Efficient Neural Network Compression Process to Facilitate Hardware Implementation

Antoine Siebert[1,3], Guillaume Ferré[1], Bertrand Le Gal[2], Aurélien Fourny[3]
[1]Univ. Bordeaux, CNRS, Bordeaux INP, IMS, UMR 5218, F-33400, Talence, France [2]IRISA/INRIA Laboratory
[3]THALES, France Email: forename.name@{ims-bordeaux.fr[1], inria.fr[2], thalesgroup.com[3]}

***Abstract*—Neural networks deliver unparalleled performance across numerous domains, yet they are often burdened by increasing parameter counts and computational costs. In scenarios where real-time or embedded device deployment is required, cloud computing may not be feasible. While various neural network compression techniques have been proposed, typically a compromise between different methods is necessary. This paper introduces a new compression method that enables simultaneous quantization and pruning. Subsequently, a low-rank decomposition is performed. This approach is applied to a Long-Short Term Memory (LSTM)-based neural network and evaluated against other compression methods using a language dataset. Our results demonstrate that incorporating the proposed method into the compression process not only is feasible but also achieves significant compression efficacy.**

***Index Terms*—Neural network compression, quantization, pruning, low-rank decomposition**

I. Introduction

In recent years, neural networks have gained significant popularity by delivering impressive performance across a diverse range of tasks, from signal processing to, more recently, language modeling. As these networks become increasingly complex, both in terms of parameter numbers and computational demands, they are generally run in the cloud or on multiple GPUs, consuming substantial energy. However, there is a growing shift towards executing neural networks on embedded devices. This trend is driven by several factors, including the considerable carbon footprint associated with cloud-based machine learning tasks and potential security risks to data. Moreover, cloud inference may not always be feasible for real-time applications, necessitating that algorithms be directly embedded into the devices. To facilitate this shift, numerous methods have been developed, aimed at making neural networks lighter, faster, and less computationally intensive while preserving their performance. There are several main types of compression methods, including quantization [1], pruning [2], low-rank decomposition [3], and knowledge distillation [4]. Historically, most of the compression research has focused on convolutional neural networks (CNNs) due to their greater embedded use in real-time signal processing applications. Recurrent neural networks are also used in signal processing applications like image compression, [5], or many natural language processing (NLP) tasks. But, the compression methods developed are not often tested on this type of networks. Authors of [6] present a low-rank method to compress Long-Short-Term-Memory (LSTM) neural network. A bayesian pruning method is presented in [7], which produces sparse weight matrices. However, this type of matrix is not always advantageous for hardware implementation. There are also few compression approaches that attempt to merge several methods [8] in order to benefit from the advantages of each. In [**?**] authors propose a scheme to double the weight bitwidths at each stage which allows pruning, but restricts quantization to a power of two and was not implemented on recurrent networks. However, implementing multiple compression techniques simultaneously can be complex. It often introduces new issues regarding the order of processing, or potentially results in an over-parameterized system where the parameter selection becomes critically important. Usually, a challenge in neural network compression lies in selecting an appropriate method.

In this paper, we naturally brought together and combined different neural network compression methods in order to create a complete compression process adapted for hardware implementation by avoiding sparse matrices. We propose a new simultaneously quantization and pruning algorithm tested on recurrent neural networks. The proposed method is presented in Section II and the obtained results are presented in Section III. The paper is concluded and perspectives are given in Section IV.

II. Presentation of the neural network compression process

In this section, we elaborate on the comprehensive neural network compression method and describe the steps involved in the process.

A. Process principle

The various stages of the compression process are described in Fig. 1. The starting point is a pre-trained model, and the end product is a compressed version of that model. The

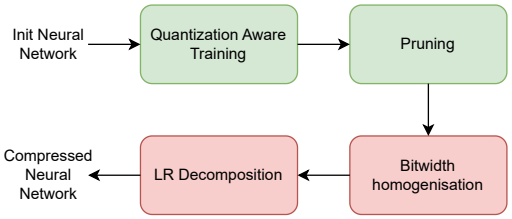

Fig. 1. Compression process from an initialized trained neural network. Green boxes represent steps during which the bitwidths of the layers are still learnable, and the red ones represent fixed bitwidths.

