MulCode: A Multiplicative Multi-way Model for Compressing Neural Language Model

Yukun Ma[1]*, **Pei-Hung(Patrick) Chen**[2*], **Cho-Jui Hsieh**[2]
[1]AIR Labs, Continental Automotive Group, Singapore [2]Department of Computer Science, UCLA

Abstract

It is challenging to deploy deep neural nets on memory-constrained devices due to the explosion of numbers of parameters. Especially, the input embedding layer and Softmax layer usually dominate the memory usage in an RNN-based language model. For example, input embedding and Softmax matrices in IWSLT-2014 German-to-English data set account for more than 80% of the total model parameters. To compress these embedding layers, we propose MulCode, a novel multi-way multiplicative neural compressor. MulCode learns an adaptively created matrix and its multiplicative compositions. Together with a prior weighted loss, MulCode is more effective than the state-of-the-art compression methods. On the IWSLT-2014 machine translation data set, MulCode achieved 17 times compression rate for the embedding and Softmax matrices, and when combined with quantization technique, our method can achieve 41.38 times compression rate with very little loss in performance.

1 Introduction

Deep neural language models with a large number of parameters have achieved great successes in facilitating a wide range of Natural Language Processing (NLP) applications. However, the overwhelming size of parameters used by these models stands out as the main obstacle hindering their deployment on memory-constrained devices. Given the over-parameterization of deep neural nets, the effective compression of them has been receiving increasing attention from the research community. The neural language models typically consist of three major components: the recurrent layers (e.g., a LSTM cell), an embedding layer for representing input tokens, and a Softmax layer for generating output tokens. The dimension of recurrent layers is typically small and independent of the size of input/output vocabulary. In contrast, the embedding layer and Softmax layer use a dense layer to map the vocabulary to a lower dimensional space, which grows linearly with the vocabulary size and contributes mostly to memory consumption. For example, the One Billion Word language modeling task (Chelba et al., 2013) has a vocabulary size around 800k, and more than 90% of the memory usage is used to store the input and Softmax matrices. Compressing the word embedding, therefore, is the key to reducing the memory usage of deep neural language models.

There exist several studies for compressing neural language models. In particular, quantization (Lin et al., 2016; Hubara et al., 2016) and group-wise low rank approximation (Chen et al., 2018a) achieve the state-of-the-art performance on tasks like language modeling and machine translation. However, these methods are rather computationally inspired than being motivated by intuitions of the semantic composition and thus suffer from poor interpretability. In contrast, deep compositional models (Shu and Nakayama, 2017; Chen et al., 2018b) have been explored to represent the original word vector with a set of discrete pseudo words and thus decompose the word vector to the addition of the corresponding pseudo word vectors learned in an end-to-end manner. These methods are seen as embracing the long-existing yet still, the most popular assumption (Foltz et al., 1998) on compositionality of semantics which states that the meaning of an element (e.g., a word, phrase, or sentence) can be obtained by taking the addition of its constituent parts.

Despite it's more semantically meaningful, current deep compositional models have several limitations. First, additive composition might turn to be counter-intuitive for that the addition does not bring new meaning beyond the individual element to the constructed element (Pinker, 2003; Mitchell and Lapata, 2009, 2008). Besides, exist-

* The two authors contribute equally

ing deep code compressor is found hard to compress the output Softmax layer for that the conflicting codes (i.e. different words having same encoding) might make the language model less discriminative. Last, it is well-known that the distribution of word frequencies can be approximated by a power law. Defining importance of words in terms of frequencies in the corpus is also verified in (Chen et al., 2018a). However, current deep compositional frameworks encode every word equally without considering the frequency information.

In this paper, we propose MulCode, a novel and effective deep neural compressor to address the above issues. MulCode uses a multiplicative composition instead of addition. The multiplicative factors introduce a larger capacity empowering the encoding of more complicated semantic. From the perspective of semantic composition, this allows introducing new information to the base codebook vectors (sub-elements to compose a word vector), taking frequency information of each token into consideration. Technically, MulCode embraces the research line of dense matrix decomposition that has been investigated in many applications (Memisevic, 2011; Novikov et al., 2015). It trains the composition with weighted loss proportional to the frequency of each word. MulCode also adopts an adaptive way of constructing codebooks based on the frequency of each word.

