Model Compression for Data Compression: Neural Network Based Lossless Compressor Made Practical

Liang Qin, Jie Sun

Theory Lab, Central Research Institute,
2012 Labs, Huawei Technology Co., Ltd.,
Hong Kong SAR
qin.liang@huawei.com,j.sun@huawei.com

Abstract

In the recent few years, lossless compressors based on deep learning emerged as a new research direction. However, existing DL compressors focus mostly on compression ratio at the expense of compression speed (typically 10-1000 times slower than popular compressors, operating at the KB/s level), rendering them impractical to be deployed in industrial settings. To address the throughput issue of network inference, we propose to utilize model compression in DL-based lossless data compression. Our approach originates from the sparse learning framework to flexibly balance the model complexity and the compression ratio. Through systematic numerical experiments, we found that compressed models with only tens of thousands of parameters can still retain competitive compression ratios comparing with the original large models which are typically 100 times larger. The proposed scheme reveals novel properties concerning deep compression models and data—logarithmic relations between the model size and compressed data size for general mixed data, and a turning point indicating minimum representation size for relatively formatted data.

Introduction

Algorithms for data compression have been around for almost as long as information theory, initially focusing on entropy-coding (Shannon codes, Huffman codes, arithmetic coding) and later advancing into more general statistics-based approaches the most notable being the LZ series. The first time when machine learning met compression was in 1996 [1] where a data-driven compression scheme was established. Given the high flexibility in artificial NN, the DL-based compressor was improved by exploring the network architecture [2]. Those NNs were initially proposed as a general replacement of various models in the context mixing class of traditional compressors such as PAQ[1] and CMIX[2]. However, the game changed as extensive supporting platforms and hardware emerged in the last decade. DeepZip [3] as well as its adaptive version DZip [4] adapted long short-term memory (LSTM) and gated recurrent units (GRU) as their building blocks of a deep compressor. Partly taking the fruit from natural language processing, NNCP [5] generalized the SOTA LSTM and attention-based autoregressive network, whose third version[3] achieved a record 0.882 bits per character in compressing the enwik9 benchmark, the best result ever reached.

[1] http://mattmahoney.net/dc/zpaq.html
[2] http://www.byronknoll.com/cmix.html
[3] http://mattmahoney.net/dc/text.html#1102

On the other hand, deep generative models as a big breakthrough of the 21st century in DL, also provide a systematic methodology for data compression. Volume-preserving flow model [6, 7] and variational autoencoder [8] stood out and became two successful lossless compression approaches, respectively based on injective and stochastic mappings between real and latent spaces. Such a connection extends how people think of compression, although their compression rates are reportedly poorer than autoregressive models.

Since most deep autoregressive compressors originate from AI tasks that emphasize model accuracy, previous work focuses mostly on adopting such models to achieve high compression ratio. However, they come at the cost of slowing down the compression throughput. For instance, the NNCP compressor reaches an extreme compression capability but can only process data at the level of KB/s, making them impractical for daily use. In fact, large-scale deployment relies critically on a compressor being able to achieve a balance between its throughput and compression rate, a tradeoff often underexplored in the existing literature of AI-based data compression. To fill in this gap, lightweight models at ∼100MB/s inference speed and compatible GPU-based entropy encoders are indispensible for a practical DL-based compressor, both of which are un-derexplored in the existing literature. To address the former, we develop in this paper a systematic balancing method derived from network reduction technology, which is in principle applicable to any type of data.

Our main contributions in this paper can be summarized as follows. First, we propose model compression as a way to achieve efficient data compression. A complete pretraining scheme Under the lasso framework is developed to gain the relation between model size and compressed data size. We solve pruning-related dynamical issues via theoretical analyses and experiments, to achieve homogeneous structure reduction. The results show that DL-based compressors can operate with refined structures that lead to practical inference speed.

