ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 978-1-7281-7605-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICASSP39728.2021.9413844

SPARSIFICATION VIA COMPRESSED SENSING FOR AUTOMATIC SPEECH RECOGNITION

Kai Zhen[1*], *Hieu Duy Nguyen*[2], *Feng-Ju Chang*[2], *Athanasios Mouchtaris*[2], *and Ariya Rastrow*[2]

[1]Indiana University Bloomington
[2]Alexa Machine Learning, Amazon, USA
zhenk@indiana.edu, {hieng, fengjc, mouchta, arastrow}@amazon.com

ABSTRACT

In order to achieve high accuracy for machine learning (ML) applications, it is essential to employ models with a large number of parameters. Certain applications, such as Automatic Speech Recognition (ASR), however, require real-time interactions with users, hence compelling the model to have as low latency as possible. Deploying large scale ML applications thus necessitates model quantization and compression, especially when running ML models on resource constrained devices. For example, by forcing some of the model weight values into zero, it is possible to apply zero-weight compression, which reduces both the model size and model reading time from the memory. In the literature, such methods are referred to as sparse pruning. The fundamental questions are when and which weights should be forced to zero, i.e. be pruned. In this work, we propose a compressed sensing based pruning (CSP) approach to effectively address those questions. By reformulating sparse pruning as a sparsity inducing and compression-error reduction dual problem, we introduce the classic compressed sensing process into the ML model training process. Using ASR task as an example, we show that CSP consistently outperforms existing approaches in the literature.

Index Terms— Model pruning, automatic speech recognition (ASR), sparsity, Recurrent Neural Network Transducer (RNN-T), compressed sensing.

1. INTRODUCTION

Automatic Speech Recognition (ASR) is an important component of a virtual assistant system. The main focus of ASR is to convert users' voice command into transcription, based on which further processing will act upon. Recently, end-to-end (E2E) approaches have attracted much attention due to their ability of directly transducing audio frame features into sequence outputs [1]. Without explicitly imposing/injecting domain knowledge and manually tweaking intermediate components (such as lexicon model), building and maintaining E2E ASR system is much more efficient than a hybrid deep neural network (DNN)-Hidden Markov Model (HMM) model.

In order to provide the best user experience, an ASR system is required to achieve not only high accuracy but also small user-perceived latency. This motivates the trend of moving the processing from Cloud/remote servers to users' device to reduce the latency further. More often than not, the hardware limitations impose strict constraints on the model complexity [2, 3]. Firstly, the hardware may only support integer arithmetic operations for run-time inference, which necessitates model quantization. Secondly, a hardware often performs multiple tasks supported by different models in sequence. The hardware, with limited memory size, thus needs to move multiple models in and out of the processing units.

To compress the model for the hardware, two widely applied methods are (a) model selection/structured pruning, i.e. choosing a model structure with pruned layers/channels and small performance degradation [4, 5], and (b) zero-weight compression/sparse pruning, i.e. pruning small-value weights to zero [6, 7]. Model selection differs from sparse pruning in that it deletes entire channels or layers, showing a more efficient speedup during inference, yet with a more severe performance degradation [4, 5]. These two types of methods are usually complementary: after being structurally pruned, a model can also undergo further zero-weight compression to improve the inference speed. In this study, we focus on sparse pruning, targeting at lowering the memory storage and bandwidth requirement as it largely contributes to the latency for on-device ASR models.

A naïve approach for sparse pruning is to push the weight values smaller than a threshold to zero after training, which often leads to significant performance degradation [6]. To mitigate this problem, Tibshirani *et. al.* applied *LASSO* regularization to penalize large-value model weights [8]. The drawbacks are twofold: firstly, it does not exert an explicit specification of the target sparsity level; secondly, it is subject to the gradient vanishing issue when the model has more and more layers. Gradual pruning approach [6] resolves those concerns by defining a sparsity function that maps each training step to a corresponding intermediate sparsity level. During the model training, the pruning threshold is adjusted gradually according to the function to eventually reach the target sparsity level. However, gradual pruning assumes that pruning the values smaller than the threshold will lead to the least degradation, which is heuristic and sub-optimal. Consequently, gradual pruning techniques provide a guidance on "when to prune and by how much", but a less satisfying answer for "which (weights) to prune", thus leading to inefficiency.

