Wang *et al. Human Genomics* 2019, **13**(Suppl 1):49
https://doi.org/10.1186/s40246-019-0225-3

Human Genomics

RESEARCH Open Access

Human mitochondrial genome compression using machine learning techniques

Rongjie Wang[1], Tianyi Zang[2*] and Yadong Wang[2*]

From IEEE International Conference on Bioinformatics and Biomedicine 2018
Madrid, Spain. 3–6 December 2018

Abstract

Background: In recent years, with the development of high-throughput genome sequencing technologies, a large amount of genome data has been generated, which has caused widespread concern about data storage and transmission costs. However, how to effectively compression genome sequences data remains an unsolved problem.

Results: In this paper, we propose a compression method using machine learning techniques (DeepDNA), for compressing human mitochondrial genome data. The experimental results show the effectiveness of our proposed method compared with other on the human mitochondrial genome data.

Conclusions: The compression method we proposed can be classified as non-reference based method, but the compression effect is comparable to that of reference based methods. Moreover, our method not only have a well compression results in the population genome with large redundancy, but also in the single genome with small redundancy. The codes of DeepDNA are available at https://github.com/rongjiewang/DeepDNA.

Keywords: Compression, Human mitochondrial genomes, Machine learning

Background

The Human Genome Project (HGP) cost about $3 billion and took about 13 years, the completion of the Human Genome Project signs of the beginning of human genome research in the life sciences has entered a new era of genome [1]. Since then, The amount of data in the genome is growing exponentially, even faster than Moore's Law [2]. Today's high-throughput sequencing technology enables sequencing of individual genomes in a matter of hours, with sequencing costs less than $1,000. These advances have allowed researchers to increase the scope for scientific discovery through large amounts of data. However, the huge amount of genomic data presents new challenges for efficient storage and transmission.

Genome sequences are generally stored in FASTA format [3]. In this format, genome sequence characters are stored in ASCII-based, and represented by four diiferent symbols (called nucleotides or bases), namely (A) adenine, (C) cytosine, (T) thymine, (G) guanine. The problem we face is how to effectively compress strings of a certain length composed of these four elements.

The existing genome compression methods were major based on the dictionary methods [4–7], and based on statistical methods [8, 9]. DeepZip [10], as a machine learning compression method, compression the general context data at a online learning model. The parameters of compression is updated once after compressing each character, trying to learn the pattern of input data. The consequence of this approach is that a lot of time is spent updating model parameters during compression and decompression. Another minor defect is that when a pattern first appears in the input data, the compression and decompression model cannot be recognized it immediately. It needed to learn the pattern, and then, can be

*Correspondence: tianyi.zang@hit.edu.cn; ydwang@hit.edu.cn
[2]School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
Full list of author information is available at the end of the article

effectively compressed when these pattern were encountered again, which affects the compression result. As we know that genomes within the same species have a highly similarity, for example, genome similarity between two human individuals up to 99.9%. Even not in the same species, the genome similarity between humans and chimpanzees can be as high as 89% [11]. The banana genome, which seems to have nothing to do with the human genome, has a similarity of 50% [12]. In this work, we proposes a static machine learning compression method, use part of experiment data as a training set, optimal the compression model parameters. Then use the other part as test data to test the effect of compression model.

In computer vision tasks, the deep learning model has achieved some good performance, such as using Convolutional Neural Network (CNN) and Long Short-Term Memory Networks (LSTM) models to solve text classification [13], image caption generation [14] and speech recognition [15], and so on. However, in genome sequence data compression, for the first time, we tried to use a deep learning model to learn the sequence of patterns in the genome, and to predict the probabilities of the next base to be encoded, followed by arithmetic coding and output compressed data stream.

We verified the validity of our proposed method in 1,000 human mitochondrial sequences, and randomly divided the data set into three parts in proportion (training set, verification set and test set). Then we trained our model with the training set, the verification set was used to select the optimal parameters, and the test set verified the effectiveness of the deep learning model.

The remainder of this paper is organized as follows: Section II describes the DeepDNA method in detail, section III reports the experimental performance of DeepDNA, conclusion is drawn in Section IV.

