PSVD: POST-TRAINING COMPRESSION OF LSTM-BASED RNN-T MODELS

Suwa Xu[1]*, Jinwon Lee*[1]*, Jim Steele*[1]

[1]Amazon Devices

ABSTRACT

RNN-T has received a lot of attention recently since it achieves state-of-art WER in automatic speech recognition. To run the RNN-T model in real-time on resource-limited edge devices, model compression is often required. However, typical compression methods are still challenging to apply to RNN-T. First, it takes a lot of fine-tuning time and computing resources (e.g., up to several weeks even with multiple GPUs) due to the large size of training speech dataset and slow training speed of LSTMs. Second, the ASR training pipeline and datasets are often proprietary and not publicly available. Thus it is hard to fine-tune the pre-trained model after compression.

In this paper, we propose PSVD (Post-training SVD) which could effectively improve the WER of a SVD compressed RNN-T model without requiring the large training dataset and costly back-propagation. Based on a small amount of test data, PSVD could quickly post-train a SVD compressed RNN-T model by leveraging a light-weight linear regression. In particular, we observed that multiple iterations of layer-sequential linear regression is effective in optimizing the compressed model. Compared to SVD compression without fine-tuning, PSVD could improve WER by up to 8.36% at a fraction of the compute time. To the best of our knowledge, our work is the first to leverage post-training compression for LSTM and RNN-T.

Index Terms— speech recognition, RNN-T, LSTM, SVD, Post-training model compression.

1. INTRODUCTION

As speech recognition technologies continue to improve, they are becoming increasingly essential for edge devices (e.g., Amazon Alexa, Google Assistant, Apple Siri); they are widely used to remotely interact with applications (e.g., music player, alarms) and control smart home devices (e.g., bulb, thermostat). With increasing demands on privacy and low-latency interaction from customers, speech recognition is recently processed directly on devices rather than in the cloud.

RNN-Transducer (RNN-T) has emerged as state-of-the-art neural networks for automatic speech recognition (ASR) tasks [1][2]. Since typical size of RNN-T is still large (e.g., 480MB with float32 parameters [1]), it is infeasible to run the RNN-T model in real-time on resource-limited edge devices. Some level of model compression is required to deploy the RNN-T on edge devices.

Singular Value Decomposition (SVD) is a known effective compression method for LSTM-based neural networks like RNN-T [3][4]. It factorizes a large weight matrix into two small weight matrices by truncating low-rank singular values. Since accuracy is quite degraded after SVD compression [4][5], full fine-tuning is often used to recover accuracy. Unfortunately, this is not simple for an RNN-T model for two reasons. First, RNN-T takes a lot of fine-tuning time and computing resources (e.g., up to several weeks even with multiple GPUs) due to the large size of training speech datasets (e.g., up to 100K hours [6][7]) and slow training speed of LSTMs (relatively slower acceleration than CNNs on GPU [8]). Second, commercial-grade ASR training pipeline and datasets are often proprietary and not publicly available (e.g., MLPerf only provides the pretrained RNN-T model [9]) and thus fine-tuning compressed models is infeasible in practice.

In this paper, we propose PSVD which post-trains SVD compressed RNN-T models to improve WER effectively. The key idea of PSVD is to optimize the weights of a SVD compressed model by minimizing the mean squared error (MSE) between uncompressed and compressed model outputs per layer. Then, the compressed model better mimics the original uncompressed model and thus recovers the accuracy. This process can be quickly done through a linear regression with a small set of test data. However, the recurrent and multi-layer nature of LSTM-based RNN-T diminishes the effectiveness of post training. PSVD addresses this challenge by devising *multi-iteration layer-sequential linear regression*, which applies linear regression multiple times from the first to the last layer in sequence. In this way, PSVD could improve WER up to 8.36% on Librispeech dataset [10] compared to SVD compression.

Our key contributions are summarized as follows.

- We propose PSVD which can post-train SVD compressed RNN-T model without training data and back propagation.
- Considering the recurrent and multi-layer characteristic of LSTM-based RNN-T, we develop multi-iteration layer-sequential linear regression method.

- We empirically validated that PSVD effectively improves the WER of SVD compressed RNN-T model with a small set of test data within a few hours.
- We extensively study diverse aspects of post-training compression on RNN-T model (e.g., SVD types like stacked vs. rollout, the number and order of iterations).

