Convolutional codes

KG

May 3, 2013

1 Setting

We have a tuple of infinite sequences of message symbols

$$\begin{gathered} m_{10}, m_{11}, m_{12}, \dots \\ m_{20}, m_{21}, m_{22}, \dots \\ \vdots \\ m_{k0}, m_{k1}, m_{k2}, \dots \end{gathered}$$

that we want to encode as a tuple of infinite sequences of codeword symbols

$$\begin{gathered} c_{10}, c_{11}, c_{12}, \dots \\ c_{20}, c_{21}, c_{22}, \dots \\ \vdots \\ c_{n0}, c_{n1}, c_{n2}, \dots \end{gathered}$$

The ith *information frame* is the vector $(m_{1i}, m_{2i}, \dots, m_{ki})$, while the ith *codeword frame* is $(c_{1i}, c_{2i}, \dots, c_{ni})$.

We shall study injective maps from k-tuples of infinite series to n-tuples of infinite series, and how to recover the k-tuple from the n-tuple in the presence of errors.

The *information rate* of the code is the fraction k/n. Obviously, one goal is to have the information rate as high as possible.

Often, we shall consider the sequences as produced sequentially in time, so that a symbol m_{i1} comes before m_{i2} in time, and m_{1i} comes before m_{3i}:

$$m_{10}, m_{20}, \dots, m_{k0}, m_{11}, \dots, m_{k1}, m_{12}, \dots$$

2 Power Series

Let $\mathbf{F}$ be a field. The power series ring $\mathbf{F}[\![x]\!]$ is the set of infinite formal sums

$$\left\{ a_0 + a_1 x + a_2 x^2 + \cdots = \sum_{i=0}^{\infty} a_i x^i \middle| a_i \in \mathbf{F} \right\}.$$

Addition is coefficient-wise, and multiplication of $\sum a_i x^i$ and $\sum b_i x^i$ is the series $\sum c_i x^i$ where the coefficients are given by the sums

$$c_i = \sum_{j=0}^{i} a_j b_{i-j}.$$

It is easy to show that this really is a ring.

We note that the polynomials form a subring of the power series ring. We say that a power series has *finite weight* if it has only a finite number of non-zero coefficients, that is, if it is a polynomial. If it is not a polynomial, we say that it has *infinite weight*.

It is easy to show that $\mathbf{F}[\![x]\!]$ has a single maximal ideal generated by x ($\mathbf{F}[\![x]\!]$ is a local ring), and that the elements of this ideal (those with zero constant term) are all the non-invertible elements. This means that we can always write a non-zero power series as $x^d f(x)$, where $d \geq 0$ and $f(x)$ is invertible.

Next, we consider the field of fractions $\mathbf{F}(\!(x)\!)$ of $\mathbf{F}[\![x]\!]$. Consider a fraction $a(x)/b(x)$. We can always write $b(x) = x^d c(x)$, where $c(x)$ is invertible, say with inverse $e(x)$. Then

$$\frac{a(x)}{b(x)} = \frac{a(x)}{x^d c(x)} = \frac{a(x)e(x)}{x^d} = x^{-d} a(x)e(x).$$

We have a canonical form for the non-zero elements of the field of fractions: $x^d f(x)$, where $f(x)$ is invertible and d is any integer, both positive and negative.

The rational functions (fractions where the numerator and denominator are both polynomials) form a subfield of $\mathbf{F}(\!(x)\!)$. We say that an element of $\mathbf{F}(\!(x)\!)$ has *finite weight* if it is of the form $x^d f(x)$, where $f(x)$ is a polynomial.

3 Convolutional Codes

First, we assume that our symbols come from a finite field $\mathbf{F}$. We shall then represent each infinite sequence as a power series from the power series ring $\mathbf{F}[\![x]\!]$,

and k-tuples and n-tuples as vectors of power series. To simplify notation, we denote these tuples of power series by $\mathbf{m}(x)$ and $\mathbf{c}(x)$.