Methods

Overview

For a length of T sequence steam $x_{1:T} = x_1, x_2, \ldots, x_T$, where each variable $x_i \in \Sigma, i \in [1, T]$, for genome sequence, $\Sigma = \{A, C, G, T\}$. The probability of the entire sequence $x_{1:T}$is:

$$\begin{aligned} p(x_{1:T}) &= p(x_1)\, p(x_2|x_1)\, p(x_3|x_{1:2}) \cdots p\left(x_T|x_{1:(T-1)}\right) \\ &= \prod_{t=1}^{T} p\left(x_t|x_{1:(t-1)}\right) \end{aligned} \tag{1}$$

Therefore, the probability density estimation problem of sequence data can be converted into a univariate conditional probability estimation, that means, the probability of a sequence can be viewed as the product of its probability of sub-sequence. The more likely a sequence is to occur, the lower its entropy value and the better compression result. In other words, the more accurate the prediction of conditional probability events of sub-sequence, the better the compression effect achieved. The conditional probability can be expressed as $p(x_t|x_{1:(t-1)})$ of x_t given $x_{1:(t-1)}$, which fed into the arithmetic encoding tool, to get the final compressed file.

Given N sequences data, the sequence probability model needs to learn a model $p_\theta\left(x_t|x_{1:(t-1)}\right)$, to maximize the log likelihood function of the entire data set.

$$\max_\theta \sum_{n=1}^{N} \log p_\theta\left(x_{1:T_n}^{(n)}\right) = \max_\theta \sum_{n=1}^{N} \sum_{t=1}^{T_n} \log p_\theta\left(x_t^{(n)}|x_{1:(t-1)}^{(n)}\right) \tag{2}$$

We can use the neural network model to estimate the conditional probability $p_\theta\left(x_t|x_{1:(t-1)}\right)$. Suppose a neural network $f(\theta)$, whose input is the historical information $x_{1:(t-1)} = x_1, x_2, \ldots, x_{t-1}$, the output is the occurrence of next base x_t, the output four nucleotides probabilities satisfy:

$$\sum_{x_t \in \{A,C,G,T\}} p_\theta\left(\mathbf{x_t}|\mathbf{x_{1:(t-1)}}\right) = 1 \tag{3}$$

Where θ represents the neural network parameter, the conditional probability $p_\theta\left(x_t|x_{1:(t-1)}\right)$ can be obtained from the output of the neural network.

The architecture of our neural network model DeepDNA is shown in Fig. 1, the framework mainly has six layers: the first layer is the single one-hot representation, which convert the genome sequence nucleotides $\{A, C, G, T\}$ to vectors. The second convolution layer extracts the context short-term correlation in the genome. The third layer is the pooling layer to remove the noise. The fourth layer, LSTM, extracted the long-term correlation in the genome. The fully connected layer and the last layer are used for outputting next nucleotides $\{A, C, G, T\}$ probabilities.

The following subsections describe how we apply CNN to extract genome sequences local features, LSTM to capture long-term dependencies over window features sequence and fed output layer to the arithmetic coder for getting the bit-stream respectively.

Convolutional Neural Network (CNN)

One-dimensional convolution operation corresponds a series of filters sliding over the genome sequence identify the sequence characteristics at different positions in the genome, it takes the one-hot encoding of the genome as input, where one can be defined as:

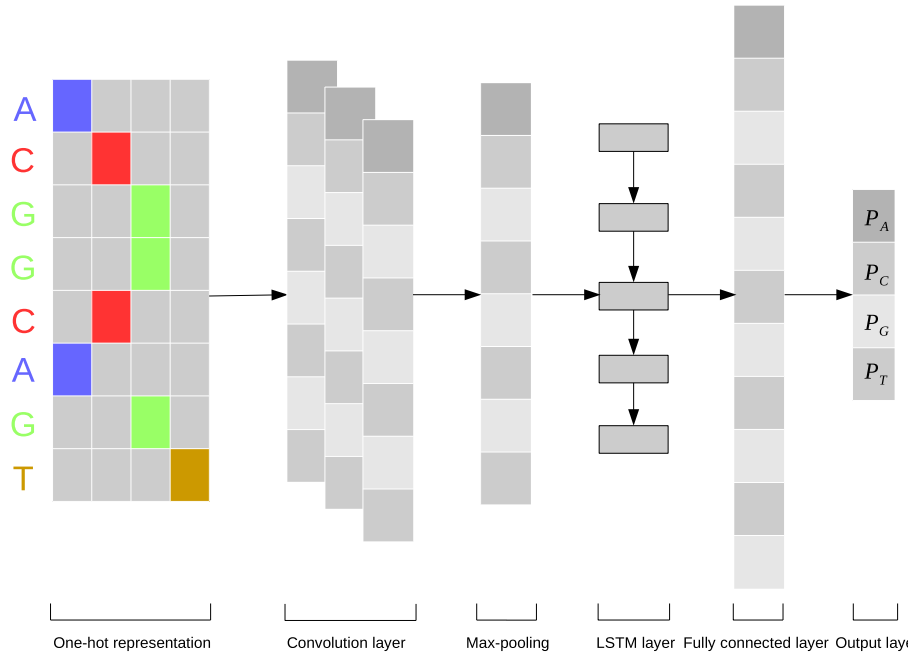

Fig. 1 The architecture of the DeepDNA model. Firstly, the input genome sequence is transformed into one-hot 4-dimensions bit matrix; A convolution layer activated by a rectified linear units acts as a local feature extractor, its output is a matrix with column matrix of the convolution filter and the row matrix of the position in the input sequence; A max-pooling procedure is used to reduce the size of the output matrix and only preserve the main features; The subsequent Long Short-Term Memory network (LSTM) layer is considered as acting the role of capturing sequence long-term features; A flattened fully connected layer is to collect LSTM outputs; The last layer performs a sigmoid non-linear transformation to a vector that serves as probability predictions of the sequence base

$$\begin{aligned} A &\rightarrow [1,0,0,0] \\ C &\rightarrow [0,1,0,0] \\ G &\rightarrow [0,0,1,0] \\ T &\rightarrow [0,0,0,1] \end{aligned} \tag{4}$$

Let $x \in \mathbb{R}^{T\times 4}$ represents the genome sequence with input length of T, and $x_i \in \mathbb{R}^4$ is the base vector representation of the ith position in the sequence. There are m filters in total, and the output of filters are $o \in \mathbb{R}^{(N-k+1)\times m}$, the convolution operation of each element $o_{i,j}$ is defined as follows:

$$o_{i,j} = \delta(w_j \odot [x_i, x_{i+1}, \ldots, x_{i+k-1}] + b_j) \tag{5}$$

where $w_j \in \mathbb{R}^{k\times 4}$ is the jth filter vector, and $b_j \in \mathbb{R}$ is a shared value for the bias for filter w_j, symbol $\odot$ is a convolution operation, define as follow:

$$w_j \odot [x_i, x_{i+1}, \ldots, x_{i+k-1}] = \sum_{n=0}^{k-1} w_{nj} x_{i+n} \tag{6}$$

Symbol δ is the neural nonlinear activation function, we select the ReLU [16] operation as the activation function, which outputs negative value to 0 and as defined below:

$$ReLU(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ if } x \leq 0 \end{cases} \tag{7}$$

We use the max-pooling as the convolution layer after the output processing for the output vector feature extraction. Max- pooling operation selects the maximum value in an unit area as the representative feature of the sequence, which is used to extract the sequence higher scale features in the next layer.

To reduce over-fitting, a dropout layer is connected after the maximum pooling layer. The term "dropout" refers to the temporary deletion a apart of units in the neural network, deleting all the links associated with it. The probability of choosing which unit to be dropped is independent of the others, and with a fixed probability p.

Long short-term memory networks (LSTM)

Recurrent neural network (RNNs) propagates historical message through a concatenation network structure, but has the problem of long gradient disappearance. Long Short-Term Memory Networks (LSTM) [17], as a special recurrent neural network, was proposed in 1997 to solve the problem that recurrent neural network (RNNs) cannot learn long-term dependent correlation. They are

now widely used in various time series problems and have achieved a series of excellent results.