2. RELATED WORK

A good overview of diverse model compression methods is summarized in [11] including SVD [3][5][4], Tensor decomposition, Pruning [12][13] and Quantization [14][15]. In general, we can further classify these compression methods into three levels. The definition of each level depends on the amount of training data and computation complexity for compression.

- **Level 1: Data-free compression (DFC)** [3][5]. No data is required. Neither post-training nor fine-tuning is required after model compression with SVD or Pruning.
- **Level 2: Post-training compression (PTC)** [16][12][14]. A limited set of the test data are used to post-train compressed models and improve their accuracy. No back-propagation is required. Our PSVD is in this level.
- **Level 3: Compression-aware training (CAT)** [2][4][15][13]. A full set of the training data with ground-truth labels is used to fine-tune the compressed models. Full back-propagation and training pipeline are required.

Typically, CAT shows better accuracy than PTC which shows better accuracy than DFC. However, PTC has recently received a lot of attention since it could address the key limitations of CAT. First, PTC could be still applicable even when only the pre-trained model is available; training pipeline and datasets of commercial-grade ASR are often proprietary and not available in public and thus CAT cannot be used in such cases. Also, PTC can be quickly performed whereas CAT takes a lot of training time and compute resource since its complexity is comparable to that of the original model training procedure due to full back-propagation.

Recently, several PTC methods have been proposed. Asym3D [16] and Channel Pruning [12] leverage layer-wise non-linear optimization to quickly post-train the compressed CV models. The authors of [14] post-train quantized CV models through layer-wise bias correction and weight equalization to recover the accuracy. These works basically try to reconstruct original feature maps similar to knowledge distillation [17]. However, all previous PTC work is based on convolution neural networks (CNNs) in the context of computer vision. To the best of our knowledge, our PSVD work is the first to leverage post-training compression on LSTM and RNN-T. Different from stateless CNN, the stateful nature of LSTMs makes it hard to apply post-training compression. Our PSVD addresses this challenge through multi-iteration layer-sequential linear regression by fully considering the characteristics of LSTM and RNN-T networks.

3. SVD COMPRESSION OF RNN-T

RNN-T consists of an encoder, decoder (a.k.a. prediction network), and joint network. The encoder is served as an Acoustic Model (AM) while decoder is served as a Language Model (LM). Both encoder and decoder are based on multi-layer LSTMs while the joint network is a feed-forward network which combines outputs from encoder and decoder. Typically, encoder dominates the size of RNN-T model. For example in pre-trained RNN-T model from MLPerf [9], encoder is 42.97M parameters while decoder is 1.64M and joint network is 0.7M parameters. Thus, we will focus on compressing the encoder later in the paper.

LSTM is the core component of the encoder. In a high level, each LSTM consists of two weight matrices (i.e.,W_x: input weight matrix, W_h: recurrent weight matrix) along with a chain of activation functions. Later in this paper, we will focus on compressing W_x and W_h weight matrices. Note that there are two different ways to represent W_x and W_h. As shown in Figure 1, each weight matrix can be either rolled out into 4 small matrices (i.e., W_{xf},W_{xo},W_{xz},W_{xi}) or stacked into one matrix (i.e., W_x). We will evaluate the effect of different weight representation of LSTM on PSVD in section 5.4.

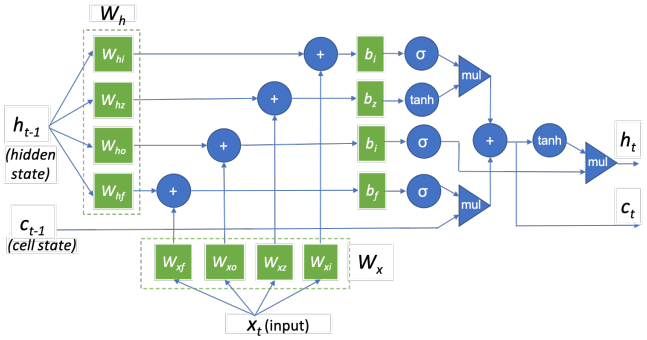

Fig. 1. LSTM with stacked or rollout weight matrices [18]

Singular Value Decomposition (SVD)-based model compression is one of widely-used compression methods. For a given weight matrix $W \in \mathbb{R}^{M \times N}$, $W = USV$, where S is a singular value matrix which is diagonal. SVD compression only keeps the k largest singular values of W. Then, we could have a compressed weight matrix $\widetilde{W}$ which is comprised of the two small matrices $\widetilde{U}$ and $\widetilde{V}$ (i.e., $\widetilde{W} = \widetilde{U}\widetilde{S}\widetilde{V}$ where $\widetilde{S}$ is fused to either of $\widetilde{U}$ or $\widetilde{V}$). Through SVD compression, the

number of parameters changes from $M \times N$ to $(M+N) \times k$. If k is small enough, the number of parameters would be largely reduced.

