A Decoding Method For Modulo Operations-Based Fountain Codes Using the Accelerated Hopfield Neural Network

Zaihui Deng[1, a], Xiaojun Tong[1, b] and Liangcai Gan[2, c]

[1]School of Mathematics and computer science, Wuhan Textile University, Wuhan 430073, China;

[2]Electronic Information School, Wuhan University, Wuhan 430072, China.

[a]zaihuideng@126.com, [b]tongxiaojun@wtu.edu.cn, [c]lc_gan@hotmail.com

Abstract. This paper describes a decoding method using the accelerated Hopfield neural network, in order to address the high complexity of decoding for modulo operations-based fountain codes. The method constructs a neural network model based on a non-linear differential equation, and runs the model after setting an initial value. During the process, the model's output value first rapidly decreases under the effect of the accelerator resistor, slows down near an equilibrium point, and finally regresses to a unique equilibrium point with an arbitrarily small error. The result is half-adjusted to obtain the source data sequence. Simulated tests indicate the method to be valid, and can potentially bring the modulo fountain codes closer to practical application.

Keywords: Fountain codes; Modulo operation; Decoding; Chinese remainder theorem; Neural network.

1. Introduction

The concept of fountain codes was first proposed by Byers, Luby et al. in 1998[1]. In 2002, Luby built on his earlier work and developed the first practical implementations of fountain code – Luby transform (LT) codes[2], which were followed up by implementations from other researchers, including online codes[3], raptor codes[4], turbo fountains[5], and so on[6,7,8,9]. These earlier fountain codes all use XOR operations for their encoding processes. Ref [10] proposed Chinese transform (CT) codes, a new class of fountain code based on the Chinese remainder theorem, in order to address the low packet efficiency of LT-based encoding. Ref [11] suggested two variants specialized for transmission and computation efficiency. The CT codes all generate packets by performing modulo operation on the decimal numbers of the source sequence, thus will be referred to as "modulo fountain codes" hereafter. The two papers' methods somewhat differ in structure, with Ref [10] using a chaotic position scrambling algorithm to generate packets.

However, when it comes to decoding, their approaches have all relied on the relatively complex Dayan seeking-unity method, which is a traditional algorithm for solving remainder problems. Improving decoding efficiency will be crucial for modulo fountain codes to become more practical. The Hopfield neural network excel at inverse operations[12], as their basic model is a first-order non-linear differential equation, which can quickly regress to a stable equilibrium point in the state space. On this basis, this paper puts forward a fast decoding method, using an improved Hopfield neural network to perform inverse operations on each received packet to restore the source data sequence. This method employs a simple algorithm that omits the need to calculate modular multiplicative inverses, effectively addressing the decoding's complexity problem. Simulated tests have indicated the method to be valid and feasible.

2. Chinese remainder theorem and Dayan seeking-unity method

2.1 Chinese remainder theorem

The Chinese remainder theorem is one of the crowning achievements of ancient Chinese mathematics. It can be summarized as [13]: let $m_1, m_2, \dots, m_k$ be positive integers that are pairwise coprime, and $a_1, a_2, \dots, a_k$ be the respective remainders of a positive integer x divided by $m_1, m_2, \dots, m_k$, then the congruence set

$$\begin{cases} x \equiv a_1 (\bmod m_1) \\ x \equiv a_2 (\bmod m_2) \\ \quad \vdots \\ x \equiv a_k (\bmod m_k) \end{cases} \tag{1}$$

Must have a unique solution, and its smallest positive integer solution is:

$$\mathrm{x} = a_1 M_1 y_1 + a_2 M_2 y_2 + \cdots + a_k M_k y_k \tag{2}$$

Where: M is the least common multiple of $m_1, m_2, ..., m_k$, i.e. $M = \prod_{i=1}^{k} m_i$, $M_i = M / m_i$, $M_i y_i = 1 (\bmod m_i)$; $i = 1, 2, ..., k$; and y_i is referred to as the modular multiplicative inverse (hereafter "modular inverse").

2.2 The Dayan seeking-unity method

From the previous section, we know that the key step in solving Equation 1 lies in determining the modular inverse y in the congruencey$G \equiv 1(modm)$; once determined, the solution can be calculated using Equation 2. Since the encoding and decoding processes of modulo fountain codes can be treated as respectively the generation and solution of congruence sets in a remainder theorem problem, the main costs of the decoding algorithm are incurred by determining the modular inversey, making it the core problem in the decoding algorithm.

