Structured Pruning of LSTMs via Eigenanalysis and Geometric Median for Mobile Multimedia and Deep Learning Applications

Nikolaos Gkalelis, Vasileios Mezaris
Information Technologies Institute
Centre for Research and Technology Hellas
Thessaloniki, Greece
Email: {gkalelis,bmezaris}@iti.gr

***Abstract*—In this paper, a novel structured pruning approach for learning efficient long short-term memory (LSTM) network architectures is proposed. More specifically, the eigenvalues of the covariance matrix associated with the responses of each LSTM layer are computed and utilized to quantify the layers' redundancy and automatically obtain an individual pruning rate for each layer. Subsequently, a Geometric Median based (GM-based) criterion is used to identify and prune in a structured way the most redundant LSTM units, realizing the pruning rates derived in the previous step. The experimental evaluation on the Penn Treebank text corpus and the large-scale YouTube-8M audio-video dataset for the tasks of word-level prediction and visual concept detection, respectively, shows the efficacy of the proposed approach[1].**

***Keywords*-deep learning; mobile multimedia; LSTM; automatic structured pruning; eigenanalysis; geometric median**

I. Introduction

Deep learning (DL) is currently becoming a game changer in most industries, ranging from media and mobile communications to health care and security [1]–[5]. However, it is well known that top-performing DL models consist of millions or billions of parameters and their deployment to resource-limited applications such as smartphones and other mobile devices is challenging [1].

Structured network pruning has been identified as a remedy to the above problem [6]–[9]. However, the structured recurrent neural network (RNN) pruning techniques introduced so far (i.e. [8], [9]) require the modification of the loss function in order to impose sparsity constraints, which may lead to numerical instabilities and performance reduction when a high degree of sparseness is pursued [10]. In order to address the above limitations, we follow a different path, taking advantage of recent advances in structured deep convolutional neural networks (DCNN) pruning [6], [11]. Firstly, the eigenanalysis of the sample covariance matrix computed using a LSTM layer's responses is utilized to quantify correlations among units in the same LSTM layer and derive automatically the pruning rates at layer level, as for instance it is done for convolutional layers in [11]. Subsequently, a GM-based criterion, which has shown superior performance in comparison to criteria exploiting sparsity-inducing constraints in the domain of structured DCNN pruning [6], is used to identify and prune the most replaceable RNN structures according to the pruning rates computed above. Experimental results show that the proposed approach provides competitive performance on the Penn Treebank text corpus [12] and the YouTube-8M audio-video dataset [13] for the tasks of word-level prediction in text and concept detection, respectively.

The rest of the paper is structured as follows: Section II reviews related work on pruning. Section III details the proposed method. The experimental evaluation is described in Section IV and conclusions are drawn in Section V.

II. Related work

DNN compression and acceleration approaches can be roughly categorized to quantization, low-rank approximations, knowledge distillation and pruning [1], [2], [14]. The latter is currently getting increasing attention mainly because the methods falling in this category can achieve high compression rates while maintaining a stable model performance. In the following, we briefly review several pruning approaches in order to put ours into context.

Pruning techniques typically consist of the definition of the elementary network structures as candidates for pruning, an importance estimation criterion to rank the above structures, and a pruning strategy defining how pruning is performed [6]–[9], [11], [15]. Depending on whether a pruning approach removes individual network weights or well-defined network components, is characterized as unstructured or structured, respectively. Concerning pruning strategies, most approaches either prune a pre-trained model or incorporate pruning into the training procedure. Another important aspect of pruning is whether the layers' pruning rates are fixed or obtained automatically [11]. The latter can usually provide more efficient architectures and is closely related with the network architecture search paradigm [16].

