WEST: WORD ENCODED SEQUENCE TRANSDUCERS

Ehsan Variani[*]*, Ananda Theertha Suresh*[*]*, Mitchel Weintraub*

Google Inc., USA

{variani, theertha, mweintraub} @google.com

ABSTRACT

Most of the parameters in large vocabulary models are used in embedding layer to map categorical features to vectors and in softmax layer for classification weights. This is a bottleneck in memory constraint on-device training applications like federated learning and on-device inference applications like automatic speech recognition (ASR). One way of compressing the embedding and softmax layers is to substitute larger units such as words with smaller sub-units such as characters. However, often the sub-unit models perform poorly compared to the larger unit models. We propose WEST, an algorithm for encoding categorical features and output classes with a sequence of random or domain dependent sub-units and demonstrate that this transduction can lead to significant compression without compromising performance.

1. INTRODUCTION

A standard way of representing categorical features or classes as continuous features is to use embedding look-ups e.g., word-to-vector representation [1]. The corresponding matrix which contains one embedding per row is called embedding matrix and its size scales with the number of categorical features (or output classes) and the embedding dimension. For the input layer (or *embedding* layer), the embedding matrix stores the representation for categorical features while for the output layer (or *softmax* layer), it stores the classification weights corresponding to each output class.

For large vocabulary tasks, embedding layers might not even fit within the memory capacity of a single accelerated computation unit like Graphical Processing Unit (GPU) [2, 3]. The situation is even worse when using these models for on-device training such as federated learning [4, 5] or on-device inference such as ASR on the phone [6] where there is also communication bandwidth constraint between server, device and CPU, accelerator respectively. Furthermore, these models inherently suffer from unbalanced topology since many core layers like recurrent layers are allocated a small percentage of the total number of parameters; for Penn TreeBank (PTB) [7, 8] or YouTube language model [2], the embedding and softmax layers contain nine times more parameters than the recurrent layers. Finally, due to data sparsity, there might not be enough training examples per embedding parameter, particularly for the infrequent features. The techniques to address the above challenges fall under a wide category of the *compression* algorithms.

The compression algorithms are either *multi-stage* or *single-stage*. In multi-stage compression techniques, a larger model is first trained and then embedding or softmax layers are compressed via some compression algorithm. This includes scalar quantization [9, 10, 11, 12, 13], vector quantization [14], product quantization [15], combination of vector and product quantization [2], Huffman coding [16], and low rank approximation [17]. These techniques do not optimize the same objective function as the neural network training objective function, thus often require another stage of retraining of the uncompressed model parameters. Alternatively, single-stage approaches impose a structure on the parameter space apriori of training and directly optimize the smaller model. The most common single stage compression technique for embedding layer is to use hashing [18], where the input vocabulary is hashed to a smaller vocabulary. The hashing mechanism is not one-to-one and is lossy. Another common single-stage compression technique that also addresses data sparsity is to use sub-word units like word-pieces or characters. While these methods naturally lead to high compression rate, there is still a gap between their performance with that of larger unit models. This paper tries to shed light on the modeling performance gap between using smaller and larger units.

2. WEST: WORD ENCODED SEQUENCE TRANSDUCERS

We hypothesize that the reason for the performance difference between larger unit and smaller sub-unit models is the way smaller unit models are structured. Consider the example of language models in Figure 1. The main difference between word model and character model is the cycle that these models operate on. In the word model, input word is fed to predict the next word with word level cycle. The sub-unit models perform on sub-unit cycle which is smaller than word. Hence, the recurrent units need to remember longer sub-unit contexts. Furthermore, since these models are normalized on sub-units, they assign probabilities to character sequences that are not in the vocabulary. We propose Word Encoded Sequence Transducers (WEST) model which performs over larger units which are internally represented via smaller sub-units, which can be language dependent (like characters) or random. Next, we formally define WEST model, its compoenents and finally its probabilistic interpretation.

[*]equal contributions.

