Compressing Genomic Sequences by Using Deep Learning

Wenwen Cui, Zhaoyang Yu, Zhuangzhuang Liu, Gang Wang, and Xiaoguang Liu(✉)

College of CS, TJ Key Lab of NDST, Nankai University, Tianjin, China
{cuiww,yuzz,liuzhuangzhuang,wgzwp,liuxg}@nbjl.nankai.edu.cn

Abstract. Huge amount of genomic sequences have been generated with the development of high-throughput sequencing technologies, which brings challenges to data storage, processing, and transmission. Standard compression tools designed for English text are not able to compress genomic sequences well, so an effective dedicated method is needed urgently. In this paper, we propose a genomic sequence compression algorithm based on a deep learning model and an arithmetic encoder. The deep learning model is structured as a convolutional layer followed by an attention-based bi-directional long short-term memory network, which predicts the probabilities of the next base in a sequence. The arithmetic encoder employs the probabilities to compress the sequence. We evaluate the proposed algorithm with various compression approaches, including a state-of-the-art genomic sequence compression algorithm DeepDNA, on several real-world data sets. The results show that the proposed algorithm can converge stably and achieves the best compression performance which is even up to 3.7 times better than DeepDNA. Furthermore, we conduct ablation experiments to verify the effectiveness and necessity of each part in the model and implement the visualization of attention weight matrix to present different importance of various hidden states for final prediction. The source code for the model is available in Github (https://github.com/viviancui59/Compressing-Genomic-Sequences).

Keywords: Genomic sequence compression · Deep learning · Arithmetic coding

1 Introduction

Due to the development of high-throughput sequencing technologies, the cost of sequencing has gradually decreased and the amount of genomic sequences has increased explosively [3]. Both researchers and doctors can gain the scientific knowledge of genomic sequences information, thereby promoting the development of biological science, as well as the therapy and diagnosis of patients.

This work is partially supported by National Science Foundation of China (61872201, 61702521, U1833114) and Science and Technology Development Plan of Tianjin (18ZXZNGX00140, 18ZXZNGX00200).

But the storage, processing and transmission of genomic sequences data have become major challenges [17]. Data compression can not only reduce the size of data storage, but also reduce the cost of processing and the time of transmission, which makes it become one of key ways to alleviate these problems [3].

A genomic sequence mainly consists of four nucleotides that is adenine (A), cytosine (C), guanine (G), and thymine (T). There are also many unknown nucleotides in genomic sequences which are generally represented by N [16]. General compression algorithms are designed dedicatedly for English text compression and they are not able to get an ideal compression performance for genomic sequences [1], in which the regularities are very small. The standard compression tools such as **compress**, **gzip** and **bzip2** have poor compression ratios [2]. Besides, the content of genomic sequences are different from general random text sequences, and the characteristics of sequences need to be taken into account for better compression, such as exact repeat, approximate repeat of a sub-string and complementary palindromes that may occur many times in a given genome sequence [1]. Furthermore, the probabilities of these base occurrence are not so different and inappropriate compression methods will decrease the speed of decompression, both making genomic sequence compression become a tough task [15,20]. Consequently, the compression of genomic sequence has become an important challenge and a specific method is needed urgently.

Arithmetic coding, a lossless data compression method, can minimize the redundancy of information and is optimal within one bit of the Shannon limit of $log_2 1/P(x)$ bits, where $P(x)$ is the probability of the entire input sequence x. However, its compression ratio depends almost entirely on the predictor [14], which is the key to high-quality compression. Recently, deep learning has developed as one of the most promising prediction methods. In particular, Recurrent Neural Networks (RNNs) have been widely used to deal with sequence data, such as the tasks of text, audio sequence prediction, and language translation, which are able to extract better structured information from vast amounts of sequences and find dependent relationship between contexts.

In this paper, we propose an effective genomic sequence compression algorithm combining a deep learning model comprising Convolutional Neural Network (CNN) and attention-based Bi-directional Long Short-Term Memory Networks (BiLSTM), and an arithmetic encoder. In summary, the contributions of this article are as follows:

- We introduce a deep learning model that automatically learns and captures features of genomic sequences to predict the probability of the each base at the next position.
- We use CNN to exact the local features of the sequences and use BiLSTM to learn the long-term dependence features of sequences as well as the characteristics of palindromes, considering both forward and backward directions.
- We exploit an attention mechanism to learn the importance of the hidden states and features provided by the BiLSTM for global situation. Higher weights are given to more important features to estimate the final probability more accurately.