The major advantage of structured pruning techniques over the unstructured ones is that the former do not require the use of special-purpose accelerators [8], [15] and thus can

[1]Source code is made publicly available at: https://github.com/bmezaris/lstm_structured_pruning_geometric_median

take advantage of cheap, widely available devices such as conventional GPUs. The structured pruning of DCNNs has been studied extensively in the literature [6], [7], [11]. In contrast, structured RNN pruning is a much less investigated topic [8], [9]. More specifically, in [8] intrinsic sparse structures (ISSs) of LSTMs are defined and a Group Lasso-based (GL-based) approach is used for ISS pruning. Similarly, the authors in [9] utilize the L_0 norm to constrain network parameters and subsequently prune the ISS components which are close to zero, achieving higher pruning rates than [8]. Both the above works utilize sparsity-inducing regularizers to modify the loss function, which may lead to numerical instabilities and suboptimal solutions for certain network architectures [10]. To this end, inspired from best practices in the DCNN structured pruning domain, we quantify the redundancy at layer level using the eigenanalysis of the covariance matrix formed by the layer's responses (e.g. similar to the way it is done in [11] for DCNN filters), and utilize a GM-based criterion [6] to rank and prune the most replaceable LSTM structures in each layer.

III. Proposed method

A. Formulation

Suppose an annotated training dataset $\mathcal{X}$ of N sequences and C classes

$$\mathcal{X} = \{(\mathbf{X}_\kappa, \mathbf{y}_\kappa) | \kappa = 1, \ldots, N\}, \tag{1}$$

where, the matrix $\mathbf{X}_\kappa = [\mathbf{x}_{\kappa,1}, \ldots, \mathbf{x}_{\kappa,T}] \in \mathbb{R}^{F \times T}$ represents the κth sequence of length T (without loss of generality it is assumed that all sequences have the same length), $\mathbf{y}_\kappa \in \mathbb{R}^C$ is its class indicator vector, whose ιth element is 1 if $\mathbf{X}_\kappa$ belongs to class ι and zero otherwise, $\mathbf{x}_{\kappa,t}$ is the tth feature vector of the κth sequence, and F is the input space dimensionality.

A DNN consisting of L LSTM layers is utilized for learning the above classes. The computations in the lth LSTM layer with respect to the κth input sequence at a specified time step t are performed as [17]

$$\mathbf{i}^{[l]}_{\kappa,t} = \sigma(\mathbf{W}^{[l]}_{ix}\mathbf{x}^{[l]}_{\kappa,t} + \mathbf{W}^{[l]}_{ih}\mathbf{h}^{[l]}_{\kappa,t-1} + \mathbf{b}^{[l]}_i), \tag{2}$$

$$\mathbf{f}^{[l]}_{\kappa,t} = \sigma(\mathbf{W}^{[l]}_{fx}\mathbf{x}^{[l]}_{\kappa,t} + \mathbf{W}^{[l]}_{fh}\mathbf{h}^{[l]}_{\kappa,t-1} + \mathbf{b}^{[l]}_f), \tag{3}$$

$$\mathbf{o}^{[l]}_{\kappa,t} = \sigma(\mathbf{W}^{[l]}_{ox}\mathbf{x}^{[l]}_{\kappa,t} + \mathbf{W}^{[l]}_{oh}\mathbf{h}^{[l]}_{\kappa,t-1} + \mathbf{b}^{[l]}_o), \tag{4}$$

$$\mathbf{u}^{[l]}_{\kappa,t} = tanh(\mathbf{W}^{[l]}_{ux}\mathbf{x}^{[l]}_{\kappa,t} + \mathbf{W}^{[l]}_{uh}\mathbf{h}^{[l]}_{\kappa,t-1} + \mathbf{b}^{[l]}_u), \tag{5}$$

$$\mathbf{c}^{[l]}_{\kappa,t} = \mathbf{f}_{\kappa,t} \odot \mathbf{c}^{[l]}_{\kappa,t-1} + \mathbf{i}^{[l]}_{\kappa,t} \odot \mathbf{u}^{[l]}_{\kappa,t}, \tag{6}$$

$$\mathbf{h}^{[l]}_{\kappa,t} = \mathbf{u}^{[l]}_{\kappa,t} \odot tanh(\mathbf{c}^{[l]}_{\kappa,t}), \tag{7}$$

