EE 121: Digital Communications

April 24, 2008

The Viterbi Algorithm

Introduction

In this lecture, we will show that by buffering and implementing a block decoding scheme even while transmitting bit by bit, we can implement the ML receiver and perform a kind of decoding that is known as *Viterbi Decoding* or the *Viterbi Algorithm*. We will also get a sense of what computational (complexity) issues we face when we actually try to implement this kind of ML receiver.

Block Decoding and the Viterbi Algorithm for the 2-tap ISI channel

At the end of last lecture, we said that the ISI might actually benefit us while decoding if we decode all the bits being transmitted as a *block* since the ISI both explicitly contains information about the bit that was sent in the time instant before the present one, and implicitly contains information about all the bits that were sent before the present time instant. A particular scheme that we shall study right now that is based on this type of thinking is known as the *Viterbi Decoder*. The idea behind this is block decoding: after n time instants, decode all the n bits $(b_1, b_2, \ldots, b_n)$ together. We have as a starting point the 2-tap ISI channel

$$y[m] = x[m] + h[1]x[m-1] + w[m] \tag{1}$$

We are sending one bit every time instant and we use the mapping

$$x[m] = \begin{cases} \sqrt{E} & b_m = 1 \\ -\sqrt{E} & b_m = 0 \end{cases}$$

Recall that when we had to implement the ML receiver for block signaling, we used a vector notation where each vector of voltages had as elements the voltages at the time instant that corresponded to the index of that element- for example, the received vector $\mathbf{y}$ had elements y_m which were the received voltages at time instant m. Here, we have a *meta-state*, or a *state vector*

$$\mathbf{s}[m] = \begin{bmatrix} x[m-1] \\ x[m] \end{bmatrix} \tag{2}$$

The state vector, or simply the state at time instant m contains the voltage from the previous time instant as the first element and the current transmission voltage as the second element. The state vectors will provide us with the capability to come back to a channel model that looks like the AWGN channel, where we are more comfortable while doing any decoding. Once we solve for a unique set of states using the ML rule, we have a set of voltages we have decided on by using the ML rule, since the states are nothing but a pair of voltages. Once we have obtained a set of voltages from the states using the ML rule, we can use the set of

voltages that is decided by the ML rule to decide the set of n bits that were sent, since we know the bits to voltages mapping. This is the manner in which we can decode a block of bits. We can write the channel model as

$$y[m] = x[m] + h[1]x[m-1] + w[m] = \begin{bmatrix} h[1] & 1 \end{bmatrix} \mathbf{s}[m] + w[m] \tag{3}$$

This then resembles an AWGN channel where we are more accustomed to applying the ML rule. However, here the quantity that we can estimate is not the voltage $x[m]$, but it is the state $\mathbf{s}[m]$, but we already know from construction how to go from the states to the voltages and then to the bits. If we denote the sequence of states after n time instants- we will have n states since we have one state for each time instant- by $\mathbf{s}^{(n)}$,

$$\mathbf{s}^{(n)} = (\mathbf{s}[1], \mathbf{s}[2], \ldots, \mathbf{s}[n]) \tag{4}$$

Much like how there were 2^n bit sequences from which we had to decide the most likely one, we have here 2^n possible sequences $\mathbf{s}^{(n)}$- each state can take only 2 values , with the first element for the first state being set to some arbitrary bit that we had transmitted before time 1 (this we don't really care about). For every subsequent state, the first element is fixed by the second element of the previous state, and the second element can take one of two values $\pm\sqrt{E}$. Thus we can define the ML receiver problem in the following way: Given a sequence of received voltages $(y[1], y[2], \ldots, y[n])$, find that sequence of states $\mathbf{s}^{(n)}$ that maximizes the likelihood of receiving that particular received voltage vector. Like the AWGN case, we can proceed by saying

- The probability of receiving a particular voltage at time m, $y[m]$, given a particular state $\mathbf{s}[m]$ at time m is simply the probability that the noise equals the received voltage minus $\begin{bmatrix} h[1] & 1 \end{bmatrix} \mathbf{s}[m]$, which can be evaluated using the Gaussian pdf.
- The likelihood of receiving a particular sequence of voltages $\mathbf{y}$ given a sequence of states $\mathbf{s}^{(n)}$, is simply the product of the probabilities of the noise being the received signal minus the transmitted signal component at each time instant, since the noises at each time instant are independent. This gives the following equation:

$$L_{\mathbf{s}^{(n)}} = \prod_{m=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\left(y[m] - \begin{bmatrix} h[1] & 1 \end{bmatrix} \mathbf{s}[m]\right)^2}{2\sigma^2}} \tag{5}$$