Lossless Compression with Neural Networkss

Consider a sequence of data $\boldsymbol{x} = (x_1, x_2, ..., x_N)$ of length N where $x_i \in \mathcal{S}$ of cardinality $|\mathcal{S}| \leq 2^K$ (typically $K = 8$ meaning that each original symbol occupies a single byte). The goal for lossless compression is to find an invertible transformation $F : \boldsymbol{x} \to \boldsymbol{z} = (z_1, \ldots, z_M)$ with $z_i \in \{0, 1\}$. The compression ratio is computed as $\rho(\boldsymbol{x}, \boldsymbol{z}) = \frac{NK}{M}$, with the larger the value the more compressed the original data has become; an alternative quantification is *bits per character* (bpc), defined as $\text{bpc}(\boldsymbol{x}, \boldsymbol{z}) = \frac{8M}{NK}$.

Neural Network Predictor

The autoregressive approach treats the joint probability as the product of conditional probabilities of each symbol over all previous symbols

$$\Pr(\mathbf{x}) = \prod_{i=1}^{N} \Pr_i(x_i) = \prod_{i=1}^{N} \Pr(x_i | x_1 ... x_{i-1}) \tag{1}$$

where the $i = 1$ term should be understood as the prior likelihood of a string starting with x_1. A typical autoregressive compressor requires the predictor to output all N distributions $\Pr_i(\cdot)$ with $i = 1, ..., N$ each represented by $|\mathcal{S}|$ positive real numbers that sum up to 1.

As long as all distributions $\Pr_i(\cdot)$ and the true sequence $\mathbf{x}$ are known to the entropy encoder, it can encode $\mathbf{x}$ into a bitstream $\mathbf{z}$, with the minimum code length

$$L = -\sum_{i=1}^{N} \log \Pr_i(x_i). \tag{2}$$

Machine learning parametrizes the above conditional probability with learnable coefficients $\boldsymbol{\theta}$ so that $\Pr(\cdot|x_1...x_{i-1}, \boldsymbol{\theta})$ can be easily computed. Minimizing the target sequence code length coincides with the training objective

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta} \in \mathcal{H}} L(\boldsymbol{\theta}) \tag{3}$$

where $\mathcal{H}$ is the hypothesis class of $\boldsymbol{\theta}$.

Many predictors such as feedforward NNs and those in PAQ and CMIX effectively truncate the input of $\Pr_i(\cdot)$ to previous k symbols, so that

$$\Pr(\mathbf{x}) = \prod_{i=1}^{N} \Pr_i(x_i) \approx \prod_{i=1}^{N} \Pr\left(x_i | x_{\max(1,i-k)}...x_{i-1}\right) \tag{4}$$

and k serves as an algorithmic tunable parameter that indicates the correlation length of the data.

Entropy Encoder

Entropy encoders represent $\mathbf{x}$ by a bitstream with the information of $\Pr_i(\cdot)$. The popular Huffman coding [9] encodes symbols separately with code length given by equation 2 and at most one redundant bit for each symbol. To circumvent the additional bit, the arithmetic code (AC) expresses $\mathbf{x}$ with an interval of length $\Pr(\mathbf{x})$ within $(0, 1)$ and the overall extra-bit length is a constant. Nevertheless, adaptive quantization and updating the segment in arithmetic code is still time-consuming for compressors of daily use. In 2009, the asymmetric numeral system (ANS) was proposed [10] to achieve both the optimal bit length and fast speed so long as the symbol probabilities are static. Although entropy encoders have achieved throughputs much faster than our final model inference speed, they mostly operate on CPU while our NN model requires special processing units for acceptable speeds which in the experiments is an NVIDIA V100 training card. A fully GPU-based entropy encoder compatible with our pruned NN models is beyond the scope of this paper, and we leave this demand in the future work and use a CPU version for end-to-end compression and decompression tests.

Methods

Despite being able to reach high compression ratios, the obvious drawback of using neural networks as the predictor is that artificial neural networks typically contain