– We conduct experiments based on several real-world genomic data sets and demonstrate the superior performance of the proposed approach for compressing genomic sequences. Ablation experiments verify the effectiveness of each part of the model and the visualization of attention weight matrix shows the various importance of hidden states and features of sequences for the probability estimation.

The rest of the paper is organized as follows: Sect. 2 discusses related work. Section 3 presents the overview of the proposed algorithm. Section 4 details the experimental settings and analyzes the results of experiments, and we conclude in Sect. 5.

2 Related Work

In this section, we report some main studies on genomic sequences compression, and divide them into two categories which are traditional methods and learning-based methods, according to whether deep learning methods are used. To compress genomic sequences, the former detects exact repeats or approximate repeats , while the latter resorts to deep learning technologies.

Traditional Methods. Biocompress [9] was the first genomic sequence compression algorithm, which detects exact and palindromes repeats in DNA sequence through an antomaton and encodes the lengths and positions of their earliest occurrences through Fibonacci encoding. The extension version, Biocompress-2 algorithm [10] uses a Markov model of order-2 to encode when there is non-repeat region. Besides, other algorithms exploit approximate repeats to encode sequences, such as GenCompress proposed by Chen et al. [5]. The followed DNA-Compress algorithm [6] searches all approximate repeats in the first phase and uses the Ziv and Lempel compression algorithm to encode them in the second phase. Behzadi and Le Fessant proposed DNAPack [2] which uses dynamic programming to find better repeats, achieving better average compression ratios.

In addition, some studies provide the probability of next symbol to the arithmetic coder in various ways. XM introduced by Cao et.al [4], applies Bayesian averaging of a mixture of experts to provide the probability, including an order-1 Markov, an order-2 Markov, and a copy expert. Although it gains better compression performance, it is not suitable for long sequences. MFCompress [18] uses single finite-context models for encoding the header text, as well as multiple competing finite-context models for encoding the main stream of the DNA sequences, but its main problem is memory limitation.

Learning-based Methods. Deep learning approaches such as CNN can be applied for predicting various biological targets through genomic sequences [19,22]. Inspired by DeepZip [8], Wang et al. [21] introduced DeepDNA for the human mitochondrial genomes compression. It relies on CNN to extract the local features and LSTM to capture the long-term dependence of the sequences to provide the probability for compression. Since only approximately 0.1% of 3GB human genome is specific [7], so DeepDNA shows excellent performance compared with **gzip**. But it does not consider the importance of chunks of sequences for the probability estimation of next base. Nor does it consider the feature of global sequences, such as complementary palindromes.

We improve DeepDNA by proposing a compression algorithm using deep learning based on local and global features of genomic sequences, including palindromes, approximate repeats and exact repeats.

3 Proposed Algorithm

The overview of our algorithm is illustrated in Fig. 1, including the following two main parts: a deep learning model to estimate probability of next base and an arithmetic encoder to encode this base. The process of compression is follows. A sliding window of a genomic sequence as the input is transformed into vectors in the pre-processing stage. Then the CNN and attention-based BiLSTM are used to extract the local and global features respectively. Through the fully connected layer and softmax layer on the top, we obtain a vector of the probabilities of bases. The probabilities are send to arithmetic encoder to obtain the final compressed data.

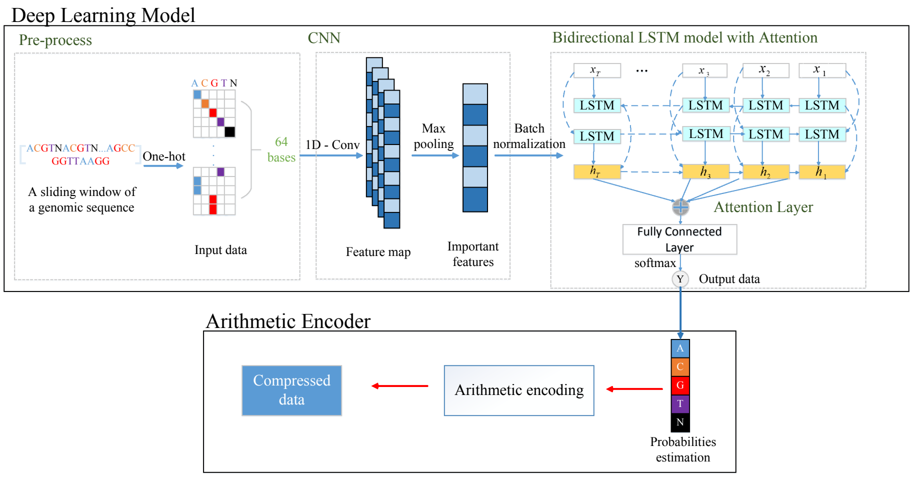

Fig. 1. The overview of proposed algorithm.