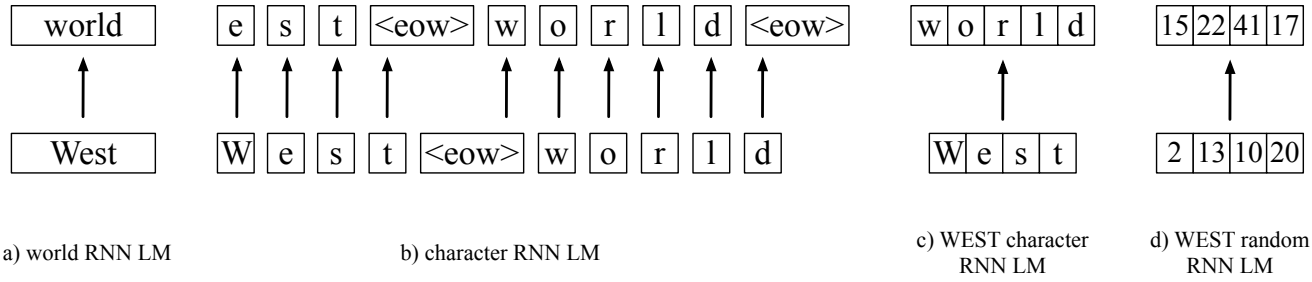


Fig. 1. Different recurrent neural network language models (RNN LM) of sentence 'West world'. (a) In the word model, the input word is fed to predict the next word. (b) In character model, the input and output operate on character level. (c, d) In WEST, while the word is represented by smaller sub-units (language dependent or random), the model still operates on larger units i.e., word in this example.

2.1. Definition

WEST is a single-stage compression technique that compresses the embedding and softmax layers from the beginning of the training by factoring them as a product of a structured sparse matrix and a structured dense matrix. Let V be the vocabulary size i.e., the number of categorical features or number of output classes and d be the embedding dimension. Then embedding and softmax matrices are of size $V \times d$. Let E denote softmax or embedding matrix. We propose to factorize E as

$$E = C \times E^c,$$

where C is a sparse structured matrix of size $V \times n \cdot k$ such that each row of C is concatenation of n weighted one hot vectors of length k and E^c is a structured dense matrix of size $n \cdot k \times d$.

2.2. Structured sparse matrix

Let $c : [1, V] \rightarrow \cup_{i \leq n}[1, k]^i$ be a mapping from the vocabulary to the set of sequences of length at most n, where each entry ranges from 1 to k. We refer to n as the *code length* and k as the *alphabet size*. Let $c_i(w)$ be the i$^{\text{th}}$ entry of the code for word w [1] . Given such a codebook c, we construct a sparse matrix C of size $V \times n \cdot k$ as follows. For $i \leq n$ and $j \leq k$,

$$C_{w,(i-1)\cdot k+j} \neq 0 \text{ if and only if } c_i(w) = j.$$

Furthermore, we define $\lambda_{w,i}$ to be the weight corresponding to the entry corresponding to $c_i(w)$. We differentiate between two types of sparse code books, the *weighted* sparse matrix where the non-zero entries can take any value and the *unweighted* sparse matrix where the non-zero entries are restricted to be one, i.e., $\lambda_{w,i} = 1 \ \forall w, \forall i$. To store the codebook (the sparse matrix entries indices), $V \cdot n \cdot \lceil \log_2 k \rceil$ bits are needed. This can also be prohibitive in many applications. Hence, we propose to use sparse codebooks that can be stored succinctly with fewer than $V \cdot n$ additional parameters e.g., language codes and random codes.

Language coding: Let F be a collection of sub-units such as characters or word-pieces [19] and let $w = w_1, w_2, \ldots w_{n'}$ where each $w_i \in F$ and $n' \leq n$. If $F(w_i)$ be the index of w_i in F, then

$$c(w) = F(w_1), F(w_2), \ldots, F(w_n).$$

where $F(j >= n') = |F|$ is the padding symbol. For example, if the set of words are {i, it, he, she, you, they}, and sub-units are {I, t, he, s, you, y}, then c(she) is $(4, 3)$. The advantage of the language codebook is that the word-to-character mapping is typically stored along with the network and hence does not require any additional space.

