Estimating Lossy Compressibility of Scientific Data Using Deep Neural Networks

Zhenlu Qin, Jinzhen Wang, Qing Liu, Jieyang Chen, Dave Pugmire, Norbert Podhorszki, and Scott Klasky

Abstract—Simulation based scientific applications generate increasingly large amounts of data on high-performance computing (HPC) systems. To allow data to be stored and analyzed efficiently, data compression is often utilized to reduce the volume and velocity of data. However, a question often raised by domain scientists is the level of compression that can be expected so that they can make more informed decisions, balancing between accuracy and performance. In this letter, we propose a deep neural network based approach for estimating the compressibility of scientific data. To train the neural network, we build both general features as well as compressor-specific features so that the characteristics of both data and lossy compressors are captured in training. Our approach is demonstrated to outperform a prior analytical model as well as a sampling based approach in the case of a biased estimation, i.e., for SZ. However, for the unbiased estimation (i.e., ZFP), the sampling based approach yields the best accuracy, despite the high overhead involved in sampling the target dataset.

Index Terms—High-performance computing, data reduction, deep learning

1 Introduction

Simulation based scientific applications generate increasingly large amounts of data on high-performance computing (HPC) systems. To dramatically reduce the storage footprint and allow data analysis to be done more efficiently, data compression is often utilized to reduce the volume and velocity of data prior to data at rest. Recent effort in lossy compression managed to achieve much higher compression ratios than lossless compression by allowing for controlled information loss. Nevertheless, the performance outcome of lossy compression is highly variable, and is shown to be both data and compressor dependent. As a result, a question often raised by domain scientists is the level of compression that can be expected for a given error tolerance so that they can make more informed decisions, balancing between accuracy and performance. Unfortunately, without testing compression on the data, which is computationally expensive for large datasets, it is often hard to give a quantitative answer even for compressor developers. In this paper, we aim to address the challenge of estimating the compressibility of scientific datasets leveraging deep neural networks, and make comparisons with a gray-box analytical model and a sampling based approach.

For the deep learning based approach, the idea is to train a deep neural network with a set of features that captures the characteristics of both data and compressor, and learn the intrinsic relationships between features and compression performance. In particular, we extract two types of features that impact the overall compression performance: 1) the statistics of data to measure the smoothness of data, a primary indicator of compressibility (Section 3.2.1); 2) compressor related features to capture the inner mechanisms that a compressor takes to reduce data. This is driven by the fact that loss compressors are designed differently. For example, ZFP [1] uses a transformation based method to discard high-frequency components followed by the embedded encoding, while SZ [2] is based upon curve-fitting and Huffman tree encoding. These differences in design lead to the discrepancy in the performance outcome among compressors, and therefore need to be accounted for in the performance estimation. In particular, we capture hit ratio, quantization intervals, Huffman tree related features for SZ, and bits per bitplane and maximum precision for ZFP (Section 3.2.2).

This work further compares the deep learning based approach with a gray-box analytical model [3], which extrapolates the compression ratio from a base error bound (e.g., 10^{-9}) to a target error bound for a given dataset. In particular, the idea is to collect the distribution of compression features, and use the similarity of these features across error bounds to further derive the compression ratios. Meanwhile, as a baseline, we also obtain the compression ratios using a sampling based approach [4]. The major contributions of this paper are as follows:

- This paper is among the first to use deep learning to understand HPC performance, particularly that of data reduction. In particular, we have shown that when compressor related features are incorporated in training, the prediction performance can be significantly improved.
- Overall, the analytical model achieves the worst performance, as a result of the approximation of compression features across error bounds. For the sampling based estimation, if it is a biased estimation due to the lack of the bounded locality (e.g., SZ) [4], the deep model outperforms the sampling based approach. In contrast, for an unbiased estimation (e.g., ZFP), the sampling based approach achieves the best accuracy, despite that sampling needs to be done on a per dataset basis.

The remainder of this paper is organized as follows. Section 2 provides the background and motivation. Section 3 presents the design of compressibility estimation using deep neural networks, with a focus on extracting meaningful features that will be used for training. Section 4 discusses the evaluation results, along with conclusions in Section 5.