where, the superscript l is the layer index (i.e. $l = 1, \ldots, L$); $\mathbf{i}^{[l]}_{\kappa,t}$, $\mathbf{f}^{[l]}_{\kappa,t}$, $\mathbf{o}^{[l]}_{\kappa,t}$, $\mathbf{u}^{[l]}_{\kappa,t}$, $\mathbf{c}^{[l]}_{\kappa,t}$ and $\mathbf{h}^{[l]}_{\kappa,t}$ are the $H^{[l]}$-dimensional input gate, forget gate, output gate, input update, unit state and hidden state vectors; $H^{[l]}$ is the number of the layer's units, $\mathbf{x}^{[l]}_{\kappa,t} \in \mathbb{R}^{F^{[l]}}$ is the layer's input vector at time step t; and $\mathbf{W}^{[l]}_{ix}, \mathbf{W}^{[l]}_{fx}, \mathbf{W}^{[l]}_{ox}, \mathbf{W}^{[l]}_{ux} \in \mathbb{R}^{H^{[l]} \times F^{[l]}}$, $\mathbf{W}^{[l]}_{ih}, \mathbf{W}^{[l]}_{fh}, \mathbf{W}^{[l]}_{oh}, \mathbf{W}^{[l]}_{uh} \in \mathbb{R}^{H^{[l]} \times H^{[l]}}$, $\mathbf{b}^{[l]}_i, \mathbf{b}^{[l]}_f, \mathbf{b}^{[l]}_o, \mathbf{b}^{[l]}_u \in \mathbb{R}^{H^{[l]}}$, are the layer's weight matrices and vectors. Based on the above formulation, the goal of structurally pruning LSTM architectures can be stated as follows: given a target pruning rate $\theta \in (0, 1)$ for the overall network, estimate the pruning rate $\theta^{[l]}$ and subsequently select the less significant (in terms of their influence to the overall network classification performance) $\theta^{[l]}H^{[l]}$ units to prune at layer l so that $\frac{\sum_{l=1}^{L} \theta^{[l]} H^{[l]}}{\sum_{l=1}^{L} H^{[l]}} = \theta$.

B. Computation of layer's pruning rate

Due to its high representational power, the hidden state vector $\mathbf{h}^{[l]}_{\kappa,T}$ of any LSTM layer l at the last time step has been often used for representing the overall sequence at the output of the layer (e.g. see [18]). Based on this fact, the whole training set at the output of the lth layer is represented using the data matrix

$$\begin{aligned} \mathbf{Z}^{[l]} &= [\mathbf{h}^{[l]}_{1,T}, \ldots, \mathbf{h}^{[l]}_{N,T}] \\ &= [\mathbf{z}^{[l]}_1, \ldots, \mathbf{z}^{[l]}_N], \end{aligned} \tag{8}$$

where for simplicity we set $\mathbf{z}^{[l]}_\kappa = \mathbf{h}^{[l]}_{\kappa,T}$. Given $\mathbf{Z}^{[l]}$, the sample covariance matrix associated with the responses of the lth layer can be computed using

$$\mathbf{S}^{[l]} = \frac{1}{N} \sum_{\kappa=1}^{N} (\mathbf{z}^{[l]}_\kappa - \mathbf{m}^{[l]})(\mathbf{z}^{[l]}_\kappa - \mathbf{m}^{[l]})^T, \tag{9}$$

where $\mathbf{m}^{[l]} = \frac{1}{N} \sum_{\kappa=1}^{N} \mathbf{z}^{[l]}_\kappa$ is the sample mean vector. The above matrix is symmetric positive semidefinite, thus, with real nonnegative eigenvalues, which can be efficiently computed using appropriate techniques [19]. Sorting $\mathbf{S}^{[l]}$'s eigenvalues into descending order and normalizing them to sum to one, we car represent them as

$$\lambda^{[l]}_1, \ldots, \lambda^{[l]}_{H^{[l]}}, \tag{10}$$

where, $\lambda^{[l]}_1 \geq \cdots \geq \lambda^{[l]}_{H^{[l]}} \geq 0$ and $\sum_{i=1}^{H^{[l]}} \lambda^{[l]}_i = 1$.

