DZip: improved neural network based general-purpose lossless compression

by Mohit Goyal, Kedar Tatwawadi, Shubham Chandak, Idoia Ochoaγ

ECE 563 Project

Students: Huyen Nguyen
Linjie Tong

Outline

1. Introduction
    - Arithmetic Coding
    - Adaptive Arithmetic coding
    - Advantages of neural network-based compressing method
2. DZip: improved neural network based general-purpose lossless compression
    - Motivation of DZip
    - DZip approach and training procedure
    - Compression results
1. Conclusion

Huffman coding

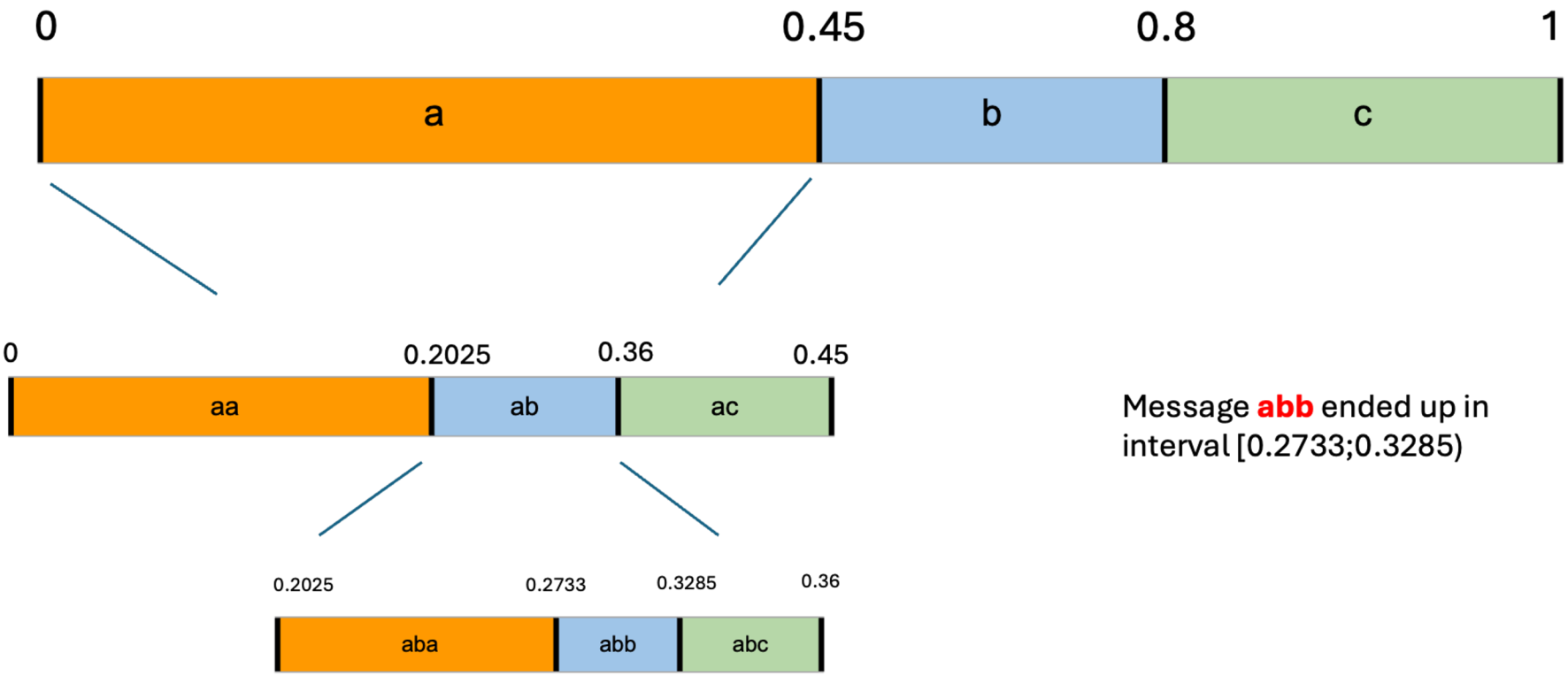
Example of Huffman Coding (not optimal):

+ message of length 10

Symbol	a	b	c
Frequency	0.45	0.35	0.2
Codeword	0	10	11
Entropy: 1.513		**Average length: 1.55**	