Random coding: Let Rand (k, n) be a random map where each word is mapped to a random sequence of length n with integeres less than alphabet size k such that no two words have the same code. The resulting codebook is uniquely decodable by design. Furthermore, for any two words w_1, w_2, the probability of collision at any index is :Prob $(c_i(w_1) = c_i(w_2)) = 1/k$, for $i \leq n$. Note that for alphabet size k, this is the minimum possibility of collision for any algorithm. The advantage of the random codebook is that it can be generated on-the-fly, given a seed. Hence, storing random codebook needs just V parameters.

2.3. Structured dense matrix

The structured dense matrices has been investigated in the literature with the aim of reducing computational complexity of matrix-vector multiplication and memory consumption [20]. Here we only consider two types of such matrices, block-diagonal and band matrices.

Block-diagonal structure [21]: Let E^is be the *sub-unit embedding matrices* of size $k \times d/n$, then E^c is a block-diagonal matrix with E^is as the diagonal entries i.e.,

$$E^c_{(i-1)\cdot k+i',(i-1)\cdot d/n+j'} = E^i_{i',j'},$$

for $i \leq n$, $i' \leq k$ and $j' \leq d/n$, else 0. If $c(w) = c_1(w), \ldots, c_n(w)$ is the code for w and $\lambda_{w,i}$ is the corresponding weights in the structured sparse matrix, then the embedding for word w with block diagonal matrix is

$$E_w = \left[\lambda_{w,1} E^1_{c_1(w)}; ...; \lambda_{w,n} E^n_{c_n(w)}\right], \tag{1}$$

which is weighted concatenation of the rows of the sub-unit embedding matrices corresponding to the code $c(w)$. Note

[1] In the rest of the paper, we use "word" for categorical features and output classes as our main application is language models.

that the concatenation using unweighted sparse matrix does not require any extra computation.

Band structure: Let E^is be matrices of size $k \times d$, then E^c is the matrix obtained by stacking entries of E^is one below another i.e.,

$$E^c_{(i-1)\cdot k+i',j} = E^i_{i',j}.$$

Under the band structure, the embedding can be computed as

$$E_w = \lambda_{w,1} E^1_{c_1(w)} + ... + \lambda_{w,n} E^n_{c_n(w)}, \quad (2)$$

which is the weighted sum of rows corresponding to the code $c(w)$. Note that to store the structured dense matrix E^c it is sufficient to store the sub-unit embedding matrices E^is and hence the space used in the block-diagonal structure is $k \times d$ and the space used by the band structure is $k \times d \times n$. If the parameters are tied i.e., all E^is are equal, then the number of parameters reduces by a factor of n.

2.4. WEST interpretation

In WEST, the input and output operates on word level while the internal representations function on sub-unit levels. For example, the input embedding of each word can be obtained either by concatenation (1) or sum of sub-unit embeddings (2). Further, WEST for softmax layer can be interpreted as MaxEnt model over sub-unit weights. If h is the penultimate layer activation and E^c is a band structured matrix, the logit for class w is

$$l_w = E_w \cdot h = \sum_{i=1}^{n} \lambda_{w,i} E^i_{c_i(w)} \cdot h,$$

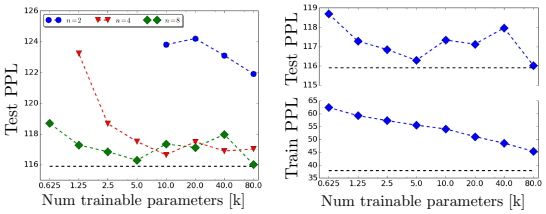
and the posterior probability is

$$\begin{aligned} P(w|h) &= \frac{\exp(l_w)}{\sum_{w'} \exp(l_{w'})} \\ &= \frac{1}{\mathbb{Z}(h)} \exp\left(\sum_{i=1}^{n} \lambda_{w,i} E^i_{c_i(w)} \cdot h\right), \end{aligned} \quad (3)$$