The ML receiver dictates that we pick that particular $\mathbf{s}^{(n)}$ for which $L_{\mathbf{s}^{(n)}}$ is maximum. Once we pick such an $\mathbf{s}^{(n)}$, we can find the sequence of states. From the sequence of states we can find the sequence of voltages and from the sequence of voltages we can find the sequence of bits $(b_1, b_2, \ldots, b_n)$. In order to maximize the likelihoods over the states $\mathbf{s}[m]$, note that the product of exponentials has in the power term in Equation(5), the sum of the individual power terms, which is $\sum_{m=1}^{n} \left(y[m] - \begin{bmatrix} h[1] & 1 \end{bmatrix} \mathbf{s}[m]\right)^2$. With the negative sign in front of this term in the power of the exponential, we have the simplified version of the ML rule:

$$\text{Pick } \mathbf{s}^{(n)} \text{ such that } \sum_{m=1}^{n} \left(y[m] - \begin{bmatrix} h[1] & 1 \end{bmatrix} \mathbf{s}[m]\right)^2 \text{ is minimized.}$$

If we were actually doing this kind of computation and comparison, how many computations would it take to implement this receiver? For each $\mathbf{s}^{(n)}$, not counting the subtractions inside each term of the summation we have n additions and n multiplications. Therefore, we have $n2^n$ additions and $n2^n$ multiplications on the whole. For a few tens of bits even, this computation becomes very cumbersome and takes up a large number of cycles. We would like a smarter algorithm to make the likelihoods comparison and implement the ML receiver. It turns out that such an algorithm exists and is known as the *Viterbi Algorithm*, which we shall now study.

The Viterbi Algorithm[1]

At the end of the previous section, we looked at the computational "cost"- the number of additions and multiplications it would take to implement the ML receiver. We saw that a brute-force exhaustive search would require a complexity that grows *exponentially* with the block length n. An efficient algorithm needs to exploit the structure of the problem and moreover should be recursive in n so that the problem does not have to be solved from scratch for determining each state $\mathbf{s}[m]$. The solution lies in an algorithm known as the *Viterbi algorithm*- an algorithm that lays out the states for each time instants $1 \rightarrow n$ on a plane as a collection of nodes and uses the metric $\left(y[m] - \begin{bmatrix} h[1] & 1 \end{bmatrix} \mathbf{s}[m]\right)^2$ as a "cost" of passing through a particular state node $\mathbf{s}[m]$ at time m, given that we have received a voltage $y[m]$. The state sequence that is estimated to be the transmitted sequence is then that sequence that yields the minimum total cost of going from time instant 1 to time instant n. We present in Figure 1 four states that represent the four possible states while transmitting bit by bit. The $x[m]$ are normalized by a factor of $\sqrt{E}$. This "state machine" builds up to the Viterbi Algorithm by showing how we go from one state to another based on a value of transmitted voltage at the present instant $x[m]$. The spirit in which we proceed is : *If we are in a given state , and if at the present time instant a particular voltage was transmitted, what state would we go to?*

With the 2 tap channel, we have defined the state to contain two elements, the interfering element and the current transmission element. We can extend this concept to an L tap channel and say that at time m,the state is an L-dimensional vector,

$$\mathbf{s}[m] := \begin{bmatrix} x[m-L+1] \\ x[m-L+2] \\ \cdot \\ x[m] \end{bmatrix} \tag{6}$$

The optimization problem of finding the most efficient ML receiver can be represented as the problem of finding the shortest path through an n-stage *trellis*, as shown in Figure 3. We

[1]Adapted from "Fundamentals of Wireless Communications", by David Tse and Pramod Viswanath, Cambridge University Press,2005.