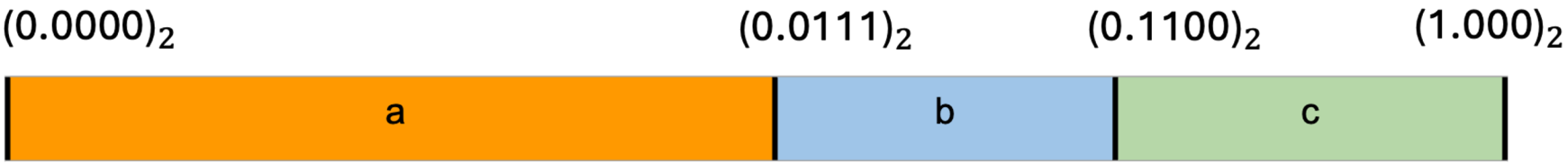
Can we do better?

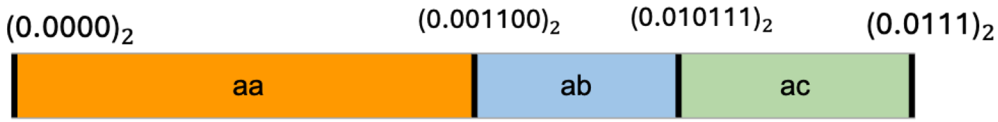
Arithmetic coding:



Message **abb** ended up in interval [0.2733;0.3285)

Arithmetic coding:





Message **ab** encoded in binary: in interval of $[0.001100 ; 0.010111)_2$

$(0.1)_2 = 1 \times \frac{1}{2} = (0.5)_{10}$

If choose $x = 0.0101$ to encode message **ab**

$(0.0101)_2 = 1\times2^{-2} + 1 \times2^{-4} = (0.125)_{10}$

Huffman coding vs Arithmetic Coding

Example:

+ message of length 20: aababaaabaabaaaabaa<eos>

Symbol	a	b	c
Frequency	0.7	0.2	0.1
Codeword	0	10	11
Entropy: 1.16		**Average length of Huffman: 1.30**	

+ Huffman Coding requires: 26 bits
+ Arithmetic Coding requires: 20 bits

Arithmetic Coding

Arithmetic Coding require information from the sequence data distribution.

Symbol	a	b	c
Frequency	0.7	0.2	0.1
Codeword	0	10	11
Entropy: 1.16		**Average length: 1.30**	

Question: How can we estimate the true data distribution accurately and efficiently at the time of message passing?

Adaptive Arithmetic Coding

- In static Arithmetic Coding above, the data distribution used to encode the next token is static for the entire message passing process.

- However, in real life, true distribution of the sequences proved hard to compute, predefined distribution hardly approximate the true data distribution.

- Idea of **Adaptive Arithmetic Coding**:
  At each step, the probability distribution will be changed according to the data observed up until that point. The decoded data matches the original data as long as the frequency table in decoding is replaced in the same way and in the same step as in encoding.

- The many approaches used to estimate probability distribution at each iteration.

Adaptive Arithmetic Coding (1)

- **Idea**: use a dynamic probability distribution which changes with each character read. Initially, each character is assumed to have the same probability, and each time a particular symbol S_i is read, its probability for future iterations increases.

- This is based on counting the frequency of occurrence of each characters as we observe the streamed data at each iteration. (assume the independence of our data).

- At initialization, uniform distribution can be used, or more sophisticated, to prevent the redundant data storage (a placeholder for characters that never occur), an escape character can be used.

Adaptive Arithmetic Coding (2)

- The above modelling may not be the best in estimating the true data distribution. However, the pros is little overhead.

- More advanced adaptive models an increase the performance by introducing more overheads:

  - Weight probabilities at each step based on predictions of the next character.
  - Using the previous characters or previously seen subsequences as part of the modelling process (multi-symbol adaptive model, prediction by partial matching).
  - Adding extra metadata symbols to allow to account for out-of-band information.

Adaptive Arithmetic Coding (3)

- **Prediction by Partial matching (PPM):**

Idea: We take into account the dependency of the next token and previous observed subsequences. Recurring patterns of any lengths can be used to predict the next character and assign the token a probability value. For example, if the sequence ‘SALSA’ has been seen five times, the sequence ‘SALAD’ has seen twice, then if the characters ‘SAL’ are encountered in the future, we want to weight the probability of S higher than that of A.

Adaptive Arithmetic Coding (3)

- Example: A context table showed the occurrence frequency of each character after a subsequence of different lengths (3 and 2 in this case).