where $\mathbb{Z}(h)$ is the normalization factor. We speculate this property distinguishes WEST from other sub-unit models (like character or word-piece) in the way it models the probability of next word given the history. The character model predicts the probability of next word `west`, given history ϕ via the following chain rule:

$$P(\texttt{w}|\phi) \cdot P(\texttt{e}|\phi,\texttt{w}) \cdot \cdots \cdot P(\texttt{<eow>}|\phi,\texttt{w},\texttt{e},\texttt{s},\texttt{t}),$$

where <eow> is the end of word symbol. Estimating the word probability with the above equation is difficult as the recurrent model needs to remember longer contexts of sub-units. Furthermore, since these models are normalized on sub-units, they assign probabilities to character sequences that are not in the vocabulary e.g., "`wese`". Similar to character or word-piece models, WEST uses sub-units, but the predictions are over larger units (e.g, words). This has two advantages: it has a smaller memory footprint as it uses sub-units internally and it has modelling advantage as it outputs normalized probabilities over larger units. WEST estimation of $P(\texttt{west}|h)$ via Eq. (3) is very similar to MaxEnt framework [22]. Similar to MaxEnt model the word level scores are derived by weighted sum of sub-unit features, however the features in WEST are trainble through all network components below the last layer.



(a) Compression for different code-lengths (b) Generalization ($n = 8$)

Fig. 2. WEST embedding performance on PTB with random codes. WEST models use an additional 10k non-trainable parameters to store the seeds for random codes.

3. EXPERIMENTS

Experiments are designed to evaluate different aspects of WEST compression in terms of *maximum achievable compression rate* for embedding and softmax, *effect of word-level normalization*, *choice of sub-units* and performance for an *embedded ASR task*.

Datasets. We use *Penn Treebank* (PTB) public dataset that [23] consists of 929k words in the training corpus and 82k words in the test corpus with 10k vocabulary size. The model from [8] is used as the baseline (see Table 1 in [8] for details). The *embedded task* presented here is a state-of-the-art ASR on the phone which uses a RNN LM for second pass rescoring. The vocabulary size is 64k and the model size is 15 MB on disk (Embedding: 4.1 MB, LSTMs: 8.45 MB and Softmax: 4.1 MB). The WER for this model is reported on a set of 14k anonymized, hand-transcribed voice search utterances extracted from live traffic [6].

Maximum compression rate: The performance of WEST embedding with random codes, unweighted sparse matrices, and tied block-diagonal structure for PTB is presented in Figure 2. The number of trainable parameters in the embedding matrix can be compressed up to 1000 times with perplexity (PPL) close to baseline (dashed horizontal line). At the same compression ratio, longer code length achieves better perplexity as larger n increases sampling space size k^n, and thus reduces the collision probability. Comparing the gap between the train and test perplexity for WEST models with baseline suggests that the baseline model is prone to over-fitting, while WEST models achieve similar test PPL with larger training PPL and hence generalize better, see Figure 2(b). For softmax, the maximum compression rate which achieves the same performance as baseline is about two (fourth row of Table 2).

Effect of word-level normalization: To examine the effect of word-level normalization (3), we design the following ex-

periment. For each word w, we first optimize the model that estimates $P(w|h)$ by estimating the character probability individually i.e., for the word `west`, we estimate probability as $P_1(\texttt{w}|\phi) \cdot P_2(\texttt{e}|\phi) \cdot P_3(\texttt{s}|\phi) \cdot P_4(\texttt{t}|\phi) \cdot P_5(\texttt{<eow>}|\phi)$, where $P_i(c|\phi)$ is the probability of i^{th} character being c with history ϕ. This corresponds to the band structure without normalization. The word level test PPL of this model is 366k (char-normalized in Table 1). However, if we train the model using WEST (3), the performance improves to 533. We further added word-level bias term and sparse weights $\lambda_{w,i}$. This increases the number of parameters only by 0.07M, but improves the perplexity to 149.72. By increasing the number of parameters of LSTM, the perplexity can be further improved to 114.59 while keeping the model size same as the baseline.