3.1 Deep Learning Model

Pre-processing. After inputting a sliding window of length n to the model, each base in it is firstly processed through one-hot encoding and represented by a 5-dimensional vector $x_i \in \mathbb{R}^5$, where i is the i-th base in the sliding window. Each dimension represents one kind of base, and the dimension corresponding to the current base is set to 1 while the other dimensions are set to 0. For example, we transform a A into $\{1,0,0,0,0\}$ and a C into $\{0,1,0,0,0\}$. So the sequences can be denoted by a set of vectors finally.

CNN. In this stage, the input set of n vectors of the sliding window $\mathbf{x}$ is processed by the weight-sharing strategy of filters to capture simple local patterns through a convolution unit. The exact repeated patterns have the same features, and the approximate repeated patterns have similar features. The calculation process of a convolution unit is represented by

$$z_{i,f}\left(\mathbf{x}\right) = \sigma\left(W_f \widehat{x}_i + b_f\right) \tag{1}$$

where $z_{i,f}\left(\mathbf{x}\right)$ gives the feature map of filter f for the i-th base of $\mathbf{x}$, W_f is the parameters matrix of filter f for the feature map, $\widehat{x}_i = [x_i, x_{i+1}, ..., x_{i+k-1}]$ denotes the concatenation of the vectors for k bases from the i-th base to the $i+k-1$-th base of input $\mathbf{x}$, b_f is a shared value of filter f for bias, and $\sigma\left(\cdot\right)$ is a non-linear activation function.

The max-pooling operation is taken for each feature map to select the most important features by choosing the max value in each block. Meanwhile, the size of the output is reduced by one third in the proposed model, reducing the complexity of computation.

As network depth increases, the changes of parameter during training process will cause the hidden layer to face the covariate shift problem that could reduce the convergence rate. We use batch normalization [12] to perform the normalization for each mini-batch, making the input of each layer keep the same distribution. It improves the training speed and the generalization of models, accelerates the convergence process, and alleviates over-fitting.

BiLSTM with Attention mechanism. LSTM was proposed to solve existing long-term dependencies problem of RNN. We use LSTM to extract sequential features between local patterns captured by CNN. The main idea of it is to use cell state with gates. Firstly, the forget gate decides what information is maintained from the last cell state. Next, the input gate controls what information updates the new cell state. Finally, the output gate decides what information of new cell state to be the next state. However, unidirectional LSTM is not enough for the genomic sequences because they contain many special features such as palindrome. Therefore, we employ an advanced bi-directional long short-term memory(BiLSTM), which is able to process sequences in both forward and backward directions for better capturing the long dependence in genomic sequences.

We use $\overrightarrow{h}$ and $\overleftarrow{h}$ to represent the hidden state of forward and backward direction respectively. The output of the t-th hidden state is a list of two directions of hidden states that can be represented as $h_t = \left[\overrightarrow{h_t}, \overleftarrow{h_t}\right]$. Due to the occurrence of approximate repeats in genomic sequences, various hidden states contribute differently to probability estimation. Therefore, we apply an attention mechanism to learn the importance of various parts automatically, which improves the performance of prediction and overall compression ratio.

Assuming that the LSTM layer produces m hidden states h_t, $t = 1, 2, ..., m$, and a score u_t is calculated by Eq. (2) to evaluate the importance for each hidden state.

$$u_t = \sigma(W^T h_t + b) \tag{2}$$

where W^T is the parameter vector and b is the bias. Finally, the softmax function α_t given by Eq. (3) is used to calculate the weighted average coefficient of score u_t, and the output of attention mechanism o is the sum of the products of hidden states and theirs scores, which is shown in Eq. (4).