CTX	S	FREQ
ABC	A	3
	B	5
	C	6

CTX	S	FREQ
BC	A	5
	B	10
	C	8
	D	10
	E	15

For context ‘ABC’, the occurrence of the next token being B is 5, so PPM encodes B with probability 5/14.

CTX table with length is s subset of CTX table with length 2 for the overlapping characters. Ex: ‘A’ 3 times after ’ABC’, ‘A’ occurs 5 times after ‘BC’.

Adaptive Arithmetic Coding (3)

- What if the next token has not showed up previously in the data stream? Next token can be ‘D’, which was never recorded in CTX table of ‘ABC’.

CTX	S	FREQ
ABC	A	3
	B	5
	C	6
	⊗	1

CTX	S	FREQ
BC	A	5
	B	10
	C	8
	D	10
	E	15
	⊗	1

To account for new character, a symbol is added, which is called the ‘escape symbol’ (⊗) and a probability is assigned to that symbol.
Whenever there is a new character, the escape symbol informs us to jump to the next table (Ex: jump from CTX table of ‘ABC’ to ‘BC’ to search for ‘D’, and if ‘D’ never shows up in context of any lengths, jump to uniform distribution).

As we observed more data, the probability distribution gets closer to the true distribution, and probability of the escape symbol gets smaller.

Advantage of Using Neural Network over Trandition Method

Ability to Model Complex Data Distributions: Neural networks excel at learning non-linear and complex relationships in data. This allows them to model intricate dependencies that traditional methods may overlook.

Adaptability: Neural networks can generalize well across diverse datasets without requiring extensive manual adjustments. Traditional approaches often rely on domain-specific knowledge or assumptions that may not always hold.

Better Compression Ratios for High-Dimensional Data: High-dimensional data like images or audio benefit from neural networks' ability to extract and utilize latent structures effectively. This often leads to better compression ratios compared to traditional methods.

Integration with Modern Systems: Neural networks integrate easily with other machine learning tasks like feature extraction or prediction. This makes them ideal when compression is part of a larger automated system.

When Neural Networ is Needed for Compression

1. **High Dimensional and Complex Data**: Neural networks can model complex, high-dimensional data distributions better than traditional methods. For example
2. **Nonlinear Dependencies:** Neural networks excel at capturing nonlinear relationships.
3. **High Compression Ratio:** Neural network can achive high compression ratio for its learning capacity. If compression ratio is a more significant factor than compression speed, using Neural Network.

Dzip data compressor motivation

Types of model to estimate the conditional probability $P(S_r | S_{r-1}, \ldots, S_{r-K})$

- ➔ **Static:**
  - model first trained on some external training data and available to both the compressor and decompressor.
  - restricted to cases where similar training data available, not for general-purpose compression tasks.
- ➔ **Adaptive:**
  - both the compressor and the decompressor initialized with ... updated adaptively based on the sequence seen up to some point; ... ng data.
- ➔ **Semi-adaptive:**
  - model trained based only on input sequence, training proc... through the input data; trained model parameters and arithmetic co... sed file
  - additional cost expected to compensate a better predictive model and smaller arithmetic coding output.

DZip: combines elements of semi-adaptive and adaptive approaches to achieve better prediction

Dzip training procedure

- Data stream $S_N = \{s_1, s_2, \ldots, s_N\}$ over an alphabet $\mathcal{S}$, the size is $|\mathcal{S}|$
- K: the length of previous tokens used to estimate next token probability

- **Training procedure:**
  - For the current r^{th} symbol S_r :
  - The model aim is to estimate $P(s_r | s_{r-1}, \ldots, s_{r-K})$, s_r is the current r^{th} symbol
  - Let $\underline{\hat{y}}_r \in \mathbb{R}^{|\mathcal{S}|}$: the vectors of predicted probability, the k^{th} index of $\underline{\hat{y}}_r$
  - $\hat{y}_{rk} = P(S_r = x_k | s_{r-1}, \ldots, s_{r-K})$, $x_k \in \mathcal{S}$
  - Let $\underline{y}_r \in \mathbb{R}^{|\mathcal{S}|}$: the one-hot encoding with $y_{rk} = 1$ if $s_r = x_k$, $y_{rk} = 0$ otherwise
  - Cross entropy (CE) loss:

$$\mathcal{L} = \sum_{r=K+1}^{N} CE(\underline{y}_r, \underline{\hat{y}}_r) = \sum_{r=K+1}^{N} \sum_{k=1}^{|S|} y_{rk} \log_2 \frac{1}{\hat{y}_{rk}}$$