Model	Test PPL	# trainable params [M]		
		Emb	LSTM	Soft
Char-normalized	366k	2.0	0.5	0.20
Word-normalized	533.39	2.0	0.5	0.20
+ Word biases	238.06	2.0	0.5	0.21
+ Sparse weights	149.72	2.0	0.5	0.27
+ Wider lstm	114.59	2.0	2.24	0.27
Baseline	115.91	2.0	0.5	2.01

Table 1. Softmax with language codes.

Choice of sub-units. Table 2 presents the effect of various coding schemes. The first row is the performance of the character codebook with bias and weights as in Table 1. The second row uses random codes with alphabet size 49 (which is equal to number of unique characters in PTB) and code length of 12 which matches the total parameters of the character coding. The PPL for the random coding is better than the character model (row 2 and row 1 in Table 2). This shows that the model need not use the language structure and it can learn features with random encoding. Let Rand(k, n, t), denote the code where the most frequent t words are coded as themselves and rest of the words are assigned randomly. To be more concrete, let words be ordered in decreasing order for frequency. For $w \leq t$, let $c(w) = (k + w)$ and for $w > t$, the codes are randomly assigned using Rand(k, n). Note that under this construction the alphabets assigned for top t words are unique and are not shared with any other codes. As shown in row 3 and 4 in Table 2, the PPL gets better. Finally compressing both embedding and softmax layers and distributing the saved parameters to the LSTM, the PPL reduces to 92.36, which is the best test PPL for this model size for PTB to the best of our knowledge.

Large-scale embedded ASR. Table 3 presents performance of WEST compression on a second pass RNN LM of a large-scale on-device ASR task. The number of trainable parameters in the embedding layer is compressed 500 times using WEST with random codes, unweighted sparse matrices, and tied block diagonal structure. This doesn't change the WER and saves 4 MB. Storing the random codebook needs another 64k parameters. This gives a compression of 15.5 times for the embedding layer. For softmax, we use band structure and language codes where the sub-units include most frequent 16k words and characters. This results in a moderate softmax compression of 3.1 and the model size drops to 4.75 MB. Note that we do not need extra space for storing the language codebook as the information is already present in the symbol table. Finally, to demonstrate the flexibility of WEST to integrate with other compression techniques, we apply scalar quantization on top of already compressed WEST model to reduce the model size to 1.35 MB, resulting in a total of 11 times compression without any performance degradation. If the parameter savings from embedding and softmax is used in LSTMs while keeping the same size of the baseline model, WER improves by 3% relative (last row of Table 3). We remark that we obtained similar compression rates using random codes instead of language codes.

Sub-unit	Test PPL	# trainable params [M]		
		Emb	LSTM	Soft
Character	149.72	2.0	0.5	0.27
Rand(49, 12)	134.58	2.0	0.5	0.25
Rand(49, 12, 2000)	121.73	2.0	0.5	0.63
Rand(49, 12, 4000)	116.84	2.0	0.5	1.00
Rand(49, 12)	101.58	2.0	2.24	0.25
Rand(49, 12, 2000)	92.36	0.5	3.38	0.63
Baseline	115.91	2.0	0.5	2.01

Table 2. Performance of different sub-units, character, Rand(k, n) and Rand(k, n, t). The band structure was used for the softmax.