4. PSVD COMPRESSION

Adopting similar notation as [19], we denote the hidden state of the l-th hidden layer at time $t-1$ by $\mathbf{h}_{t-1}^l \in \mathbb{R}^{N^l}$ and the inputs to this layer at time t, which are hidden states from the previous layer or the input features, are denoted by $\mathbf{h}_t^{l-1} \in \mathbb{R}^{N^{l-1}}$. The weighted sum of $\mathbf{h}_{t-1}^l$ and $\mathbf{h}_t^{l-1}$ are denoted by $\mathbf{y}_t \in \mathbb{R}^{N^l}$. Then the equations which define a standard RNN are

$$\mathbf{p}_t^l = W_x^{l-1}\mathbf{h}_t^{l-1} + \mathbf{b}_x^l \tag{1}$$
$$\mathbf{q}_t^l = W_h^l\mathbf{h}_{t-1}^l + \mathbf{b}_h^l \tag{2}$$
$$\mathbf{y}_t^l = \mathbf{p}_t^l + \mathbf{q}_t^l \tag{3}$$
$$\mathbf{h}_t^l = g(\mathbf{y}_t^l) \tag{4}$$

where $\mathbf{b}_x^l \in \mathbb{R}^{N^l}$ and $\mathbf{b}_h^l \in \mathbb{R}^{N^l}$ represent bias vectors, and $W_x^{l-1} \in \mathbb{R}^{N^{l-1} \times N^l}$ and $W_h^l \in \mathbb{R}^{N_l \times N_l}$ refer to the input and the recurrent weight matrices, respectively. $g(\cdot)$ denotes a non-linear activation function. Note that when $l = 1$, $\mathbf{h}_t^{l-1} = \mathbf{x}_t$. The above procedure can be easily extended to case of LSTM. The input and recurrent weight matrix in the case of LSTM is then the concatenation of input, forget, cell and output gate weight matrices. Using $\mathbf{i}_t^l$, $\mathbf{f}_t^l$, $\mathbf{z}_t^l$, $\mathbf{o}_t^l$ and $\mathbf{c}_t^l$ to respectively denote the input, forget, cell, output gates and cell state at time t of layer l, hs to denote the number of hidden states, $\sigma(\cdot)$ is the sigmoid function, and $\odot$ is the Hadamard product, then

$$\mathbf{i}_t^l = \sigma(\mathbf{y}_t^l[:, 0:hs]) \tag{5}$$
$$\mathbf{f}_t^l = \sigma(\mathbf{y}_t^l[:, hs:2*hs]) \tag{6}$$
$$\mathbf{o}_t^l = \sigma((\mathbf{y}_t^l[:, 2*hs:3*hs]) \tag{7}$$
$$\mathbf{z}_t^l = \tanh(\mathbf{y}_t^l[:, 3*hs:4*hs]) \tag{8}$$
$$\mathbf{c}_t^l = \mathbf{f}_t^l \odot \mathbf{c}_{t-1}^l + \mathbf{i}_t^l \odot \mathbf{z}_t^l \tag{9}$$
$$\mathbf{h}_t^l = \mathbf{o}_t^l \odot \tanh(\mathbf{c}_t^l) \tag{10}$$

As described in section 3, our PSVD is based on SVD, so decomposing W_x^{l-1} and W_h^l produces $W_x^{l-1} = U_x^{l-1}S_x^{l-1}V_x^{l-1^T}$ and $W_h^l = U_h^l S_h^l V_h^{l^T}$. Keeping the top k singular values approximates the factors as $\widetilde{U_x^{l-1}}\widetilde{S_x^{l-1}} \in \mathbb{R}^{N^{l-1} \times k}$, $\widetilde{V_x^{l-1^T}} \in \mathbb{R}^{k \times N^l}$, $\widetilde{U_h^l}\widetilde{S_h^l} \in \mathbb{R}^{N^l \times k}$ and $\widetilde{V_h^{l^T}} \in \mathbb{R}^{k \times N^l}$. The total number of parameters of W_x^{l-1} is reduced from $N^{l-1} \times N^l$ to $(N^{l-1} + N^l) \times k$. Similarly, the total number of W_h^l is reduced from $N^l \times N^l$ to $(N^l + N^l) \times k$. Based on the truncated weights, we have