Dzip training procedure

- Arithmetic coding inference procedure:
  - For the current r^{th} symbol s_r
  - P(s_r| s_{r-1},…, s_{r-K}) is estimated, and used to encode s_r
  - s_r then fed into the arithmetic encoding block which recursively updates its state: s_r is used as input to the model for the conditional probability of the next token: P(s_{r+1}| s_r,…, s_{r-K+1})

Dzip inference procedure

- Arithmetic coding inference procedure:
  - For the current r^{th} symbol s_r
  - P(s_r| s_{r-1},…, s_{r-K}) is estimated, and used to encode s_r
  - s_r then fed into the arithmetic encoding block which recursively updates its state: s_r is used as input to the model for the conditional probability of the next token: P(s_{r+1}| s_r,…, s_{r-K+1})

Model architecture

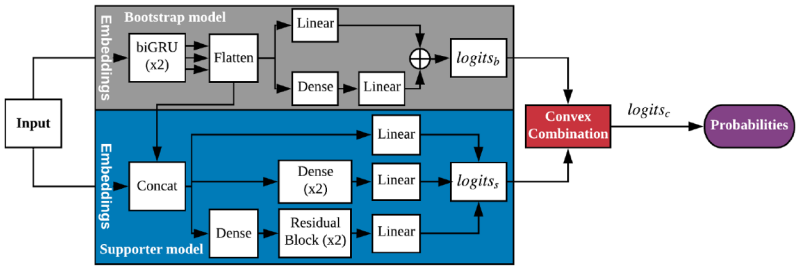


Figure 2: Combined model architecture consisting of bootstrap and supporter models. Dense layers correspond to fully connected layers with ReLU activation. Linear layers are also fully connected layers but do not incorporate a non linear transformation. Concat block denotes concatenation of all the input vectors.

Convex Combination

$$logits_c = \lambda * logits_b + (1 - \lambda) * logits_s \text{ , s.t. } \lambda \in [0, 1],$$

Bootstrap model: Achieve trade-off between model size and prediction capability.

- biGRU helps network learn long-term dependencies
- Dense layer is important for learning long-term relationships

Supporter model: The supporter model architecture is designed to adapt quickly and provide better probability estimates than the bootstrap model.

- The supporter model consists of three sub-NNs which act as independent predictors of varying complexity. The first sub-NN is linear, the second sub-NN has two dense layers and the third sub-NN uses residual blocks.

Training process

First stage: reading the input file byte-by-byte and, based on the vocabulary size, automatically selecting the hyperparameters for the bootstrap and supporter models.

Second stage: training the bootstrap model by performing multiple passes through the sequence.

Third stage:

- Compression ratio: Frozen parameter of Boostrap model and update parameters in Supporter model.
- Encoding/decoding speed: Just use bootstrap model to compress.

Results

Table 1: Bits per character (bpc) achieved by the tested compressors on the real datasets. Best results are boldfaced. $\log_2 |\mathcal{S}|$ represents the bpc achieved assuming an independent uniform distribution over the alphabet of size $|\mathcal{S}|$. For DZip, we specify the total bpc and the size of the model (in % space occupied). Spec. Comp. stands for specialized compressor.

File	Len/$\log_2\|\mathcal{S}\|$	Gzip	BSC	7zip	ZPAQ	LSTM Compress	NNCP	CMIX	DZip		Spec. Comp.
									bpc	Model	
webster	41.1M/6.61	2.32	1.29	1.70	1.09	1.23	0.98	**0.83**	1.44	31.33%	0.83
mozilla	51.2M/8.00	2.97	2.52	2.11	1.88	2.05	1.63	**1.39**	2.15	25.37%	1.39
h. chr20	64.4M/2.32	2.05	1.73	1.77	1.68	7.82	1.66	**1.62**	1.63	0.92%	1.62
h. chr1	100M/2.32	2.14	1.78	1.83	1.74	7.36	**1.67**	**1.67**	**1.67**	0.58%	1.65
c.e. genome	100M/2.00	2.15	1.87	1.89	1.80	7.51	1.80	**1.74**	1.81	0.53%	1.72
ill-quality	100M/2.00	0.50	0.35	0.35	0.34	6.48	0.34	**0.33**	0.34	2.79%	0.51
text8	100M/4.75	2.64	1.68	1.93	1.52	1.76	1.48	**1.31**	1.74	9.38%	1.31
np-bases	300M/2.32	2.16	1.86	1.93	1.79	7.34	**1.70**	1.73	1.73	0.19%	1.75
np-quality	300M/6.51	5.95	5.69	5.71	5.53	5.51	5.50	**5.49**	5.56	1.13%	5.35
enwiki9	500M/7.69	2.72	1.64	1.94	1.43	1.66	1.21	**1.05**	1.47	3.67%	1.05
num-control	159.5M/8.00	7.57	7.66	7.41	6.96	6.82	6.72	**6.63**	6.83	2.67%	7.12
obs-spitzer	198.2M/8.00	6.50	2.51	2.27	2.20	2.87	1.73	**1.58**	2.18	6.70%	7.74
msg-bt	266.4M/8.00	7.08	6.96	5.76	6.29	6.22	5.36	5.24	**5.21**	2.08%	6.67
audio	264.6M/8.00	5.75	4.63	4.98	4.17	4.92	3.49	3.44	**3.40**	3.29%	N/A

