DZip: improved general-purpose lossless compression based on novel neural network modeling

Mohit Goyal*, Kedar Tatwawadi, Shubham Chandak, Idoia Ochoa*
UIUC*, Stanford University

Introduction

A wide class of (lossless) compressors utilize the "prediction + entropy coding" approach, wherein a statistical model generates predictions for the upcoming symbols given the past and an entropy coder uses the predicted probabilities to perform compression. In this general framework, a better prediction model directly induces a better compressor.

We propose DZip, a general-purpose compressor for sequential data that exploits the well-known modeling capabilities of neural networks (NNs) for prediction, followed by arithmetic coding. DZip uses a novel hybrid architecture based on adaptive and semi-adaptive training. Unlike most NN based compressors, DZip does not require additional training data and is not restricted to specific data types, only needing the alphabet size of the input data.

Background

Based on how the model (predictor) is trained, lossless compression schemes can be categorized into three types:

Static
- The model is trained offline on some external training data and is available during encoding/decoding.

Adaptive
- The model is pseudorandomly initialized and then updated adaptively during encoding/decoding.
- This approach does not require the availability of training data and works quite well for small models.

Semi-adaptive
- The model is trained based only on the input sequence and the training procedure can involve multiple passes through the input data.
- The trained model is included as part of the compressed file.
- The additional cost is expected to be compensated by the fact that the sequence-specific training will lead to a better predictive model.

DZip Framework

Bootstrap Model: This model is designed keeping in consideration the trade-off between the model size and the effective compression obtained. This model is trained on the input sequence multiple times and is then stored as a part of the compressed file.

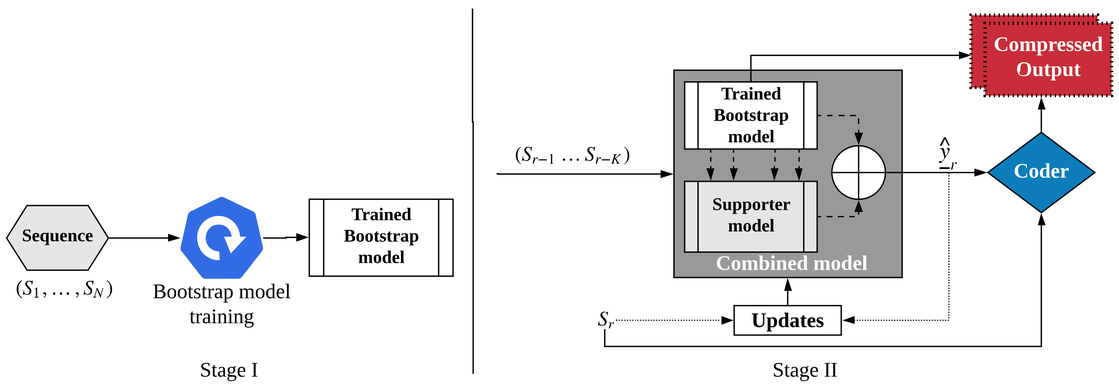
Supporter Model: This is a significantly larger model in comparison to the Bootstrap model. It is randomly initialized at the start of encoding/decoding and is then adaptively updated to better estimate the probability of upcoming symbols.

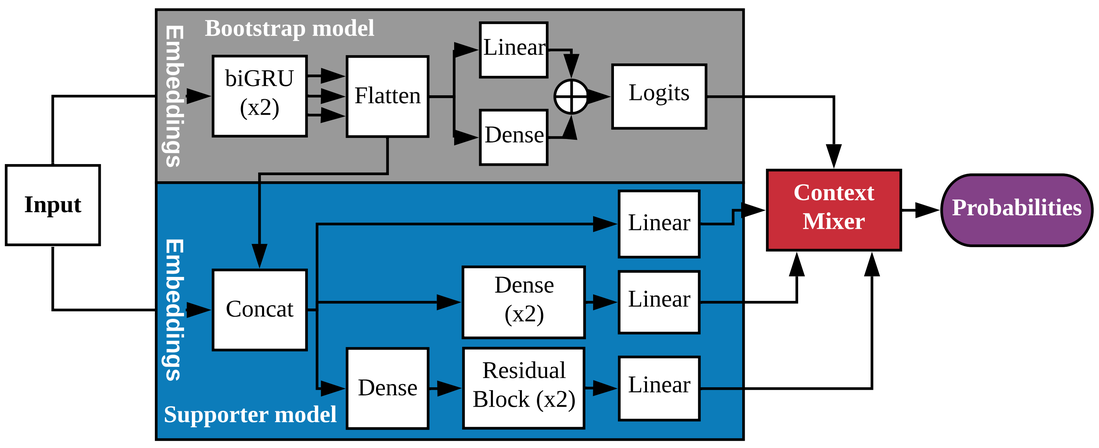
Combined Model: The combined model is a combination of Bootstrap model and the Supporter Model. The logits (unscaled probabilities) from both the models undergo convex combination to generate the final proabability that is fed to the entropy coder.

Notations S_r denotes the r^{th} symbol of the sequence $\mathcal{S}$, K denotes the context length, $\underline{\hat{y}}_r$ is the probability vector for symbol S_r, $\underline{y}_r$ is the onehot encoded ground truth.