LSTM hierarchical transformation function is defined as follows:

$$\begin{aligned} f_t &= \delta(W_f \cdot [x_t, h_{t-1}] + b_f) \\ i_t &= \delta(W_i \cdot [x_t, h_{t-1}] + b_i) \\ o_t &= \delta(W_o \cdot [x_t, h_{t-1}] + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot tanh(W_c \cdot [x_t, h_{t-1}] + b_c) \\ h_t &= o_t \odot tanh(c_t) \end{aligned} \tag{8}$$

Where $f_t \in \mathbb{R}^h$ is a forgetting gate to control which messages in the old memory unit will be discarded; $i_t \in \mathbb{R}^h$ is an inputting gate to control how much new messages will be recorded in the current memory unit; $o_t \in \mathbb{R}^h$ is an output gate to control output in the history unit. Through the joint control of the above three gates, the problem of long-term gradient disappearance is solved, and the dependence of long-term context can be excavated.

The terms W_f, W_i, W_o, and W_c denote weight matrices for forgetting gate, inputting gate, outputting gate, and unit state connections. The terms b_f, b_i, b_o, and b_c denote the bias vectors of the forgetting gate, inputting gate, outputting gate, and unit state connections. x_t, h_{t-1} is the input sequence data for the current time and state output for previous time separately.

The symbol δ is function of logistic sigmoid, which limits output range to $[0, 1]$, defined as:

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

The function of *tanh* limits output range to $[-1, 1]$, which is expressed as hyperbolic tangent function and defined by:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{10}$$

The symbol $\odot$ represents the dot product operation of the element, as described in the formula 6.

Arithmetic encoder

The arithmetic encoder [18] was proposed in 1976 by Rissanen and Pasco to solve the problem of infinite decimal precision, is a coding approach closest to information entropy when data distribution was fixed. Instead of encoding each character to an integer, the arithmetic encoder encrypts the sequence as a sufficiently precise numeric value in the interval $(0, 1)$, called sequence identifier or label. As the encoding progresses, the label interval becomes smaller and smaller, and the next interval range is fixed by the probability of encoding character. The decoding operation is similar to encode, given the character probabilities, the arithmetic decoder predict the probability to the corresponding character interval. As long as the decimal representation of the interval identifier or label is accurate enough, the decoder is able to recover the whole encoding sequence in lossless.

Figure 2 demonstrates the identifier determination procedure of arithmetic encoder, for example, coding a sequence 'CGTA', we assume that the probability values of each base are: $p(A) = p(T) = 0.2, p(C) = 0.5, p(G) = 0.1$. At beginning, the initial interval is $(0, 1)$, then the first base 'C' limited the interval to $(0.2, 0.7)$, base 'G' limited the interval to $(0.55, 0.6)$, and so on $\cdots$. The latter interval is a subset of the former interval, so the range will be more and more smaller, lastly, the identifier is limited the interval to [0.59, 0.592]. We can choose any point within this interval, like a middle value 0.591, its binary value stream

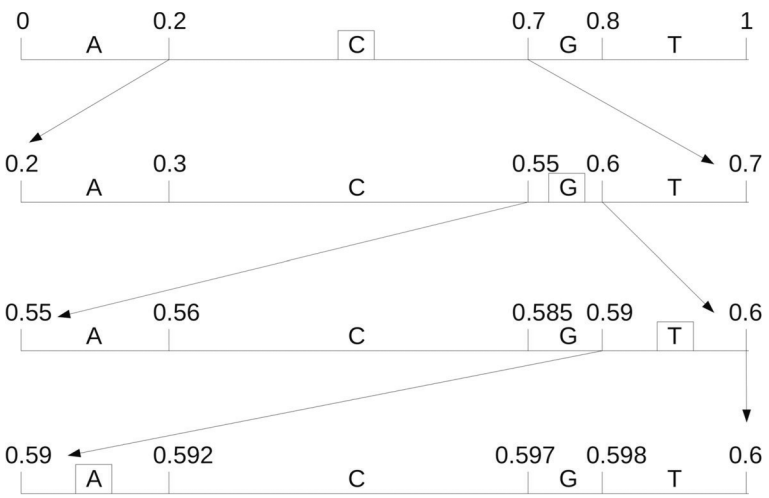

Fig. 2 Arithmetic encoding process. It illustrates the sequence label determination process when encoding a sequence 'CGTA', assume that the probability values of each base: $p(A) = p(T) = 0.2, p(C) = 0.5, p(G) = 0.1$

is the arithmetic coded description of the raw sequence 'CGTA'.