Traditional compressors: Gzip, BSC and 7zip.
NN-based compressors: ZPAQ, LSTM Compress, NNCP, CMIX

Model is tested on various type of data:

- Genomic data :h. chr1, h. chr20, c.e. genome, npbases np-quality, ill-quality
- Text: webster, text8, enwiki9.
- Executable files: mozilla.
- Double precision floating point data: num-control, obs-spitzer, msg-bt.
- Audio data: audio.

Results

Table 2: Compression in bpc obtained by (i) only the bootstrap model and (ii) DZip (combined model). Improv. stands for the improvement of the combined model with respect to the bootstrap model (in bpc).

FILE	*webster*	*mozilla*	*h. chr1*	*text8*	*np-bases*	*np-quality*	*enwiki9*	*obs-spitzer*	*msg-bt*	*audio*
Length	41.1M	51.2M	100M	100M	300M	300M	500M	198.2M	266.4M	264.6M
Bootstrap Only	1.474	2.233	1.720	1.789	1.755	5.588	1.596	2.445	5.259	3.405
DZip	1.443	2.150	1.673	1.737	1.725	5.562	1.470	2.181	5.214	3.389
Improv.	0.031	0.083	0.047	0.052	0.03	0.026	0.126	0.264	0.045	0.016

Combined model improves the compression by 0.072 bpc on the real datasets. DZip in bootstrap only mode still outperforms Gzip, 7zip, BSC and LSTM-Compress on most of the selected datasets, while being more practical due to its reduced running time.

Conclusion

Advantage:

- DZip achieves improvements over Gzip, 7zip, BSC, ZPAQ and LSTM-Compress.
- DZip also compares favorably with the state-of-the-art NN-based compressors CMIX and NNCP, achieving similar compression while being substantially faster.

Disadvantage:

- As a deep learning-based compression method, encoding/decoding is still time-consuming.

Appendix

Example to show computation of loss

$\mathcal{L} = \sum_{r=1}^{N} CE(y_r, \hat{y}_r) = \sum_{r=1}^{N} \sum_{k=1}^{|S|} y_{r,k} \log_2 \frac{1}{\hat{y}_{r,k}}$.

Take compress 'h' in 'I am happy' as an example. For sequence length of 'I am happy' is 8, so $N = 8$. For here alphabet is English characters, $S = [a, b, c, \cdots, z]$, $|S| = 26$. 'h' is 8th in English Alphabet and 'h' is fourth characters in sequence, meaning r $= 4$.

$\mathcal{L}_4 = CE(y_4, \hat{y}_4) = \sum_{k=1}^{|S|} y_{4,k} \log_2 \frac{1}{\hat{y}_{4,k}}$.

One hot vector y_r shows which character appears in position r of sequence. $y_4 = [0, 0, \ldots, 1, 0, \ldots, 0]$ where $y_{4,8} = 1$.

Estimated probability vector $\hat{y}_r$ by NN shows probability for each character appears in position r of sequence. In this example, we assume output of NN for h is $\hat{y}_4 = [P(\alpha | I\ am)$ for α in $a, b, c, \cdots, y, z] = [0.01, \ldots, 0.88, \ldots, 0.02]$, where $\hat{y}_{4,8} = 0.88$.

When Compressed 'h' in 'I am happy', the estimated probability from NN is $\hat{y}_{4,8} = 0.88$. $\mathcal{L}_4 = 0 + 0 + \cdots + \log_2 \frac{1}{\hat{y}_{4,8}} + 0 + \cdots + 0 = \log_2 \frac{1}{\hat{y}_{4,8}}$.