2 Background and Motivation

2.1 Background

Data Reduction. Data reduction is deemed to be an integral step in scientific processes to bridge the gap between compute and I/O, and allow science to be done within the resource constraints. Depending on whether there is information lost during the process, data reduction is generally categorized into lossless compression, such as FPC [5], GZIP [6], and lossy compression, such as ZFP [1] and SZ [2]. In particular, ZFP partitions data into blocks, with each block converted into a set of floating-point mantissas along with a common exponent. Next, the floating-point mantissas are converted to fixed-point values and then taken into a reversible orthogonal transformation, resulting in the transform coefficients that are highly compressible. Meanwhile, SZ is based upon a hybrid Lorenzo curve-fitting scheme. For those data points that are curve hit, they will be further quantized and encoded using Huffman tree. For those that are curve missed, binary representation analysis is used to further compress the data.

Deep Neural Network. A deep neural network typically consists of many layers of non-linear functions in order to learn the complex structure and pattern from large datasets [7]. It is commonly used in both classification and regression. The former is to determine the category that a new sample belongs to, based upon the

- *Z. Qin, J. Wang, and Q. Liu are with the New Jersey Institute of Technology, Newark, NJ 07102. E-mail: {zq53, jw447, qing.liu}@njit.edu.*
- *J. Chen, D. Pugmire, N. Podhorszki, and S. Klasky are with the Oak Ridge National Laboratory, Oak Ridge, TN 37830. E-mail: {chenj3, pugmire, pnorbert, klasky}@ornl.gov.*

Manuscript received 17 Dec. 2019; revised 2 Feb. 2020; accepted 2 Feb. 2020. Date of publication 6 Feb. 2020; date of current version 2 Mar. 2020.
(Corresponding author: Qing Liu.)
Recommended for acceptance by G. Grider.
Digital Object Identifier no. 10.1109/LOCS.2020.2971940

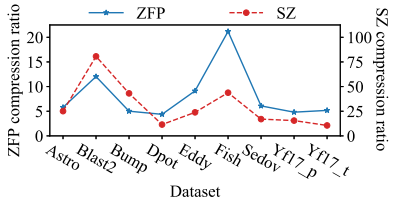

Fig. 1. Compression ratios across datasets. Note that the relative error bound is set to 10^{-3}, and the scales of ZFP and SZ differ.

knowledge of existing samples, whereas the latter is interested in the continuous decisions, e.g., the compression ratios in the context of this paper. In each layer, the output of a neuron is a weighted sum of its input values and bias, transformed by a non-linear activation function (e.g., Relu). A neural network tunes its parameters, i.e., weights and bias, by minimizing the cost function (e.g., root mean square error) that measures the error between the output values and the desired values. The error is further backpropagated through the network and the direction with gradient descent will be explored to reduce the error.

2.2 Motivation

The performance outcome of scientific data compression is highly variable across compressors and datasets. Fig. 1 shows the compression ratios across 9 datasets for both ZFP and SZ. As a result, it is beneficial to understand the compression performance prior to the run, so that domain scientists can make more informed decision—they may forgo the compression completely if the compression ratio is too low. Due to the space constraint, interested readers are referred to our prior work [4] for the details of these datasets.

In our recent work [4], we use the compression ratio of sampled data to extrapolate that of the full data, inspired by the fact that the sample data and full data maintain highly similar characteristics. Despite the good accuracy achieved by this approach, it involves substantial sampling overhead on a per dataset basis. In comparison, this work only takes features, which are substantially smaller, as input to the neural network for estimation. In addition, the previous approach is a white-box approach and needs to model the sophisticated inner mechanisms of a compressor, while this work is overall a gray-box approach that requires limited knowledge of the compressor itself.

In our recent work [3], we developed an analytical model to achieve the cross error bound prediction from a base error bound to a target error bound. A key observation that enables this estimation is the statistical similarity of compression features (e.g., *QuantInterval*) across error bounds. In our experiments, we set the base error bound to 10^{-9} and use the distribution of compression features at the base to extrapolate the compression ratios at other error bounds. While this approach does not involve sampling or training, the estimation is relatively inaccurate, as shown later, since the goal is limited to understanding the overall trend, as opposed to the individual compression ratios.