Loss Function: Categorical Cross Entropy is used to train the Bootstrap and the Combined model.

$$\mathcal{L} = \sum_{k=1}^{|\mathcal{S}|} y_{rk} \log_2 \frac{1}{\hat{y}_{rk}} \tag{1}$$





Experiments & Datasets

Datasets:
We experiment with various real and synthetic sequences as shown in the table below.

Name	Length	$\lvert\mathcal{S}\rvert$	Description
Real Datasets			
webster	41.1M	97	HTML data of the 1913 Webster Dictionary, from the Silesia corpus
mozilla	51.2M	255	Tarred executables of Mozilla 1.0, from the Silesia corpus
h. chr20	64.4M	5	Chromosome 20 of *H. sapiens* GRCh38 reference sequence
h. chr1	100M	5	First 100M bases of chromosome 1 of *H. Sapiens* GRCh38 sequence
c.e. genome	100M	4	*C. elegans* whole genome sequence
ill-quality	100M	4	100MB of quality scores for PhiX virus reads sequenced with Illumina
text8	100M	27	First 100M of English text (only) extracted from *enwiki9*
np-bases	300M	5	Nanopore sequenced reads (only bases) of a human sample
np-quality	300M	91	Quality scores for nanopore sequenced data of a human sample
enwiki9	500M	206	First 500M of the English Wikipedia dump on 2006
Synthetic Datasets			
XOR-k	10M	2	Pseudorandom sequence generated as $S_{n+1} = S_n + S_{n-k}$ (mod 2). Entropy rate 0 bits per character (bpc).
HMM-k	10M	2	Hidden Markov sequence $S_n = X_n + Z_n$ (mod 2), with $Z_n \sim Bern(0.1)$, $X_{n+1} = X_n + X_{n-k}$ (mod 2). Entropy rate 0.46899 bpc.

Results

DZip on Synthetic datasets

Compressor	*XOR-20*	*XOR-30*	*XOR-50*	*XOR-70*	*HMM-20*	*HMM-30*	*HMM-50*	*HMM-70*
Gzip	1.20	1.20	1.19	1.19	1.19	1.19	1.19	1.19
LSTM-Compress	4.23	3.19	4.77	3.43	3.02	5.19	3.64	1.01
BSC	0.10	1.01	1.01	1.01	0.69	1.01	1.01	1.01
DZip	**1e-3**	**1e-3**	**0.9e-3**	**1.00**	**0.47**	**0.47**	**0.47**	**1.00**

Fig. 3: Synthetic Datasets, XOR and HMM

Comparison of DZip performance on real datasets in bits per character (bpc)

File	Len/$\log_2\lvert\mathcal{S}\rvert$	Gzip	LSTM Compress	BSC	DZip		Specialized Compressor
					bpc	Model	
webster	41.1M/6.61	2.32	**1.23**	1.29	1.40	31.33%	1.09
mozilla	51.2M/8.00	2.97	**2.05**	2.52	2.20	25.13%	N/A
h. chr20	64.4M/2.32	2.05	7.82	1.73	**1.63**	0.92%	1.62
h. chr1	100.3M/2.32	2.14	7.36	1.78	**1.68**	0.58%	1.65
c.e. genome	100M/2.00	2.15	7.51	1.87	**1.81**	0.53%	1.72
ill-quality	100M/2.00	0.50	6.48	0.35	**0.34**	2.79%	0.51
text8	100M/4.75	2.64	1.76	**1.68**	1.73	9.45%	1.52
enwiki9	500M/7.69	2.72	1.66	1.64	**1.50**	3.59%	1.43
np-bases	609M/2.32	2.16	8.43	1.86	**1.74**	0.09%	1.75
np-quality	609M/6.51	5.91	**5.47**	5.64	**5.47**	0.57%	5.20

File	Length	Bootstrap only	DZip	Improvement (bpc)
webster	41.1M	1.450	1.399	0.051
mozilla	51.2M	2.250	2.200	0.050
h. chr1	100.3M	1.719	1.678	0.041
ill-quality	100M	0.343	0.342	0.001
enwiki9	500M	1.596	1.502	0.094
np-bases	609M	1.759	1.737	0.022

Fig. 4: DZip on Real Datasets

Computational Requirements

The main limitation of DZip in its current implementation is the encoding/decoding time. On an average, DZip takes 4-5 minutes/MB in the bootstrap training process, and 5 hours/MB for encoding/decoding. The time taken by DZip is significantly high because coding has to be performed has to be performed using a single CPU on a single thread for ensuring symmetrical updates to the model (due to Keras platform limitation).
For comparison, Gzip, LSTM-Compress, and BSC take on average 4.9 seconds/MB, 3 minutes/MB, and 0.07 seconds/MB for compression, respectively, and 0.005 seconds/MB, 4 minutes/MB, and 0.02 seconds/MB for decompression, respectively.

Code and additional details

Original Github Link: https://github.com/mohit1997/DZip
ArXiv Paper: https://arxiv.org/abs/1911.03572

We have implemented a parallelized variant of DZip using the Pytorch framework which utilizes GPU. This implementation is 60× (depending on the GPU) faster but the compression and decompression is supported only on the same machine.

Faster Implementation: https://github.com/mohit1997/Dzip-torch