initial step involves converting all basic layers into quantized layers. This includes setting bitwidth parameters that dictate the number of bits used for each neuron in a layer. Following each layer, activation quantization is performed to prevent the handling of numbers with excessively high bit counts. The bitwidths of all layers are parameters learned during the training. These activation layers also have their own learnable bitwidth parameters. This is illustrated by green boxes in the process diagram in Fig. 1. We employ a Quantization Aware Training (QAT) approach where quantization occurs during a training phase. This method delivers better performance than the Post Training Quantization (PTQ) method, but requires a training dataset. Following quantization, an automated pruning step is initiated. These two steps i.e. quantization and pruning are integrated into a single method detailed in the following subsection. After these initial modifications, a fine-tuning phase is conducted to refine the performance of the network with its new configurations. Then, a bitwidth homogenisation step is carried out, standardizing the bitwidths within each layer to a uniform value and freezing them for subsequent processes. The final bitwidth value of the layer is given by the maximum value of its bitwidth vector. These freezed bitwidths are represented by red boxes in Fig. 1. The final major step in the process involves computing a low-rank decomposition on selected weight matrices to reduce even further the model complexity and memory usage. Between each of these steps, a fine-tuning is operated to ensure optimal performance and integration of the compression effects.

B. Quantization and pruning

We consider a training set $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$, N being the number of samples in the dataset and a neural network $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\theta})$. The quantization used is the uniform symmetric quantization, i.e. for a given input $\boldsymbol{x}$ with coefficients in $[\alpha, \beta]$, the quantified version $\boldsymbol{x}_q$ of $\boldsymbol{x}$ is given by

$$\boldsymbol{x}_q = S \times \lfloor \frac{x}{S} \rceil, \tag{1}$$

with $\lfloor \cdot \rceil$ the round-to-nearest integer operator, and S the scale factor defined by $S = \frac{max(\beta, -\alpha)}{2^{b-1}-1}$, and b being the number of bits for weight encoding. All weights attached to the same neuron i are encoded on the same number of bits $b_i \in [0; 2^n]$, $n \in \mathbb{N}$. This choice was made to simplify the training with a limited number of bitwidth parameters, and also to facilitate the pruning. Thus, for a linear layer of size (in, out), $\boldsymbol{b}$ is given by $\boldsymbol{b} = [b_1, \ldots, b_{out}]$. During the half process, i.e. until the bitwidth homogenisation, the bitwidths are float numbers. It makes the training possible without having discontinuities and allowing the use of gradient descend optimization algorithm. The Eq. 1 is still valid in the case where b is a float.

The neurons of the i^{th} layer are encoded on $\boldsymbol{b}_i$ bits. Basically, when the neural network is defined, $b = 32$ for all the network layers. To avoid the manually choice of bitwidth for each layer, $\boldsymbol{b}_i$ is learned during the training. To do this, an adequate loss is used during the neural network training:

$$\sum_{i=1}^{N} L(f(\boldsymbol{x}_i, \boldsymbol{\theta}), \boldsymbol{y}_i)(1 - \lambda) + \lambda R(\boldsymbol{b}_i), \tag{2}$$

where L is the classical cross entropy loss function, and R is a regularization term relative to the neural network compression. The parameter λ controls the importance given to compression over the accuracy. More precisely, R is given by

$$R(\boldsymbol{b}_i) = \|\boldsymbol{b}_i\|_1 - \delta\|\boldsymbol{b}_i - \bar{\boldsymbol{b}}_i\|_1, \tag{3}$$

where $\bar{\boldsymbol{b}}_i$ is the vector of the same size as $\boldsymbol{b}_i$ but with every coefficients equal to the mean of $\boldsymbol{b}_i$. The parameter δ allows to control the balance between pruning and effective quantization. The regularization term is inspired from [9], but instead of using the batch normalization scale factors, it operates directly on the neuron bitwidths. The objective of the regularization term is to separate the bitwidths $\boldsymbol{b}_i$ in two clusters: one to zero and the other to 2^n. This is made possible because the term $-\|\boldsymbol{b}_i - \bar{\boldsymbol{b}}_i\|_1$ reaches its minimum value when half of the $\boldsymbol{b}_i$ coefficients are equal to zero and the other half are equal to 2^n. Additionally, the term $\|\boldsymbol{b}_i\|_1$ penalizes the high bitwidths. So, with this loss function, some $\boldsymbol{b}_i$ coefficients will be pushed near to zero and important weight values will be kept. When it passes under a certain threshold γ, the associated weights of the neurons are removed, this is the pruning part. This allows to remove whole lines of coefficients in the linear layer weight matrix and effectively reduces the memory occupation and the computational complexity of the neural network. Given the size reduction of the output layer, it automatically adjusts the input size of the next layer. So, the pruning allows the effective weight matrix size reductions, without obtaining sparse matrices with only weights pruned.

