words as the "dictionary," i.e., $B = 8000$.

[4]http://en.wikipedia.org

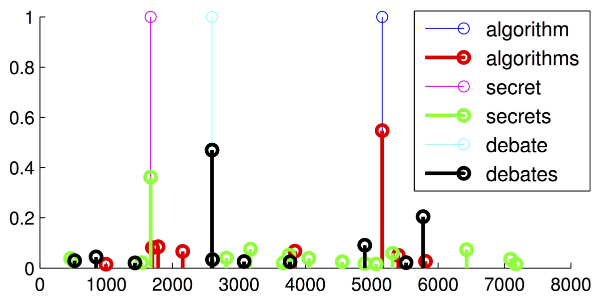


Figure 3: The sparse representations of selected words. The x-axis is the dictionary of 8k common words; the y-axis is the coefficient of sparse coding. Note that *algorithm*, *secret*, and *debate* are common words, each being coded by itself with a coefficient of 1.

We had 2k–42k uncommon words in different settings. We first pretrained word embeddings of both rare and common words, and obtained 200d vectors $\boldsymbol{U}$ and $\boldsymbol{w}$ in Equation (5). The dimension was specified in advance and not tuned. As there is no analytic solution to the objective, we optimized it by Adam (Kingma and Ba, 2014), which is a gradient-based method. To filter out small coefficients around zero, we simply set a value to 0 if it is less than $0.015 \cdot \max\{v \in \boldsymbol{x}\}$. w_α in Equation (6) was set to 1 because we deemed fitting loss and sparsity penalty are equally important. We set w_β in Equation (7) to 0.1, and this hyperparameter is insensitive.

Figure 3 plots the sparse codes of a few selected words. As we see, *algorithm*, *secret*, and *debate* are common words, and each is (sparsely) coded by itself with a coefficient of 1. We further notice that a rare word like *algorithms* has a sparse representation with only a few non-zero coefficient.

Moreover, the coefficient in the code of *algorithms*—corresponding to the base word *algorithm*—is large (~ 0.6), showing that the words *algorithm* and *algorithms* are similar. Such phenomena are also observed with *secret* and *debate*.

The qualitative analysis demonstrates that our approach can indeed learn a sparse code of a word, and that the codes are meaningful.

4.3 Quantitative Analysis of Compressed Language Models

We then used the pre-computed sparse codes to compress neural LMs, which provides quantitative analysis of the learned sparse representations of words. We take perplexity as the performance measurement of a language model, which is defined by

$$\mathrm{PPL} = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 p(w_i|h_i)}$$

where N is the number of running words in the test corpus.

4.3.1 Settings

We leveraged LSTM-RNN as the `Encoding` subnet, which is a prevailing class of neural networks for language modeling (Sundermeyer et al., 2015; Karpathy et al., 2015). The hidden layer was 200d. We used the Adam algorithm to train our neural models. The learning rate was chosen by validation from $\{0.001, 0.002, 0.004, 0.006, 0.008\}$. Parameters were updated with a mini-batch size of 256 words. We trained neural LMs by NCE, where we generated 50 negative samples for each positive data sample in the corpus. All our model variants and baselines were trained with the same pre-defined hyperparameters or tuned over a same candidate set; thus our comparison is fair.

We list our compressed LMs and competing methods as follows.

- **KN3**. We adopted the modified Kneser-Ney smoothing technique to train a 3-gram LM; we used the SRILM toolkit (Stolcke and others, 2002) in out experiment.
- **LBL5**. A **L**og-**BiL**inear model introduced in Mnih and Hinton (2007). We used 5 preceding words as context.
- **LSTM-s**. A standard LSTM-RNN language model which is applied in Sundermeyer et al. (2015) and Karpathy et al. (2015). We implemented the LM ourselves based on Theano (Theano Development Team, 2016) and also used NCE for training.
- **LSTM-z**. An LSTM-RNN enhanced with the `ZRegression` mechanism described in Section 3.4.
- **LSTM-z,wb**. Based on LSTM-z, we compressed word embeddings in `Embedding` and the output weights and biases in `Prediction`.
- **LSTM-z,w**. In this variant, we did not compress the bias term in the output layer. For each word in $\mathcal{C}$, we assigned an independent bias parameter.

4.3.2 Performance

Tables 3 shows the perplexity of our compressed model and baselines. As we see, LSTM-based LMs significantly outperform the log-bilinear

Vocabulary	10k	22k	36k	50k
KN3[†]	90. 4	125.3	146.4	159.9
LBL5	116. 6	167.0	199.5	220.3
LSTM-s	107. 3	159.5	189.4	222.1
LSTM-z	75. 1	104.4	119.6	130.6
LSTM-z,wb	73. 7	103.4	122.9	138.2
LSTM-z,w	72. 9	101.9	119.3	129.2

Table 3: Perplexity of our compressed language models and baselines. [†]Trained with the full corpus of 1.6 billion running words.