MulCode outperforms state-of-the-arts methods on compressing language models and neural machine translation models. For example, on One-Billion-Word data set, MulCode achieves 18.2 times compression rate on embedding and Softmax matrices without losing much of performance. On IWSLT-14 DE-EN machine translation task, MulCode achieves 17 times compression with BLEU score difference smaller than one percent. Results can be further improved to 41.38 times compression rate when combined with quantization technique.

2 Related Work

2.1 Model Compression for Convolutional Neural Network

Low-rank Approximation The original word embedding vectors can be viewed as a large matrix with the rows being the words and columns being the vector values. One natural way to approximate it is to obtain the low-rank approximation of the matrix using SVD. Based on this idea, (Sainath et al., 2013) compressed the fully connected layers in neural nets. For convolution layers, (Jaderberg et al., 2014; Denton et al., 2014) applied higher-order tensor decomposition to compress CNN. Similarly, (Howard et al., 2017) developed another structural approximation. (Kim et al., 2016) proposed an algorithm to select rank for each layer. (Yu et al., 2017) reconstructed the weight matrices by using sparse plus low-rank approximation.

Pruning Many algorithms have been proposed to remove those unimportant weights in deep neural nets. To do this, one needs to define the importance of each weight. (LeCun et al., 1990) showed that the importance can be estimated by using the Hessian of loss function. (Han et al., 2015b) considered adding ℓ_1 or ℓ_2 regularization and applied iterative thresholding approaches to achieve very good compression rates. Later on, (Han et al., 2015a) demonstrated that CNNs can be compressed by combining pruning, weight sharing and quantization.

Quantization Using lower precision representations to store parameters has been used for model compression. (Hubara et al., 2016) showed that a simple uniform quantization scheme effectively reduces both the model size and the prediction time of a deep neural net. (Lin et al., 2016) showed that non-uniform quantization can further improve the performance. Recently, several advanced quantization techniques have been proposed for CNN compression (Xu et al., 2018; Choi et al., 2018).

2.2 Model Compression for Neural Language Models

Despite model compression has been studied extensively for CNN models, fewer work have focused on the compression for deep neural nets for NLP applications. In fact, directly applying popular approaches to NLP problems does not provide a satisfactory result. (Hubara et al., 2016) showed that the naive quantization can only achieve less than 3X compression rate on PTB data with no performance loss. (Lobacheva et al., 2017) showed that for word-level LSTM models, the pruning approach can only achieve 4 times compression with more than 5% performance loss. Most recently, (Chen et al., 2018a) proposed a decomposition scheme based on block-wise low

rank approximation of embedding matrix. Although this method achieves competitive empirical results, the learned model does not have a strong semantic interpretation.

Additive Composition One of the most popular assumptions about the composition of semantics is called the additive composition stating that the meaning of a unit (e.g., word, phrase, or sentences) can be obtained by summing up the meaning of its constituents. At the word level, a word might be decomposed into a set of subword units. For example, "disagree" = "dis"+"agree". Alternatively, a word can also be represented by a set of relevant words, e.g., "king" = "man" + "crown". (Gittens et al., 2017) has validated this particular assumption for a popular word embedding approach (i.e., Skip-Gram (Mikolov et al., 2013)).

Based on this assumption, (Chen et al., 2016) created a codebook by splitting the vocabulary into two disjoint sets based on the word frequency: the most frequent words and the rest. Less frequent words are represented using a sparse linear combination of the vectors of more frequent words. Instead of using an explicit set of words, (Shu and Nakayama, 2017) designed the codebooks in a more data-driven fashion where the selection of pseudo words and code vectors are learned automatically using the Gumbel-Softmax trick (Jang et al., 2016). Following the same approach, (Chen et al., 2018b) propose additional training objective to integrate the learning of discrete codes with the training of the language model. Our method, based on the multiplicative composition rather than addition, follows this research line. The effectiveness of multiplicative composition is verified in some language modeling tasks (Mitchell and Lapata, 2009, 2008).

3 Proposed Method

3.1 Multi-way Multiplicative Codes

Compositional models start with defining a set of d_m dimensional vectors, which serve as basic codes to compose the targeted embedding matrix. These vectors could further be separated into M groups and say each group contains K vectors. We call each group a codebook, and each d_m dimensional vector in the codebook a codeword.