For recurrent layers, things are a bit more complex because there are multiple weights inside the layers but the idea is the same as the linear layers. For LSTM or Gated Recurrent Unit (GRU) layers, there are multiple weight matrices. The functioning is illustrated with a LSTM layer, but it can be easily adapted with GRU. A LSTM layer is given by

$$\boldsymbol{i}_t = \sigma(\boldsymbol{x}_t \boldsymbol{W}_{ii}^T + \boldsymbol{b}_{ii} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hi}^T + \boldsymbol{b}_{hi}), \tag{4}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{x}_t \boldsymbol{W}_{if}^T + \boldsymbol{b}_{if} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hf}^T + \boldsymbol{b}_{hf}), \tag{5}$$

$$\boldsymbol{g}_t = \tanh(\boldsymbol{x}_t \boldsymbol{W}_{ig}^T + \boldsymbol{b}_{ig} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{hg}^T + \boldsymbol{b}_{hg}), \tag{6}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{x}_t \boldsymbol{W}_{io}^T + \boldsymbol{b}_{io} + \boldsymbol{h}_{t-1} \boldsymbol{W}_{ho}^T + \boldsymbol{b}_{ho}), \tag{7}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t, \tag{8}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t), \tag{9}$$

with σ the sigmoid function, $\boldsymbol{h}_{t-1}$ the previous hidden state and $\odot$ the element-wise multiplication operator. The matrices can not be singularly considered as multiple linear layers because it will imply sparse matrices which are not always hardware friendly. So, in order to prune effectively these layer, weights have to be reunified and pruned together. Thus, we define

$$\boldsymbol{W} = [\boldsymbol{W}_i, \boldsymbol{W}_h] \in \mathcal{M}_{h,4(h+i)}(\mathbb{R}), \tag{10}$$

with $\boldsymbol{W}_i = [\boldsymbol{W}_{ii}, \boldsymbol{W}_{if}, \boldsymbol{W}_{ig}, \boldsymbol{W}_{io}] \in \mathcal{M}_{h,4i}(\mathbb{R})$ and $\boldsymbol{W}_h = [\boldsymbol{W}_{hi}, \boldsymbol{W}_{hf}, \boldsymbol{W}_{hg}, \boldsymbol{W}_{ho}] \in \mathcal{M}_{h,4h}(\mathbb{R})$ where h is the size of the hidden state and i the size of the input. Therefore, with a bitwidth vector $\boldsymbol{b}$ of size h, all the weights are pruned the same way. So if some values of $\boldsymbol{b}$ are below γ, the corresponding lines in $\boldsymbol{W}_i$ and $\boldsymbol{W}_h$ are removed. Moreover, the corresponding rows in $\boldsymbol{W}_h$ are also removed because it is composed of

matrices of sizes of h by h. For the GRU layers, the idea is completely the same with $\boldsymbol{W} \in \mathcal{M}_{h,3(h+i)}(\mathbb{R})$.

C. Low-rank decomposition

There are multiple techniques to perform low-rank decomposition on weight matrices, but in this context the main method used is the Singular Value Decomposition (SVD). In this task, the main issue is the rank choice. Even if methods exist to learn ranks of different weight matrices during training [10], it requires an additive term in the loss function used. As there is already the regularization term and in order to limit the number of extra-parameters, the decomposition rank is empirically chosen. The impact of this choice is measured in the result section.

D. Fine Tuning

At each process stage, fine-tuning is conducted by initiating a new training phase following any modification to the network. This adjustment is crucial for adapting the weights to new conditions such as quantization, pruning, or following low-rank factorization, thereby, ensuring the extraction of optimal performance from the network. The same parameters as in the main training phase are retained, except the λ value is reduced by a factor 5 to avoid large variations of bitwidths.

III. Experiments and results

The experiments were made with medium LSTM described in [11] on PennTreeBank (PTB) language model dataset [12] with a vocabulary size of $V = 10000$. The neural network is composed of an embedding layer, followed by two LSTM layers of size 650, and end with a linear layer of size V. The optimizer used is SGD with an initial learning rate of 1, with a linear decay of 5/6 every epochs from the 6^{th}. The value of δ in the loss is set to $\delta = 0.125$. The γ treshold for the pruning part is set to 1. The impact of the λ value is inspected through different trainings, without LR factorization. Then the impact of chosen ranks during LR factorization is measured and finally some of the obtained networks are kept. Then, LR factorization is performed in order to decrease the memory occupation as well as the computational complexity. The final compressed neural networks are obtained.