As explained in [11], the set of the eigenvalues provides insight on the correlation of the responses produced by the different units of the layer. An eigenvalue close to zero implies that the variables along the corresponding principal component of $\mathbf{S}^{[l]}$ are linearly dependent. Therefore, the situation where all the energy in the output of a layer (represented by its hidden state vectors) is accumulated to only a small fraction of eigenvalues indicates that there are many redundant units in this layer. Based on the above analysis, we proceed to express the layer pruning rate with respect to the derived eigenvalues. We define the following two sets of variables, $\zeta^{[l]}_1, \ldots, \zeta^{[l]}_{H^{[l]}}$, $\delta^{[l]}_1, \ldots, \delta^{[l]}_{H^{[l]}}$, where, $\zeta^{[l]}_i = \sum_{j=1}^{i} \lambda^{[l]}_j$,

$$\delta^{[l]}_i = \begin{cases} 1, & \text{if } \zeta^{[l]}_i \leq \alpha, \\ 0 & \text{else.} \end{cases} \tag{11}$$

In the equation above, α a is a parameter in $[0, 1]$ defining the amount of energy to keep at the output of a layer, and thus closely related with layer's pruning rate. The pruning rate $\theta^{[l]}$ for the lth layer can then be computed using

$$\theta^{[l]} = 1 - \frac{\sum_{i=1}^{H^{[l]}} \delta_i^{[l]}}{H^{[l]}}. \quad (12)$$

Thus, our goal is now to identify α by solving the following single-variable optimization problem

$$\underset{\alpha}{\operatorname{argmin}} \left| \frac{1}{L} \sum_{l=1}^{L} \theta^{[l]} - \theta \right|. \quad (13)$$

Because α is bounded in [0,1], the above problem can be efficiently solved using an appropriate iterative method.

C. Minibatch computation of sample covariance matrix

The computation of the sample covariance matrix for each LSTM layer requires high memory storage space for retaining the layer's output along with all epoch steps, and may be even infeasible when processing large-scale datasets in devices with limited computational resources. To this end, we propose a minibatch algorithm for the computation of this matrix, as explained in the following. For simplicity of illustration, let us consider the case that the dataset is split into two partitions, with one partition consisting of the $\breve{N}$ sequences processed so far, and the new minibatch of $\tilde{N}$ sequences, i.e. $N = \breve{N} + \tilde{N}$. Dropping the superscript layer index $[l]$ for simplicity, the data matrix at the output of any layer can be then represented as

$$\begin{aligned} \mathbf{Z} &= [\mathbf{z}_1, \ldots, \mathbf{z}_{\breve{N}}, \mathbf{z}_{\breve{N}+1}, \ldots, \mathbf{z}_N] \\ &= [\breve{\mathbf{z}}_1, \ldots, \breve{\mathbf{z}}_{\breve{N}}, \tilde{\mathbf{z}}_1, \ldots, \tilde{\mathbf{z}}_{\tilde{N}}] \\ &= [\breve{\mathbf{Z}}, \tilde{\mathbf{Z}}], \end{aligned} \quad (14)$$

where the block matrices $\breve{\mathbf{Z}}$, $\tilde{\mathbf{Z}}$ contain the hidden state vectors corresponding to the already processed sequences and the minibatch of new sequences, respectively. The sample mean vector can then be written as

$$\mathbf{m} = \tfrac{1}{N}(\textstyle\sum_{\kappa=1}^{\breve{N}} \breve{\mathbf{z}}_\kappa + \sum_{\kappa=1}^{\tilde{N}} \tilde{\mathbf{z}}_\kappa) = \tfrac{\breve{N}}{N}\breve{\mathbf{m}} + \tfrac{\tilde{N}}{N}\tilde{\mathbf{m}}, \quad (15)$$