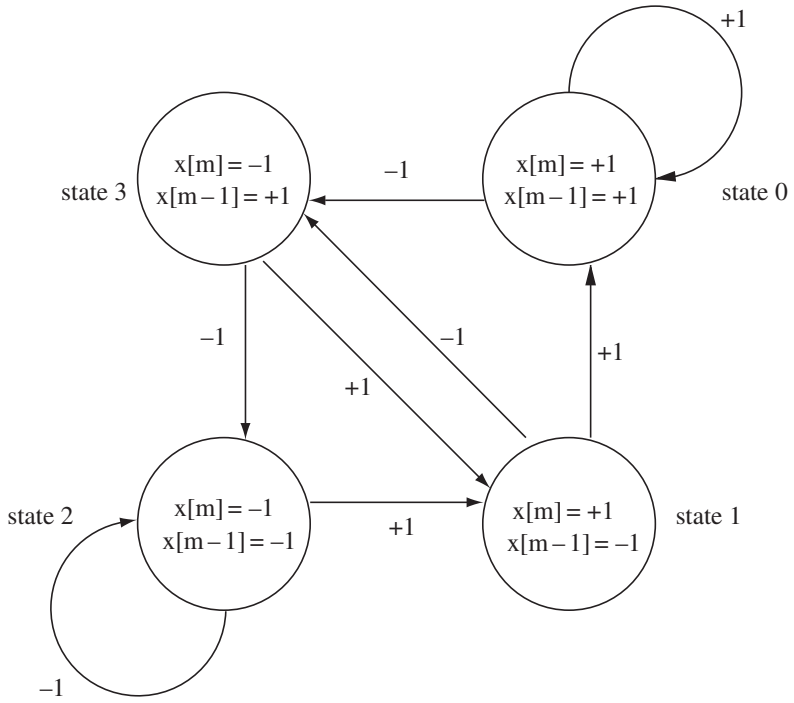

Figure 1: A finite state machine when $\frac{x[m]}{\sqrt{E}} = \pm 1$. There is a total of four states.

have the 4 possible states at each time instant laid out as nodes for each time instant. The index m indicate the time instant we are at. The $s[m]$ at the top indicate which exact state we are going to decide as being the transmitted state at time m (this will become clearer in a moment). Each state sequence $\mathbf{s}^{(n)} = (\mathbf{s}[1], \ldots, \mathbf{s}[n])$ is visualized as a path through the trellis, and given the received sequence $y[1], \ldots, y[n]$, the cost associated with the mth transition is

$$c_m(\mathbf{s}[m]) := \left(y[m] - \begin{bmatrix} h[1] & 1 \end{bmatrix} \mathbf{s}[m]\right)^2 \tag{7}$$

The solution is given recursively by the *optimality principle* of dynamic programming. Let $V_m(\mathbf{s})$ be the cost of the shortest path to a given state $\mathbf{s}[m]$ at stage m. Then $V_m(\mathbf{s})$ for all states $\mathbf{s}[m]$ can be computed recursively:

Here the minimization is over all possible states $\mathbf{u}$, i.e., we only consider the states that the finite state machine can be in at stage $m-1$ and, further, can still end up at state $\mathbf{s}[m]$ at stage m. The correctness of this recursion is based on the following intuitive fact: if the shortest path to state $\mathbf{s}[m]$ at stage m goes through the state $\mathbf{u}^*$ at stage $m-1$, then the part of the path up to stage $m-1$ must itself be the shortest path to state $\mathbf{u}^*$. See Figure 2. Thus, to compute the shortest path up to stage m, it suffices to augment only the shortest paths up to stage $m-1$, and these have already been computed.

Once $V_m(\mathbf{s})$ is computed for all states $\mathbf{s}[m]$, the shortest path to stage m is simply the minimum of these values over all states $\mathbf{s}[m]$, $m : 0 \to n$. Thus, the optimization problem is solved. Moreover, the solution is recursive in n. Figure 3 shows the "trellis" that is used in

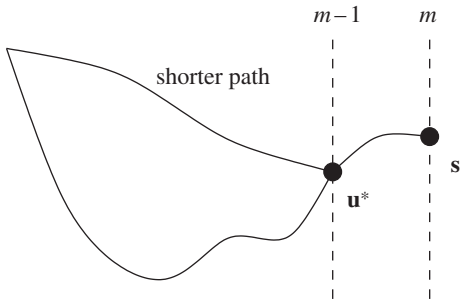

Figure 2: The dynamic programming principle. If the first $m-1$ segments of the shortest path to state $\mathbf{s}$ at stage m were not the shortest path to state $\mathbf{u}^*$ at stage $m-1$, then one could have found an even shorter path to state $\mathbf{s}$.