The Euler function and the Dayan seeking-unity method are both traditional methods for determining modular inverses, with the latter being more practical and programmable as an algorithm. When solving a problem as complicated as $873201\mathrm{b} \equiv 1(\mathrm{mod}9829)$, the Euler function has a computational complexity of $873201^{\phi(9289)}$, which cannot be quickly accomplished even by high performance computer, while the Dayan seeking-unity method can complete the task in a significantly shorter time, making it the generally preferred method for calculating modular inverses.

Let the congruence bey$G \equiv 1(modm)$, where m and G are pairwise coprime positive integers, the Dayan solution for modular inverse y can be represented by the following recurrence relations:

$$\begin{cases} c_0 = 1 \\ c_1 = q_1 \\ c_k = q_k c_{k-1} + c_{k-2} (1 < k \le n) \end{cases} \tag{3}$$

Where: n is the total number of non-zero remainders from Euclidean divisions on m and$G$, and $q_k(1 < k \le n)$ represents the quotient of each division. Therefore the modular inverse in question is:

$$\mathrm{y} \equiv (-1)^n c_n (modm) \tag{4}$$

3. The neural network-based decoding algorithm

The complexity issue of modulo fountain codes is found in their decoding algorithms, as their encoding algorithms consist of simple modulo operations. This issue can be solved with the characteristics of Hopfield neural network [14].

3.1 Modulo operations

The encoding components of the existing modulo fountain codes divide data blocks in different ways. The computation-efficient approach focuses on minimizing block sizes, while the transmission-efficient approach adjusts block sizes with modular decomposition according to conditions of the transmission channel[11]. Ref [10] uses an approach that balances transmission and computation with chaotic position scrambling.

Regardless of methods, these encoding processes always involve dividing the source binary sequence into blocks, converting each block into a decimal number, choosing corresponding moduli to calculate its remainders, and generating transmission packets from the moduli and remainders. The packets are to be decoded using the remainder theorem at the receiving end. As shown in Fig. 1, the source block's decimal number (e.g. 125) is divided by a series of primes (e.g. 7, 11, 13…), and the resulting remainders are sent as packets. The receiving end will have a 100% chance of recovering the original value (e.g. 125) after receiving a sufficient number of the packets.

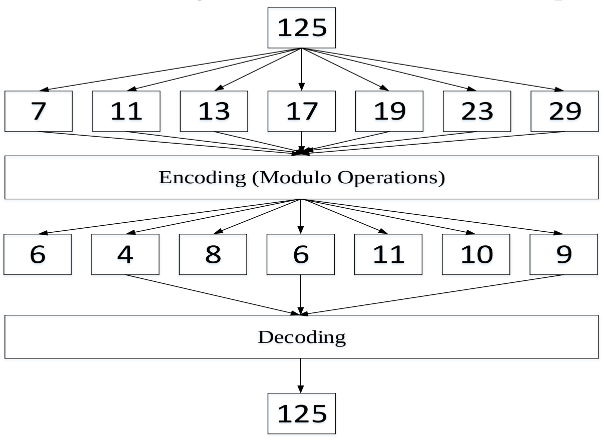


Fig. 1 Encoding and decoding for modulo fountain codes

3.2 Accelerated decoding method using the Hopfield neural network

The decoding process of modulo fountain codes involves multiple inverse operations of the remainder operations on decimal numbers.The Hopfield neural network are well-suited for inverse operations[12], as their basic models are first-order non-linear differential equations, and can rapidly regress to a stable equilibrium point in the state space. The decoding of each block can be treated as solving a non-linear differential equation, which can be represented by a Hopfield neural network. Equation 5 shows the differential equation constructed for this purpose:

$$\frac{C}{\beta}\frac{dx}{dt}=-\frac{1}{R\beta}x-U(x)\cdot\sum_{i=1}^{L}\gamma(<x>_{m_i}-a_i^{*})-(1-U(x))\cdot\frac{\gamma x}{R_d} \tag{5}$$

Where: $U(x)=\begin{cases}1\begin{cases}when U(x(t-\delta))=0,\ \ \varphi_i(x)>0 (for\ all\ i)\\ when U(x(t-\delta))=1,\ \ \varphi_i(x)>0\ or\ \varphi_i(x)<0\end{cases}\\ 0,\ \ others\end{cases}$; C, β, γ and R are positive real numbers; $<x>_{m_i}$ is the remainder of x divided by m_i; $a_i^{*}=a_i+\varDelta$; δ and $\varDelta$ are non-integers larger than 0, with $0<\delta<\Delta$ and $0<\Delta<0.5$; $x(0)=M-1$; $U(x(0))=0$; $\varphi_i(x)=\gamma(<x>_{m_i}-a_i^{*})$; $i=1,2,...,L$; and t is the time variable.