where, $\breve{\mathbf{m}} = \frac{1}{\breve{N}} \sum_{\kappa=1}^{\breve{N}} \breve{\mathbf{z}}_\kappa$, $\tilde{\mathbf{m}} = \frac{1}{\tilde{N}} \sum_{\kappa=1}^{\tilde{N}} \tilde{\mathbf{z}}_\kappa$. On the other hand, the sample covariance matrix (9) can be decomposed as

$$\mathbf{S} = \mathbf{\Sigma} - \mathbf{m}\mathbf{m}^T, \quad (16)$$

where $\mathbf{\Sigma} = \frac{1}{N} \sum_{\kappa=1}^{N} \mathbf{z}_\kappa \mathbf{z}_\kappa^T$. The latter can be further expressed as

$$\begin{aligned} \mathbf{\Sigma} &= \frac{1}{N}\Big(\sum_{\kappa=1}^{\breve{N}} \mathbf{z}_\kappa \mathbf{z}_\kappa^T + \sum_{\kappa=\breve{N}+1}^{N} \mathbf{z}_\kappa \mathbf{z}_\kappa^T\Big) \\ &= \frac{\breve{N}}{N}\breve{\mathbf{\Sigma}} + \frac{\tilde{N}}{N}\tilde{\mathbf{\Sigma}}, \end{aligned} \quad (17)$$

Algorithm 1: Minibatch computation of $\mathbf{S}$ (9)

Input: Current $\mathbf{\Sigma}$, $\mathbf{m}$, N; new batch $\tilde{\mathbf{Z}}$
Output: Updated $\mathbf{S}$, $\mathbf{\Sigma}$, $\mathbf{m}$, N
Compute $\tilde{\mathbf{\Sigma}}$, $\tilde{\mathbf{m}}$, $\tilde{N}$ using $\tilde{\mathbf{Z}}$
Update $\mathbf{S}$, $\mathbf{\Sigma}$, $\mathbf{m}$, N using (15), (16), (17):
$\breve{N} \leftarrow N$; $N \leftarrow \breve{N} + \tilde{N}$
$\mathbf{m} \leftarrow \frac{\breve{N}}{N}\mathbf{m} + \frac{\tilde{N}}{N}\tilde{\mathbf{m}}$; $\mathbf{\Sigma} \leftarrow \frac{\breve{N}}{N}\mathbf{\Sigma} + \frac{\tilde{N}}{N}\tilde{\mathbf{\Sigma}}$
$\mathbf{S} \leftarrow \mathbf{\Sigma} - \mathbf{m}\mathbf{m}^T$

where, $\breve{\mathbf{\Sigma}} = \frac{1}{\breve{N}} \sum_{\kappa=1}^{\breve{N}} \breve{\mathbf{z}}_\kappa \breve{\mathbf{z}}_\kappa^T$, $\tilde{\mathbf{\Sigma}} = \frac{1}{\tilde{N}} \sum_{\kappa=1}^{\tilde{N}} \tilde{\mathbf{z}}_\kappa \tilde{\mathbf{z}}_\kappa^T$. Based on the above formulation, the minibatch computation of the sample covariance matrix for an arbitrary number of minibatches is presented in Algorithm 1.

D. LSTM unit importance estimation and pruning

Without loss of generality we examine the unit selection and pruning procedure for a popular LSTM architecture, consisting of a biderectional LSTM (BLSTM) [20] with layer indices $l = (1, 1), (1, 2)$, for the forward and backward LSTMs, respectively, and a regular LSTM with layer index $l = 2$, as shown in Fig. 1. For each layer the weight matrices can be stacked to form the following block matrices

$$\mathbf{W}^{[l]} = [\mathbf{W}_x^{[l]}, \mathbf{W}_h^{[l]}], \quad (18)$$

$$\mathbf{W}_x^{[l]} = [\mathbf{W}_{ix}^{[l]}, \mathbf{W}_{fx}^{[l]}, \mathbf{W}_{ux}^{[l]}, \mathbf{W}_{ox}^{[l]}], \quad (19)$$

$$\mathbf{W}_h^{[l]} = [\mathbf{W}_{ih}^{[l]}, \mathbf{W}_{fh}^{[l]}, \mathbf{W}_{uh}^{[l]}, \mathbf{W}_{oh}^{[l]}], \quad (20)$$