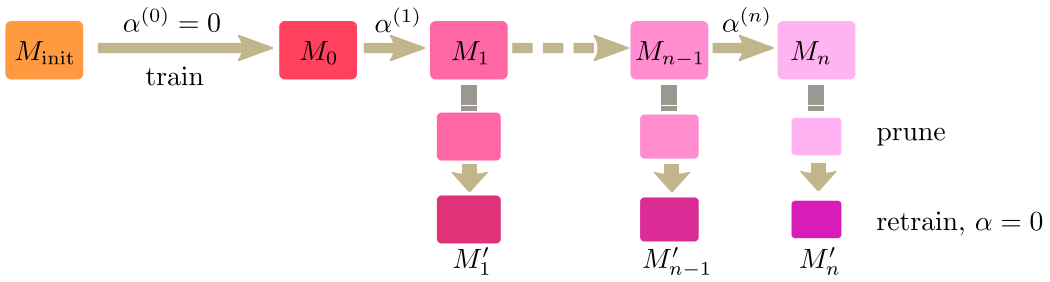

Figure 1: Three-step model reduction process with the outcome models at the bottom. We continue training a model M_0 with an increasing lasso term, and trim all suppressed nodes in the meanwhile. Then the trimmed networks are pushed to retrain without the presence of the lasso term.

way too many parameters, causing significantly more computation flops than practically used algorithms. Model compression technology was proposed to reduce the memory usage of usually giant NN models, in order to accommodate limited storage, communication bandwidth, computation and power, especially in portable devices.

Approaches for model compression approaches include weight sharing, pruning, quantization and distillation. We choose structured model pruning as the first trial toward a practical deep compressor because it depends less on the hardware architecture. People have proposed a variety of model pruning frameworks [11] starting with greedy deletions, and a more systematic way is by penalizing non-zero weights with additional regularization terms in loss function such as L_1 and L_2 norms [12]. Besides punishing individual edges, structured pruning imposes penalties for network nodes, filters or layers, making the outcome network faster without specialized platform support.

Lasso for DL-based Compression

In this work we adopt the lasso method for the scenario of DL-based compression as it is known as the standard pruning scheme [13–15] for both structured and non-structured deletion. The ordinary lasso introduces L_1 regularization while group lasso imposes additional terms that are proportional to the L_2 norm (square root of L_2 regularization) of parameters in each group

$$\mathcal{L} = L + \sum_{g \in \mathcal{G}} \alpha_g \sqrt{\sum_{w \in g} w^2}, \tag{5}$$

where L comes from equation 2 and $\mathcal{G}$ denotes the set of lasso groups. Each group in $\mathcal{G}$ defines a NN structure whose removal is believed to bring significant acceleration. In this work we perform only node pruning and figure 2 illustrates typical lasso groups for our tested NN structure. The lasso L_2 form ensures its continuity as well as non-zero gradient around the point where $w = 0, \forall w \in g$ given a group g, so it leads to a local minimum at zero.

Pruning after Pretraining

Three DL-based compression schemes exist in the literature with varied model generalizability [4]. In the static way, a model is trained over samples of one data type (e.g., enwik8), and is applicable only to compressing this data type (e.g., all English text) but the model size is regarded as part of the algorithm. The semi-adaptive way trains the model exactly on the data to be compressed, and the model must be saved in the compressed stream. The third one is adaptive compression in which we scan the sequence and the NN model updates at the same time of compression. We choose the static approach to demonstrate our pruning method though it also works for the semi-adaptive way.

As for the pruning procedure, people typically perform it after the training converges. Other schemes were also proposed such as periodic pruning [16]. We first optimize under the cross-entropy L and then increment lasso terms after each epoch. Any node with a noise level less than the predetermined threshold will be removed from the network, and we will obtain a series of pruned models of decreasing sizes. Finally, all the pruned models have to be retrained using the same data as training. The whole procedure is given in figure .

Lasso Dynamics

Weight Grouping

Figure 2 illustrates lasso groups in our predicting NN. Choosing either the left-side or right-side weights is usually satisfying as one latent node's lasso group. However, due to the increasing popularity of rectified linear units $\mathrm{ReLU}(x) = \max(x, 0)$, we propose double-sided lasso groups for pruning latent nodes. Let p be a latent node with bias b, $\mathbf{w}_i$ be incoming weights, and $\mathbf{w}_o$ be outgoing weights. The following equation holds for any node with ReLU activation

$$\mathrm{ReLU}\left(\alpha \mathbf{w}_i^\top \cdot \mathbf{v} + b\right) = \alpha \mathrm{ReLU}\left(\mathbf{w}_i^\top \cdot \mathbf{v} + b/\alpha\right) \tag{6}$$

where elements of $\mathbf{v}$ are outputs of the last latent layer. Thus for the total loss function

$$L\left(\alpha \mathbf{w}_i, b, \mathbf{w}_o\right) = L\left(\mathbf{w}_i, b/\alpha, \alpha \mathbf{w}_o\right). \tag{7}$$