The decoding process is similar to encoding. First, base 'C' is decoded according to 0.591 within the interval (0.2,0.7), while the next decoding process 0.591 within the interval (0.55,0.7), the base 'G' is decoded, and so on $\cdots$, until decoded the last base 'A', and the whole sequence 'CGTA' is decoded.

Consider an input sequence $x_{i-1}, x_{i-2}, \ldots, x_{i-k+1}$ for the compression model, the output for the model is the predict the next base x_i. The estimate probabilities $p(x_i|(x_{i-1}, x_{i-2}, \ldots, x_{i-k+1}), h_{i-1})$ is provided into arithmetic encoder to obtain the final output bit-streams, where h_{i-1} is the deep learning model former state. According to the theory of Shannon entropy [19], the number of output bits for the base x_i is determined by:

$$H = -log_2(p(x_i|(x_{i-1}, x_{i-2}, \ldots, x_{i-k+1}), h_{i-1}) \quad (11)$$

That is, the more accurate the probability of our model estimation, the higher the probability of corresponding coding base, the smaller the output bit-tream, and the better the compression effect we get.

Model setting & training

In the deep learning model, we make comprehensive use of the local feature capture ability of CNN and the long-term feature extraction ability of LSTM. Our deep learning model implementation uses the Keras [20] library, which is an open resource for deep learning derived on the backend of Theano [21].

Comprehensive setting for the model structure are described as follows:

- Input layer (Input nucleotides: 64 ×4)
- Convolutional Neural Network (CNN) layer (Filters no.: 1024, window size: 24 ×4, stride: 1.)
- Max-Pooling layer (Window size: 3 ×4, stride: 1.)
- Dropout layer (Probability: 0.1)
- Long Short-Term Memory networks (LSTM) layer (LSTM units: 256)
- Dropout layer (Probability: 20%)
- Fully connected layer (Units: 1024)
- Sigmoid output layer (Units: 4)

In the deep learning model, all parameters are initialized according to random and uniform distribution $unif(-0.05, 0.05)$, and entire biases are initialized to 0. We use mini-batch training (default size: 64) to minimize the cross entropy loss function on the training data set. Validation losses were assessed at the end of each training epoch to monitor the convergence. We utilized about 10 epochs to complete the training, each of which took about ~6 h.

The loss function of cross-entropy is defined as:

$$L(y, \widehat{y}) = -\frac{1}{n}\sum_{t=1}^{n}\sum_{i=1}^{4} y_i^{(t)} log(\widehat{y}_i^{(t)}) \quad (12)$$

Where $\widehat{y}_i^{(t)}$ is the probability of prediction character at time t being nucleotide i, $y_i^{(t)}$ is the one-hot vector represent the real nucleotide at time t, and mini-batches sample size is n. We exploited the adaptive learning rate RMSprop [22] designed by Geoff Hinton as the learning rate of the model.

Both in the encoding and decoding process, it calculated the nucleotide probability based on the same deep learning network parameters, so they get the same prediction probability value, therefore, the original sequence can be lossless reconstructed by arithmetic coding.

Results

Dataset

In order to validate the effectiveness of our proposed model, 1,000 complete human mitochondrial genome sequences were used as experimental data, and the average sequence length of human mitochondrial genome sequence is 16,500bp. All data were download from the MITOMAP [23] database in the March 2019. We randomly selected 700 sequences being the training data set, 200 sequences being the verification data set, and 100 sequences being the test data set. In order to make the sequence consists of only 4 nucleotides (A, C, G, T), for simplicity, the fuzzy symbols was replaced with a fixed base "A" in the training process, and all lowercase nucleotides in the data set were converted to uppercase. In the compression process, we record the nucleotides not in the $\{A, C, G, T\}$ and its position information, record whether the base is in lowercase or not, so that the original sequence can be reconstructed in lossless when decompressing.