In this work, we propose a compressed sensing (CS)-based pruning method, referred to as CSP subsequently, that is sparse-aware and addresses both "*when to prune*" and "*which to prune*". CSP reformulates the feedforward operations in machine learning architectures, such as Long Short Term Memory (LSTM) or Fully-Connected (FC) cells, as a sensing procedure with the inputs and hidden states being random sensing matrices. Under that perspective, a sparsification process is to enhance the sparsity and reduce the compression error, due to pruning, simultaneously. Following [9], we adopt the ℓ_1 regularization to enforce sparsity and the ℓ_2 regularization to mitigate the compression loss, and reformulate the

*This work was conducted during Kai's internship in Amazon Pittsburgh, PA, USA.

sparsification procedure as an optimization problem. We demonstrate the effectiveness of our method by compressing recurrent neural network transducer (RNN-T), one of the E2E ASR models. The RNN-T model is sparsified via a hybrid training mechanism in which CSP is conducted during the feedforward stage, along with the back propagation for the global optimization. Our proposed method constantly outperforms the state-of-the-art gradual pruning approach in terms of the word error rate (WER) under all settings. In particular, with a sparisty ratio of 50% where half of the weights are 0, CSP yields little to no performance degradation on LibriSpeech dataset.

The rest of the paper is structured as follows. In Sec. 2, we briefly review related pruning methods. Our CSP method is introduced in Sec. 3. Sec. 4 describes our experiment setup and results. Finally, we conclude our work with some remarks in Sec. 5.

2. RELATED WORK

One of the most straightforward approaches for sparse-aware training is applying ℓ_k regularization, where $k = 0, 1, 2$, etc. There has been a rich literature in comparing various forms of sparsity regularizers. Consider the model training: $\mathbb{W} \leftarrow \arg\min_{\mathbb{W}} \mathcal{L}_{\text{accuracy}}(\mathbb{W}) + ||\mathbb{W}||_1$, where ℓ_1 norm is used on the regularization. The fundamental idea is to optimize the model prediction while penalizing large weight values. In DNN model compression, the regularization is usually implemented as an extra loss term for the training.

This training-aware sparsity regularization leads to promising pruning results especially for convolutional neural networks (CNN) [10] with residual learning techniques [11, 12], but may not apply well to models employing recurrent neural network (RNN) components such as LSTM. The error due to a global sparsity constraint ℓ_1 will be propagated to all time steps. Additionally, such drawback is much more severe for architectures, such as RNN-T, which contains feedback loop from one part of the model to the others.

Another well-known, state-of-the-art, pruning method for ML models is gradual-pruning [6]. This method does not resort to ℓ_1 regularization for sparsity, but dynamically updates the pruning threshold during model training, as is indicated by its name. To answer the question "when to prune", the authors defines a sparsity function parametrized by the target sparsity $\boldsymbol{s}_f$ at $\boldsymbol{t}_n$ step with an initial pruning step $\boldsymbol{t}_0$. Concretely, at training step $\boldsymbol{t}$, the pruning threshold is adjusted to match the sparsity $\boldsymbol{s}_t$ calculated in Eq. 1. The main complication is to adjust the pruning procedure such that the model weights are relatively converged and the learning rate is still sufficiently large to reduce the pruning-induced loss.

$$\boldsymbol{s}_t = \boldsymbol{s}_f * \left(1 - \left(1 - \frac{\boldsymbol{t} - \boldsymbol{t}_0}{\boldsymbol{t}_n - \boldsymbol{t}_0}\right)^3\right) \tag{1}$$

One concern is that finding an optimal setup for these hyperparameters can be hard without going through a rigorous ablation analysis. Furthermore, with gradual pruning, gradient-updating back-propagation is the only mechanism to limit the degradation. Most importantly, gradual pruning only addresses the question "*when to prune*" but not "*which (weights) to prune*". At each time step, the weights below the (gradually increased) threshold are to be pruned. This is based on the premise that the smaller the weight, the less important it is, which is heuristic and sub-optimal.