We want to study injective maps $G : \mathbf{F}[\![x]\!]^k \to \mathbf{F}[\![x]\!]^n$ and the possibility of recovering the starting point after perturbing the code word tuple with an error vector,

$$\mathbf{m}(x) \xmapsto{G} \mathbf{c}(x) \rightsquigarrow \mathbf{c}(x) + \mathbf{e}(x) \rightsquigarrow \mathbf{m}(x).$$

We begin by formulating a number of restrictions on what maps we are interested in.

1. The ith codeword frame should only depend on the first i information frames: if $\mathbf{a}(x) \equiv \mathbf{b}(x) \pmod{x^{i+1}}$, then $G(\mathbf{a}(x)) \equiv G(\mathbf{b}(x)) \pmod{x^{i+1}}$.

2. The map G should be $\mathbf{F}$-linear. That is, for any $\mathbf{a}(x)$, $\mathbf{b}(x)$ and any $c \in \mathbf{F}$, we have that $G(c\mathbf{a}(x)) = cG(\mathbf{a}(x))$ and $G(\mathbf{a}(x)+\mathbf{b}(x)) = G(\mathbf{a}(x))+G(\mathbf{b}(x))$.

3. The map G should respect multiplication by x. That is, $G(x^i\mathbf{m}(x)) = x^iG(\mathbf{m}(x))$ for any $i \geq 0$.

4. The map G should have *finite constraint lenght* d. That is, the ith codeword frame should only depend on the ith information frame and the d preceeding information frames.

Injective maps from $\mathbf{F}[\![x]\!]^k$ to $\mathbf{F}[\![x]\!]^n$ that satisfy these four points are called *convolutional encoders*, and the image of these maps are called *convolutional codes*.

Three of the four restrictions can be interpreted in terms of time as follows:

1. Codeword frames should not depend on information frames from the future.

3. Delaying the information frames is equivalent to delaying the codeword frames.

4. A finite memory for message symbols should be sufficient to compute the codeword information.

At this point, we can begin to deduce various properties of these encoders. Consider a k-tuple $\mathbf{m}(x) = (m_1(x), m_2(x), \ldots, m_k(x))$, $m_j(x) = \sum_{i=0}^{\infty} m_{ji}x^i$, $j = 1, 2, \ldots, k$. We shall first look at the first l codeword tuples, so modulo x^l we

have (by the first, third and second property, followed by rearranging sums) that

$$\begin{aligned}
G(\mathbf{m}(x)) &\equiv G\left(\left(\sum_{i=0}^{\infty} m_{1i}x^i, \sum_{i=0}^{\infty} m_{2i}x^i, \ldots, \sum_{i=0}^{\infty} m_{ki}x^i\right)\right) \\
&\equiv G\left(\left(\sum_{i=0}^{l-1} m_{1i}x^i, \sum_{i=0}^{l-1} m_{2i}x^i, \ldots, \sum_{i=0}^{l-1} m_{ki}x^i\right)\right) \\
&\equiv \sum_{i=0}^{l-1} x^i G\left((m_{1i}, m_{2i}, \ldots, m_{ki})\right) \equiv \sum_{i=0}^{l-1} x^i \sum_{j=1}^{k} m_{ji} G(\mathbf{e}_j) \\
&\equiv \sum_{j=1}^{k} G(\mathbf{e}_j) \sum_{i=0}^{l-1} m_{ji}x^i \equiv \sum_{j=1}^{k} G(\mathbf{e}_j) m_j(x) \pmod{x^l},
\end{aligned}$$

where $\mathbf{e}_j$ is the vector with 1 in its jth coordinate and 0 everywhere else. Since this holds for any l, we have that