where, $\mathbf{W}_x^{[l]} \in \mathbb{R}^{H^{[l]} \times 4F^{[l]}}$, $\mathbf{W}_h^{[l]} \in \mathbb{R}^{H^{[l]} \times 4H^{[l]}}$ and $l = (1, 1), (1, 2), 2$. The block matrix $\mathbf{W}^{[l]}$ can be then represented as

$$\mathbf{W}^{[l]} = [\mathbf{w}_1^{[l]}, \ldots, \mathbf{w}_{H^{[l]}}^{[l]}]^T, \quad (21)$$

where $\mathbf{w}_j^{[l]} \in \mathbb{R}^Q$ is the jth row of $\mathbf{W}^{[l]}$, directly related with the jth unit of the lth layer, and $Q = 4(H^{[l]} + F^{[l]})$.

Based on the above formulation, an importance score $\eta_j^{[l]}$ for each unit in the lth layer can be derived using a GM-based function, which has shown excellent performance in DCNN pruning [6], [7],

$$\eta_j^{[l]} = \frac{1}{H^{[l]}} \sum_{k=1}^{H^{[l]}} \|\mathbf{w}_j^{[l]} - \mathbf{w}_k^{[l]}\|_2. \quad (22)$$

The value $\eta_j^{[l]}$ quantifies the dissimilarity between the jth unit and all other units in the layer. Therefore, a small $\eta_j^{[l]}$ denotes that in average this unit is highly correlated with the other units in the layer and thus can be discarded safely without harming the classification performance of the network.

Proceeding to the definition of weight structures for structurally pruning the network, we learn ISSs for both LSTMs and BLSTMs by extending the approach presented in [8], as explained in the following. Let us suppose that the kth

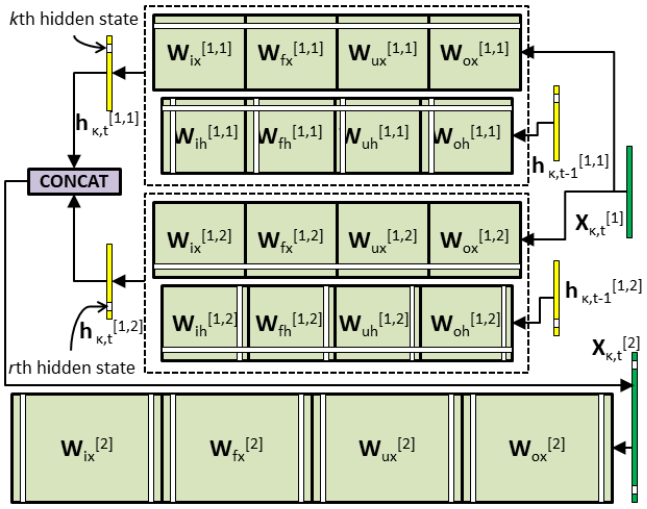

Figure 1: Applying ISSs in BLSTM.

and rth hidden states of the forward and backward LSTM, respectively, have been selected to be removed, as shown in Fig. 1. Then, the kth and rth row of $\mathbf{W}_x^{[1,1]}$, $\mathbf{W}_h^{[1,1]}$, and, $\mathbf{W}_x^{[1,2]}$, $\mathbf{W}_h^{[1,2]}$, respectively, contributing to the generation of these states should be removed as well, as shown by the four white horizontal lines in the figure. Moreover, due to their correspondence to the connections receiving the above hidden states from the previous time step, the kth and rth column of each matrix block of $\mathbf{W}_h^{[1,1]}$ and $\mathbf{W}_h^{[1,2]}$, should also be removed, as shown by the eight vertical lines in the matrix blocks of the BLSTM in the figure. Finally, the kth and rth columns of the four matrix blocks in $\mathbf{W}_x^{[2]}$ receiving the above states are also set to zero, as shown by the eight white vertical lines in the weight matrices of the second layer LSTM in Fig. 1.