$$\mathbf{p}_t^l \approx \widetilde{\mathbf{p}_t^l} = (\widetilde{U_x^{l-1}}\widetilde{S_x^{l-1}})\widetilde{V_x^{l-1^T}}\widetilde{\mathbf{h}_t^{l-1}} + \mathbf{b}_x^l \tag{11}$$
$$\mathbf{q}_t^l \approx \widetilde{\mathbf{q}_t^l} = (\widetilde{U_h^l}\widetilde{S_h^l})\widetilde{V_h^{l^T}}\widetilde{\mathbf{h}_{t-1}^l} + \mathbf{b}_h^l \tag{12}$$

Note that the hidden states in (11) and (12) starting from the second layer are different from original ones because the input and recurrent weights have been SVD compressed in (1) and (2), so the hidden states are updated accordingly. To optimize the weights of the SVD compressed model, we want to minimize the mean squared error between uncompressed and compressed models' outputs per layer. In this way, the compressed model would be closer to the uncompressed model. By assuming the uncompressed model's output is approximately linear with compressed model's output, this optimization problem could be solved via linear regression with a small set of sampled data. More concisely, we target to determine M_x^{l-1} and M_h^l as the solution to the following least-squares problems:

$$M_x^{l-1} = \text{argmin}_M ||M\widetilde{\mathbf{p}_t^l} - \mathbf{p}_t^l||_{\mathcal{F}}^2 \tag{13}$$
$$M_h^l = \text{argmin}_M ||M\widetilde{\mathbf{q}_t^l} - \mathbf{q}_t^l||_{\mathcal{F}}^2 \tag{14}$$

where $||X||_{\mathcal{F}}$ denotes the Frobenius norm of the matrix. However, the recurrent and multi-layer nature of LSTMs make this approach less effective. If each layer is post-trained independently, the error of shallower layers will get amplified and accumulate to deeper layers [16]. Therefore, post-training all layers at once would not be very effective. We propose a new algorithm Post-training compression SVD (PSVD) which applies linear regression multiple times from the first to the last layer in sequence to alleviate the error accumulation issue. In our experiment we found that using it significantly improves accuracy.

The inputs to Algorithm 1 are the original $\mathbf{p}_t^l$, $\mathbf{q}_t^l$ and SVD compressed $\widetilde{\mathbf{p}_t^l}$, $\widetilde{\mathbf{q}_t^l}$. In each iteration, $M_{x,i}^{l-1}$ is calculated to make $\widetilde{\mathbf{p}_t^l}$ as close to $\mathbf{p}_t^l$ as possible. Similarly, $M_{h,i}^l$ is calculated to adjust $\widetilde{\mathbf{q}_t^l}$. With N iterations, the final post-trained $\hat{\mathbf{p}_t^l}$ and $\hat{\mathbf{q}_t^l}$ are

$$\hat{\mathbf{p}_t^l} = M_{x,N}^{l-1}M_{x,N-1}^{l-1}...M_{x,1}^{l-1}\widetilde{\mathbf{p}_t^l} \tag{15}$$
$$\hat{\mathbf{q}_t^l} = M_{h,N}^{l}M_{h,N-1}^{l}...M_{h,1}^{l}\widetilde{\mathbf{q}_t^l} \tag{16}$$

Since the weights in (11) and (12) are in SVD decomposition form, by multiplying $M_{x,N}^{l-1}M_{x,N-1}^{l-1}...M_{x,1}^{l-1}$ with $(\widetilde{U_x^{l-1}}\widetilde{S_x^{l-1}})$, the resulting matrix has the same dimension as $(\widetilde{U_x^{l-1}}\widetilde{S_x^{l-1}})$. The same applies for $\hat{\mathbf{q}_t^l}$ case. In Section 5, we will discuss how many samples and iterations are needed.