Such invariance also approximately applies to other activation functions around linear regions defined by equation 6, which means that penalizing one side of edges can be compensated by enhence the other side. Including both sides of weights in the node's lasso group can suppress the above invariant transform that escapes node penalties. In our experiments, we introduce two weight groups for one latent node, the left-side weights $g^{(l)}_{\mathrm{latent}}$ and the right-side weights $g^{(r)}_{\mathrm{latent}}$, as illustated in figure 2.

Lasso Prefactor

We assign the same prefactor α_g to identical structures, such as nodes of the same latent layer, while the ratio between different structures remains flexible. The simplest choice is to use a constant prefactor

$$\alpha_g = \alpha. \tag{8}$$

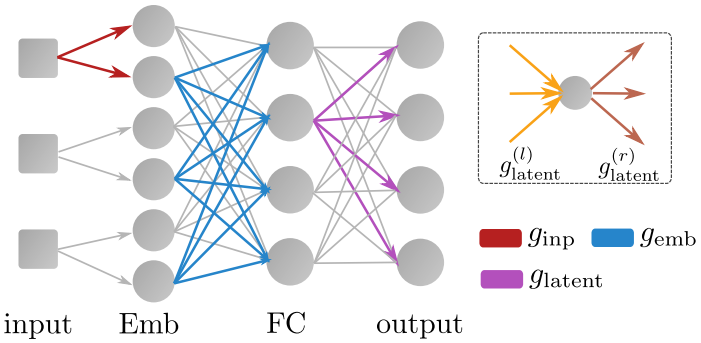

Figure 2: Examples of lasso groups in a deep neural network consisting of one embedding layer and one latent layer. Three types of lasso groups are colored corresponding to removing an input node, an embedding dimension and a latent node respectively. *Inset*: left group and right group of an intermediate latent node.

Published works [14, 15] also propose prefactors that are proportional to group cardinalities

$$\alpha_g = \alpha|g| \tag{9}$$

so that the magnitude of the derivative with respect to $w_i \in g$ remains unchanged with varied size of g.

As the training dynamics on practical NNs are difficult to analyze, the preference of one prefactor convention is mostly empirical. Both options converge to the minimum network whereas under the constant prefactor all structures shrink in a more simultaneous and stable manner. Thus we choose the constant convention for later experimental settings.

Pruning Threshold

The training of a NN usually converges to a bounded motion around the minimum for the stochastic nature of mini-batch dynamics and optimization finite step length. Hence thresholds are needed as indicators for zero-weight groups. Such thresholds should be greater than, but of the same order of the noise level of group weights around the minimum. We simulate the optimizing process of the lasso term alone without the real loss, for the real part becomes negligible with increasing α. We conduct an experiment of a vector $\mathbf{w}$ optimized under the lasso potential $\alpha\|\mathbf{w}\|_2 = \alpha\left(\sum w^2\right)^{\frac{1}{2}}$ ($\mathbf{w}$ is treated as a single lasso group), and choose the root mean square of a lasso group as an indicator of its zero level, i.e., $\mathcal{S}_g = \sqrt{\frac{1}{|g|}\sum_{w\in g} w^2}$.

We find that $\mathcal{S}_g$ falls and then fluctuates around zero and its expectation is affected by the prefactor α_g, learning rate γ and the dimension of $\mathbf{w}$. It is straightforward to obtain analytical results when stochastic gradient descent (SGD) is employed

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \gamma\nabla\left(\alpha\|\mathbf{w}(t)\|_2\right) = \left(1 - \frac{\alpha\gamma}{\|\mathbf{w}(t)\|_2}\right)\mathbf{w}(t), \tag{10}$$

which means $\mathbf{w}(t)$ is parallel with $\mathbf{w}(t+1)$ and $\|\mathbf{w}(t+1)\|_2 = |\|\mathbf{w}(t)\|_2 - \alpha\gamma|$. Hence if $\|\mathbf{w}(t)\|_2 \leq \alpha\gamma$, $\mathbf{w}(t+2) = \mathbf{w}(t)$. It concludes that the noise around the L_2 minimum