An 1-way codebook then is defined as a $R^{1\times M\times K\times d_m}$ tensor. Correspondingly, we define N-way codebooks as $U \in R^{N\times M\times K\times d_m}$, where each of the N ways consists of a set of M codebooks, each codebook contains K words, and each codeword is associated with a d_m dimensional vector representing its semantics.

Since U is a high-order tensor which is not memory-friendly, we model the composition of U as applying a customized multiplicative operator on two tensors $C \in R^{M\times K\times d_m}$ and $S \in R^{N\times M\times d_m}$ as

$$U = C \odot S,$$

where $\odot$ is a multiplicative operator defining that

$$U_{i,j,k} = C_{j,k} \circ \sigma(S_{i,j}),$$
$$\forall i \in [1,N], j \in [1,M], \text{and } k \in [1,K],$$

$\sigma(\cdot)$ stands for the tangent hyperbolic function, and $\circ$ is the Hadamard product (entry-wise product). Tensor C is referred to as the base codebook, consisting of M codebooks where each codebook contains K codeword vectors of d_m dimensions. S is called rescaling codebooks. Each codeword in the base codebook is injected with new meanings by a code vector in the rescaling codebook.

Despite rescaling codebook uses much less memory (O(NMd_m)) than simply creating a N times larger set of vectors (O($NMKd_m$)), using rescaling codebook still introduces additional costs of the memory. We propose to further reduce the cost by allowing rescaling code vectors to be shared. We replace S with a $\tilde{S} \in R^{N\times d_m}$. That is, for each of the N way codebook, we use only one d_m-dimensional vector to rescale the base codebook C. The number of parameters in S can thus be reduced and the computation of U becomes

$$U_{i,j,k} = C_{j,k} \circ \sigma(\tilde{S}_i),$$
$$\forall i \in [1,N], j \in [1,M], \text{and } k \in [1,K].$$

Now, with the N-way codebook defined by C and $\tilde{S}$, given an embedding matrix E with V vocabularies (i.e. $E \in R^{V\times d_m}$), we could represent E by finding an encoding $Q \in R^{V\times N\times M}$ to compose E from C and $\tilde{S}$. Q_i contains encoding of corresponding vocabulary V_i. From each of N-way n, and each of M codebooks m, $Q_{i,n,m}$ indicates which codeword to compose. Let the word vector for the ith word be e_i. We could construct e_i by

$$e_i \approx \hat{e}_i = \sum_{n=1}^{N}\sum_{m=1}^{M} U_{n,m,Q_{i,n,m}}$$
$$= \sum_{n=1}^{N}\sum_{m=1}^{M} C_{m,Q_{i,n,m}} \odot \sigma(\tilde{S}_n).$$

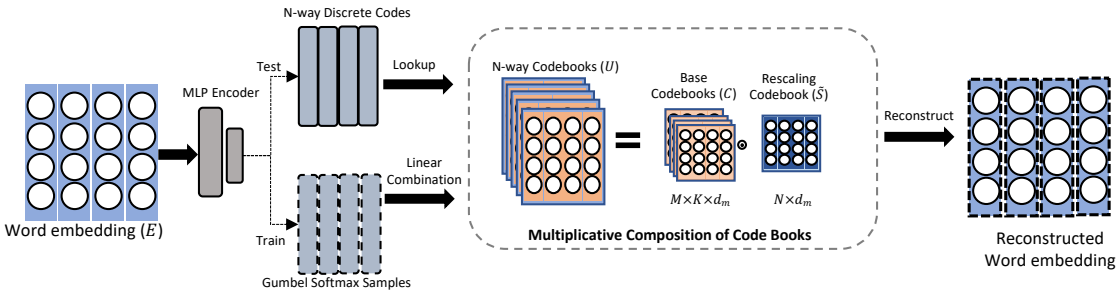

Figure 1: Overall architecture of the multi-way multiplicative compressor

The N-way discrete code could be learned in an end-to-end manner by using the Gumbel-Softmax trick (Jang et al., 2016). We first compute an encoding vector for the original word vector e_i by feeding it to a neural network

$$a_i = Softmax(\sigma(W^\top \phi(W'^\top e_i + b) + b')),$$

where W, W', b and b' are the parameters of the network, and ϕ is the softplus function. a_i represents a real value tensor in a shape $N \times M \times K$, which is then fed to the Gumbel-Softmax to generate a continuous approximation of drawing discrete samples with respect to the last dimension