Vocabulary	10k	22k	36k	50k
LSTM-z,w	17.76	59.28	73.42	79.75
LSTM-z,wb	17.80	59.44	73.61	79.95

Table 4: Memory reduction (%) by our proposed methods in comparison with the uncompressed model LSTM-z. The memory of sparse codes are included.

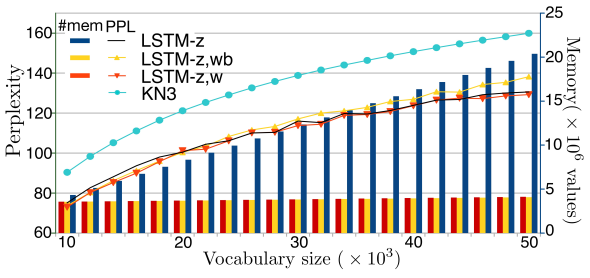


Figure 4: Fine-grained plot of performance (perplexity) and memory consumption (including sparse codes) versus the vocabulary size.

model as well as the backoff 3-gram LM, even if the 3-gram LM is trained on a much larger corpus with 1.6 billion words. The `ZRegression` mechanism improves the performance of LSTM to a large extent, which is unexpected. Subsection 4.4 will provide more in-depth analysis.

Regarding the compression method proposed in this paper, we notice that LSTM-z,wb and LSTM-z,w yield similar performance to LSTM-z. In particular, LSTM-z,w outperforms LSTM-z in all scenarios of different vocabulary sizes. Moreover, both LSTM-z,wb and LSTM-z,w can reduce the memory consumption by up to 80% (Table 4).

We further plot in Figure 4 the model performance (lines) and memory consumption (bars) in a fine-grained granularity of vocabulary sizes. We see such a tendency that compressed LMs (LSTM-z,wb and LSTM-z,w, yellow and red lines) are generally better than LSTM-z (black line) when we have a small vocabulary. However, LSTM-z,wb is slightly worse than LSTM-z if the vocabulary size is greater than, say, 20k. The LSTM-z,w remains comparable to LSTM-z as the vocabulary grows.

To explain this phenomenon, we may imagine that the compression using sparse codes has two effects: it loses information, but it also enables more accurate estimation of parameters especially for rare words. When the second factor dominates, we can reasonably expect a high performance of the compressed LM.

From the bars in Figure 4, we observe that traditional LMs have a parameter space growing linearly with the vocabulary size. But the number of parameters in our compressed models does not increase—or strictly speaking, increases at an extremely small rate—with vocabulary.

These experiments show that our method can largely reduce the parameter space with even performance improvement. The results also verify that the sparse codes induced by our model indeed capture meaningful semantics and are potentially useful for other downstream tasks.

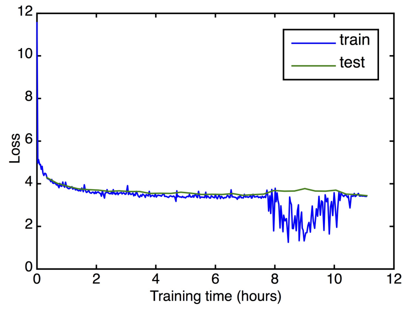
4.4 Effect of `ZRegression`

We next analyze the effect of `ZRegression` for NCE training. As shown in Figure 5a, the training process becomes unstable after processing 70% of the dataset: the training loss vibrates significantly, whereas the test loss increases.

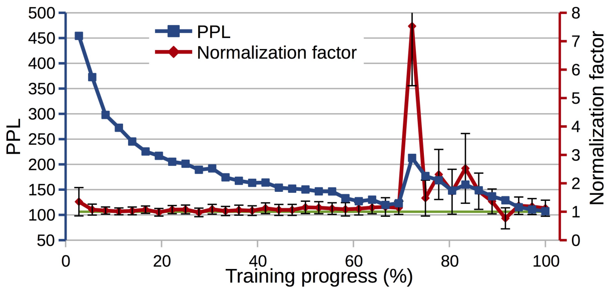
We find a strong correlation between unstableness and the Z_h factor in Equation (10), i.e., the sum of unnormalized probability (Figure 5b). Theoretical analysis shows that the Z_h factor tends to be self-normalized even though it is not forced to (Gutmann and Hyvärinen, 2012). However, problems would occur, should it fail.

In traditional methods, NCE jointly estimates normalization factor Z and model parameters (Gutmann and Hyvärinen, 2012). For language modeling, Z_h dependents on context h. Mnih and Teh (2012) propose to estimate a separate Z_h based on two history words (analogous to 3-gram), but their approach hardly scales to RNNs because of the exponential number of different combinations of history words.