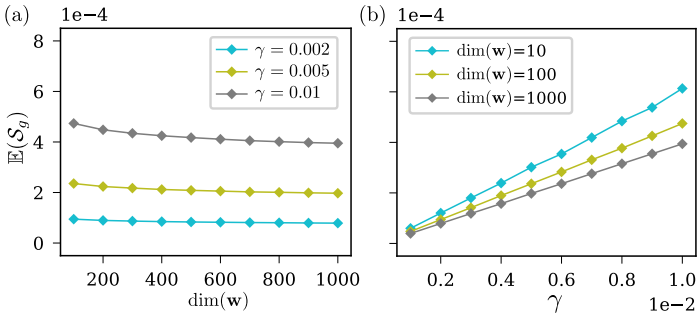

Figure 3: Expected values of noise magnitude $\mathcal{S}_g$ for a free vector $\mathbf{w}$ under L_2-form potential and Adam optimizer. Experiments have been repeated and errors are less than the marker size. (a) is for variable $\mathbf{w}$ dimension and (b) is for variable learning rate γ.

under SGD is

$$\mathbb{E}(\mathcal{S}_{g,\mathrm{SGD}}) = \frac{\alpha\gamma}{2\sqrt{|g|}}. \tag{11}$$

Other optimizers have more complicated mechanisms so we measure $\mathbb{E}(\mathcal{S}_g)$ experimentally for the popular Adam optimizer [17] without weight decay. We find that the potential strength does not affect the fluctuation, and for other factors, figure 3 shows the noise L_2-norm expectation for variable learning rates and dimensions. A good estimate of $\mathbb{E}(\mathcal{S}_g)$ is

$$\mathbb{E}(\mathcal{S}_{g,\mathrm{Adam}}) \approx 0.5\gamma. \tag{12}$$

Numerical Experiments

	size /	$\lvert\mathcal{S}\rvert$ /	gzip 19MB/s	DZip 85KB/s	description
enwik8	96M	205	2.92	1.80	English Wikipedia in XML format
text8	96M	27	2.65	1.74	English text extracted from Wikipedia
chr1	238M	5	2.06	1.67	H. Sapiens GRCh38 sequence
c.e.gene	96M	4	2.24	1.82	C. elegans whole genome sequence
qs36	100M	38	3.74	2.94	quality scores (36 bytes per read) of a human genome sample
qs148	140M	39	2.48	1.54	quality scores (148 bytes per read) of genome sample NA12878
Mozilla	49M	256	2.98	2.65	tarred executables of Mozilla 1.0

Table 1: Realistic datasets used in experiments. The columns Gzip and DZip record the resulting bpc using corresponding compressors, with their typical compression throughputs shown in the second row. DZip's ratio and speed here are different from the original paper because static compression strategy and independent test data are employed. chr1 and c.e.gene are introduced in DZip.

We conduct experiments on realistic datasets. The details of the datasets are listed in table 1. The seven datasets cover four classes: English texts, genome sequence,

quality score in genome sequencing and mixed data. Quality scores are integer ranging from 30 to 70 encoded each with a single byte, resembling time series. For each dataset, 4/5 is used for training and evaluation while the rest is for tests.

As for the NN model, we take a feedforward NN with two fully connected latent layers each with 512 initial nodes. A final layer fully connected to the second latent layer is present with nodes equal to $|\mathcal{S}|$ and is equipped with a softmax filter to output $\Pr_t(\cdot)$. The inputs are treated in bytes, and the $k-$byte sequence $x_{t-k}...x_{t-1}$ will be taken as the input k nodes. A learnable embedding layer is inserted between input nodes and the first latent layer mapping a byte onto a D_{emb}-dimensional vector. Throughout our numerical tests, $k = 16$ is fixed, because it exhibits more complicated pruning dynamics if we apply lasso terms to input nodes.