Equation 5 represents a model where x is first assigned an initial value, and then as the initial value of $U(x)$ is 0, $x(t)$ begins a monotonic decrease over time. After x enters the region of regression near the solution, the value of $U(x)$ will stay constant at 1; due to its feedback information, x will tend towards equilibrium, and the optimal solution x_0 can be obtained. While there is some error between the optimal solution and the accurate solution, the error can be kept in the -0.5~0.5 range by adjusting the values of γ , β, and R. The result can then be half-adjusted to obtain the accurate solution x^*, which is the decimal number from the source data block.

The proof of the principle above can be found in Ref [12]. As long as we choose large enough values for γ , β, and R, the approximate solution x_0 can be half-adjusted to obtain the accurate solution x^*. This completes the decoding process. The error ε^* can be represented by Equation 6.

$$\varepsilon *=-\frac{1}{\gamma\beta RL+1}x^{*}+\frac{1}{1+\frac{1}{\gamma\beta R}}\varDelta \tag{6}$$

Equation 5 differs from the differential equation provided by Ref [12] by having an additional third term on the right-hand side. When implemented as Fig. 2's circuit schematic, the first term on the right-hand side is the effect of x on the discharging of resistor R; the second term is the effect of feedback information; and the third term is the acceleration effect. With properly tuned values for γ , β, and R, the third term should be able to accelerate the decoding process. Resistor R_d is the discharging acceleration resistor; with the third term added, the discharging of R_d can accelerate the monotonic decrease of$x(t)$, without influencing the error between the accurate solution and the stable solution, i.e. it has no effect on the result of decoding. The majority of the system's running time is taken up by the discharging process, hence shortening the discharging will also reduce the total time, and achieve the acceleration of the decoding.

There are the following relationships between Equation (5) and the circuit schematic shown in Fig. 2: R is the network resistor; R_d is the acceleration resistor; C is the capacitor; γ is the weighted resistor; β is the amplification factor; a_i^* is the input current; x is the output voltage; $\varphi_i(x)$ is the input voltage of L constraining amplifiers; $< \mathrm{x} >_{m_i}$ can be implemented with asymmetric Hopfield neural network; the constraint $U(x)$ can be implemented with simple asynchronous sequential circuits; the operational amplifier's input-output relationship is $f(v) = v$.

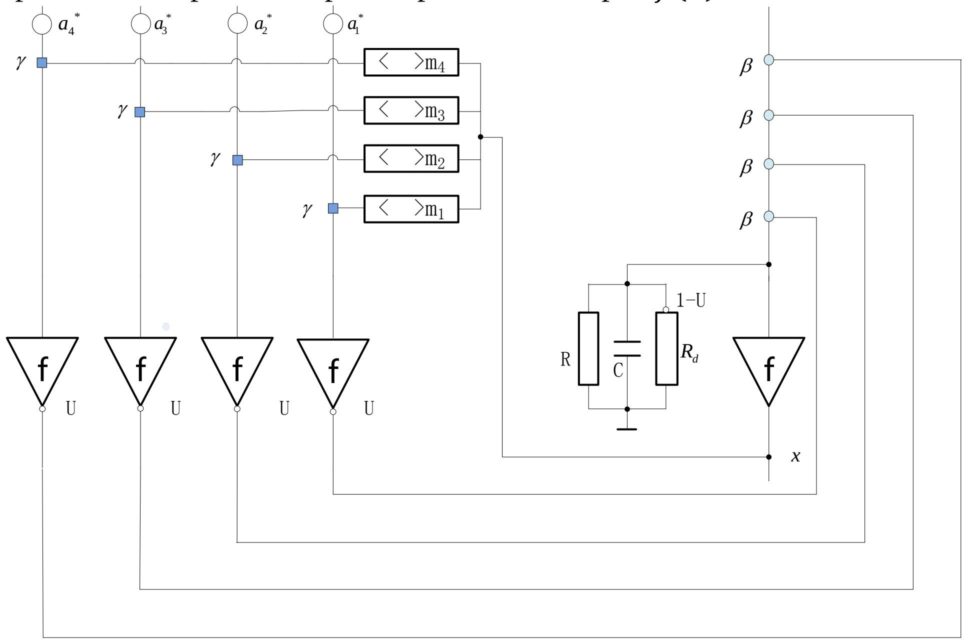


Fig. 2 Circuit diagram of the accelerated Hopfield neural network decoder

Based on the previous sections, the full decoding process consists of the following:

(1) The parameters γ , β, R and Δ are set to values based on the length of the source data block, so that the equilibrium point of Equation 5 has an optimal error when compared to the source data.

