Lossless Data Compression using Recurrent Neural Networks

Code: https://github.com/TheGrayFrost/Text-Compression

Introduction

The Big Data revolution has led to collection of huge amounts of different types of different types of datasets such as images, texts, log data etc. Thus, there is a need for coming up with statistical models and efficient compressors for these data formats.

There has been significant advancement on the problem of lossless compression in the past 50 years. In his seminal work, Claude E. Shannon showed that the Entropy rate is the best possible compression ratio for the given source, and gave a methodology to achieve it (though impractical). J. Rissanen proposed Arithmetic coding [2], which is an efficient procedure for achieving the entropy bound for known distributions. For sources with unknown distributions adaptive variants of Arithmetic encoding has been designed, which try to compress by learning the conditional k-gram model distribution. However, as the complexity increases exponentially, the value of k is limited to 20 symbols. Implementations of different types of Arithmetic coding can be found here [4]. This can lead to a significant loss of compression ratio, as the models are not able to capture long term dependencies.

We know that Recurrent Neural Network (LSTM/GRU) based models are good at capturing long term dependencies [5] and can predict the next character/word very well. In this project we explore the utility of RNN based models along with Arithmetic coding to improve lossless compression.

Nuances of Theoretical Data Compression

Although entropy is often used as a characterization of the information content of a data source, this information content is not absolute: it depends crucially on the probabilistic model. A source that always generates the same symbol has an entropy rate of 0, but the definition of what a symbol is depends on the alphabet. Consider a source that produces the string ABABABABAB… in which A is always followed by B and vice versa. If the probabilistic model considers individual letters as independent, the entropy rate of the sequence is 1 bit per character. But if the sequence is considered as "AB AB AB AB AB ..." with symbols as two-character blocks, then the entropy rate is 0 bits per character.

However, if we use very large blocks, then the estimate of per-character entropy rate may become artificially low. This is because in reality, the probability distribution of the sequence is not knowable exactly; it is only an estimate. For example, suppose one considers the text of every book ever published as a sequence, with each symbol being the text of a complete book. If there are *N* published books, and each book is only published once, the estimate of the probability of each book is $1/N$, and the entropy (in bits) is $-\log_2(1/N) = \log_2(N)$. As a practical code, this corresponds to assigning each book a unique identifier and using it in place of the text of the book whenever one wants to refer to the book. This is enormously useful for talking about books, but it is not so useful for characterizing the information content of an individual book, or of language in general: it is not possible to reconstruct the book from its identifier without knowing the probability distribution, that is, the complete text of all the books. The key idea is that the complexity of the probabilistic model must be considered. Kolmogorov complexity is a theoretical generalization of this idea that allows the consideration of the information content of a sequence independent of any particular probability model; it considers the shortest program for a universal computer that outputs the sequence. A code that achieves the entropy rate of a sequence for a given model, plus the codebook (i.e. the probabilistic model), is one such program, but it may not be the shortest.

Compressor Framework:

We take a look at the compressor model we used for the experiments. The framework can be broken down into two blocks:

1 **RNN Probability Estimator block**: For a data stream S_0, S_1, S_2, …., S_N, the RNN probability estimator block estimates the conditional probability distribution of S_k, based on the previously observed symbols. This probability estimate is fed into the Arithmetic encoding block.

2 **Arithmetic Coder block**: The Arithmetic coder block can be thought of as an FSM which takes in the probability distribution estimate for the next symbol and encodes it into a state (decoder operations are reversed)

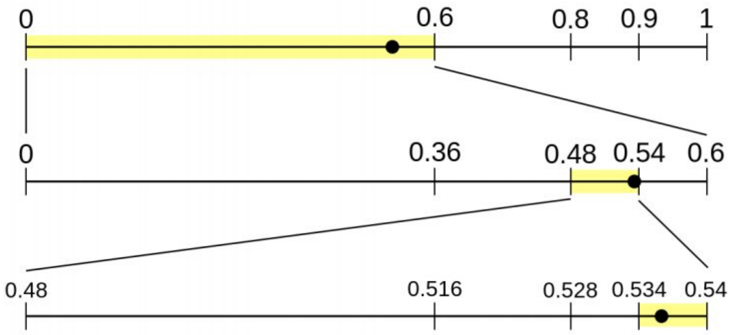

Figure 1: Arithmetic Encoding of the sequence (0,2,3) for an i.i.d. (0.6, 0.2, 0.1, 0.1) distributed source

Encoder and Decoder Operations

1. The arithmetic encoder block always starts with a default probability distribution estimate for the first symbol S_0. This is done so that the decoder can decode the first symbol.
2. Both the arithmetic encoder and the RNN Estimator blocks pass on the state information across iterations. The final state of the Arithmetic Encoder acts as the compressed data.
3. If the model is trained for more than an epoch, the weights of the RNN Estimator Block also need to be stored, and would count in the compression size.