3 Design

3.1 Goal

In this paper, we aim to use deep learning to understand and predict the compressibility of a dataset for a given lossy compressor. Our goal is twofold: 1) *compressibility classification*, where we classify a dataset into either compressible or incompressible. In particular, if the compression ratio achieved is higher than a prescribed threshold, we label the dataset as compressible. Otherwise, it is labeled as incompressible. 2) *compression ratio regression*, where we aim to estimate the compression ratio through learning the non-linearity of lossy compression. This is motivated by the need to understand the trade-off between compression ratio and error tolerance. For example, it would be very appealing for a user to relax the error tolerance, if the compression ratio can be drastically improved. Note that we take features of the target data as input to the deep neural network in order to make estimation, and thus this work allows one to dynamically select the compressor (SZ or ZFP) that achieves a higher compression ratio on a per dataset basis, similar to prior work [8].

3.2 Feature Extraction

In this section, we discuss the the rationale behind the features that are selected for training. Overall the features include both general features as well as compressor-specific features, with the former capturing the data characteristics as well as the compression parameters, while the latter capturing the inner mechanisms used for compression.

3.2.1 General Features

For lossy compressors, error bound is a key parameter that affects the outcome of compression. Intuitively, the looser the error bound is, the larger the compression ratio will be, despite that the counter-intuitive cases were also occasionally observed [4]. In order to choose a set of error bounds covering the full range of a dataset without bias, we use the relative error bound in this paper. Note that since the native ZFP only supports the absolute error bound from the library interface, we use the product of the relative error bound and the root mean square (RMS) as the input to ZFP when measuring the real compression ratio [3].

Meanwhile, it is evident that data characteristics also play an important role in data compression—a smoother dataset is in general more compressible than a dataset that fluctuates. Here we compute simple statistics to reflect data characteristics, including mean, variance, standard deviation, quartiles, as well as histograms, for this purpose.

3.2.2 Compressor-Specific Features

As shown in Fig. 1, the compression performance is highly dependent on the compressor chosen. For example, ZFP is block transform based compression and uses the embedded encoding for each bit plane, while SZ is based upon curve-fitting and Huffman tree. Therefore, features that capture how a compressor reacts to a dataset also need to be incorporated into training. Note that, to reduce the cost of feature extraction, we only build these features from sampled datasets, e.g., 10 percent of the original data. We next discuss the features we build for ZFP and SZ, respectively.

BitsPerBitplane and MaxPrec (ZFP). ZFP partitions data into blocks, and for each block, it uses a reversible orthogonal transformation, such as discrete cosine transform (DCT), to generate near-zero transform coefficients for each data point, which leads to higher compressibility. To achieve different levels of precision, compression is done on a per bit plane basis using embedded encoding. The less the number of bit planes, the lower the precision is. In particular, *BitsPerBitplane* is the average number of bits in a bit plane after encoding, and *MaxPrec* is the number of bit planes to be encoded.

HitRatio (SZ). In contrast, SZ was developed around the idea of curve-fitting and entropy encoding. For curve-fitted data points, quantization and Huffman tree encoding are utilized, motivated by the observation that after quantization, duplicated quantization intervals can occur, which can be well compressed by Huffman tree. Meanwhile, for curve-missed points, binary representation analysis is further conducted for compression. As observed by our previous work [4], most of the compression performance is contributed by curve-fitting, and as such, *HitRatio*, defined as the ratio of

TABLE 1
Description of Training/Testing Datasets (ZFP).

Problem	Purpose	# of Compressible	# of Incompress.
Classification	Train	648	402
	Test	167	97
Problem	**Purpose**	**Min**	**Max**
Regression	Train	0.97	35.77
	Test	0.97	25.66

TABLE 2
Description of Training/Testing Datasets (SZ).

Problem	Purpose	# of Compressible	# of Incompress.
Classification	Train	617	318
	Test	153	82
Problem	**Purpose**	**Min**	**Max**
Regression	Train	1.29	99.74
	Test	1.29	99.38

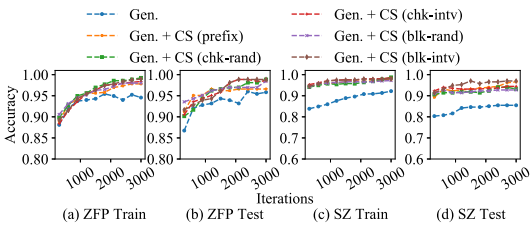

Fig. 2. Binary classification. The notation of *Gen.* indicates that the general features are incorporated in training and testing, whereas *CS* denotes compressor-specific features are further included, followed by the sampling method annotated in the bracket. In particular, *chk* and *blk* denote the sampling granularity of chunk (40 floating-point numbers) and block (4), respectively. The suffix *rand* and *intv* denote the random and interval sampling, respectively. For example, *Gen.+CS (chk-rand)* indicates that both general and compressor-specific features of ZFP are used, and the sampling is chunk-based random sampling.

the number of points that can be curve-fitted to the total number of points, is a key indicator for compression performance.