$$\hat{D}_i = Softmax((\log a_i + g)/\tau),$$

where g is a random noise vector sampled from Gumbel distribution, and τ is the Softmax temperature controlling how close is the sample vector to a uniform distribution. $\hat{D}_i$ is an approximately one-hot out of K drawing for each way N and codebook M. We can then compute approximation of each dimension of e_i as

$$\hat{e}_{i,d} = U_{:,:,:,d} \odot \hat{D}_i$$

Note that during training $\hat{D}_i$ is generated as a continuous approximation of the N-way discrete code. At the testing phase, the fixed encoding Q_i of vector e_i is directly computed as $Q_{i,n,m} = \arg\max_k a_{n,m,k}$. More details on Gumbel-trick could be found in (Jang et al., 2016).

3.2 Group Adaptive Coding

Given the rescaling codebooks $\tilde{S}$ is small, the memory consumption mainly consists of two parts: code vectors C and discrete codes Q. On the one hand, the code vectors normally account for the major memory usage when dealing with relatively smaller vocabulary size. On the other hand, the size of discrete codes grows linearly with the size of vocabulary and the logarithm of K. To further reduce the memory usage, we propose to use codes of adaptive length and dimensions to deal with the linear dependency with vocabulary size. The intuition is to encode frequent words with a longer code length to achieve a relative lower reconstruction loss while representing rare words with fewer codes. To achieve this, we first sort the words according to their frequency and then split words into fixed number of groups G. The ith group could access only $\gamma_{G_i} = M \times (1 - \frac{i-1}{|G|})$ codebooks.

In the same time, we store the low-rank version of code vectors for some codebooks that were mainly used for representing rare words.

$$C_{i,j} = W_{G_i} c_i,$$

where $c_i \in R^{d_{G_i}}$ and $W_{G_i} \in R^{d_{G_i} \times d_m}$ is the linear transformation matrix to be shared by all the codebooks in the ith group. We resolve d_{G_i} in an intuitive way as shown in Algorithm 1.

In practice, high frequency words tend to get accesses to codebooks with higher rank while fewer frequent words can only access codebooks of lower dimension. Thanks to the long tail of rare words, this could actually help save considerable memory space. Normally, using a low compression rate results in $d_{G_i} \ll \gamma_{G_i} \times K$ so that it is guaranteed to reduce the number of parameters in the base codebook.

3.3 Prior-weighted Reconstruction Loss

According to Zipf's law, the frequency of words conforms to a power law distribution. It means that word which falls to the long tail may rarely appear in the sentence. It motivates the modification on the learning objective so that the compressor will be more focused on words with high frequency. In this paper, we use the distribution of

Sort set G according to frequency;
d': the minimum dimension ;
δ : targeted compression rate;
f_{G_i} : frequency of group G_i;
o_{G_i} : bits consumed by discrete code matrix Q for each word in G_i;
$n \leftarrow \delta \times V \times d_m \times 32(bits)$;
for $i \leftarrow 1$ *to* $|G|$ **do**
  compute a ratio based on frequency $\Delta_{G_i} \leftarrow \frac{f_{G_i}}{\sum_{i=1}^{|G|} f_{G_i}}$;
  calculate the dimension;
  $d_{G_i} \leftarrow (n - o_{G_i}) \times \Delta_{G_i} / (\gamma_{G_i} + d_m)$;
  ensure dimension is valid within (d', d_m);
  $d_{G_i} \leftarrow \min(\max(d_{G_i}, d'), d_m)$;
  update n by subtracting used bits;
  $n \leftarrow n - o_{G_i} - (\gamma_{G_i} - \gamma_{G_{i+1}} + d_m) \times d_{G_i}$
end

Algorithm 1: Algorithm to resolve the dimension of adaptive codebooks towards achieving a targeted compression rate.