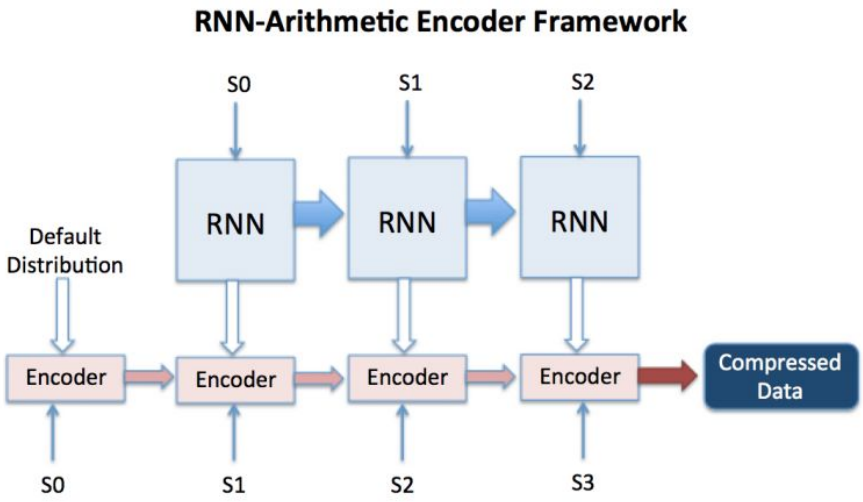

Figure 2: Encoder Model Framework

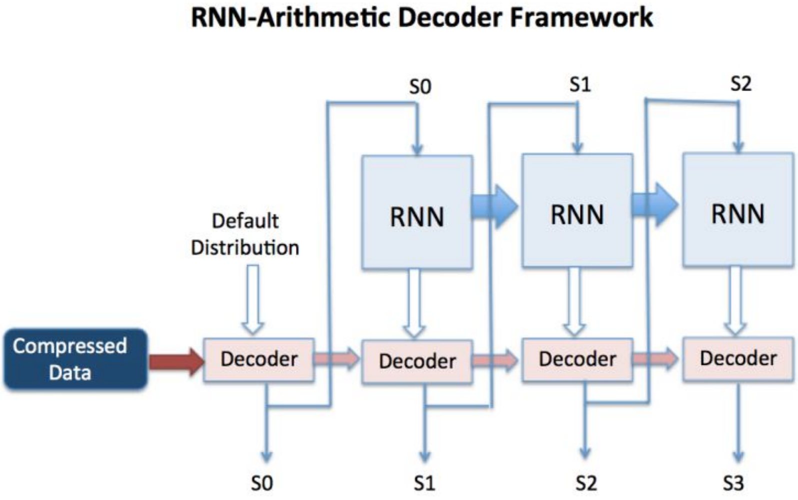

Figure 3: Decoder Model Framework

Intuition: Why should this work?

Our vague understanding of the notion of entropy in information talks about overall log probability sums. But as we can appreciate from the nuances of theoretical data compression section, the entropy that really matters is the probability for the local patterns. These biases in local short-term probability distributions are what we observe as ‘patterns’ and better prediction of these is what will truly lead to the maximum achievable compression.

In an extreme case, if our model can always predict what the next token in a sequence will be (and that is what we are trying to achieve from our RNNs), the compressed size boils down to just the first character (plus, the RNN description).

Mathematically, the compression size L_{AE} (average number of bits used per symbol) of the Arithmetic encoder is very close to the normalized cross-entropy loss which we use in the RNN based estimator. Thus, CE-loss is in fact the optimal loss function for this framework.