Results and model analysis

We experimented DeepDNA method for training set, verification set and test set respectively. The training set was utilized for learning model parameters, the verification set was utilized to determine network structure and model parameters, and the test set was utilized to verify the performance of the final selection of model parameters.

As we can be seen from Fig. 3, with the increase of the number of training mini-batches, the loss function gradually decreases and ultimately tends to converge. There was a trivial fluctuation point at the final of training, owing to a few genomics structural variation sites, which will affected the prediction performance. However, these structural changes accounted for less than 1% of the total sequence, that is, a few differences in the data sites of human mitochondrial genome did not have much impact on the final compression results.

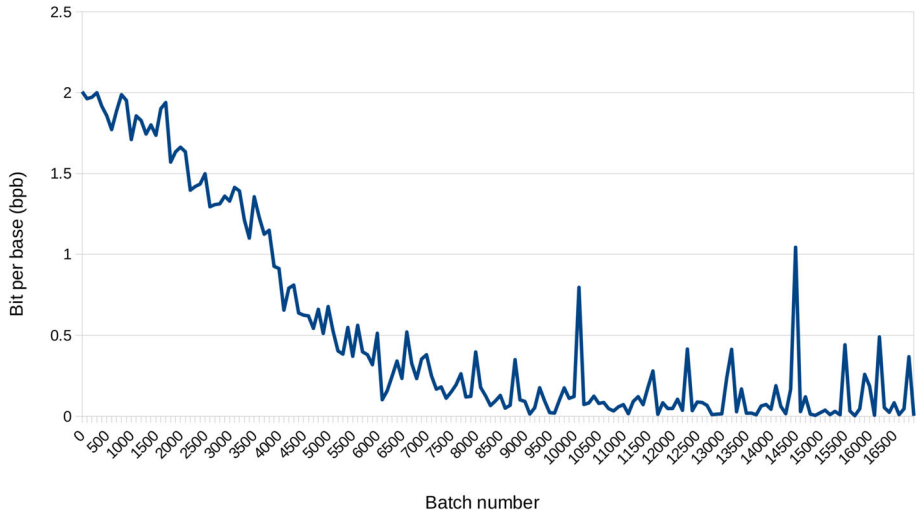

Fig. 3 The training loss function values (bpb) as the number of training mini-batches for DeepDNA model. 700 human mitochondrial genome sequences were trained, and the input length of the base sequence was 64, and the output was the classification of the corresponding four nucleotides

In order to verify the validity of our proposed DeepDNA model, we tested our method and the other four methods on the test set (100 human mitochondrial genome sequence data). Table 1 lists the compression results of the DeepDNA method, the Gzip [24] method, MFCompress [9] method, DMcompress [25] method, which we proposed earlier, the unit of compression results is in bits per base (bpb). The compression result of DeepDNA corresponds to the average of the lengths of all base prediction probability output codes in the genome sequence, namely:

$$DeepDNA(bpb) = -\frac{1}{T}\sum_{i=1}^{T} log_2(P(\widehat{y_i})) \qquad (13)$$

Where T is the sequence length of compression genome, and $P(\widehat{y_i})$ is the predicted probability value of the DeepDNA model output corresponding to the base y_i of the genome sequence at i-th position, which can be fed into arithmetic coding [26] directly and got the compression bit-streams file.

Table 1 shown that on the human mitochondrial genome test data set, the compression result of the normal text compression method Gzip is 1.45 bpb, and the DMcompress method proposed in our previous work and MFCompress method both are 0.07 bpb. The deep learning-based compression method DeepDNA proposed by us, has a compression result of 0.03 bpb. Our method DeepDNA has a better result to the other three methods in the human mitochondrial genome dataset.