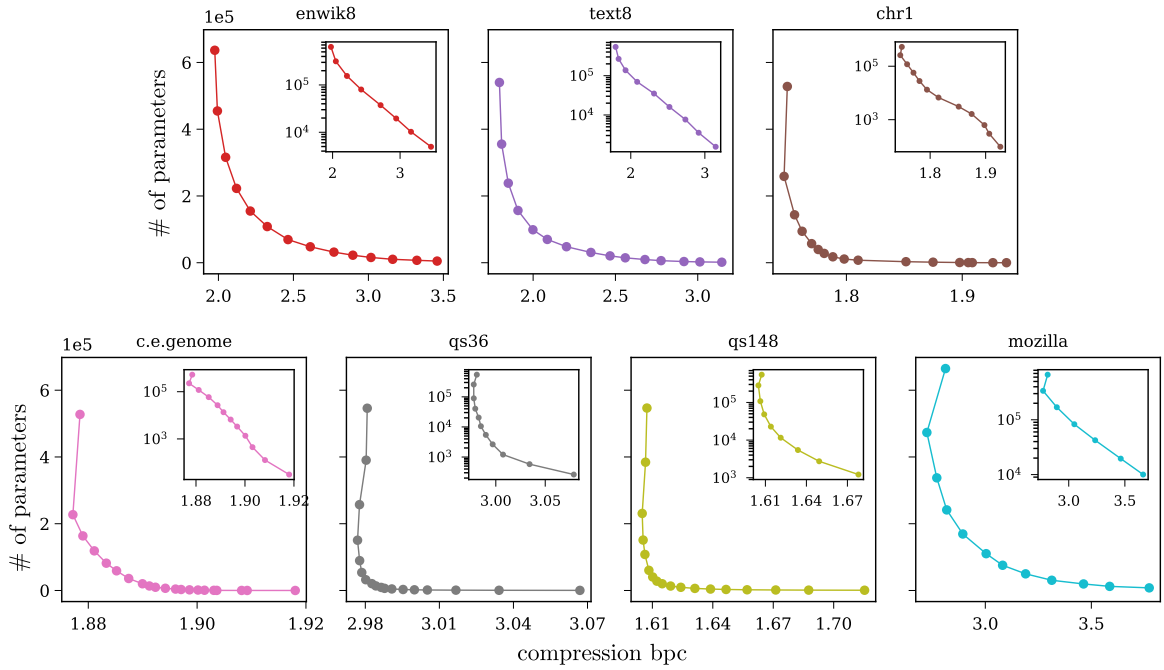

Figure 4: Trade-off curves between the model complexity and the compression rates. Each panel represents one dataset. Insets are the same curves on logarithmic scales.

We then perform the sparsity learning on the training data of the above datasets. 8192 batches are run in parallel on the NVIDIA Tesla V100 training card and PyTorch platform. When the Adam optimizer is employed with a learning rate $\gamma = 0.002$, 32 epochs are adequate for all data types to converge without lasso terms. Then, group lasso losses are introduced by setting $\alpha_g = 5 \times 10^{-7} t^2$ where t is the epoch since the lasso losses are present. Each latent node is attached with two lasso groups involving incoming or outgoing edges, and the input-node pruning is turned off. Any node whose root mean square falls below 10^{-3} is removed from the network. A series of sparse models are produced as α increases. We retrain all pruned NNs with only the cross-entropy loss for a maximum of 10 epochs, then evaluate for their final performance on the test sets.

The initial models before pruning have typically 5×10^5 parameters and reach

$\sim$5MB/s inference speed on GPU. Their compression ratios are comparable with DZip in genomic datasets, and are lower than DZip for mixed texts such as enwik8, text8 and Mozilla. The results are expected because DZip utilizes combined recurrent units and linear regression, while we choose only linear layers for the pruning to work.

Figure 4 shows the sparsity-quality tradeoff. Compressed file sizes rise as model sizes shrink as expected, except for the large-model end (upper-left corners) which can be explained by better generalizability introduced by the lasso regularization. NNs of $\sim 10^6$ parameters were proposed for autoregressive DL-based compressors [3], while we find that much smaller NNs are adequate without losing compression performance. For English texts, 10^5-parameter NNs that take only 0.4 MBs storage still outperform traditional popular compressors. For genome sequence chr1, a network as small as 10^4 parameters is only 3% worse than one with 3×10^5 parameters.