(2) The initial value is set to be the product of the received moduli $m_1, m_2, \dots, m_k$ minus 1, i.e. $x(0) = m_1 * m_2 * \dots * m_k - 1$, $U\big(x(0)\big) = 0$. The neural network described by Equation 5 is activated and reaches an equilibrium point, outputting the optimal solution to the equation.

(3) Lastly, the optimal solution is half-adjusted to obtain an accurate solution, completing the decoding process.

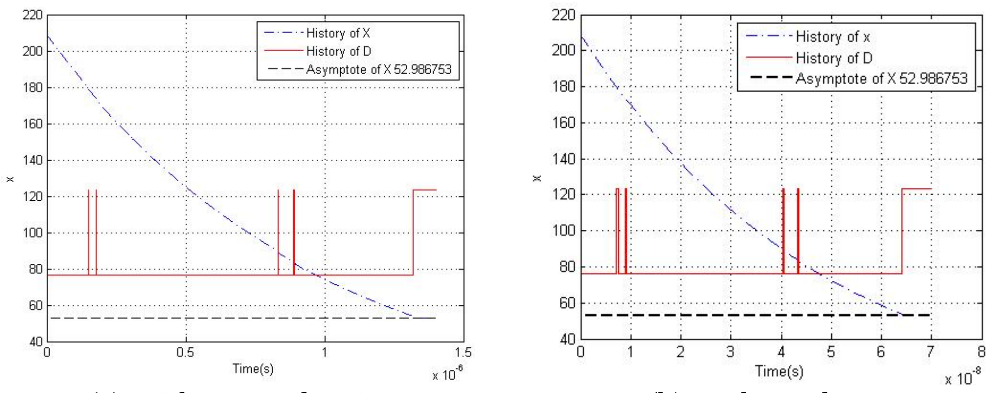
During the decoding, the neural network restores the source data block sequence from the moduli and remainders for each block, and as long as the value of γ, β and R can satisfy the maximum possible value of each block, the errors of all blocks will be kept sufficiently small.

4. Simulated tests and analysis

The neural network decoding structure of modulo fountain codes can be realized using electronic circuit, thus can be used in high-speed and real-time processing environment. However, due to the limitation of hardware, the binary number of single group cannot be too long. In order to compare different decoding method of the binary number of single group, we select the decimal number 53 and 179, using fourth-order Runge-Kutta method to perform the simulation. In the experiment, we calculate the average of the result after 1000 simulations.

The simulated tests were run on a computer with a Intel i5-4210 2.6GHz CPU and 8G RAM, using the Matlab R2014a software. The first test was to verify the performance of accelerated Hopfield network decoding, and the second test was to compare the traditional method and the Hopfield neural network method.

Test 1: Let the source data block be the decimal number 53, and the parameters be $m_1 = 2,\ m_2 = 3,\ m_3 = 5,\ m_4 = 7,\ R = 1\text{k}\Omega,\ C = 1000\text{pf},\ R_d = 50\Omega.\ \gamma = 1,\ \ \beta = 1,$ and $\Delta = 0.1$. The Hopfield neural network with and without acceleration are tested, with results shown in Fig. 3.



(a) Without acceleration (b) With acceleration

Fig. 3 Performance comparison: the Hopfield neural network with and without acceleration

As shown in Fig. 3, using both networks, the value of the output voltage X started from the initial value and regressed towards 52.986573, which is half-adjusted to the accurate value 53. That is to say, the result of decoding is 53. The value of D only stayed at the constant 1 when the curve was within X's region of attraction. However, the computation time of the network has been significantly shortened with the addition of the acceleration resistor. With all parameters being the same, the network without the acceleration resistor regressed in $1.320 \times 10^{-6}s$, and the one with the resistor took$6.408 \times 10^{-8}s$, which is faster by an order of magnitude.

Test 2: Let the source data block be the decimal number 179, and the parameters be$m_1 = 5,\ m_2 = 7,\ m_3 = 11,\ m_4 = 13,\ R = 120\text{k}\Omega,\ C = 1000\text{pf},\ R_d = 0.24\Omega.\ \gamma = 1,\ \ \beta = 1,$ and $\Delta = 0.1$. The Danyan seeking-unity method, the Hopfield neural network and the acceleration Hopfield neural network are tested respectively, with results shown in Table1.