$$\alpha_t = \frac{\exp(u_t)}{\sum_{t=1}^{m} \exp(u_t)} \tag{3}$$

$$o = \sum_{t=1}^{m} \alpha_t h_t \tag{4}$$

3.2 Arithmetic Coder

Arithmetic coding maps the data to be encoded, that is the gene sequence, to a decimal between the interval $[0, 1]$. The concrete algorithm is described in Algorithm 1. $level_{low}$ and $level_{high}$ represent the lower and upper boundaries of the current interval. $delta$ represents the length of the interval. $base.level_{low}$ and $base.level_{high}$ represent the lower and upper boundaries of the coding interval for each base respectively. When coding, starting from the initial interval $[0, 1]$, the current interval is divided into multiple sub-intervals according to the probability of the bases (A, C, G, T, N). Then, select the corresponding sub-interval according to the current input base and use this sub-interval as the current interval for the next coding step. Repeat this process until all bases are encoded. Finally, assign a unique decimal from the final sub-interval as the encoding result of the gene sequence.

Algorithm 1. Arithmetic Encoding

Input: The base table $dict = \{A, C, G, T, N\}$, the probability P of five bases in $dict$ and the gene sequence Seq

Output: a decimal

```
1: $level_{low}, inter \leftarrow 0$
2: $level_{high}, delta \leftarrow 1$
3: for base in $dict$ do
4:     $base.level_{low} \leftarrow inter$
5:     $base.level_{high} \leftarrow inter + P_{base}$
6:     $inter \leftarrow base.level_{high}$
7: end for
8: for base in $Seq$ do
9:     $delta \leftarrow level_{high} - level_{low}$
10:    $level_{high} \leftarrow level_{low} + delta * base.level_{high}$
11:    $level_{low} \leftarrow level_{low} + delta * base.level_{low}$
12: end for
13: return a random decimal between $level_{low}$ and $level_{high}$
```

Algorithm 2. Arithmetic Decoding

Input: The base table $dict = \{A, C, G, T, N\}$, the probability P of five bases in $dict$, the encoded decimal d and the length of compressed sequence len
Output: the gene sequence Seq

```
 1: $inter \leftarrow 0$
 2: $num \leftarrow 1$
 3: for base in $dict$ do
 4:     $base.level_{low} \leftarrow inter$
 5:     $base.level_{high} \leftarrow inter + P_{base}$
 6:     $inter \leftarrow base.level_{high}$
 7: end for
 8: while $num \leq len$ do
 9:     find the base $base$ whose interval covers $d$
10:     append $base$ to $Seq$
11:     $delta \leftarrow base.level_{high} - base.level_{low}$
12:     $d \leftarrow (d - base.level_{low})/delta$
13:     $num \leftarrow num + 1$
14: end while
15: return the gene sequence $Seq$
```

The process of decoding which is described in Algorithm 2 is equivalent to the inverse operation of the encoding process. When decoding, only one decimal is entered. First, divide the initial interval $[0, 1]$ according to the probability of bases. Second, observe which sub-interval the input decimal is in and output the corresponding base. Third, select this sub-interval and divide it continually. Repeat this process until all bases of the gene sequence are decoded.

4 Experiments and Evaluation

4.1 Data Sets and Baselines

The effectiveness of proposed algorithm is evaluated through several data sets, including 2851 mitochondrial sequences of various fishes species and 745 mitochondrial sequences of various bird species downloaded from National Center for Biotechnology Information[1], 303 transcriptome sequences of ray-finned fishes used in [11] and 1000 human mitochondrial sequences obtained from DeepDNA [21] respectively. We use 70% of each data set for training, 20% for validation, and the rest of 10% for testing. The statistics of data sets are summarized in Table 1.

Table 1. The statistics of data sets

Data sets	Bases	Sequences	Size (KB)	Proportion				
				A	C	G	T	N
Fishes	5	2851	53336	29.28%	27.86%	16.37%	26.48%	0.0024%
Birds	5	745	12703	30.44%	31.22%	14.35%	23.95%	0.04%
Human	4	1000	16532	30.90%	31.25%	13.15%	24.70%	
Ray-finned fishes	5	305	169385	14%	17%	17%	12%	40%

[1] https://www.ncbi.nlm.nih.gov/.

Based on these data sets, we compare the performance of proposed algorithm with (a) **Gzip** and **7-zip**, classic standard compression tools, (b) **MFCompress**: a traditional and state-of-the-art compression algorithm, which can compress multi-fasta files, and (c) **DeepDNA**, a learning-based method, which applies deep learning methods, i.e. CNN and LSTM, for compressing genomes.