More interesting results are present in the curves for qs36 and qs148 that both take a characteristic L-shape. A turning point is manifest around $N_p = 10^3 \sim 10^4$, and above this point the model size drops vertically without losing compression quality. What is special about datasets qs36 and qs148 is that they are actually one-dimensional integer values featuring limited patterns, so there exist countable rules to determine the distribution of the next value. Our model compression reveals the minimum NN-representation of such rules though efforts are still needed to bridge the gap between NN-representations and explicit formulas.

The trends in texts and mixed data are monotonic, and any reduction in the model size will result in a lower compression rate. Their curves in logarithmic scale exhibit apparent linearity, which implies the following relation

$$M \approx -b \log Q + c \tag{13}$$

where Q is the model size, M is the compressed file size and (b, c) is a pair of positive constants. People have established universal approximation theorems for various NNs [18, 19] which states that large enough NN models can approximate any continuous function, while the loss in accuracy with limited NN size remains unanswered.

The nature of structural pruning makes the results easy for network acceleration. The inference speed of our tests for a NN of 4×10^4 parameters can achieve $\sim$50 MB/s on NVIDIA Tesla V100 if batch parallelization is fully exploited, much faster than the speed reported in DeepZip [3] which is of KB/s level. The end-to-end compression speed is of the same magnitude while decompression speed drops significantly, supposedly due to heavy CPU-GPU communication during decompression in our implementation.

Conclusions

To make DL-based compressors practical, we propose to utilize sparse learning to compress a trained NN model which leads to a condensed network model that runs orders of magnitude faster than the original prediction model. The proposed framework enables to flexibly balance the compression ratio and throughput through tuning sparsity parameters. For various data types including text, genomes and Mozilla, the experimental results exhibit a minimum model size of only thousands of parameters,

with the same representability as large NNs; for mixed data, an interesting logarithmic relation between model and compressed data sizes is evident. These findings indicate a promising path to develop small-sized neural networks for lossless compression.

References

[1] J Schmidhuber and S Heil, "Sequential neural text compression.," *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 7, no. 1, pp. 142–6, 1996.

[2] Matthew V Mahoney, "Fast text compression with neural networks.," in *FLAIRS conference*, 2000, pp. 230–234.

[3] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa, "Deepzip: Lossless data compression using recurrent neural networks," *arXiv preprint arXiv:1811.08162*, 2018.

[4] Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, and Idoia Ochoa, "Dzip: improved general-purpose loss less compression based on novel neural network modeling," in *2021 Data Compression Conference (DCC)*. IEEE, 2021, pp. 153–162.

[5] Fabrice Bellard, "Lossless data compression with neural networks," 2019.

[6] Emiel Hoogeboom, Jorn WT Peters, Rianne van den Berg, and Max Welling, "Integer discrete flows and lossless compression," *arXiv preprint arXiv:1905.07376*, 2019.

[7] Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole, "Discrete flows: Invertible generative models of discrete data," *Advances in Neural Information Processing Systems*, vol. 32, pp. 14719–14728, 2019.

[8] James Townsend, Tom Bird, and David Barber, "Practical lossless compression with latent variables using bits back coding," *arXiv preprint arXiv:1901.04866*, 2019.

[9] David A Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[10] Jarek Duda, "Asymmetric numeral systems," *arXiv preprint arXiv:0902.0271*, 2009.

[11] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag, "What is the state of neural network pruning?," *arXiv preprint arXiv:2003.03033*, 2020.

[12] Song Han, Jeff Pool, John Tran, and William J Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.

[13] Robert Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[14] Ming Yuan and Yi Lin, "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 68, no. 1, pp. 49–67, 2006.

[15] Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini, "Group sparse regularization for deep neural networks," *Neurocomputing*, vol. 241, pp. 81–89, 2017.

[16] Trevor Gale, Erich Elsen, and Sara Hooker, "The state of sparsity in deep neural networks," *arXiv preprint arXiv:1902.09574*, 2019.

[17] Diederik Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[18] George Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[19] Ding-Xuan Zhou, "Universality of deep convolutional neural networks," *Applied and computational harmonic analysis*, vol. 48, no. 2, pp. 787–794, 2020.