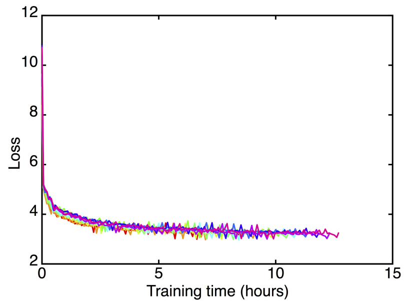
We propose the `ZRegression` mechanism in Section 3.4, which can estimate the Z_h factor well (Figure 5d) based on the history vector $\boldsymbol{h}$. In this way, we manage to stabilize the training process (Figure 5c) and improve the performance by



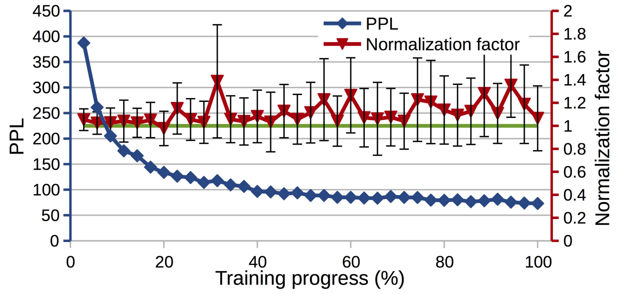
(a) Training/test loss vs. training time w/o `ZRegression`.



(b) The validation perplexity and normalization factor Z_h w/o `ZRegression`.



(c) Training loss vs. training time w/ `ZRegression` of different runs.



(d) The validation perplexity and normalization factor Z_h w/ `ZRegression`.

Figure 5: Analysis of `ZRegression`.

a large margin, as has shown in Table 3.

It should be mentioned that `ZRegression` is not specific to model compression and is generally applicable to other neural LMs trained by NCE.

5 Conclusion

In this paper, we proposed an approach to represent rare words by sparse linear combinations of common ones. Based on such combinations, we managed to compress an LSTM language model (LM), where memory does not increase with the vocabulary size except a bias and a sparse code for each word. Our experimental results also show that the compressed LM has yielded a better performance than the uncompressed base LM.

Acknowledgments

This research is supported by the National Basic Research Program of China (the 973 Program) under Grant No. 2015CB352201, the National Natural Science Foundation of China under Grant Nos. 61232015, 91318301, 61421091 and 61502014, and the China Post-Doctoral Foundation under Grant No. 2015M580927.

References

Jacob Andreas and Dan Klein. 2014. When and why are log-linear models self-normalizing. In *Proceedings of the Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 244–249.

Yoshua Bengio and Jean-Sébastien Senécal. 2003. Quick training of probabilistic neural nets by importance sampling. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3:1137–1155.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 535–541.

Welin Chen, David Grangier, and Michael Auli. 2015. Strategies for training large vocabulary neural language models. *arXiv preprint arXiv:1512.04906*.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *Proceedings of the 52rd Annual Meeting of the Association for Computational Linguistics*, pages 1370–1380.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A. Smith. 2015. Sparse overcomplete word vector representations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, pages 1491–1500.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

Michael Gutmann and Aapo Hyvärinen. 2012. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 13(1):307–361.

Zhanying He, Chun Chen, Jiajun Bu, Can Wang, Lijun Zhang, Deng Cai, and Xiaofei He. 2012. Document summarization based on data reconstruction. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 620–626.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference*.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2014. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*.

Dan Jurafsky and James H. Martin. 2014. *Speech and Language Processing*. Pearson.

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. 2006. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, pages 801–808.

Omer Levy and Yoav Goldberg. 2014. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the Eighteenth Conference on Natural Language Learning*, pages 171–180.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Tomas Mikolov, Anoop Deoras, Daniel Povey, Lukas Burget, and Jan Cernocký. 2011. Strategies for training large scale neural network language models. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 196–201.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Andriy Mnih and Geoffrey Hinton. 2007. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine learning*, pages 641–648.

Andriy Mnih and Yee-Whye Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*.

Fréderic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, pages 246–252.

Lili Mou, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. 2015a. Distilling word embeddings: An encoding approach. *arXiv preprint arXiv:1506.04488*.

Lili Mou, Rui Yan, Ge Li, Lu Zhang, and Zhi Jin. 2015b. Backward and forward language modeling for constrained natural language generation. *arXiv preprint arXiv:1512.06612*.

Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389.

Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. 2015. A neural network approach to context-sensitive generation of conversational responses. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 196–205.

Andreas Stolcke et al. 2002. SRILM—An extensible language modeling toolkit. In *INTERSPEECH*, pages 901–904.

Martin Sundermeyer, Hermann Ney, and Ralf Schlüter. 2015. From feedforward to recurrent LSTM neural networks for language modeling. *IEEE/ACM Transactions on Audio, Speech and Language Processing*, 23(3):517–529.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*.

Meng Yang, Lei Zhang, Jian Yang, and David Zhang. 2011. Robust sparse coding for face recognition. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, pages 625–632.