IV. Experiments

A. Datasets

1) Penn Treebank (PTB): This is one of the most widely used datasets for evaluating the performance of statistical language models [12]. It consists of 1086k tokens in ASCII format and 10k classes (i.e. unique tokens). It is partitioned to training, validation, and testing sets with 930k, 74k and 82k tokens, respectively.

2) YouTube-8M (YT8M): The large-scale YT8M video dataset is utilized to evaluate the proposed approach for the task of audiovisual concept detection [13]. This dataset consists of 3862 classes (semantic concepts) and 6134598 videos. Visual and audio feature vectors have been pre-extracted and provided at frame-level (1 frame per second) with dimensionality 1024 and 128, respectively.

Table I: Evaluation results on PTB (lower PPL values are better).

	Dropout keep rate	*ISS # in (1st, 2nd)*	*PPL (validate, test)*
baseline [21]	0.35	(1500, 1500)	(82.57, 78.57)
ISS-GL [8]	0.60	(373, 315)	(82.59, 78.65)
ISS-L_0 [9]	0.65	(296, 247)	(81.62, 78.08)
ISS-GM (prop.)	0.50	(236, 297)	(**81.49**, **77.97**)

Table II: Evaluation results on YT8M (higher GAP@20 values are better).

	GAP@20	T_{tr}
no pruning	84.33%	6.73
ISS-GL [8] ($\theta = 30\%$)	83.20%	7.82
ISS-GM (prop.) ($\theta = 30\%$)	**84.12%**	15.40
ISS-GL [8] ($\theta = 70\%$)	82.2%	7.43
ISS-GM (prop.) ($\theta = 70\%$)	**83.10%**	14.54

B. Setup

The proposed method, called hereafter ISS-GM, is evaluated against ISS-GL [8] and ISS-L_0 [9] in the PTB dataset. In this experiment, a two-layer stacked LSTM model [21] is utilized, and the training procedure described in [9] is followed. For the evaluation in the YT8M, a variant of the BLSTM architecture presented in Section III-D is utilized to compare ISS-GM and ISS-GL (the software implementation of ISS-L_0 is not provided in [9] and for this reason ISS-L_0 is not included in this experiment). In more detail, the forward and backward layers of the BLSTM consist of 512 units each, while 1024 units are used for the LSTM layer. Each video is represented with a feature vector sequence of $T = 300$ length. Both models are trained for 10 epochs using CE loss with minibatch SGD, batch size of 256, an exponential learning rate schedule with initial learning rate of 0.0002, learning rate decay of 0.95 at every epoch, and pruning is applied every 200 training steps.

The performance evaluation on the PTB and YT8M datasets is performed using the per-word perplexity (PPL) and the global average precision at 20 (GAP@20) [13], respectively. ISS-GM is implemented in PyTorch and Tensorflow for the evaluation in PTB and YT8M, respectively. For the ISS-GL method, the Tensorflow code provided in [8] is adapted for the YT8M experiments. The evaluation is performed in an Intel i7-3770K PC with 32 GB RAM, Windows 10, and Nvidia GeForce GPU (GTX 1080 Ti).

C. Results

The experimental results in terms of PPL on the PTB dataset are shown in Table I. Table II depicts GAP@20 rates and training times in hours per epoch (T_{tr}) for the evaluation on the YT8M dataset with pruning rates 30% and 70%. From the obtained results we conclude the following: i) The proposed ISS-GM achieves the best performance

in all experiments. More specifically, on the PTB dataset a small but significant PPL gain is obtained using ISS-GM (considering that ISS-L_0 is the previous state-of-the-art approach), while, on the YT8M dataset a quite large GAP@20 improvement of approximately 1% is attained over ISS-GL for both 30% and 70% pruning rates. ii) ISS-GM exhibits a high degree of robustness against large pruning rates, making it suitable for compressing deep networks and allowing their deployment in mobile and other resource-constrained environments. For instance, only 0.21% and 1.23% performance drop is observed on the YT8M dataset for 30% and 70% pruning rates, respectively. iii) Concerning training times, we observe that ISS-GM is approximately two times slower than ISS-GL in the YT8M experiment, mainly because ISS-GL computes the eigenvalues of the covariance matrix for each layer every time the pruning procedure is applied. However, concerning that the training is performed off-line, this time overhead is considered insignificant.

