Part 2: Convolutional Codes

Jinsong Han, Aaron Andrews, Patrick Tam

MATH 2605: Calculus III for Computer Science
Georgia Institute of Technology

March 31, 2015

A convolutional code is a way to encode a binary stream being fed into a shift-register circuit into a multiple output output streams determined by a linear combination of various elements in indexes relative to the current index being processed. The main objective of this section was to properly encode and decode convolutional codes through the use of a transformation matrix and two iterative techniques, Jacobi and Gauss-Seidel. The results determined that Gauss-Seidel is the more efficient at decoding convolutional codes, as the number of iterations necessary to decode remained at a constant value of two, whereas Jacobi required more iterations as the length of the stream to be decoded increased.

Introduction

A convolutional code generally consists of a binary data stream, $\overline{x}$, being fed into a shift-register circuit, where the values of memory elements stored within the binary stream $\overline{x}$ are removed and then added to various output streams based on specific patterns having to do with linear combinations of various indexes relative to the current index being processed of the original

input stream, $\overline{x}$. The various output streams can then be concatenated in a specific way or form to form $\overline{y}$, the convolutional code word. In a simpler sense, this can be seen as an encoding process. A certain binary stream is being manipulated into different output streams which take in values from the original stream through a set of specific patterns. The end result can be decoded to return the original binary stream if the same formula used to encode the stream is utilized.[1]

In this case, the memory elements from $\overline{x}$ are being shifted and encoded into two output streams, $\overline{y}^0 = (y_0^0, y_1^0, y_2^0, ...)$ and $\overline{y}^1 = (y_0^1, y_1^1, y_2^1, ...)$, based on two specific patterns. The two output streams can be concatenated to create $\overline{y} = (y_0^0, y_0^1, y_1^0, y_1^1, y_2^0, y_2^1, ...)$, a single coded data stream. This is known as the convolutional code word.

Each element in the output stream $\overline{y}$ is the result of a linear combination of the elements in the original input stream $\overline{x}$. For this specific example, for an arbitrary element index, j of $\overline{x}$, y_j^0 and y_j^1 can be represented as

$$y_j^0 = x_j + x_{j-2} + x_{j-3} \tag{1}$$

$$y_j^1 = x_j + x_{j-1} + x_{j-3} \tag{2}$$

For example, when the $\overline{x}$ stream is $(0, 1, 0, 0, 0)$, the following coded output sequences are obtained. Please note that the $\overline{x}$ stream can be interpreted as $(0, 1, 0, 0, 0, 0, 0, 0)$.

$$\overline{y}^0 = (0, 1, 0, 1, 1, 0, 0, 0) \tag{3}$$

$$\overline{y}^1 = (0, 1, 1, 0, 1, 0, 0, 0) \tag{4}$$

The convolutional code word would be the combination of the two generated output streams, and would be the following.

$$\overline{y} = (00, 11, 01, 10, 11, 00, 00, 00) \tag{5}$$

The objective of this part of the project is to be able to encode any input stream $\overline{x}$ to become an encoded $\overline{y}$ - the convolutional code. It is also necessary that we can supply any proper $\overline{y}$ word and decode it to obtain the original $\overline{x}$ stream. This encoding and decoding process can be seen as an equation in linear algebra, $A\overline{x} = \overline{b}$, where A is the transformation matrix, $\overline{x}$ is the original binary stream, and $\overline{b}$ is the convolutional word $\overline{y}$.

Approach

In order to encode the binary stream $\overline{x}$ to become the word $\overline{y}$, the streams were treated as matrices with the dimensions n x 1, and were multiplied by a transformation matrix, $A^0 or A^1$, to produce the output stream $\overline{y}^0$ or $\overline{y}^1$, respectively. In order to account for the fact that the patterns can go past the element with the highest index, three zeroes are appended to the end of the x stream, making its dimensions $n + 3$ x 1. The transformation matrices are square $n + 3$ x $n + 3$ matrices that utilize the equations that convert the stream $\overline{x}$ into a result of a linear combination of elements based on a specific pattern. The A transformation matrix would only vary based on the size of the input stream. Thus, when the transformation matrix is multiplied by $\overline{x}$, the elements in the target indexes will be added up. The resulting equation can be written as $A^{0,1}\overline{x} = \overline{y}^{0,1}$.

In order to account for overflow (i.e. all three target indexes have the value "1", making the output stream contain a value of "3", which is neither "0" nor "1"), the results are all modded by two, thus ensuring that all values are either 0 or 1. This will result in two binary streams, $\overline{y}^0$ and $\overline{y}^1$, which can then be concatenated properly to create the convolutional word $\overline{y}$.

In order to decode the binary stream $\overline{y}$, a similar technique to the encoding process, but in reverse, must be done. The first step is to ensure that the binary stream entered is a valid convolutional code - meaning that its length is even, and that the stream is able to be generated by some original binary stream $\overline{x}$. If so, the convolutional code $\overline{y}$ will first be separated into

the two original output streams, y^0 and y^1. Afterwards, using the equation $A^{0,1}x = y^{0,1}$, the original binary stream $\overline{x}$ will be produced. The same stream, $\overline{x}$, should be produced regardless of whether $A^{0,1}\overline{x} = \overline{y}^{0,1}$ were used.