Structure	PPL	WER [%]		Size
	Test	VS	Dict	[MB]
Baseline	68.07	13.7	7.3	15
+ Embedding$(15.5X)$	70.1	13.7	7.3	11.25
+ Softmax$(3.1X)$	71.0	13.6	7.3	4.75
+ Quantization	71.0	13.7	7.3	1.35
Reallocation	56.3	13.3	7.1	15

Table 3. WEST for on-device second pass rescoring.

4. CONCLUSION

We proposed WEST, a single-stage compression technique for reducing the size of embedding and softmax layers. WEST bridges the gap between larger unit and smaller sub-unit models and provides a general framework that can be interpreted as MaxEnt model over sub-units. We reported significant compression rates for embedding and moderate rates for the softmax layer. Our experiments showed that the choice of sparse and dense matrices do not matter for the embedding layer, but they do matter for the softmax layer. In particular for softmax compression, variations of random codes performed slightly better than character codes and weighted sparse matrices performed significantly better than unweighted sparse matrices. Finally, we demonstrated that beside compression, while keeping overall model size fixed, reallocating saved parameters from WEST to other network components can result in even better performance.

5. REFERENCES

[1] T.Mikolov, K.Chen, G.Corrado, and J.Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[2] S.Kumar, M.Nirschl, D.Holtmann-Rice, H.Liao, A. T.Suresh, and F.Yu, "Lattice rescoring strategies for long short term memory language models in speech recognition," *arXiv preprint arXiv:1711.05448*, 2017.

[3] H.Soltau, H.Liao, and H.Sak, "Reducing the computational complexity for whole word models," in *Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2017.

[4] J.Konečnỳ, B.McMahan, and D.Ramage, "Federated optimization: Distributed optimization beyond the datacenter," *arXiv preprint arXiv:1511.03575*, 2015.

[5] J.Konečnỳ, H. B.McMahan, F. X.Yu, P.Richtárik, A. T.Suresh, and D.Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[6] I.McGraw, R.Prabhavalkar, R.Alvarez, et al., "Personalized speech recognition on mobile devices," in *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, 2016.

[7] T.Mikolov, M.Karafiát, L.Burget, J.Černockỳ, and S.Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[8] W.Zaremba, I.Sutskever, and O.Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[9] M.Courbariaux, Y.Bengio, and J.-P.David, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[10] S.Anwar, K.Hwang, and W.Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, 2015.

[11] K.Hwang and W.Sung, "Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, 2014.

[12] Y.Wu et al., "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[13] R.Alvarez, R.Prabhavalkar, and A.Bakhtin, "On the efficient representation and execution of deep acoustic models," *arXiv preprint arXiv:1607.04683*, 2016.

[14] Y.Gong, L.Liu, M.Yang, and L.Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv preprint arXiv:1412.6115*, 2014.

[15] A.Joulin, E.Grave, P.Bojanowski, M.Douze, H.Jégou, and T.Mikolov, "Fasttext. zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[16] S.Han, H.Mao, and W. J.Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[17] P. H.Chen, S.Si, Y.Li, C.Chelba, and C.-j.Hsieh, "Groupreduce: Block-wise low-rank approximation for neural language model shrinking," *arXiv preprint arXiv:1806.06950*, 2018.

[18] W.Chen, J.Wilson, S.Tyree, K.Weinberger, and Y.Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015.

[19] M.Schuster and K.Nakajima, "Japanese and korean voice search.," in *ICASSP*, 2012, pp. 5149–5152.

[20] V.Pan, "On computations with dense structured matrices," *Mathematics of Computation*, vol. 55, no. 191, pp. 179–190, 1990.

[21] Z.Li, R.Kulhanek, S.Wang, Y.Zhao, and S.Wu, "Slim embedding layers for recurrent neural language models," *arXiv preprint arXiv:1711.09873*, 2017.

[22] A. L.Berger, V. J. D.Pietra, and S. A. D.Pietra, "A maximum entropy approach to natural language processing," *Computational linguistics*, vol. 22, no. 1, pp. 39–71, 1996.

[23] M. P.Marcus, M. A.Marcinkiewicz, and B.Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.