TABLE I
λ value impact on PTB dataset and comparison with other neural networks of similar size

λ	BOPS(B)	Param(M)	Size(Mo)	Ppl	LSTM1	LSTM2
0.1	1.887	19.80	19.6	84.8	650	650
0.6	1.017	15.50	14.1	104.6	647	400
0.7	0.277	9.06	9.28	108.7	163	179
0.8	0.070	7.27	7.27	143.4	73	53
0.9	0.004	6.57	6.54	400.5	18	2
[11]	2.888	4.64	18.6	117.7	200	200
[6]	1.018	4.7	4.64	118.2	200	200
[6]	/	4.2	16.8	92.9	650	650
[7]	/	3.31	13.25	109.2	153	153
[7]	/	1.672	6.69	120.2	207	207

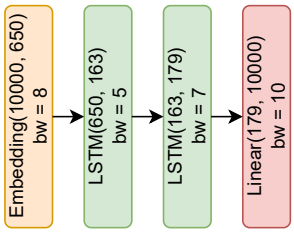

Fig. 2. Illustration of the neural network obtained with $\lambda = 0.7$, bw represents the layer weight bitwidths decided during the training.

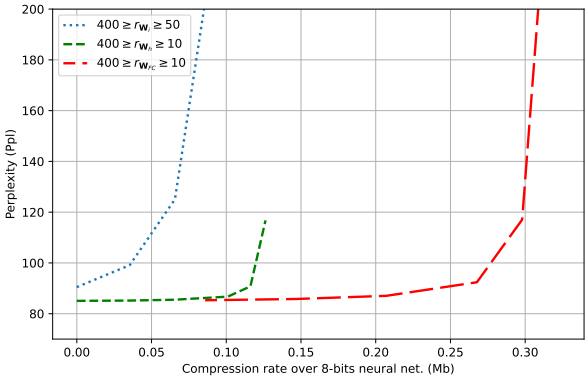

Fig. 3. Perplexity obtained for different weight matrix SVD. It allows to measure the impact of a single weight matrix SVD on performance. $r_{\boldsymbol{W}_i}$, $r_{\boldsymbol{W}_h}$, and $r_{\boldsymbol{W}_{FC}}$ refer to the rank chosen for SVD of $\boldsymbol{W}_i$, $\boldsymbol{W}_h$ and $\boldsymbol{W}_{FC}$, respectively.

A. Impact of λ value

The influence of the λ parameter is detailed in Table I. We compare the number of parameters, bit operations (BOPS) calculated from [13], network size, and perplexity (Ppl) for the approach that excludes low-rank decomposition with networks of similar size. Perplexity is defined as the exponential of the cross entropy loss. Thus, the lower it is, the better. The LSTM sizes are also reported to account for the effective pruning. The λ values ranging from 0.2 to 0.5 did not result in the pruning of any neurons, leading to performance closely mirroring the one observed with $\lambda = 0.1$. Consequently, these values were not included in the Table I. The number of parameters obtained is between 19.8M for no pruning at all and 6.57M. We remind that the embedding layer can not be pruned because the vocabulary size is fixed. Thus, in our implementation there are 6.5M irreducible parameters. The networks obtained with $\lambda = 0.6$ and $\lambda = 0.7$ reach perplexity relatively close to the references for less BOPS. In order to provide greater visibility on the method effect, the neural network trained with $\lambda = 0.7$ is illustrated in Fig. 2. The different bitwidths and size for each layer are represented. In this specific network configuration, all activations are quantized to 8 bits. It is important to note that this is not an observation across all different networks.