4.2 Configurations of Our Deep Learning Model

We set the sliding window size to be 64, the number of filters of CNN layer to be 1024 and the size of the convolution unit to be 24 with step of 1. For max-pooling layer, the size of the window is 3 with the same size of step. The dimension of LSTM is set to be 256, which makes BiLSTM to be 512 dimensions for each hidden state. We train the deep learning model until it converges.

The loss function in the training process based on cross entropy. The smaller the value of loss function is, the better the model's performance manifests. And we use a optimizer in [13] to directly minimize the loss function with a mini-batch size of 128. The learning rate is set to 0.0001 at the beginning and changes adaptively by the optimizer during the training process. The size of the model weights is only 11.8MB. We explore more innovations in the model structure to achieve superior compression performance rather than just stacking parameters and the experimental results are shown in Sect. 4.3. So our model can be used in various sequences with a wider application prospect.

4.3 Experimental Results and Analysis

Compression Performance. Table 2 presents the results of the experiment comparing the proposed algorithm with the baselines based on the data sets, in terms of bpb (bit per base).

Table 2. Compression performance of various methods (bpb)

Methods	Name	Fishes	Birds	Human	Ray-finned fishes
Traditional	Gzip	2.4968	2.47	2.519	1.69
	7-zip	1.2628	1.1692	0.0929	**0.6698**
	MFCompress	1.42	1.364	1.5572	1.1155
learning-based	DeepDNA	1.3591	1.363	0.0548	1.4693
	Proposed algorithm	**0.7036**	**0.6655**	**0.01456**	0.8193

From Table 2, the following observations can be made. First, The proposed algorithm beats all the traditional or learning-based methods on Fishes, Birds and Human data sets. That is because the proposed algorithm can learn more complicated patterns from the sequences to be utilized in compression. Second, the learning-based methods achieve even greater advantages over the traditional methods on the Humans data set, since it has a higher genetic similarity than other data sets. Meanwhile, the performance of the proposed method outperforms DeepDNA's, even up to 3.7 times. Third, the traditional methods perform better on the data set of Ray-finned fishes than the learning-based method DeepDNA. The 7-zip achieves the best result of 0.6698, because this data set contains many continuous 'N' bases, fitting in the mechanism of traditional compression algorithm. However, the proposed algorithm obtains the performance that is next to 7-zip closely.

Comparison with DeepDNA. Learning-based methods can mine the potential connections of genetic information more intelligently meeting the compression needs, so we compare learning ability of the deep learning model in our proposed algorithm with another learning-based method DeepDNA. From the results of experiments in Table 2, our algorithm performance better than DeepDNA on all data sets. Due to the limitation of space, we only display the results on Fishes data. To verify its convergence, we plot the average loss of validation set after each epoch in the training process in Fig. 2. We can see that the average loss of our model steadily decreases as traing processes while the loss of DeepDNA increases after epoch 1 and the loss of subsequent epochs fluctuates up and down repeatedly, which shows that our model gradually converges and learns the features of sequences better than DeepDNA during the training process.

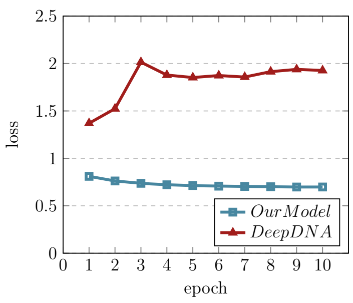

Fig. 2. The average loss of validation set during training.

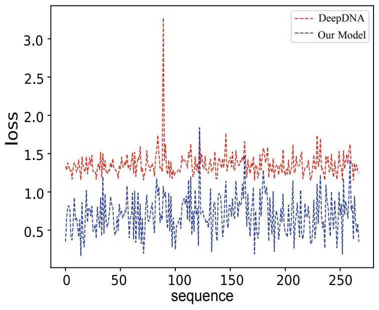

Fig. 3. The test loss of our model compared with DeepDNA

We then calculate the loss of all sequences in the test data sets as shown in Fig. 3 to verify the efficiency of the proposed model. Each sample of the horizontal axis represents a complete genomic sequence and the vertical axis shows its average loss. As the figure shown, for each sample (281 in total), the loss of our model is lower obviously than that of DeepDNA.