QuantInterval (SZ). *QuantInterval* is used to measure the number of quantization levels used by SZ for curve-fitted points. The larger the error bound is, the smaller the *QuantInterval* will be, thus the higher chance that two data points fall into the same quantization level. After entropy encoding, each quantization level is mapped to one Huffman tree node and further encoded.

NodeCount, EncodeSize, and TreeByteSize (SZ). During the entropy encoding for the curve-fitted points, the total number of Huffman tree nodes is denoted by *NodeCount* and the total tree size is *TreeByteSize*. The resulting data size is denoted by *EncodeSize*.

4 Evaluation

We use TensorFlow (1.11.0) and Keras (2.2.4) as the programming environment for training and prediction. To obtain sufficient training data, we vary the relative error bound from 10^{-9} to 10^{-1} for both ZFP (0.5.0) and SZ (1.4.9), and collect all features from 22 different datasets spanning across 14 applications. For binary classification, we empirically select the threshold of ZFP and SZ to be the median compression ratios of all datasets. More statistics of the datasets used for training and testing can be found in Tables 1 and 2, respectively. Note that the training and testing datasets can overlap over a particular application. The more complex cross-application estimation is left for future study, which will result in relatively lower

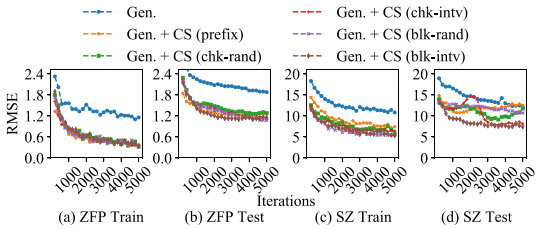

Fig. 3. Regression.

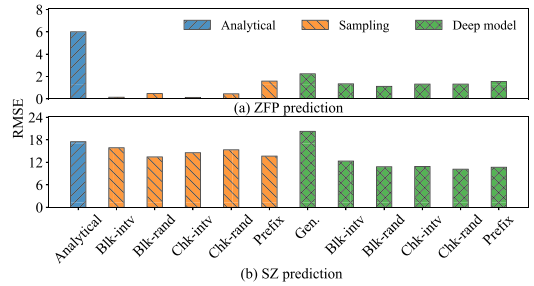

Fig. 4. Compressor ratio estimation. The legend *analytical* denotes the analytical model, while the legend of *sampling* denotes the compression ratio measured on the sampled (10%) data. The compression ratio predicted by the deep neural network is denoted as *deep model*. Note that the resulting RMSE is calculated across all 22 testing datasets.

accuracy and higher RMSE, because the training will not fully capture the characteristics of the target applications. We utilize a simple fully connected neural network with three hidden layers, with Relu being the activation function. This setup achieves the best training accuracy, as compared to other configurations of the neural networks, and therefore is chosen. More details regarding the neural network can be found at this URL.

We first evaluate the training and testing accuracy for both binary classification and regression in Figs. 2 and 3, respectively. The overall trend is that the accuracy improves with the number of iterations, illustrating that patterns do exist in compression and can be well learned by the neural network. The testing accuracy of using the general features reach as high as 94.83 and 87.47 percent for ZFP and SZ, respectively. By further incorporating compressor related features, the accuracy can be substantially improved up to 98.86 and 97.01 percent, respectively. A key finding of this result is that, despite that these compressor-specific features are collected from a small sample (e.g., 10 percent) of the full data, they are sufficient to capture the compressor characteristics and can dramatically improve the accuracy. On the other hand, the training and testing accuracy of SZ are lower than those of ZFP under the same settings. The reason is the effectiveness of learning heavily depends on the training data. Intuitively, the more statistically rich the training data are, the better the results will be. We define the statistical richness as the ratio of the number of training datasets and the compression ratio range. In particular, the statistical richness of ZFP and SZ training datasets are 23.42 and 7.82, respectively. As a result, the neural network can learn more effectively for ZFP than for SZ. Similar results are observed for regression in Fig. 3.