the Viterbi decoder. Once again, we lay out the four possible states that are related to the transmit signals (the relationship can be seen in the state diagram of Figure 1), for each time instant m going from 0 to n. The state at time 0 provides an initial condition from where we can work our way forward in the trellis and estimate the transmitted bits by calculating the "costs" c_m. Our movement among the possible states while going from one time instant to another is restricted by the fact that the second element of the state at time $m-1$ must be the first element of the state at time m (here we are considering a 2-tap channel). Keeping this in mind, the dashed lines represent all possible paths that we can take in the trellis, or equivalently, all the valid sequences of states from which we can choose one as the sequence most likely to represent the transmitted bits. The decoding procedure involves two steps:

1. Calculate the costs along each transition from one node to another, or the "cost" of traversing through a particular state represented by a node. Imagine we are "sitting"in a state at time $m-1$- we take the received signal $y[m]$ and calculate the cost of going to each state in time instant m using the cost metric of Equation(7). Bear in mind that we have information about $y[m]$ for all m when we are decoding since this is a block scheme. In this manner, we can calculate the cost of all transitions, or passing through each state node at each time instant.

2. Find the minimum cost path from time instant 0 to n and decide on the best sequence- this is where the dynamic programming principle comes in. If we want to find the minimum cost path from time instant 0 to n, this path must necessarily include the minimum cost path from time instant 0 to $n-1$. We can start at time instant n and recursively apply this principle to find the best sequence. We find the shortest path from $m-1$ to m, and then find the shortest path to that node in time instant $m-1$ that is part of the shortest path from time instant $m-1$ to m. This procedure is repeated over all $m : n \rightarrow 1$.

The number of computations we have to make here is equal to the number of states times the number of bits or time instant, which is $4n$. Thus, the complexity of the Viterbi algorithm

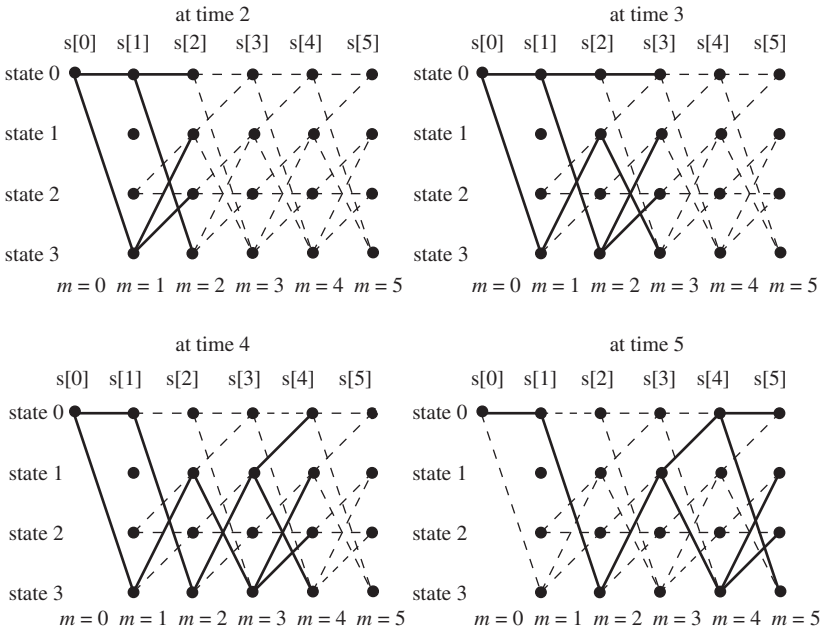

Figure 3: The trellis representation of the channel.

is linear in the number of time instants n. Thus, the cost is constant per symbol, a vast improvement over brute-force exhaustive search. However, its complexity is also proportional to the size of the number of states, which itself depends on the number of taps in the channel as we have shown earlier. In our example, the number of elements we have to store cost information about, is the number of nodes in the trellis, which is the number of possible states 2^L (since each state has L elements) times the number of time instants, $n2^L$. For a large number of taps, we might end up with a problem of *storage complexity*- we might run out of space in memory when we are writing software for this decoder. This is one of the major drawbacks of the Viterbi Algorithm.

Looking Ahead

We have taken a major shift in view and said that instead of trying to cancel out ISI, we can use it to our advantage and perform block decoding. This was a receiver based technique. In the next lecture, we will look at a transmitter based technique employed in the DSL modem, called OFDM.