the words in the training set as a prior knowledge to guide the learning of the compressor. In addition, similar to (Lin et al., 2017) we also want to let each of the N way encoding focus on different aspects of the original word embedding. The proposed training objective function then becomes

$$\mathcal{L} = \sum_{i=1}^{V} (-\log \hat{p}_i \|\hat{e}_i - e_i\|_2^2 + \lambda \|\hat{v}_i \hat{v}_i^\top - I\|_2^2 + \lambda \|\hat{v}_i^\top \hat{v}_i - I\|_2^2),$$

where $\hat{p}_i$ represents the empirical distribution of word V_i in the training set, e_i is original word vector, $\hat{e}_i$ is the reconstructed vector and $\hat{v}_i \in R^{N \times d_m}$ is a compilation of reconstrcuted vectors from all N-way codebooks in a matrix form (i.e. $\hat{v}_{i,n} = \sum_{m=1}^{M} C_{m,Q_{i,n,m}} \odot \sigma(\tilde{S}_n)$ is the reconstructed vector from nth-way codebook). The new objective function thus focuses on high-frequency words and in the meanwhile allows the N-way coding to encode different information.

4 Experiment

4.1 Data Sets and Models

Following the experimental protocol of (Chen et al., 2018a), we evaluate our proposed method with two important NLP tasks: language modeling and machine translation. Table 1 summarizes the key characteristics of the four data sets

Table 1: Statistics of data sets. The number in parenthesis shows the ratio of embedding layers (input plus Softmax) respective to the entire model size

Models	Vocab. Size	Dimension	Model Size
PTB-small	10k	200	17.7MB(85.8%)
PTB-large	10k	1500	251MB(45.4%)
OBW	793k	1024	6.8GB(91.2%)
NMT: DE-EN	30k	500	195MB(83.1%)

and models. PTB-small is using 2-layer LSTM-based language model built on Penn Tree Bank (PTB) data set. The vocabulary size is 10k. Input embedding layer and output Softmax layer are both set to 200 dimensions. PTB-large is trained on the same PTB data set as PTB-small except the dimensions for input and output are increased to 1500. Neural machine translation (NMT: DE-EN) is a Seq2seq model initialized using Pytorch checkpoints provided by Open-NMT (Klein et al., 2017). The model is to perform German to English translation tasks on IWSLT-14 (Cettolo et al., 2014) data set. One billion words (OBW) uses a 2-layer LSTM (referred to as "2-LAYER LSTM-8192-1024" by (Jozefowicz et al., 2016)) trained on the OBW data set and the vocabulary size is 793,471. For LSTM-based language models, the input embedding matrix and Softmax embedding matrix account for the major memory usage (up to 91.2%). Therefore, we target compressing both the input and Softmax embedding matrices.

4.2 Implementation Details

We compress both input embedding and Softmax matrices. We trained MulCode by using Adam optimizer with learning rate 0.001. For PTB data set, we group the vocabulary using 3 groups and 8 groups for OBW. To resolve the dimension of codebooks for adaptive coding, we use targeted compression rate $\delta = 0.2$ for PTB-small and $\delta = 0.05$ for the rest three models[1].

After approximation, we retrain the rest of parameters by SGD optimizer with initial learning rate 0.01. Whenever the validation perplexity does not drop down, we decrease the learning rate to an order smaller. We did not include results of fine-tuning on OBW for that the re-training process takes too long (few days) which is not compliant with our motivation to compress the given pre-trained embeddings.

[1]The original language models can be downloaded from https://github.com/mayktian/MulCode.git

The compression rate and corresponding performance could certainly be plotted as a spectrum graph. The more we compress, the larger the performance drop. In this paper, as far as BLEU score is concerned, we report results of compressed models when the BLEU falls within 3 percent difference from the original score. For PTB data set, we target 3 percent drop of perplexity (PPL) after retrain. For OBW data set, since it has a larger vocabulary size, we report results within 10 percent difference from the PPL achieved by the uncompressed model. For each method we tested various parameters and report the smallest model size of the compression fulfilling above criteria.

Notice that some previous methods compress model directly during training phase (Khrulkov et al., 2019; Wen et al., 2017). In contrast, our problem setup follows (Chen et al., 2018a; Shu and Nakayama, 2017; Chen et al., 2018b) that given a pre-trained model, we want to compress the model with limited fine-tuning.

4.3 Comparison with Baseline Models

We refer to our proposed method as MulCode (Mul stands for both multi-way and multiplicative composition). We mainly compare with two state-of-the-art baseline compressors targeting compressing the embedding layer.

1. **GroupReduce** We refer to the results reported in (Chen et al., 2018a).

2. **DeepCode** The additive composition model by (Shu and Nakayama, 2017). We use the pytorch code[2] released by (Shu and Nakayama, 2017) to produce the results.