$$G(\mathbf{m}(x)) = \sum_{j=1}^{k} G(\mathbf{e}_j) m_j(x).$$

In other words, the map G is uniquely determined by its action on the unit vectors $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_k$. If $G(\mathbf{e}_j) = (g_{j1}(x), g_{j2}(x), \ldots, g_{jn}(x)) \in \mathbf{F}[\![x]\!]^n$, then the map G is quite simply right-multiplication by the matrix

$$\begin{pmatrix}
g_{11}(x) & g_{12}(x) & \cdots & g_{1n}(x) \\
g_{21}(x) & g_{22}(x) & \cdots & g_{2n}(x) \\
\vdots & & & \vdots \\
g_{k1}(x) & g_{k2}(x) & \cdots & g_{kn}(x)
\end{pmatrix}.$$

We shall identify the map G with this matrix, called a *generator matrix* for the convolutional code, and denote the application of G to $\mathbf{m}(x)$ by $\mathbf{m}(x)G$, right-multiplication by G.

Finally, the fourth property says that the ith information frame should influence at most the next $d+1$ codeword frames. In other words, the power series $m_{ji}x^i g_{jl}(x)$, which describes the influence of the jth bit of the ith information frame, should have only a finite number of non-zero coefficients. That is, it should have finite weight, which means that $g_{jl}(x)$ should be a polynomial.

Let us now consider the vector space map from $\mathbf{F}(\!(x)\!)^k$ to $\mathbf{F}(\!(x)\!)^n$ defined by G. We know that for any vector $\mathbf{m}(x) \in \mathbf{F}(\!(x)\!)^k$, there is an integer $d \geq 0$ such

that $x^d \mathbf{m}(x) \in \mathbf{F}[\![x]\!]^k$. It follows that the action of G on $\mathbf{F}[\![x]\!]^k$ is essentially the same as the action on $\mathbf{F}(\!(x)\!)^k$, in particular, G is still injective.

Since G is an injective vector space map from $\mathbf{F}(\!(x)\!)^k$ to $\mathbf{F}(\!(x)\!)^n$, we have that $k \leq n$ and that the matrix has rank k.

We see that a convolutional encoder is a $k \times n$ matrix with polynomial entries that has a right inverse (though not necessarily with polynomial entries). An (n,k)-convolutional code is essentially a k-dimensional subspace of $\mathbf{F}(\!(x)\!)^n$.

4 Choosing Encoders

For any convolutional code, there are many different encoders G. Which shall we use?

Catastrophic Encoders We shall first discuss a property we do not want our encoder to have. Suppose we have some message $\mathbf{f}(x)$ of infinite weight such that $\mathbf{e}(x) = \mathbf{f}(x)G$ has finite weight. It can then happen that a finite error in transmission, namely $\mathbf{e}(x)$, will lead to an infinite number of errors in the decoding, namely $\mathbf{f}(x)$.

If there exists some message $\mathbf{f}(x)$ of infinite weight such that its encoding $\mathbf{f}(x)G$ has finite weight, we say that the encoder is *catastrophic*. We do not want to use catastrophic encoders.

It is easy to see that a sufficient condition for a generator matrix G to be non-catastrophic is the following:

> The generator matrix G has a right inverse where every entry has finite weight.

It can be shown that this is also a necessary condition.

Canonical Generator Matrices We would like a generator matrix that makes the computation of the codeword as simple as possible. One measure of simplicity is the number of information frames required to compute a given codeword frame.

When $\mathbf{c}(x) = \mathbf{m}(x)G$, we know that

$$c_i(x) = \sum_{j=1}^{k} m_j(x) g_{ji}(x).$$

If $d_{ji} = \deg g_{ji}(x)$, we need d_{ji} symbols from $m_j(x)$ to compute each symbol in $c_i(x)$. In other words, we need

$$d_j = \max_i d_{ji}$$

symbols from $m_j(x)$ to compute $\mathbf{c}(x)$. In total, we need

$$d = \sum_j \max_i \deg g_{ji}(x)$$

memory to compute the codeword frames of $\mathbf{c}(x)$. This number d is called the *external degree* of G.