Table 1 lists the results of compression of all 100 human mitochondrial genomes as a group data-set. To verify their efficiency of compressing on individual genome sequence, we randomly selected 5 human mitochondrial genomes. The data is independently compressed and compared the results. The Table 2 lists the compression results of the 5 human mitochondrial genome sequences. It can be seen from the table that our proposed DeepDNA method achieves the best compression results on all five independent genomes. Comparing with the current normal text compression method Gzip, the finite context model based compression method MFCompress, and our earlier information entropy-based compression method DMcompress, the result of compression of DeepDNA is less than 0.05 bpb.

Discussion

The Tables 1 and 2 show that our proposed method can not only achieve compression effect on the multi-genomes, but also achieve good compression effect on individual genome data. The other three compression

Table 1 Results for DeepDNA and the other methods compression for 100 human mitochondrial genomes

Dataset	Total size (nucleotides)	Gzip (bpb)	MFCompress (bpb)	DMcompress (bpb)	DeepDNA (bpb)
100 human Mitochondrial genomes	1,656,779	1.45	0.07	0.07	0.03

The measure of space occupied is evaluated in bits per base (bpb)

Table 2 Detailed results for DeepDNA and the other methods on randomly selected five sequences from 100 human mitochondrial genome sequences

Genome ID	Gzip (bpb)	MFCompress (bpb)	DMcompress (bpb)	DeepDNA (bpb)
KF162105.1	2.63	2.09	2.07	0.01
MF058266.1	2.64	2.09	2.07	0.05
KC911416.1	2.64	2.09	2.06	0.01
AY339411.1	2.63	2.09	2.07	0.01
JQ702777.1	2.64	2.08	2.06	0.04

The measure of space occupied is evaluated in bits per base (bpb)

methods have better compression on the multi-genomes than on the individual genome, because the data redundancy on the multi-genomes is higher than the individual genome. The neural network model we proposed for lossless genome compression is not affected by this limitation.

The neural network parameters, as part of the compression model, obtained through training and learning directly participate in the compression decompression process. Thus it avoided the process of continuing to update parameters during compression/decompression, saving a lot of time. Because of the consistency between genomes, we can train the compression model in advance, and then used directly for compression.

Conclusions

We designed a novel, machine learning method, DeepDNA, which integrates the convolutional neural network (CNN) and the long short-term memory network (LSTM) for compressing the genome sequences. Experiment on 1,000 complete human mitochondrial genome sequences have shown that our method can learn local features of sequences through convolution layer, and can learn advanced representations of long-term dependence of sequences through long short-term memory network (LSTM). We evaluated the performance of deep learning model on 100 human mitochondrial genome sequences compression task and obtained an acceptable result.

Our model indicated the feasibility of compressing genome sequences via CNN and LSTM network models. This work will help to better explore the patterns and rules in genome sequences, assist in decoding the functional characteristics of sequences, and to help resolve the link between genes and disease. Because better sequence prediction model will be achieved better compression effect, and a better sequence prediction model can help solve all above problems.

In addition, with the exploration of genome sequence characteristics in future biological analysis, the compression method can obtain more redundant information, and get a better improved compression performance. In the next work, we can explore methods of lossless compression of the entire human genome, making full use of as much background information as possible, for instance, mutations, tandem repeats, motifs, etc., to train the machine learning model for compression genome sequences.

Acknowledgements
Not applicable.

About this supplement
This article has been published as part of *Human Genomics Volume 13 Supplement 1, 2019: Selected articles from the IEEE BIBM International Conference on Bioinformatics & Biomedicine (BIBM) 2018: human genomics*. The full contents of the supplement are available online at https://humgenomics.biomedcentral.com/articles/supplements/volume-13-supplement-1.

Authors' contributions
RW designed the experiment and was a major contributor in writing the manuscript. TZ performed the reliability of the experiment. All authors read and approved the final manuscript.

Funding
Publication costs were funded by the National Key Research and Development Program of China [Grant No.: 2016YFC0901605, 2016YFC1201702-01], the National High-tech R&D Program of China [Grant No.: 2015AA020108, 2012AA02A604].

Availability of data and materials
The dataset generated and analysed during the current study are available in the Mitomap repository, https://www.mitomap.org/MITOMAP

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Peng Cheng Laboratory, ShenZhen, China. [2]School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China.