Since the average loss of a sequence is susceptible to the extreme loss of some slide windows in the sequence, so we randomly select a sequence whose ID is NC_011180.1 in the test set and plot the average loss of every ten consecutive sliding windows for further exploration and illustration. The result is shown

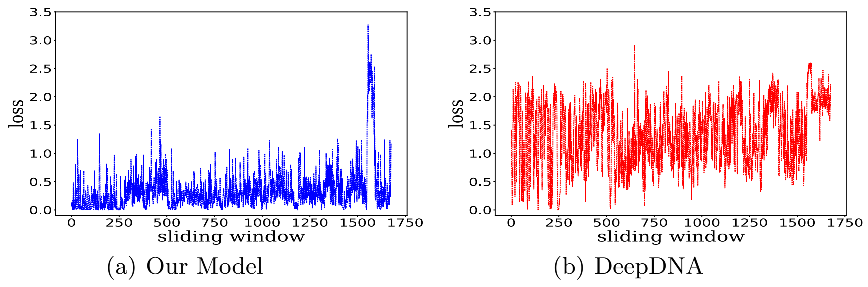

Fig. 4. Average loss of ten consecutive sliding windows in sequence NC_011180.1.

in Fig. 4. We can see that the values of loss fluctuation are relatively smooth between samples, which are in 0-0.5 in general from Fig. 4 (a) while in 0.5-2 for DeepDNA in Fig. 4 (b). The average loss of this sequence is 0.35 when using our model and 1.33 when using DeepDNA.

Ablation Experiments. We also conduct ablation experiments that delete or replace one part of our model to verify their importance based on training, validation, and test data sets respectively. The results are shown in Table 3, in which **Without CNN** means removing the CNN layer and the max-pooling layer, **Without BiLSTM** means replacing the BiLSTM with a unidirectional LSTM, **Without Attention** means removing the top-layer attention mechanism of the model and **Complete Model** denotes the one combining all the parts. Except for the removed parts, the remaining parts of these models are consistent with those in the complete model.

Table 3. Results of ablation experiments

Model	Training	Validation	Test
Without CNN	0.6978 + 15.2%	0.7232 + 3.5 %	0.7137 + 1.4 %
Without BiLSTM	0.6710 + 10.8%	0.7321 + 4.8 %	0.7320 + 4.0 %
Without Attention	**0.5964−1.4%**	0.7107 + 1.7 %	0.7174 + 1.9 %
Complete Model	0.6053	**0.6982**	**0.7036**

As shown in the table, each part of our model is indispensable. Although **Complete Model** is 1.4% worse than **Without Attention** on the training set, it has the best performance on both the validation set and the test set and outperforms other models up to 4.8% and 4% respectively. In addition, the results of **Without BiLSTM** show that bi-directional LSTM is able to capture long-term dependency and learn the special features such as complementary palindromes much better in the genomic sequences.

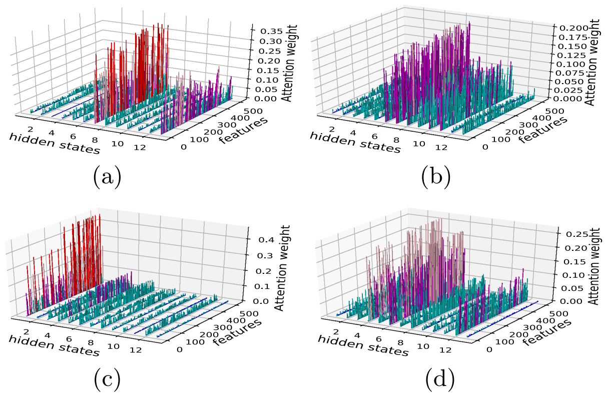

Fig. 5. The attention weights in four sliding windows

Visualizing Attention. For LSTM network without attention mechanism, each hidden state has the same weight of the final feature representation or is merely based on the last hidden state. However, various hidden states contribute to the final prediction differently. The attention mechanism automatically learns the importance of hidden states to assign higher weights for more important ones. To better interpret the attention mechanism, as shown in Fig. 5, we visualize the attention weight matrix of four randomly selected sliding windows and there are 512 features being generated in each hidden state. Different colored bars display the different ranges of the weight values of high-level features learned by BiLSTM.

We can observe that the seventh hidden state and the last hidden state are assigned higher weights in Fig. 5 (a), while the first hidden state has a higher weight in Fig. 5 (c), and there is the same regularity in (b) and (d). What's more, it can be seen that the contributions of 512 features in each hidden state are different, too. So the learned varieties of weight values promoting the accuracy for the final prediction, indicting the effectiveness of attention mechanism.