Table 2 summarizes the comparison between the proposed methods and state-of-the-art baselines for the four benchmark data sets and LSTM models. MulCode manages to compress the input embedding layer and Softmax embedding layer 6 to 18 times without suffering a significant loss in the performance.

In comparison, all the baseline models achieve much lower compression rate with PTB-small which has only 200 dimensions. It is reasonable since embedding layers of PTB-small contains less redundant information and thus can be hardly compressed. As compared with DeepCode, our method achieves much higher compression rate[3] for all the four models. Our method also consistently and significantly outperforms GroupReduce.

4.4 Comparison with Quantization

Quantization has been proven to be a strong baseline. In fact, the discrete coding of MulCode can be considered equivalent to a trainable quantization. On the other hand, we need to point out that quantization is not orthogonal to MulCode. MulCode could be combined with quantization to achieve better performance. Specifically, the $M \times K$ base codebooks as well as the rescaling codebook could be quantized to further reduce the memory usage. We summarize the results in Table 3. Quantized MulCode could achieve more than 30.8 times compression for both input embedding and Softmax matrix in OBW. In addition, on machine translation task, it achieves 41.38X with BLEU score drops around only 1% after retraining. In particular, we observe that the effect of retraining is more prominent for MulCode and simple quantization compared to GroupReduce. This implies that local precision lost for low-rank basis in GroupReduce is more difficult to be recovered. In contrast, the collective information of MulCode due to the compositional property is more robust when imprecise local vectors present.

4.5 Ablation Analysis

We summarize the ablation analysis in Table 4. The major technical features of MulCode are rescaling codebooks (RC), prior-weighted reconstruction loss (PRL), and the group adaptive coding (GAC). Besides, we also consider using full rescaling books (Full) in place of the one shared across M codebooks. We remove or add these features one at a time. First, the rescaling codebook is shown contributing largely to the performance. On the other hand, since removing the rescaling codebook is equivalent to degrading to using pure additive composition, it also suggests the multiplicative composition is a better alternative to the additive composition. In the meantime, removing the rescaling codebook is equivalent to repeatedly selecting from the base codebook. It may force the codeword vector to encode the original meaning of

[2] https://github.com/zomux/neuralcompressor

[3] Aligned with what has been mentioned by (Shu and Nakayama, 2017) in their rebuttal, we found the DeepCode unable to work with the Softmax layer of OBW no matter how hard we have tuned it (99.5 as shown in Table 2).

Table 2: Compressed model evaluation with 3 language models and 1 machine translation model. Memory usage is reported as compared with the original input embedding and Softmax embedding. For example, 30x means the compressed embeddings use only 1/30 the memory space of the original embedding.

Model	Metric	Original	GroupReduce	DeepCode	MulCode
PTB-small	Memory	1X	4X	3.9X	6.5X
	PPL(before retrain)	112.28	115.38	120.05	115.38
	PPL(after retrain)	-	113.81	115.57	113.34
PTB-large	Memory	1X	8X	3.8X	17.1X
	PPL(before retrain)	78.32	84.79	84.73	83.85
	PPL(after retrain)	-	79.83	81.80	79.89
NMT: DE-EN	Memory	1X	8X	5.7X	17X
	BLEU(before retrain)	30.33	29.31	26.58	29.48
	BLEU(after retrain)	-	29.96	29.37	30.08
OBW	Memory	1X	6.6X	5.7X	18.2X
	PPL	31.04	32.47	99.59	32.66

Table 3: Results of compressing all input and Softmax embedding layers on three data sets. Experiments compare proposed Quantized MulCode with traditional Quantization and Quantized GroupReduce. 20x means approximated embedding uses 20 times smaller memory compared to original input embedding layer and Softmax layer..

Model	Metric	Original	Quantization	Quantized GroupReduce	Quantized MulCode
PTB-small	Memory	1X	6.4X	16X	16.3X
	PPL(before retrain)	112.28	115.81	116.54	116.33
	PPL(after retrain)	-	114.14	114.39	113.34
PTB-large	Memory	1X	6.4X	20X	28.2X
	PPL(before retrain)	78.32	81.69	81.53	81.55
	PPL(after retrain)	-	79.22	78.61	78.91
NMT: ED-EN	Memory	1X	6.4X	32X	41.38X
	BLEU(before retrain)	30.33	27.41	29.33	29.32
	BLEU(after retrain)	-	30.19	29.65	29.94
OBW	Memory	1X	6.4X	26X	30.8X
	PPL(before retrain)	31.04	32.63	34.43	34.36

words as a mix of different aspects of the semantics, which may turn out hard to be recovered. In fact, we find it difficult to train a model that can achieve low perplexity loss (33) on OBW with the rescaling codebooks removed.