V. Conclusion

In this paper, a new LSTM structured pruning approach was proposed that utilizes the sample covariance matrix of layer's responses and a GM-based criterion to automatically derive pruning rates at layer level and compress the network, to make it more suitable for deployment in mobile or other resource-constrained environments. The proposed approach was evaluated on two datasets for the tasks of word-level prediction in text and concept detection in audiovisual sequences, providing competitive performance at high pruning rates.

Acknowledgment

This work was supported by the EU Horizon 2020 research and innovation programme under grant agreement H2020-780656 ReTV.

References

[1] K. Ota, M. S. Dao, V. Mezaris, and F. G. B. D. Natale, "Deep learning for mobile multimedia: A survey," *ACM Trans. Mutlimedia Comput., Commun. and Appl.*, vol. 13, no. 3s, pp. 34:1–34:22, Jun. 2017.

[2] W. Bailer and H. Fassold, "Resource-efficient object detection by sharing backbone CNNs," in *IEEE ISM*, San Diego, California, USA, Dec. 2019, pp. 196–199.

[3] D. Zeng, Y. Yu, and K. Oyama, "Audio-visual embedding for cross-modal music video retrieval through supervised deep CCA," in *IEEE ISM*, Taichung, Taiwan, Dec. 2018, pp. 143–150.

[4] P. Budikova, M. Batko, and P. Zezula, "Multi-modal image retrieval for search-based image annotation with RF," in *IEEE ISM*, Taichung, Taiwan, Dec. 2018, pp. 52–60.

[5] E. M. Ibrahim *et al.*, "Neural networks based fractional pixel motion estimation for HEVC," in *IEEE ISM*, Taichung, Taiwan, Dec. 2018, pp. 110–113.

[6] Y. He *et al.*, "Filter pruning via Geometric median for deep convolutional neural networks acceleration," in *IEEE CVPR*, Long Beach, CA, USA, Jun. 2019.

[7] N. Gkalelis and V. Mezaris, "Fractional step discriminant pruning: A filter pruning framework for deep convolutional neural networks," in *IEEE ICMEW*, London, UK, Jul. 2020, pp. 1–6.

[8] W. Wen *et al.*, "Learning intrinsic sparse structures within long short-term memory," in *ICLR*, Vancouver, BC, Canada, Apr-May 2018.

[9] L. Wen, X. Zhang, H. Bai, and Z. Xu, "Structured pruning of recurrent neural networks through neuron selection," *Neural Networks*, vol. 123, pp. 134–141, Mar. 2020.

[10] H. Xu, C. Caramanis, and S. Mannor, "Sparse algorithms are not stable: A no-free-lunch theorem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 1, pp. 187–193, Jan. 2012.

[11] X. Suau, U. Zappella, and N. Apostoloff, "Filter distillation for network compression," in *IEEE WACV*, Snowmass Village, CO, USA, Mar. 2020, pp. 3129–3138.

[12] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn Treebank," *Comput. Linguist.*, vol. 19, no. 2, p. 313–330, Jun. 1993.

[13] J. Lee, A. P. Natsev, W. Reade, R. Sukthankar, and G. Toderici, "The 2nd YouTube-8M large-scale video understanding challenge," in *ECCV Workshops*, Munich, Germany, Sep. 2018.

[14] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018.

[15] H. Kang, "Accelerator-aware pruning for convolutional neural networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 55, pp. 2093–2103, Jan. 2020.

[16] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, Jan. 2019.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, vol. 2, Montreal, Canada, Dec. 2014, pp. 3104–3112.

[19] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 4th ed. Baltimore, USA: The Johns Hopkins University Press, 2013.

[20] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, Mar. 2005.

[21] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1804.03209, 2014.