For any given convolutional code, there is at least one encoder with minimal external degree. Such an encoder is called a *canonical generator matrix* for the convolutional code. The minimal external degree of the encoders is called the *degree* of the convolutional code.

For any vector $\mathbf{a}(x)$ of polynomials, let

$$\deg \mathbf{a}(x) = \max_i \deg a_i(x).$$

Let G be a generator matrix, and let $\mathbf{g}_j(x)$ denote its jth row. It can be shown that G is canonical if and only if the following two conditions are met:

1. There exists a right inverse for G with polynomial entries.
2. For any vector $\mathbf{m}(x)$ of polynomials,

$$\deg(\mathbf{m}(x)G) = \max_j(\deg m_j(x) + \deg \mathbf{g}_j(x)).$$

Note that any canonical generator matrix is also a non-catastrophic encoder. It is easy to see that the converse does not hold.

5 Representing Encoders

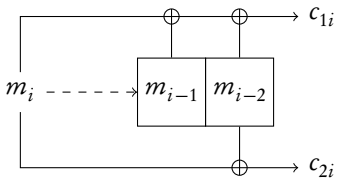
We discuss a number of representations for encoders by way of an example over $\mathbf{F}_2$. So consider the generator matrix

$$G = \begin{pmatrix} 1+x+x^2 & 1+x^2 \end{pmatrix}.$$

We see that the ith codeword frame can be computed from the ith, the $i-1$th and the $i-2$th information frames using the formulae

$$\begin{aligned} c_{1i} &= m_i + m_{i-1} + m_{i-2}, \qquad \text{and} \\ c_{2i} &= m_i + \qquad\quad m_{i-2}. \end{aligned}$$

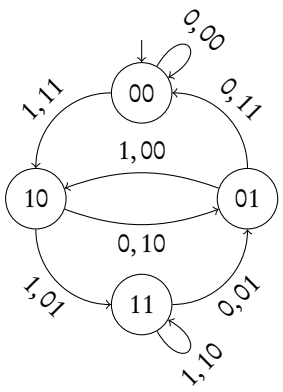
Such formulae can be efficiently implemented using shift registers, as illustrated by the following diagram:



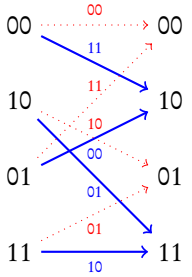
Note that the number of memory cells used by this circuit is equal to the degree of the generator matrix.

After feeding a sequence of message symbols to this circuit, the output need not be the complete codeword. To get a codeword, one needs to feed a (fixed number) of zeros to the circuit such that the memory is guaranteed to be all zeros.

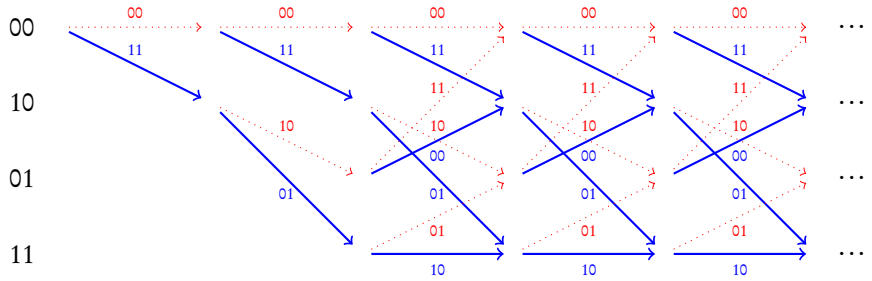
This circuit can in turn be represented as a finite state machine with output:



Another representation of the state machine, where dotted red arrows denotes 0 as input and blue arrows denote 1 as input, is the following:



This diagram is most useful when combined into a *trellis diagram* as follows:



Any path that starts in the top left corner corresponds to a (truncated) codeword. Any path that starts in the top left corner and terminates in the all-zero state (top row) corresponds to a codeword. The message corresponding to the path can be deduced from the colours of the edges, and the codeword corresponding to the path can be computed by concatenating the edge labels. We shall use a variant of this diagram for decoding.

