Deep Reinforcement Learning for a Dictionary Based Compression Schema (Student Abstract)

Keren Nivasch*, Dana Shapira, Amos Azaria
Data Science Center, Ariel University, Ariel 40700, Israel
{kerenni, shapird, amos.azaria}@ariel.ac.il

Abstract

An increasingly important process of the internet age and the massive data era is file compression. One popular compression scheme, Lempel–Ziv–Welch (LZW), maintains a dictionary of previously seen strings. The dictionary is updated throughout the parsing process by adding new encountered substrings. Klein, Opalinsky and Shapira (2019) recently studied the option of selectively updating the LZW dictionary. They show that even inserting only a random subset of the strings into the dictionary does not adversely affect the compression ratio. Inspired by their approach, we propose a reinforcement learning based agent, *RLZW*, that decides when to add a string to the dictionary. The agent is first trained on a large set of data, and then tested on files it has not seen previously (i.e., the test set). We show that on some types of input data, RLZW outperforms the compression ratio of a standard LZW.

Introduction

Reinforcement Learning (RL) (Sutton and Barto 2017) is a machine learning paradigm, in which an agent employs trial and error to come up with a solution to a problem, obtaining rewards or penalties for the actions it performs. The goal of the agent is to maximize the total reward. The agent starts with random trials, and might finish with sophisticated tactics and skills.

Deep RL based methods have recently gathered great success in several domains, such as playing Atari games, the game of Go, and self-driving cars. However, most domains in which deep RL has been applied enjoy a fairly straightforward translation to the agent and RL domain. In this work, we apply RL techniques to the field of data compression. We propose to view both the encoder and the decoder as agents which, in different compression schemes, may be able to pick among several actions. In the context of data compression, we must use a deterministic agent so that both the encoder and the decoder take the exact same actions, and therefore are synchronized with the same world states. This is essential as the decoder must reconstruct the original uncompressed data. Therefore, in the test phase, either any element of exploration must be completely removed, or any form of exploration must be deterministic, for example, being based upon some shared seed.

Related Work

Lossless data compression methods can be partitioned into two main encoding families, *statistical methods*, which include Huffman and arithmetic coding, and *dictionary methods*, in which LZ77 and LZ78 are the most famous ones. Lempel–Ziv–Welch (LZW), a practical implementation of LZ78, was developed by Welch (1984). LZW employs a dictionary $\mathcal{D}$ of strings. $\mathcal{D}$ is traditionally initialized by the alphabet, e.g. the set of 256 ASCII characters. $\mathcal{D}$ is dynamically updated as the input file is processed by extending existing strings in $\mathcal{D}$ by a single character, with new encountered strings that are seen in the input file for the first time. Specifically, at each stage in the compression, substrings of the input file are incrementally extended with the following character until the resulting sequence does not appear in $\mathcal{D}$. The code for the sequence (without the new character) is added to the output, and a new code (for the sequence concatenated to the new character) is added to $\mathcal{D}$. Thus, the output is a sequence of pointers to the changing dictionary. Each time the dictionary size reaches a power of 2, the number of bits used to represent the pointers increases by 1. Usually there is a bound on the dictionary size. When $\mathcal{D}$ reaches this bound, no more strings are added to it and $\mathcal{D}$ remains static. Alternatively, $\mathcal{D}$ may be restarted. Klein, Opalinsky, and Shapira (2019) studied a variant of LZW in which new strings are not always added to $\mathcal{D}$. Rather, there exists a parameter k, and a new string is added to $\mathcal{D}$ only every kth time. They found that this variant has the advantage of reducing the processing time without adversely affecting the compressing ratio. In this work, we develop RLZW, a variant of LZW, in which an RL component decides whether to insert each new string into $\mathcal{D}$ or not. Our agent uses the Q-Learning algorithm (Sutton and Barto 2017).

While, to the best of our knowledge, no previous work has introduced RL to dictionary-based compression methods, several works applied deep learning to data compression. In most cases, these methods use deep learning strategies to predict the upcoming characters or set of characters (Shermer, Avigal, and Shapira 2010; Liu et al. 2018). Combining RL and data compression has only been applied on *lossy* compression (e.g. (Xu, Nandi, and Zhang 2003; Zhu,

*Happamon 7, Kedumim, Israel. Phone: +972-58-5405402.

Lan, and van der Schaar 2013; Oladell and Huber 2012)). We note that several works have used compression in order to speed up deep learning (Ba and Caruana 2014; Amado and Meneguzzi 2018).

RLZW: Applying RL to LZW

RLZW is a neural network Q-Learning compression algorithm based on LZW. An RL agent must define states, actions and a reward function. RLZW follows the LZW algorithm, except when encountering a string that does not appear in the dictionary. Each time RLZW encounters a new string, w, RLZW is required to select between two actions: inserting it into $\mathcal{D}$ or not. The reward function is set to the difference, in bits, between the length of the uncompressed string and the length of the corresponding pointer to the dictionary: $r = 8|w| - \lceil \log_2 |\mathcal{D}| \rceil$, where $|w|$ denotes the size of w, and $|\mathcal{D}|$ denotes the number of words in the dictionary.