TABLE II
RELATIVE PERFORMANCE OF COMPRESSED NEURAL NETWORKS WITH QUANTIZATION, PRUNING, AND LOW-RANK DECOMPOSITION

Model	BOPS	Param	Size	Ppl	$r_{\boldsymbol{W}_h}$	$r_{\boldsymbol{W}_{FC}}$
Fp32	1.00x	1.00x	1.00x	82.0	/	/
$\lambda = 0.1$	7.70x	1.00x	3.98x	84.8	/	/
$\lambda = 0.1$	23.12x	1.71x	6.66x	87.0	100	100
$\lambda = 0.7$	52.35x	2.18x	8.41x	108.7	/	/
$\lambda = 0.7$	54.51x	2.43x	9.51x	113.4	50	100

B. Effect of low-rank decomposition

We first examine the effects of low-rank decomposition with a λ value of 0.1, which results in quantization without pruning. We then explore various LR approximations applied to different weight matrices. Specifically, LR factorization was performed on $\boldsymbol{W}_i$ and $\boldsymbol{W}_h$ the two matrices in an LSTM layer as well as $\mathbf{W}_{FC}$, the matrix in the final linear layer. The embedding layer was excluded from this LR factorization due to our focus on reducing computational complexity, though it could be considered if the goal was to minimize memory usage. The Fig. 3 shows that the lower the rank selected for LR factorization, the greater the perplexity. For higher ranks applied to $\boldsymbol{W}_h$ and $\boldsymbol{W}_{FC}$, perplexity remains almost unchanged. However, this is not the case for $\boldsymbol{W}_i$, where even high-rank approximations significantly impair performance. Thus, while the different matrices can be approximated using LR decompositions without compromising performance to a certain threshold, $\boldsymbol{W}_i$ stands as an exception and will not be decomposed moving forward.

C. Combination of the compression methods

In the concluding part of our results section, we implemented low-rank decomposition on the matrices $\boldsymbol{W}_h$ and $\boldsymbol{W}_{FC}$ from the networks trained with $\lambda = 0.1$ and $\lambda = 0.7$ as detailed in Table II. We then evaluated the overall performance of the compressed neural networks resulting from these modifications. For both configurations, the low-rank decomposition led to reductions in memory usage and computational complexity, although it adversely affected performance. Notably, the network trained with $\lambda = 0.1$ derived greater benefits from the LR decomposition compared to the one trained with $\lambda = 0.7$. This is attributed to the larger sizes of $\boldsymbol{W}_h$ and $\boldsymbol{W}_{FC}$ in the $\lambda = 0.1$ configuration. This process allows to obtain a neural network with 23.12 times less BOPS, almost 7 times lighter and with close performance compared to the float32 model.

IV. CONCLUSION

In this paper, we introduce a novel quantization technique that concurrently supports pruning. This compression approach enables independent quantization and pruning of each layer, eventually followed by low-rank decomposition. It allows to decrease both the memory usage and computational demands of the neural network, with an aim to preserve performance. The regularization term added to the loss function encourages certain bitwidths to approach zero, while preserving those associated with critical neurons. Performance of this process was evaluated on PTB dataset with LSTM-based neural networks. The performance evaluation shows the method effectively operates on quantization and pruning, and can offer a reasonable compromise between computational complexity, memory occupation, and perplexity of the network. In future works, it could be interesting to measure performance of this process on different sizes and types of neural networks, and properly evaluate the impact of the δ parameter.

REFERENCES

[1] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*. Chapman and Hall/CRC, 2022, pp. 291–326.

[2] R. Reed, "Pruning algorithms-a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[3] L. T. Nguyen, J. Kim, and B. Shim, "Low-rank matrix completion: A contemporary survey," *IEEE Access*, vol. 7, pp. 94 215–94 237, 2019.

[4] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[5] G. Toderici, D. Vincent, N. Johnston, S. Jin Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[6] A. M. Grachev, D. I. Ignatov, and A. V. Savchenko, "Compression of recurrent neural networks for efficient language modeling," *Applied Soft Computing*, vol. 79, p. 354–362, Jun. 2019. [Online]. Available: http://dx.doi.org/10.1016/j.asoc.2019.03.057

[7] N. Chirkova, E. Lobacheva, and D. Vetrov, "Bayesian compression for natural language processing," 2018.

[8] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2020.

[9] T. Zhuang, Z. Zhang, Y. Huang, X. Zeng, K. Shuang, and X. Li, "Neuron-level structured pruning using polarization regularizer," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 9865–9877.

[10] Y. Idelbayev and M. Carreira-Perpiñán, "Low-rank compression of neural nets: Learning the rank of each layer," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 8046–8056.

[11] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," 2014. [Online]. Available: https://arxiv.org/abs/1409.2329

[12] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. Interspeech 2010*, 2010, pp. 1045–1048.

[13] P. Freire, S. Srivallapanondh, A. Napoli, J. E. Prilepsky, and S. K. Turitsyn, "Computational complexity evaluation of neural network applications in signal processing," 2024.