5. EXPERIMENT

Our experiment is based on the Pytorch inference benchmark pipeline provided by MLPerf [9][20] which is the industry

Result: Optimized compressed RNNT model:
calculate $\mathbf{p}_t^l$ and $\mathbf{q}_t^l$ for all layers without doing any compression;
$i \leftarrow 0$;
while $i <=$ *max number of trails or WER change* $<=$ *threshold* **do**
  for $l \in L$ **do**
    apply SVD compression on $\mathbf{p}_t^l$ to get $\widetilde{\mathbf{p}_t^l}$;
    fit linear regression to determine $M_{x,i}^{l-1} \leftarrow \text{argmin}_M ||M\widetilde{\mathbf{p}_t^l} - \mathbf{p}_t^l||_{\mathcal{F}}^2$;
    update network by replacing $\widetilde{\mathbf{p}_t^l}$ with post-trained $M_{x,i}^{l-1}\widetilde{\mathbf{p}_t^l}$;
    apply weight SVD on $\mathbf{q}_t^l$ to get $\widetilde{\mathbf{q}_t^l}$;
    fit linear regression to determine $M_{h,i}^{l} \leftarrow \text{argmin}_M ||M\widetilde{\mathbf{q}_t^l} - \mathbf{q}_t^l||_{\mathcal{F}}^2$;
    update network by replacing $\widetilde{\mathbf{q_t^l}}$ with post-trained $M_h^{l,i}\widetilde{\mathbf{q}_t^l}$.
  end
end

Algorithm 1: Post training SVD Compression (PSVD)

standard for deep learning benchmarks. It provides a pre-trained RNN-T model consisting of an LSTM-based encoder and prediction network along with a joint network. The encoder has 5 LSTM layers with 1024 nodes (42.97M parameters) while the prediction network has 2 LSTM layers with 320 nodes (1.64M parameters). The Word Error Rate (WER) of pre-trained model is 7.45% with greedy decoding. It is evaluated on LibriSpeech dev-clean dataset [10] less that 15 seconds in length. Since the encoder is dominant in size, our paper focuses on the encoder only. We randomly sampled 400 audio WAV files, applied SVD decomposition on stacked weight matrices, and iterated three times for PSVD.

The results of our experiment are presented in Table 1. We evaluate WERs of SVD and PSVD under different thresholds (i.e, Threshold k refers to keeping the top $k \times 100\%$ singular values while Compression Ratio (**CR**) = the number of parameters in original model)/(the number of parameters in compressed model). PSVD outperforms SVD. When compression ratio is 7.71x, PSVD improved 8.36% WER than SVD. Although absolute WER improvement decreases as compression ratio (CR) decreases, relative WER improvement of PSVD over SVD (i.e., $WER_{rel} = 100 * (WER_{svd} - WER_{psvd})/(WER_{svd} - WER_{original})$) is still consistent between 20% and 30% in different compression ratios.

5.1. Effects of iterations on input and recurrent weights

To illustrate how PSVD works, we study its effects on input weight W_x^l and recurrent weight W_h^l separately. Here we only compress the first layer of the encoder using a threshold of 0.1 with randomly sampled 200 audio WAV files in Table 2. We observe that PSVD works more effectively on recurrent weight W_h^l. After applying SVD compression, the error of hidden states accumulates as t increases because W_h^l is multiplied with h_{t-1}^l at each timestamp. Multiple iterations will be helpful to ease the error accumulation issue on recurrent weight W_h^l. For input weight W_x^l with $l = 1$, h_t^{l-1} is equal to x_t, the error does not accumulate in this case. Therefore, PSVD seems to not significantly improve WER by only post-training W_x^l.

Table 1. WER (%) on LibriSpeech dev-clean set as a function of compression ratio with and without using PSVD. (WER of the original uncompressed model is 7.45%)

Threshold	CR	WER_{svd}	WER_{psvd}	WER_{rel}
0.1	7.71	35.11	26.75	30.22
0.2	3.86	14.95	13.37	21.06
0.4	1.93	9.26	8.79	25.97

Table 2. WER (%) of PSVD on recurrent weight W_h^l or input weight W_x^l

Number of iterations	WER_{psvd} *on* W_h^l	WER_{psvd} *on* W_x^l
0	12.48	12.48
1	10.87	11.74
2	10.09	11.75
3	9.65	11.78

5.2. Effects of number of iterations

Only a few iterations are needed for PSVD. We did multiple experiments to investigate how WER varies with number of iterations under different thresholds and sample sizes settings. We observed similar outcomes that 3 iterations are enough. Table 3 shows one experiment with 400 audio WAV files and a threshold of 0.1. In general, WER and number of iterations are positively correlated but with some fluctuations.