3. COMPRESSED SENSING BASED PRUNING

3.1. Adapting Compressed Sensing for Sparse Pruning

CS aims to compressing potentially redundant information into a sparse representation and reconstructing it efficiently [13, 14], which has facilitated a wide scope of engineering scenarios, such as medical resonance imaging (MRI) and radar imaging. For example, in MRI [15], high resolution scanned images are generated per millisecond or microsecond, leading to significant storage cost and transmission overhead. CS learns a sparse representation of each image with which, during the decoding time, CS can recover the reference image almost perfectly. Assume an image compression task with a reference image $\boldsymbol{x} \in \mathbb{R}^n$, where $\boldsymbol{x}$ is usually decomposed into an orthogonal transformation basis ψ and the activation $\boldsymbol{s}$, as $\boldsymbol{x} = \psi * \boldsymbol{s}$. Given that $\boldsymbol{s}$ satisfies $\mathcal{K}$-sparse property, CS is capable of locating those $\mathcal{K}$ salient activation elements. Concretely, CS introduces a sensing matrix ϕ to project $\boldsymbol{x}$ into $\boldsymbol{y}$, as $\boldsymbol{y} = \phi * \boldsymbol{x}$. In [9, 16], it has been proved that by optimizing the ℓ_2 loss in the sensing dimensionality while exerting the ℓ_1 norm regularizer to $\boldsymbol{s}$, a $\mathcal{K}$-sparse solution of $\boldsymbol{s}$, denoted as $\hat{\boldsymbol{s}}$, can be found in polynomial time. Consequently, $\psi * \hat{\boldsymbol{s}}$ can estimate the original image $\boldsymbol{x}$ with high fidelity and relatively small latency.

In this work, we investigate the effectiveness of CS based pruning for ML models. We consider the ASR task, i.e. converting audio speech to transcriptions, using an RNN-T architecture. Due to the space constraint, we only describe the transformation of LSTM cell, which is a major building block in various E2E ASR models. It is straightforward to extend the transformations to other architectures/layers like the fully-connected (FC) network and CNN.

Consider a vanilla LSTM cell: the element-wise multiplication between the input at time step t, $\boldsymbol{x}^{(t)}$, and kernels is given in Eq. 2, while that of hidden states from the previous step $\boldsymbol{h}^{(t-1)}$ and recurrent kernels is in Eq. 3. Here, $\boldsymbol{W}_f, \boldsymbol{W}_c, \boldsymbol{W}_i$, and $\boldsymbol{W}_o$ (correspondingly $\boldsymbol{U}_f, \boldsymbol{U}_c, \boldsymbol{U}_i$, and $\boldsymbol{U}_o$) denote the kernels (correspondingly recurrent kernels) weights of the cell (c), the input gate (i), output gate (o), and forget gate (f), respectively. All gating mechanisms to update the cell and hidden states are encapsulated in Eq. 4, where $\boldsymbol{C}^{(t)}$ is the cell state vector at time t and $\mathcal{G}$ denotes the transformation of $\boldsymbol{z}_x, \boldsymbol{z}_h, \boldsymbol{C}^{(t-1)}$ into $\boldsymbol{C}^{(t)}, \boldsymbol{h}^{(t)}$. The bias terms are omitted for ease of presentation.

$$\boldsymbol{z}_x = [\boldsymbol{W}_f, \boldsymbol{W}_c, \boldsymbol{W}_i, \boldsymbol{W}_o] \odot [\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t)}] \tag{2}$$

$$\boldsymbol{z}_h = [\boldsymbol{U}_f, \boldsymbol{U}_c, \boldsymbol{U}_i, \boldsymbol{U}_o] \odot [\boldsymbol{h}^{(t-1)}, \boldsymbol{h}^{(t-1)}, \boldsymbol{h}^{(t-1)}, \boldsymbol{h}^{(t-1)}] \tag{3}$$

$$[\boldsymbol{C}^{(t)}, \boldsymbol{h}^{(t)}] = \mathcal{G}(\boldsymbol{z}_x, \boldsymbol{z}_h, \boldsymbol{C}^{(t-1)}), \tag{4}$$