Table 1 Comparison of decoding performance

The Danyan seeking-unity method	The Hopfield neural network	The accelerated Hopfield neural network
2.153×10^{-3}s	3.920×10^{-4}s	4.108×10^{-9}s

As can be seen from Table 1, when using the Dayan seeking-unity method, the decoding time is $2.153 \times 10^{-3}s$; when using the neural network method, the decoding time is $3.920 \times 10^{-4}s$; and when using the accelerated neural network method, the decoding time is $4.108 \times 10^{-9}s$. The decoding speed using the accelerated Hopfield neural network is about 9.54×10^4 faster than that of the original neural network method.

During the early stage of decoding, the decrease of the system's output voltage can be described by$x(t) = x_0\exp[-t/(RC)]$, which is the equivalent of directly discharging through resistors. The acceleration resistor allows the system to rapidly reach the region of attraction near the source decimal, and fall into a local minimum in the region. The majority of the running time is spent in the early stage, which can be remarkably shortened by the addition of an acceleration resistor. Compared with the Dayan seeking-unity method, the new decoding method is much quicker, thus it increases the data processing rate. The simulation results are consistent with the theoretical analysis, as the improved method can significantly accelerate the decoding of modulo fountain codes.

5. Conclusion

This paper presents a decoding method based on the accelerated Hopfield neural network. Modulo fountain codes have several unique advantages over LT codes, including lower redundancy and greater flexibility in structures, and their disadvantage lies in the high complexity of the decoding process. The Hopfield neural network can be used to significantly increase the efficiency of decoding for modulo fountain codes, and bring them closer to practical application. The feasibility and validity of the proposed method were tested through comparison with the traditional decoding method using simulated tests.

Acknowledgements

This work was supported by the Natural Science Foundation of Hubei Province of China under Grant 2015CFB652.

References

[1]. J. Byers, M. Luby, M. Mitzenmaeher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. ACM SIGCOMM, 98, Vancouver, Aug.1998, pp.56-67.
[2]. M. Luby LT codes. The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Vancouver: 2002: 271–280.
[3]. P. Maymounkov. Online codes. NYU, Tech. Rep. TR2002-833, 2002.
[4]. A. Shokrollahi Raptor codes. IEEE Transactions on Information Theory. 2006, 52(6), pp. 2551-2567.
[5]. H. Jenkac, J. Hagenauer, T. Mayer. The turbo-fountain. European Transactions on Telecommunications. 2006, 17(3), pp. 337-349.
[6]. J. H. Sorensen, P. Popovski, J. Ostergaard. Design and Analysis of LT codes with decreasing ripple size. IEEE Tran action on Communications, 2012, 60(11), pp.3191-3197.
[7]. Yen Kuo-Kuang, Liao Yen-Chin, Chen Chih-Lung, Chang Hsie-Chia. Modified robust soliton distribution (MRSD) with improved ripple size for LT codes. IEEE Communications Letters, 2013, 17(5), pp.976-979.
[8]. Megasthenis Asteris, Alexandros G. Dimakis. Repairable fountain codes. IEEE Journal on Selected Areas in Communications, 2014, 32(5), pp.1037-1047.
[9]. Francisco Lazaro, Enrico Paolini, Gianluigi Liva, Gerhard Bauch. Distance spectrum of fixed-rate raptor codes with linear random precoders. IEEE Journal on Selected Areas in Communications, 2016, 34(2), pp.422-436.
[10]. Cheng Huang, Ben-Shun Yi, Liang-Cai Gan, et al. Fountain codes based on modulo and chaos[J].Journal of Beijing University of Posts and Telecommunications. 2010, 33(3), pp.1221-125.
[11]. Sun Hung-Min, Chang Shih-Ying, Hung Yu-Hsiang, et al. Decomposable forward error correction codes based on Chinese remainder theorem[C]. 10th International symposium on pervasive systems, Algorithms, and networks. 2009, pp.260-265.
[12]. Hong Sun, Tian-ren Yao. Mathematical principle of residue-to-decimal conversion by neural network, ACTA Electronic Sinica. 1995, 23(4), pp.48-52.
[13]. D. W. Tank, J. J. Hopfield. Simple “neural” optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit. IEEE Transactions on Circuits and Systems.1986, 33(5), pp.533-541.
[14]. O. Goldreich, D. Ron, M. Sudan. Chinese remaindering with errors[J]. IEEE Transactions on Information Theory, 2000, 46(4), pp.1330-1338.