Experiments on Synthetic Datasets:

1. I.I.D. Sources
2. Markov-k sources:

   a. Zero-entropy sources with markovity of k. These are difficult to compress. They are governed by,

   $$X_n = X_{n-1} + X_{n-k} \bmod M$$

   b. HMM sources with markovity of k. These are even tougher to compress. They are governed by,

   $$X_n = P(X_{n-1} + X_{n-k} \bmod M, 1 - (X_{n-1} + X_{n-k} \bmod M))$$

Models Tested:

1. ChGRU – Character level GRU based neural network model
2. ChFeat – The GRU based model, which takes in features which are function of all previously observed symbols instead of just the previous input.

Results and Comparative study:

1. Synthetic Datasets

Dataset	Size	ChGRU: Adam	ChGRU: AdaGrad	ChFeat: Adam	ChFeat: AdaGrad	GZIP	Adaptive AE
IID	100MB	200KB	350KB	150KB	250KB	2MB	4KB
Xen-Markov30	100MB	1MB	1.8MB	400KB	1MB	40MB	87MB
Xen-Markov50	100MB	30MB	40MB	500KB	1MB	63MB	100MB
HMM-30	100MB	4MB	6MB	1MB	1.5MB	60MB	100MB

2. Practical Datasets

Dataset	Size	ChGRU: Adam	ChGRU: AdaGrad	ChFeat: Adam	ChFeat: AdaGrad	GZIP	Best Custom
Hutter-Prize	100MB	18MB	25MB	17MB	22MB	34MB	16MB (CMIX)
EnWiki-9	256MB	178MB	190MB	168MB	179MB	330MB	154MB (CMIX)
Chromosome-1	240MB	55MB	67MB	50MB	62MB	57MB	49MB (MFCompress)

Conclusion and Further Work

We performed experiments mainly using two different models, a character-level GRU/LSTM model and a feature based GRU/LSTM model. Both of the models performed significantly well on difficult sources such as Genome Sequences. The results highlight potential for lossless compression using RNN based models.

The major drawback with the model is time. The decoder has to run the exact same RNN training as the encoder to be able to decode. This aspect, combined with the complexity of the RNN training cycle, makes the encoding and decoding equally and extremely time and resource consuming. It took us about 7 hours to compress 100MB of data i.e. a compression and decompression rate of a mere 4 kBps. The challenge is to make the model faster and work well on more complex sources. Due to the sequential nature of RNN training, and unavoidable sequence binding of the compression and decompression stage, it is not possible to parallelise the model.

Thus, further work would aim at faster learning, so that the RNN training can be stopped early and the encoder models run on those learnt tables. Another interesting extension of the project would be working with lossless compression of images. Images have significant dependencies, and modeling the same with RNN should be beneficial.

References

Data Compression

[1] Witten, Ian H., Radford M. Neal, and John G. Cleary. ”Arithmetic coding for data compression.” Communications

[2] Rissanen, Jorma, and Glen G. Langdon. ”Arithmetic coding.” IBM Journal of research and development 23.2 (1979): 149-162.

[3] ANS: Duda, Jarek. ”Asymmetric numeral systems.” arXiv preprint arXiv:0902.0271 (2009).

[4] Practical Arithmetic Coding: https://github.com/nayuki/Reference-arithmetic-coding

Deep Learning Motivation: Previous work

[5] Unreasonable Effectiveness of RNN: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[6] Cox, David. ”Syntactically Informed Text Compression with Recurrent Neural Networks.” arXiv preprint arXiv:1608.02893 (2016). APA

[7] Mahoney, Matthew V. ”Fast Text Compression with Neural Networks.” FLAIRS Conference. 2000. APA

[8] CMIX : http://www.byronknoll.com/cmix.html

Test data resources

[9] Nature Article on Genomic Boom: http://www.nature.com/news/genome-researchers-raise-alarm-over-big-data-1.17912

[10] Hutter Prize: http://prize.hutter1.net/

[11] Enwiki9: http://www.mattmahoney.net/dc/textdata

[12] HGP: Sawicki, Mark P., et al. ”Human genome project.” The American journal of surgery 165.2 (1993): 258-264.

Comparative Models

[12] GZIP: Deutsch, L. Peter. ”GZIP file format specification version 4.3.” (1996).

[13] MFCompress: Pinho, Armando J., and Diogo Pratas. ”MFCompress: a compression tool for FASTA and multi-FASTA data.” Bioinformatics 30.1 (2014): 117-118.

[14] CABAC: Prakasam, Ramkumar. ”Context adaptive binary arithmetic code decoding engine.” U.S. Patent No. 7,630,440. 8 Dec. 2009.