In order to solve for the equation, $A^{0,1}\overline{x} = \overline{y}^{0,1}$, the iterative techniques Jacobi and Gauss-Seidel were deployed. Both were used to decode the stream as an attempt to determine which iterative technique would be more efficient to decode the linear combination code into the original binary stream sequence, $\overline{x}$. The specified error tolerance for these iterations was 10^{-8}, where the error tolerance was determined by the formula $||\overline{x}_n - \overline{x}_{n-1}||$, where n stands for the current iteration. An initial guess, $\overline{x}_0$, was supplied to be the zero vector. If the iteration did not converge (go under the error tolerance) within a thousand iterations, the iteration would terminate.[2]

Discussion

The most important and apparent point for discussion would be the comparison of the decoding process based on which iterative technique was implemented: Jacobi versus Gauss-Seidel. Both iterative techniques are used to solve for the vector $\overline{x}$ in the equation $A\overline{x} = \overline{b}$

As seen through various tests with the written Jacobi and Gauss-Seidel functions unrelated to binary streams and this project, it appears as if Gauss-Seidel will generally converge, if a convergance is possible, to a solution vector $\overline{x}$ in half as many iterations as Jacobi, meaning that Gauss-Seidel seems to run faster and more efficiently than Jacobi.

For this specific project, Jacobi takes more iterations to complete decode the stream when the length of the stream increases. The general trend appears to be linear, however it largely depends on the structure of the stream, and how the zeroes and ones are ordered. But as a general trend, the longer the convolutional code $\overline{y}$ is, the more iterations it takes for Jacobi to calculate the original binary stream.

For Gauss-Seidel, something more unusual occurred. The number of iterations that it takes for the Gauss-Seidel to calculate $\overline{x}$ is always two, regardless of the size of the stream. Gauss-Seidel always manages to decode the stream with a error tolerance of 10^{-8} in only two iterations.

In order to understand what exactly was going on here, the specific steps of Gauss-Seidel were examined throughout the decoding process.

For both Jacobi and Gauss-Seidel, the iterative equation for determining the solution vector can be represented as follows:

$$S\overline{x}_{k+1} = T\overline{x}_k + \overline{b} \tag{6}$$

Where in this case, $\overline{b}$ is equal to the convolutional word, $\overline{y}$. S and T are sections of the transformation matrix, A, where $S - T$ will equal the original matrix, A. For both Jacobi and Gauss-Seidel, the matrix S will be lower triangular, which means the expression can be simplified to $A\overline{x}_{k+1} = \overline{b}$, where A is equal to S, meaning that it is lower triangular, and $\overline{b}$ is equal to $T\overline{x}_k + \overline{y}$. Since the matrix S is lower triangular, $\overline{x}_{k+1}$ can be solved for through forward substitution rather than having to multiply the inverse to the other side of the equation.

The special thing to note with Gauss-Seidel is how the matrices S and T are determined. For Gauss-Seidel, S is equal to the lower half of the original matrix plus the diagonal section, while T is equal to the negative of all the values in the upper half of the original matrix.

Recall that the transformation matrix which changes $\overline{x}$ into $\overline{y}^{0,1}$ is a square matrix, A^0 or A^1. The formulas to determine the equations $\overline{y}^{0,1}$ are linear combinations of various elements in $\overline{x}$ depending on the current index of the element being processed. All elements to be summed up, however, will only come on or before the current element being processed. In transformation matrix terms, this means that no elements above the diagonal will contain a data entry aside from 0.

This means that the T matrix for Gauss-Seidel is equal to a matrix composed of zeroes. When applied to the equation for Jacobi and Gauss-Seidel, the equation can be simplified to the

following:

$$S\overline{x}_{k+1} = \overline{b} \tag{7}$$

In other words, the result of the Gauss-Seidel iteration will not depend on the previously determined $\overline{x}$ vector from the previous iteration, nor will it depend on the initial guess vector, $\overline{x}_0$ supplied to it. The iteration will be immediately determined after one run, as all parts of the equation are constants in this case and will not change. Through simple forwards substitution ($A\overline{x} = \overline{b}$) using the lower triangular matrix S as A and the respective $\overline{y}^{0,1}$ stream for b, the original matrix stream can be determined within one run. However, since the calculation of the error depends on a previous run, if the initial guess was not equivalent to the original stream $\overline{x}$, the iteration will run one more time in order to confirm that the error tolerance has dropped below 10^{-8}.

The same thing does not occur to the Jacobi iteration since neither S nor T matrix for Jacobi is equivalent to the zero matrix. The S matrix for Jacobi is equivalent to all the values on the diagonal, while T is equal to the negative of the remaining values. Since all the values in the original matrix are either on or below the diagonal, it cannot be guaranteed that all values of S nor T will be zero. This means that Jacobi remains as an iterative technique as $\overline{x}_k$ in the equation $S\overline{x}_{k+1} = T\overline{x}_k + \overline{b}$ cannot be canceled out.

Acknowledgement

The authors thanks Dr. Luz V. Vela-Arevalo and any teaching assistants of MATH 2605 of Georgia Tech Institute of Technology for providing the authors with the knowledge necessary to take on such a topic.

References and Notes

1. ”Convolutional Coding.” Convolutional Coding (n.d.): n. pag. Massachusetts Institute of Technology. Massachusetts Institute of Technology, 4 Oct. 2010. Web. 27 Mar. 2015.

2. Strang, Gilbert. ”7.4 Iterative Methods for Ax=b.” Linear Algebra and Its Applications. San Diego: Harcourt, Brace, Jovanovich, 1988. 367-72. Print.