To prune all kernel weights (denoted as $\mathbb{W} = [\boldsymbol{W}_f, \boldsymbol{W}_c, \boldsymbol{W}_i, \boldsymbol{W}_o]$) in LSTM cells, we adapt and reformulate a CS-like pruning procedure by adopting ℓ_1 regularization for sparsity-inducing and ℓ_2 regularization for compression-error reduction, as outlined in Fig. 1. The procedure starts by a midway inference to collect the activation inputs $[\boldsymbol{z}_x, \boldsymbol{z}_h]$ as in Eq. 2 and Eq. 3. When those kernels are sparsified, the activation inputs will be consequently updated as $[\boldsymbol{z}'_x, \boldsymbol{z}'_h]$ (see Fig. 1). The goal is to sparsify and prune the kernels while preserving the value of $[\boldsymbol{z}_x, \boldsymbol{z}_h]$ to minimize the pruning-induced loss. To that end, the ℓ_1 regularizer is applied to the input kernels while the ℓ_2 regularizer controls the reconstruction loss on $\boldsymbol{z}_x$. Hence, our CS solver is embedded in a local optimizer triggered periodically by feedforward steps in a stochastic manner. As illustrated in the restricted isometry property (RIP), the sensing matrix ϕ is expected to be random for an accurate signal reconstruction [16]. The proposed CS solver satisfies the RIP

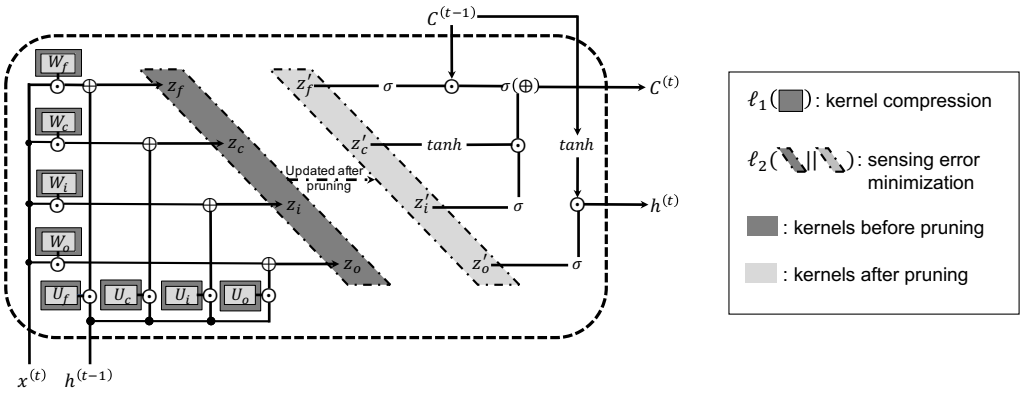

Fig. 1: A CSP-LSTM cell with the local sparse optimizer

with high probability, since the sensing matrices (the input $\boldsymbol{x}^{(t)}$ and hidden state $\boldsymbol{h}^{(t)}$) in LSTM cells vary at different time steps.

Our CS solver is kernel-wise, i.e. in each LSTM layer, there is one CS solver for each of the input-kernel and recurrent-kernel sparsification process. A general CS loss for the local optimizer is defined in Eq. 5. By adjusing the sensing coefficient, the local optimizer is capable of calibrating the model to the target sparsity by balancing the ℓ_1 and ℓ_2 regularizers,

$$\mathcal{L}_{CS}(\mathbb{W}, \boldsymbol{y}, \boldsymbol{h}) = \lambda|\mathbb{W}|_1 + ||\boldsymbol{y} - \mathbb{W} \odot \boldsymbol{h}||_2, \tag{5}$$

where the hyperparameter λ in Eq. 5 is referred to as the sensing coefficient subsequently. $\mathbb{W}$, $\boldsymbol{h}$, and $\boldsymbol{y}$ denote kernel weights, inputs, and activation input, respectively. To remove the manual-tuning, we dynamically update λ via Eq. 6.

$$\lambda \leftarrow \begin{cases} \max(\lambda_{\text{lower}}, \lambda - \epsilon), & \text{if } \boldsymbol{s} > \boldsymbol{s}_t \\ \min(\lambda_{\text{upper}}, \lambda + \epsilon), & \text{otherwise.} \end{cases} \tag{6}$$

During training, if the actual sparsity overshoots the target sparsity at a certain step, we reduce λ by ϵ. Otherwise, the λ is increased by ϵ instead. In our setup, λ is constrained between $\lambda_{\text{lower}} = 0.001$ and $\lambda_{\text{upper}} = 1.0$ with ϵ being 0.005. The pruning threshold, ρ, initialized as 0.002, is also updated as in [6] to adjust the sparsity. Algorithm 1 summarizes our proposed CSP procedure for input kernels. The CSP for recurrent kernels $\mathbb{U} = [\boldsymbol{U}_f, \boldsymbol{U}_c, \boldsymbol{U}_i, \boldsymbol{U}_o]$ is executed similarly and is omitted for brevity.