Removing the prior-weighted loss from the model results in only marginal loss in the performance except for PTB-small. It indicates that, given the limited capacity of codebooks, the performance of compressed model could benefit more from lower reconstruction loss for those words with the highest frequencies. For example, MulCode achieves 3.5% less perplexity loss compared to the model without using a prior-weighted reconstruction loss. Group adaptive coding is another key factor to reduce memory usage. It shows that it works well with all the four models. It handles the memory reduction in both ways: reducing the parameters in the code vectors and reducing the number of discrete codes to represent a word. For model with a very large vocabulary, it seems to achieve higher memory reduction than with a smaller model. We conjecture it is largely due to the prominent presence of rare words in data set like OBW. The significance of PRL and GAC verifies the importance of frequency in compressing natural words. Lastly, using the full tensor for rescaling codebook does not seem to be necessary. It fails to produce any improved performance of the compressed model at the expense of a non-trivial compression rate drop.

4.6 Selection of M,N and K

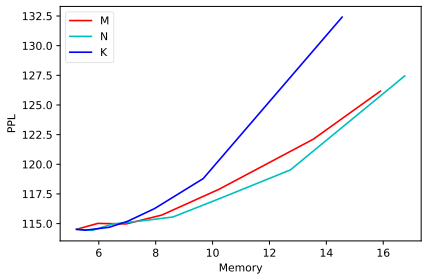

Figure 2: Influence of M, N, K for PTB-small.

To understand how the choices of M, N, K would affect the performance of the compressed model, we test the proposed model for PTB-small

Table 4: Ablation analysis of MulCode. We test the compression model by removing (or adding) one feature at a time. **-RC** stands for removing the rescaling codebook;**-PRL** stands for removing the prior-weighted reconstruction loss; **-GAC** means removing the group adaptive coding; **+ Full** means to use the untied tensor for rescaling codebook.

Model	Metric	-RC	-PRL	-GAC	+ Full	MulCode
PTB-small	Memory	3.95X	6.47X	5.68X	5.50X	6.47X
	PPL	124.78	115.58	115.65	115.96	115.38
PTB-large	Memory	5.64X	15.42X	13.08X	10.20X	17.1X
	PPL	84.72	84.82	84.60	84.36	83.85
NMT: DE-EN	Memory	9.02X	17.4X	11.24X	16X	17.44X
	BLEU	26.16	28.12	28.60	29.02	29.11
OBW	Memory	9.4X	23.8X	16.8X	23.4X	23.8X
	PPL	37.42	34.05	33.45	32.81	32.95

with varying setting of M, N, and K. The departure point of this experiment is using $M = 32, K = 32, N = 8$. We then adjust the three parameters one at a time while fixing the other two. In order to plot the results in a single figure, we set the x-axis as the memory usage instead of different values of the three parameters. As shown in Figure 2, adjusting the values of M, N, and K have similar effects on the PPL when the compression rate is low ($\leq$7X). With a larger compression rate, the performance becomes much more sensitive to the change of K. It suggests that it is safer to maintain a high value for K while tuning the rest two parameters for the purpose of securing a reasonable performance of the compressed model.

4.7 Understanding Composed Codes

At the core of our approach is the multiplicative composition from two sets of codebooks. Hence, it is interesting to investigate what has been encoded in the N-way codebooks. One assumption is that each code in the base codebooks encodes a mix of information which can be disentangled by the rescaling codebook. We encode all the words of OBW corpus by a $32 \times 8 \times 4$ codes. We compute the hamming distance of example query words (shown in Table 5) for each of the N-way codes. We select the top ones with the smallest hamming distance from the 10,000 most frequent words that are likely to have low reconstruction errors.