The state consists of the following three parameters: A 1-hot representation of the string w, a representation of $\mathcal{D}$, and the number of strings that may still be added to $\mathcal{D}$. The representation of $\mathcal{D}$ is composed of 0's and 1's indicating, for each possible string, whether it exists in the dictionary.

The neural network is composed of the input layer (including the encoding of the state), a hidden layer of size 30, and two output neurons (one for each action).

Experimental Settings

Due to the large size of the representation of $\mathcal{D}$, we used a simplified model in which the files to be compressed contain only 5 characters $\Sigma = \{\_, e, g, h, t\}$. Therefore, the initialized size of $\mathcal{D}$ is 5. We limited the maximum size for $\mathcal{D}$ to 32 and the length of the strings in $\mathcal{D}$ to 4; hence the number of possible strings in this model is $5^1 + 5^2 + 5^3 + 5^4 = 780$ (which is the length of the representation of $\mathcal{D}$).

Our dataset was composed of the *ENGLISH text collection* obtained from the *Pizza&Chili corpus*. We removed all the characters except those in Σ and created 30 files of size 18KB each. To make the compression task more challenging, we added to each file a "header" of length 50 that also contains only characters from Σ but with a different distribution than the remainder of the file. Hence, during the processing of the header, the regular LZW algorithm was expected to fill $\mathcal{D}$ with strings that do not appear much in the remainder of the file. We hypothesised that RLZW will learn to avoid these strings.

We used 24 files for training and the remaining 6 for testing. The training was performed in 50 epochs, where in each epoch a parameter ε determined the probability of exploring (as opposed to exploiting). In the first 6 epochs ε was set to 1, and then linearly decreased until, at the last epoch, it reached 0.

Results

RLZW learned to insert into $\mathcal{D}$ several commonly used strings (such as the, whereas LZW added less relevant strings. Moreover, sometimes RLZW did not fill $\mathcal{D}$ to its full capacity, showing that it learned that with a smaller dictionary it needs fewer bits for encoding. In contrast, LZW filled $\mathcal{D}$ quickly to its full capacity.

Algorithm	Compression Ratio
LZW	0.389
RLZW (train)	0.288
RLZW (test)	0.309

Table 1: Comparison between LZW and RLZW

Overall, RLZW succeeded to compress the training files 26% better than LZW, and the test files 21% better than LZW. See Table 1.

Conclusions and Future Work

In this paper we presented RLZW, an RL based agent that decides whether to insert the current string to the LZW dictionary or not. We showed that on some types of input data, RLZW outperformed the compression ratio of LZW.

The next steps are to extend this method to a larger alphabet and a larger dictionary size. We will consider additional reinforcement learning methods, such as a deep Actor-Critic learner. To the best of our knowledge, this work is the first to use a reinforcement learning agent in a dictionary based compression schema.

Acknowledgments

This work was supported by the Ministry of Science & Technology, Israel and by the Data Science and Artificial Intelligence Center of Ariel University.

References

Amado, L.; and Meneguzzi, F. 2018. Q-Table compression for reinforcement learning. *Knowledge Eng. Review* 33: e22.

Ba, J.; and Caruana, R. 2014. Do Deep Nets Really Need to be Deep? In *NIPS 2014*, 2654–2662.

Klein, S. T.; Opalinsky, E.; and Shapira, D. 2019. Selective Dynamic Compression. In *Stringology 2019*, 102–110.

Liu, H.; Chen, T.; Shen, Q.; Yue, T.; and Ma, Z. 2018. Deep Image Compression via End-to-End Learning. In *CVPR 2018*, 2575–2578.

Oladell, M. C.; and Huber, M. 2012. Symbol Generation and Grounding for Reinforcement Learning Agents Using Affordances and Dictionary Compression. In *FLAIRS 2012*.

Shermer, E.; Avigal, M.; and Shapira, D. 2010. Neural Markovian Predictive Compression: An Algorithm for Online Lossless Data Compression. In *DCC 2010*, 209–218.

Sutton, R. S.; and Barto, A. G. 2017. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition.

Welch, T. 1984. A Technique for High-Performance Data Compression. *IEEE Computer* 17(6): 8–19.

Xu, W.; Nandi, A. K.; and Zhang, J. 2003. A new fuzzy reinforcement learning vector quantization algorithm for image compression. In *ICASSP 2003*, 269–272.

Zhu, X.; Lan, C.; and van der Schaar, M. 2013. Low-complexity reinforcement learning for delay-sensitive compression in networked video stream mining. In *ICME 2013*, 1–6.