Table 3. WER (%) of PSVD as a function of iterations

Number of iterations	WER_{psvd}
0	35.11
1	29.31
2	27.47
3	26.75
4	27.58

5.3. Effects of sample size

To study the sample size needed to apply PSVD, we experimented on three different settings (100, 200 and 400 audio WAV files) with one iteration. As shown in Table 4, 200 samples achieves the best accuracy at a threshold of 0.1. In the other two cases, 400 samples is the best. Actually, the sample size needed for PSVD is directly related to the sample size needed for linear regression. Many researches suggest that a general rule of sample size needed for linear regression is 10 per each variable. 400 audio WAV files generates about 80,000 samples. Since the number of regression variables are 4096 per per layer, we have about 20 samples per each variable, which are good enough for linear regression.

Table 4. WER (%) of PSVD as a function of sample size

Threshold	*Number of WAV files*	WER_{svd}	WER_{psvd}
0.1	100	35.11	30.61
0.1	200	35.11	29.04
0.1	400	35.11	29.31
0.2	100	14.95	14.18
0.2	200	14.95	14.22
0.2	400	14.95	13.95
0.4	100	9.26	9.22
0.4	200	9.26	9.15
0.4	400	9.26	8.99

5.4. Stacked vs. rollout weight matrices

The recurrent weight matrix W_h^l is a concatenation of the four gate weight matrices, obtained by stacking them vertically, $[W_{hi}, W_{hf}, W_{hz}, W_{ho}]^T$, which corresponds to the input, forget, cell and output gates, respectively. Similarly, the inter layer matrix W_x^l is the concatenation of $[W_{xi}, W_{xf}, W_{xz}, W_{xo}]^T$ which represents the inner weights of the input, forget, cell and output gates. While applying SVD compression, we can either apply in the stacked matrix level (W_h^l, W_x^l) or in the rollout matrix level (W_{hi}, W_{hf},...). Based on our experiment, we observed that applying SVD on stacked matrix outperforms applying SVD on rollout matrix, as shown in Figure 2. Throughout this paper, we refer to applying layer-wise SVD on stacked matrix by mentioning SVD compression.

5.5. Model generalization

To demonstrate the model generalization ability, we trained PSVD on 200 randomly chosen LibreSpeech dev-clean wav files and tested on the whole test-clean data, which contains 2620 wav files and represented accuracy in Table 5. We observed consistent trends as in previous subsections. PSVD

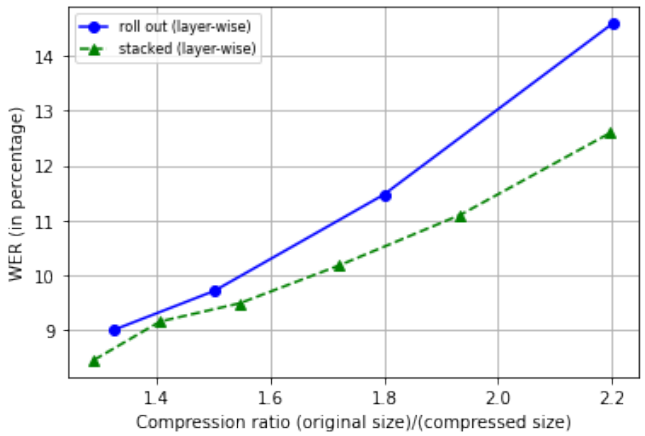

Fig. 2. WER(%) as a function of compression ratio by applying SVD compression on stacked and rollout weight matrices.

brings more significant improvements when compression ratios are relatively large and improves with more iterations. We also tested PSVD on TED-LIUMv2 test data [21] which containing 1144 wav files. WER of uncompressed model on TED-LIUM test data is 30.07%. When we compress at 0.1 (CR=7.71), WER increases to 60.21%. With three iterations of PSVD, WER drops to 53.32%. Similarly, PSVD brings WER from 41.75% down to 38.07% when compressing at 0.2 (CR=3.86) and brings WER from 33.93% down to 32.65% when compressing at 0.1 (CR=1.93).