6 Decoding

We have been given the message $\mathbf{m}(x)$ and a convolutional code with generator matrix G. The code word $\mathbf{c}(x)$ was transmitted and $\mathbf{y}(x) = \mathbf{c}(x) + \mathbf{e}(x)$ was received. The decoding problem is to recover $\mathbf{m}(x)$ (or equivalently, $\mathbf{c}(x)$) from $\mathbf{y}(x)$.

Finding the First Information Frame Observe that if we by some method can determine the first information frame, that is, the constant term of $\mathbf{m}(x)$, say $\mathbf{m}_0 = (m_{10}, m_{20}, \ldots, m_{k0})$, we can compute

$$\mathbf{y}(x) - \mathbf{m}_0 G \equiv \mathbf{c}(x) + \mathbf{e}(x) - \mathbf{m}_0 G \equiv (\mathbf{m}(x) - \mathbf{m}_0)G + \mathbf{e}(x) \equiv \mathbf{e}_0 \pmod{x},$$

where $\mathbf{e}_0$ is the constant term of the error series. Then

$$\mathbf{y}(x) - \mathbf{m}_0 G - \mathbf{e}_0 = x\mathbf{z}(x)$$

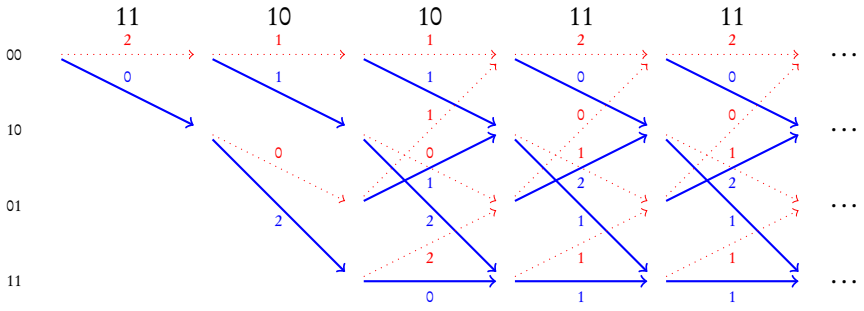
for some power series $\mathbf{z}(x)$. This means that after decoding the first information frame, we can determine the error in the first codeword frame. After removing the error and the effect of the first information frame, what remains is essentially a new power series $\mathbf{z}(x)$. We can now proceed to decode the first information frame of this power series just as we did for $\mathbf{y}(x)$, which will result in yet another power series, etc.

In other words, the decoding problem can be reduced to recovering the first information frame. While this approach works, it may not be the best approach.

Viterbi decoding Again, we illustrate with an example over $\mathbf{F}_2$ based on the generator matrix

$$G = \begin{pmatrix} 1+x+x^2 & 1+x^2 \end{pmatrix}.$$

Suppose we received the bits 11 10 10 11 11 .... We draw a variant of the the trellis diagram where each edge is labeled not with the output of the state machine, but rather the Hamming distance between the output of the state machine and the corresponding codeword frame.



This is now a weighted graph. The weight of any path starting in the top left corner and terminating on the right hand side is equal to the Hamming distance between the corresponding (truncated) code word and the received word. Finding a (truncated) codeword with minimal distance to the received bits now corresponds to finding such a path with minimal weight. Note that there may be more than one path with minimal weight. Once we have decided on a path, the corresponding message can be deduced from the colours of the edges in the path.

To ensure that we find a codeword and not a truncated codeword, we are really looking for a path from the top left corner that returns to the all-zero state. (The example shows a truncated codeword with errors.)