In Fig. 4, we further compare the outcome of using the neural network with the analytical model as well as the sampling based approach. In Fig. 4a, it is shown that the sampling based estimation achieves the best performance for ZFP, since the compression ratio of sampled data is an unbiased estimation for that of the full data, as theoretically proved in our prior work [4]. Meanwhile, the analytical model yields the highest RMSE due to the error of *BitsPerBitplane* across error bounds. For SZ estimation in Fig. 4b, using the deep model with general features achieves the highest RMSE. However, after compressor-specific features are incorporated into

TABLE 3
Compression Ratio Prediction of *Blast2_p* (ZFP).

Err bound	Real	Analytical		Sampling		Deep model	
		Value	Err%	Value	Err%	Value	Err%
10^{-9}	3.56	2.78	−21.91	3.55	−0.28	4.26	19.66
10^{-8}	4.06	3.03	−25.37	4.08	0.49	4.62	13.79
10^{-7}	4.54	3.33	−26.65	4.53	−0.22	4.89	7.71
10^{-6}	5.15	3.69	−28.35	5.14	−0.19	5.05	−1.94
10^{-5}	6.27	4.14	−33.97	6.30	0.48	6.51	3.83
10^{-4}	7.50	4.72	−37.07	7.49	−0.13	6.79	−9.47
10^{-3}	12.03	5.49	−54.36	12.00	−0.25	9.52	−20.86
10^{-2}	15.18	6.55	−56.85	15.13	−0.33	12.96	−14.62
10^{-1}	18.88	8.12	−56.99	18.82	−0.32	17.85	−5.46

training, the performance is substantially improved, and interestingly outperforms the sampling based estimation, which is proved to be a biased estimation toward that of the full data for SZ [4]. To further understand the performance, we list the real compression ratios and the predictions for the *blast2_p* dataset in Table 3. The analytical model consistently under-estimates the compression ratio and the magnitude of error increases with the error bound. The estimation error comes primarily from the over-estimation of *BitsPerBitplane*. As the block size can be estimated by multiplication of estimated *BitsPerBitplane* and *MaxPrec*, an over-estimated *BitsPerBitplane* will result in an over-estimated block size, and in turn under-estimate the compression ratio.

5 Conclusion

In this paper, we propose a deep neural network based method for estimating the compressibility of scientific data. In particular, we build a set of features to capture both data and compressor characteristics to train a deep neural network for both classification and regression. It is shown that compressor-specific features can dramatically improve the accuracy of training and testing for two leading lossy compressors, ZFP and SZ. We further compare the deep neural network model with two other compression estimation schemes to understand their pros and cons. The results show that the deep model consistently outperforms the analytical model, as well as the sampling based approach in the case of a biased estimation.

Acknowledgments

This work was supported by the US NSF under Grant No. CCF-1718297, CCF-1812861, and NJIT Research Startup Fund.

References

[1] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Trans. Vis. Comput. Graphics*, vol. 20, no. 12, pp. 2674–2683, Dec. 2014.
[2] S. Di and F. Cappello, "Fast error-bounded lossy HPC data compression with SZ," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 730–739.
[3] J. Wang, T. Liu, Q. Liu, X. He, H. Luo, and W. He, "Compression ratio modeling and estimation across error bounds for lossy compression," *IEEE Trans. Parallel Distrib. Syst.*, to be published, doi: 10.1109/TPDS.2019.2938503.
[4] T. Lu *et al.*, "Understanding and modeling lossy compression schemes on HPC scientific data," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2018, pp. 1–10.
[5] M. Burtscher and P. Ratanaworabhan, "FPC: A high-speed compressor for double-precision floating-point data," *IEEE Trans. Comput.*, vol. 58, no. 1, pp. 18–31, Jan. 2009.
[6] J.-L. Gailly, "Gzip: The data compression program," 2016. [Online]. Available: https://www.gnu.org/software/gzip/manual/gzip.pdf
[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[8] Y. Wiseman, K. Schwan, and P. Widener, "Efficient end to end data exchange using configurable compression," in *Proc. 24th Int. Conf. Distrib. Comput. Syst.*, 2004, pp. 228–235.