Since the N-way codings are generated by selecting from the base codebooks and modified by rescaling codebooks, each channel can be seen as meaningful subspace. It shows that each of the N-way codings might have encoded a different subspace of the original meaning of words, including tenses (e.g., halt v.s. halted), plurals (e.g., bank v.s. banks), synonyms (e.g., soccer v.s. football),

Word	1	2	3	4
tomorrow	today	Sunday	prompt	Tuesday
beautiful	elegant	iconic	great	fields
soccer	football	hockey	Ghana	basketball
where	when	experiencing	who	learn
bank	company	police	companies	banks
halt	stop	halted	afternoon	cover
like	just	Like	such	is

Table 5: Most similar word computed using Hamming distance for each of the N-way codings.

co-occurrence (like v.s. just), topical relatedness (soccer v.s. hockey). It verifies that the multiplicative composition used in our approach is able to introduce new information to the base codebook.

5 Conclusion

In this paper, we propose a novel compression method for neural language models. Our method applies multiplicative factors on end-to-end learned codebooks. Our method also considers the frequency information in the corpus by adding weighted loss according to the importance. At the same time, the coding scheme is made adaptive based on the frequency information. Experimental results show that our method outperforms the state-of-the-art compression methods by a large margin. In particular, on the IWSLT-14 data set, our method combined with quantization achieves 41.38 times compression rate for both the embedding and Softmax matrices. It will facilitate deployment of large neural language models on memory-constrained devices.

6 Acknowledgements

Pei-Hung Chen and Cho-Jui Hsieh acknowledge the support from NSF via IIS1719097, Intel and Google Cloud.

References

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

Patrick Chen, Si Si, Yang Li, Ciprian Chelba, and Cho-Jui Hsieh. 2018a. Groupreduce: Block-wise low-rank approximation for neural language model shrinking. In *Advances in Neural Information Processing Systems 31*, pages 10988–10998. Curran Associates, Inc.

Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018b. Learning k-way d-dimensional discrete codes for compact embedding representations. In *International Conference on Machine Learning*, pages 853–862.

Yunchuan Chen, Lili Mou, Yan Xu, Ge Li, and Zhi Jin. 2016. Compressing neural language models by sparse word representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 226–235.

Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. 2018. Universal deep neural network compression. *CoRR*, abs/1802.02271.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277.

Peter W Foltz, Walter Kintsch, and Thomas K Landauer. 1998. The measurement of textual coherence with latent semantic analysis. *Discourse processes*, 25(2-3):285–307.

Alex Gittens, Dimitris Achlioptas, and Michael W Mahoney. 2017. Skip-gram- zipf+ uniform= vector additivity. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 69–76.

Song Han, Huizi Mao, and William J. Dally. 2015a. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149.

Song Han, Jeff Pool, John Tran, and William J. Dally. 2015b. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. 2014. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan Oseledets. 2019. Tensorized embedding layers for efficient model compression. *arXiv preprint arXiv:1901.10787*.

Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. 2016. Compression of deep convolutional neural networks for fast and low power mobile applications. In *ICLR*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada. Association for Computational Linguistics.

Yann LeCun, John S Denker, and Sara A Solla. 1990. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.

Ekaterina Lobacheva, Nadezhda Chirkova, and Dmitry Vetrov. 2017. Bayesian sparsification of recurrent neural networks. *arXiv preprint arXiv:1708.00077*.

Roland Memisevic. 2011. Learning to relate images: Mapping units, complex cells and simultaneous eigenspaces. *arXiv preprint arXiv:1110.0107*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. *proceedings of ACL-08: HLT*, pages 236–244.

Jeff Mitchell and Mirella Lapata. 2009. Language models based on semantic composition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 430–439. Association for Computational Linguistics.

Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. 2015. Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450.

Steven Pinker. 2003. *The language instinct: How the mind creates language*. Penguin UK.

Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659. IEEE.

Raphael Shu and Hideki Nakayama. 2017. Compressing word embeddings via deep compositional code learning. *arXiv preprint arXiv:1711.01068*.

Wei Wen, Yuxiong He, Samyam Rajbhandari, Wenhan Wang, Fang Liu, Bin Hu, Yiran Chen, and Hai Li. 2017. Learning intrinsic sparse structures within long short-term memory. *CoRR*, abs/1709.05027.

Yuhui Xu, Yongzhuang Wang, Aojun Zhou, Weiyao Lin, and Hongkai Xiong. 2018. Deep neural network compression with single and multiple level quantization. *CoRR*.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. 2017. On compressing deep models by low rank and sparse decomposition. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 67–76.