Published: 22 October 2019

References

1. Collins FS, Morgan M, Patrinos A. The human genome project: Lessons from large-scale biology. Science. 2003;300(5617):286–90.
2. Schaller RR. Moore's law: past, present and future. IEEE Spectr. 1997;34(6):52–9.
3. Pearson WR, Lipman DJ. Improved tools for biological sequence comparison. Proc Natl Acad Sci. 1988;85(8):2444–8.
4. Grumbach S, Tahi F. Compression of dna sequences. In: Data Compression Conference; 1993. p. 340–50.
5. Grumbach S, Tahi F. Inf Process Manag. 1994;30(6):875–86.
6. Chen X, Kwong S, Li M. A compression algorithm for dna sequences and its applications in genome comparison. Genome Inform Work Genome Inform. 2000;10(4):51.
7. Li P, Wang S, Kim J, Xiong H, Ohnomachado L, Jiang X. Plos ONE. 2013;8(11):80377.
8. Kaipa KK, Bopardikar AS, Abhilash S, Venkataraman P. Algorithm for dna sequence compression based on prediction of mismatch bases and repeat location. In: IEEE International Conference on Bioinformatics and Biomedicine Workshops; 2010. p. 851–2.
9. Pinho AJ, Pratas D. Mfcompress: a compression tool for fasta and multi-fasta data. Bioinformatics. 2013;30(1):117–8.
10. Goyal M, Tatwawadi K, Chandak S, Ochoa I. Deepzip: Lossless data compression using recurrent neural networks. arXiv preprint. 2018. arXiv:1811.08162.

11. Tomkins J. Genome-wide dna alignment similarity (identity) for 40,000 chimpanzee dna sequences queried against the human genome is 86-89%. Answers Res J. 2011;4(2011):233–41.
12. D'hont A, Denoeud F, Aury J-M, Baurens F-C, Carreel F, Garsmeur O, Noel B, Bocs S, Droc G, Rouard M, et al. The banana (musa acuminata) genome and the evolution of monocotyledonous plants. Nature. 2012;488(7410):213.
13. Zhou C, Sun C, Liu Z, Lau F. A c-lstm neural network for text classification. arXiv preprint. 2015. arXiv:1511.08630.
14. Xu K, Ba J, Kiros R, Cho K, Courville A, Salakhudinov R, Zemel R, Bengio Y. Show, attend and tell: Neural image caption generation with visual attention. In: International Conference on Machine Learning; 2015. p. 2048–57.
15. Sainath TN, Vinyals O, Senior A, Sak H. Convolutional, long short-term memory, fully connected deep neural networks. In: 2015 IEEE International Conference On Acoustics, Speech and Signal Processing (ICASSP). IEEE; 2015. p. 4580–4.
16. Nair V, Hinton GE. Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10). IEEE; 2010. p. 807–14.
17. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80.
18. Salomon D, Motta G. Handbook of Data Compression: Springer; 2010.
19. Shannon CE. A mathematical theory of communication. Bell Syst Tech J. 1948;27(3):379–423.
20. Chollet F, et al. Keras, GitHub. GitHub repository. 2015. https://github.com/fchollet/keras.
21. Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, Turian J, Warde-Farley D, Bengio Y. Theano: A CPU and GPU math compiler in Python. In: Proc. 9th Python in Science Conf. vol. 1. IEEE; 2010. p. 3–10.
22. Tieleman T, Hinton G. Lecture 6.5-rmsprop, coursera: Neural networks for machine learning: University of Toronto, Technical Report; 2012.
23. Mitomap. A human mitochondrial genome database. 2018. http://www.mitomap.org.
24. Deutsch P, Gailly J-L. Zlib compressed data format specification version 3.3. Tech Rep. 1996.
25. Wang R, Teng M, Bai Y, Zang T, Wang Y. Dmcompress: Dynamic markov models for bacterial genome compression. In: 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). IEEE; 2016. p. 776–9.
26. Witten IH, Neal RM, Cleary JG. Arithmetic coding for data compression. Commun ACM. 1987;30(6):520–40.

Publisher's Note