5 Conclusion

Facing the challenges of genomic sequences compression, we propose an algorithm including a deep learning model and arithmetic encoder. The deep learning model consists of CNN, BiLSTM and attention mechanism that learns the characteristics of the genomic sequences to estimate the probability needed in the arithmetic encoder for compression. Also, we conduct experiments on various data sets, indicating the proposed algorithm outperforms the learning-based method DeepDNA and traditional compression methods. At the same time, additional ablation experiments prove the importance of each part of the proposed algorithm and the visualization of attention weights shows various importance of hidden states to the final prediction.

In the future, we intend to input pure sequences and additional information of them, such as the species of sequence or any other properties to the algorithm. It may help reduce the redundancy of genomic sequences for better compression.

References

1. Bakr, N.S., Sharawi, A.A., et al.: DNA lossless compression algorithms. Am. J. Bioinf. Res. **3**(3), 72–81 (2013)
2. Behzadi, B., Le Fessant, F.: DNA compression challenge revisited: a dynamic programming approach. In: Apostolico, A., Crochemore, M., Park, K. (eds.) CPM 2005. LNCS, vol. 3537, pp. 190–200. Springer, Heidelberg (2005). https://doi.org/10.1007/11496656_17
3. Berger, B., Peng, J., Singh, M.: Computational solutions for omics data. Nat. Rev. Genet. **14**(5), 333 (2013)
4. Cao, M.D., Dix, T.I., Allison, L., Mears, C.: A simple statistical algorithm for biological sequence compression. In: 2007 Data Compression Conference (DCC 2007), pp. 43–52. IEEE (2007)
5. Chen, X., Kwong, S., Li, M.: A compression algorithm for DNA sequences and its applications in genome comparison. Genome Inform. **10**, 51–61 (1999)
6. Chen, X., Li, M., Ma, B., Tromp, J.: Dnacompress: fast and effective DNA sequence compression. Bioinformatics **18**(12), 1696–1698 (2002)
7. Deorowicz, S., Grabowski, S.: Robust relative compression of genomes with random access. Bioinformatics **27**(21), 2979–2986 (2011)
8. Goyal, M., Tatwawadi, K., Chandak, S., Ochoa, I.: Deepzip: lossless data compression using recurrent neural networks. arXiv preprint arXiv:1811.08162 (2018)
9. Grumbach, S., Tahi, F.: Compression of DNA sequences. In: Proceedings of DCC93: Data Compression Conference, pp. 340–350. IEEE (1993)
10. Grumbach, S., Tahi, F.: A new challenge for compression algorithms: genetic sequences. Inf. Process. Manage. **30**(6), 875–886 (1994)
11. Hughes, L.C., et al.: Comprehensive phylogeny of ray-finned fishes (Actinopterygii) based on transcriptomic and genomic data. Proc. Natl. Acad. Sci. **115**(24), 6249–6254 (2018)
12. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, pp. 448–456 (2015)
13. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
14. Mahoney, M.V.: Fast text compression with neural networks. In: FLAIRS Conference, pp. 230–234 (2000)
15. Matsumoto, T., Sadakane, K., Imai, H.: Biological sequence compression algorithms. Genome Inf. **11**, 43–52 (2000)
16. Mishra, K.N., Aaggarwal, A., Abdelhadi, E., Srivastava, D.: An efficient horizontal and vertical method for online DNA sequence compression. Int. J. Comput. Appl. **3**(1), 39–46 (2010)
17. Muir, P., et al.: The real cost of sequencing: scaling computation to keep pace with data generation. Genome Biol. **17**(1), 53 (2016)
18. Pinho, A.J., Pratas, D.: Mfcompress: a compression tool for fasta and multi-fasta data. Bioinformatics **30**(1), 117–118 (2013)

19. Quang, D., Xie, X.: DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. Nucleic Acids Res. **44**(11), e107–e107 (2016)
20. Sato, H., Yoshioka, T., Konagaya, A., Toyoda, T.: DNA data compression in the post genome era. Genome Inform. **12**, 512–514 (2001)
21. Wang, R., et al.: Deepdna: a hybrid convolutional and recurrent neural network for compressing human mitochondrial genomes. In: 2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 270–274. IEEE (2018)
22. Zhou, J., Troyanskaya, O.G.: Predicting effects of noncoding variants with deep learning-based sequence model. Nat. Methods **12**(10), 931 (2015)