Algorithm 1 Proposed CSP for an LSTM cell

Inputs: input data at time step t, $\boldsymbol{x}^{(t)}$
the hidden state from the previous step, $\boldsymbol{h}^{(t-1)}$
the cell state from the previous step, $\boldsymbol{C}^{(t-1)}$
Outputs: updated hidden state at time step t, $\boldsymbol{h}^{(t)}$
updated cell state at time step t, $\boldsymbol{C}^{(t)}$
if sparsity level $<$ the target **then**
  Midway inference: $[\boldsymbol{z}_x, \boldsymbol{z}_h] = \mathcal{F}(\mathbb{W}, \boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t)})$
  Sensing: $\mathbb{W}' \leftarrow \arg\min_{\mathbb{W}} \mathcal{L}_{CS}(\mathbb{W}, \boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t)}, \boldsymbol{z}_x, \boldsymbol{z}_h)$
  Pruning: $\mathbb{W}^p_{(i,j)} \leftarrow \begin{cases} 0, & \text{if } \left|\mathbb{W}'_{(i,j)}\right| < \rho; \\ \mathbb{W}'_{(i,j)}, & \text{otherwise.} \end{cases}$
  Update coefficient and threshold λ and ρ
end if
Update cell outputs: $[\boldsymbol{z}_x, \boldsymbol{z}_h] = \mathcal{F}(\mathbb{W}^p, \boldsymbol{x}^{(t)}, \boldsymbol{h}^{(t)})$
$[\boldsymbol{C}^{(t)}, \boldsymbol{h}^{(t)}] = \mathcal{G}(\boldsymbol{z}_x, \boldsymbol{z}_h, \boldsymbol{C}^{(t-1)})$

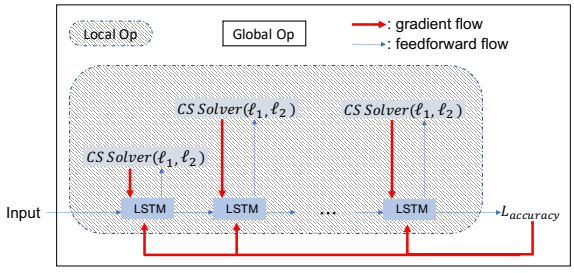

Fig. 2: Sparse-aware training scheme for CSP

3.2. Hybrid Sparse-Aware Training Scheme

The proposed CSP sparsification optimizer is combined with the conventional backpropagation algorithm in a sparse-aware training scheme (see Fig. 2). Since CSP is conducted kernel-wise for each layer, the local optimization is triggered during the feedforward stage as the samples passes through the first encoder layer to the last decoder layer sequentially. When the model makes the prediction, the global loss is calculated to update parameters in all preceding layers through backpropagation. The hybrid training scheme is not subjected to gradient vanishing thanks to the local ℓ_1 regularization, and compatible with global optimization.

4. EXPERIMENTS

4.1. Experimental Setup

We consider the ASR task with an RNN-T architecture. Comparing to the listen-attend-spell (LAS) model [17], RNN-T features online streaming capability while not requiring a separate lexicon/pronunciation system as in the connectionist temporal classification model (CTC) [18]. To rigorously evaluate CSP along with existing sparsification methods, we experiment with 4 different RNN-T based topologies, all with 5 encoding layers and 2 decoding layers. Model M-I, M-III and M-IV all have 768 units per layer (UpL) and 4000 word-pieces (WPs) while M-II uses 1024 UpL with 2500 WPs. Among 4 models, only M-III includes a joint network (J-N), which combines the outputs of RNN-T encoder and decoder to achieve better performance. M-I and M-II are trained on a far-field dataset with 25k hours of English audio, while M-III and M-IV are trained on the LibriSpeech dataset with 960 hours [19]. Note that these models are reasonably small comparing with counterparts in the literature, to highlight the effect of sparse pruning on model performance. Please refer to Table 1 for the total number of parameters of each model.