Table 5. Word error rates (%) on LibriSpeech test-clean set as a function of compression ratio and number of iterations with and without PSVD. (WER of the original uncompressed model on test-clean set is 7.67%)

Threshold	*Number of Iteration*	WER_{svd}	WER_{psvd}
0.1	1	35.73	30.62
0.1	2	35.73	29.54
0.1	3	35.73	28.35
0.2	1	15.13	14.90
0.2	2	15.13	14.50
0.2	3	15.13	14.11
0.4	1	9.50	9.35
0.4	2	9.50	9.30
0.4	3	9.50	9.26

5.6. Discussion

In our experiments, we seek to understand the impact of the proposed PSVD algorithm from different perspectives. We validated that PSVD works robustly with a small amount of

test data on different compression ratios. Also, post-training completes quickly within a few number of iterations and thus within a few hours on a commonly available CPU. Accordingly, PSVD can be easily extended to other RNN-based neural network models.

For simplicity, we used uniform thresholds for all layers. However, different layers have different sensitivities to compression and recurrent weights may also react differently from input weights. To have an optimal model to balance between model size and accuracy, we could further study autotuning compression (e.g., reinforcement learning, Bayesian optimization) on top of PSVD. In addition, we could include low-bit quantization which is orthogonal to PSVD and thus could further reduce the model size.

6. CONCLUSION

We presented a technique to post-train a SVD compressed RNN-T model without a large training dataset and costly back-propagation. The proposed technique adopts linear regression sequentially to mitigate the gap between uncompressed and compressed models, which could improve WER up to 8.36%.

7. REFERENCES

[1] Yanzhang He, Tara N Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, et al., "Streaming end-to-end speech recognition for mobile devices," in *ICASSP*, 2019.

[2] Hieu Duy Nguyen, Anastasios Alexandridis, and Athanasios Mouchtaris, "Quantization aware training with absolute-cosine regularization for automatic speech recognition," *Proc. Interspeech 2020*, pp. 3366–3370, 2020.

[3] Genta Indra Winata, Andrea Madotto, Jamin Shin, Elham J Barezi, and Pascale Fung, "On the effectiveness of low-rank matrix factorization for lstm model compression," *arXiv preprint arXiv:1908.09982*, 2019.

[4] Rohit Prabhavalkar, Ouais Alsharif, Antoine Bruguier, and Lan McGraw, "On the compression of recurrent neural networks with an application to lvcsr acoustic modeling for embedded speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 5970–5974.

[5] Jian Xue, Jinyu Li, and Yifan Gong, "Restructuring of deep neural network acoustic models with singular value decomposition.," in *Interspeech*, 2013, pp. 2365–2369.

[6] MLCommon, "People's speech dataset," in *https://mlcommons .org/en/peoples-speech/*, 2020.

[7] Sree Hari Krishnan Parthasarathi and Nikko Strom, "Lessons from building acoustic models with a million hours of speech," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 6670–6674.

[8] "The best gpus for deep learning," in *https://timdettmers.com/2020/09/07/which-gpu-for-deep-learning/*, 2020.

[9] MLCommon, "Rnn-t," in *https://github.com/mlcommons/inference/tree/master/speech*$_r ecognition/rnnt/$, 2020.

[10] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[11] Andrey Kuzmin, Markus Nagel, Saurabh Pitre, Sandeep Pendyam, Tijmen Blankevoort, and Max Welling, "Taxonomy and evaluation of structured compression of convolutional neural networks," *arXiv preprint arXiv:1912.09802*, 2019.

[12] Yihui He, Xiangyu Zhang, and Jian Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[13] Yuan Shangguan, Jian Li, Qiao Liang, Raziel Alvarez, and Ian McGraw, "Optimizing speech recognition for the edge," *arXiv preprint arXiv:1909.12408*, 2019.

[14] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling, "Data-free quantization through weight equalization and bias correction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1325–1334.

[15] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak, "Lsq+: Improving low-bit quantization through learnable offsets and better initialization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 696–697.

[16] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 10, pp. 1943–1955, 2015.

[17] Ying Zhang, Tao Xiang, Timothy M Hospedales, and Huchuan Lu, "Deep mutual learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4320–4328.

[18] NVIDIA, "Lstm on cudnn," in *https://developer.nvidia.com/blog/ optimizing-recurrent-neural-networks-cudnn-5/*, 2020.

[19] R. Prabhavalkar, O. Alsharif, A. Bruguier, and L. McGraw, "On the compression of recurrent neural networks with an application to lvcsr acoustic modeling for embedded speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 5970–5974.

[20] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al., "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.

[21] Anthony Rousseau, Paul Deléglise, Yannick Esteve, et al., "Enhancing the ted-lium corpus with selected data for language modeling and more ted talks.," in *LREC*, 2014, pp. 3935–3939.