Braden Scothern
Kyle Heaton
May 3, 2016

Hamming Codes
The First Binary Data Verification System

The Origin Of Hamming Codes

In 1947 Bell Labs was using their Bell Model V for computations throughout each week. This computer took punch cards and because of the high tendency of error that comes from human input, the computer would do one of two things to handle errors once detected. If it was during the normal work week, the computer would turn on blinkers to indicate the error so it could be corrected and if it was the weekend it would simply skip the computation and go onto the next task it had scheduled.

The tendency to come back on Monday mornings and find out that your program had an error and was unable to compute was frustrating for Richard W. Hamming so being a mathematician he decided to devise a way that he could not only detect the errors in the input but correct them. His resulting solution came to be known as Hamming Codes.

The Basic Idea of Hamming Codes

All computer data is stored as combination of 0's and 1's in a particular order and encoding which translated what the data represents. The idea behind Hamming Codes is to add parity bits to the data stream in order to verify that all data has been received correctly. These bits are organized and distributed in such a way that it is possible to identify where the error has occurred in the data stream and correct it.

It works out that if you have 4 bits then you can find out how many parity bits by finding the power of 2 that that number of bits has, in this case we get $4 = 2^2$ which gives us a power of 2. We then take that power and add 1 to it in order get the number of parity bits that are required for the data's integrity to be verified. So with 4 bits we end up with 3 parity bits meaning the final data stream will have 7 bits.

Applications Of Linear Algebra

The parity bits are all responsible to verify different pieces of the original data and have varying amounts of data that they validate. Because of how these have been organized they create a pattern that can be described with a linear code. The typical method that is used to compute and use the parity bits is to have data and organize its bits as a vector, then multiply that through a generator matrix known as G. This will correctly add in the parity bits and on the receiving end you use the parity-check matrix typically known as H to verify that data was transmitted correctly. To construct these matrices you need to understand that they have the following relationship: $GH^T = 0$. It is easiest to start with the construction of H and to use it to construct G.

Creating H

We know that H will be a (Number of Parity Bits) X (Total Number Of Bits) Matrix. So for the example above we will be creating a matrix with 3 rows and 7 columns. The construction of this matrix is quite simple, we take the bit representation of the column starting at 1 and input them into the matrix like this:

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Creating G

You then need to construct G from H. This is done by using the following relationships: $H := (A \mid I_{n-k})$ and $G := \left(I_k \mid -A^T\right)$. So we put H into proper form to construct G by swapping the columns so we have the identity matrix on the right hand side of the matrix like this:

$$H = \left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

This is done by swapping C1 with C7, C2 with C6, and C4 with C5. We now identify the right hand side as A and use it to construct G by first transforming it and setting up G like this:

$$G = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]$$

We now undo all of the previous column swaps in reverse order to construct the final form of G. So we swap C4 with C5, C2 with C6, and C1 with C7 to create this matrix:

$$G = \left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}\right]$$

The final step of constructing G is to transform it into this form:

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using G and H to verify data

Now that the matrices G and H have been constructed they can be used to correctly add parity bits to verify that data is valid. This is done by taking data such as 1011 and putting it into a vector which is multiplied with G like this:

$$Gd = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ 0 \\ 3 \\ 1 \\ 1 \\ 1 \end{bmatrix} \Rightarrow_{\text{mod } 2} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

At this point we can send the data 0001111 across and not only is the original data preserved we can use this to verify that the data was correctly sent. This is done by taking our outputted data and multiplying it with H like this:

$$Hd_2 = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 2 \end{bmatrix} \Rightarrow_{\text{mod } 2} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

The resulting zero vector means that there were no errors in the transportation of the data 1011. The Hamming codes here can detect 2 bit errors and correct single bit errors in transmission. Let's demonstrate a single bit transmission error. Here we have bit 1 of the transmission is flipped.

$$D_{error1} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad HD_{error1} = \begin{bmatrix} 4 \\ 2 \\ 3 \end{bmatrix} \Rightarrow_{\text{mod 2}} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

If the resulting matrix is read as binary we can see the final column vector shows that there was an error in the first bit. What if there is an error in bit 5?

$$D_{errror2} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \qquad HD_{error2} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} \Rightarrow_{\text{mod 2}} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Again, if we read the final column vector as binary we can see the 5th bit is the one with an error in it. If there is a 2 bit error, for example bits 2 and 3, we can plainly see that an error has been detected, but correction is not possible.

$$D_{errror3} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad HD_{error3} = \begin{bmatrix} 4 \\ 4 \\ 3 \end{bmatrix} \Rightarrow_{\text{mod 2}} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

An error has been detected, but without adding more parity bits, there is no way to know which bits are in error. It is showing the same error as if bit 1 was in error. Thus when using Hamming codes for error handling you can either use single bit error correction, or dual bit error detection, not both.

Sources:

1) http://biobio.loc.edu/chu/web/Courses/ITEC460/hamming_codes.htm
2) http://www.gaussianwaves.com/2008/05/construction-of-hamming-codes-using-matrix/
3) http://www.inference.phy.cam.ac.uk/itprnn/book.pdf
4) http://www-groups.dcs.st-and.ac.uk/~history/Biographies/Hamming.html