The Viterbi algorithm demystified

G. William Slade

Abstract

Despite being one of the most important algorithms of the 20^{th} century, the Viterbi algorithm [1], [2], [3], like the fast Fourier transform, represents a source of confusion to many people, including the author. This paper is an attempt to dispel some of the mystery by documenting the author's personal exploration of the Viterbi algorithm. As will be shown, the Viterbi algorithm is a scheme for finding the most likely sequence of hidden (in the sense of not being directly observable) states of a stochastic system using the information in a set of "imperfect" observations. In the realm of digital signal processing (DSP), most presentations of the Viterbi algorithm are based on descriptions of decoding bit streams. The connections to probability and Bayesian inference are, more often than not, buried in the descriptions of the structure of the algorithm and data-flow (trellis) diagrams with little reference to the theoretical origins of the algorithm: the hidden Markov model [4], [5], [6].

In this article, we briefly review the basics of the Hidden Markov Model. Furthermore, we will explore how the algorithm is constructed. We finish with two illustrative example calculations: a discrete observation variable (hard-decision) Viterbi algorithm, where we investigate a "toy" problem involving the weather and a character whose behavior is weather dependent, but somewhat random. We also develop a somewhat more practical application where we discuss a continuous observation variable (soft-decision) algorithm, with which we implement a forward error-correction decoder using the Viterbi algorithm, showing its relation to continuous observable variables in probabilistic systems.

Index Terms

Hidden Markov Model, Probability, Viterbi Algorithm, Bayesian Inference, Convolutional Codes, Decoding, Gaussian Noise, Binary Phase Shift Keying

I. Introduction

A. Markov model

It is a remarkable thing that many physical processes can be modeled using a simple random process that depends only on the immediately preceding state of the process. A "random walk" [7] is a classic example; a "drunken walker" stumbles left or right, forward or backward at random from his present position. The new position depends on his present position and a random change governed by a probability distribution function. Another, more pertinent example would be a stream of digital symbols whose transitions are determined by the system of coding used (and the intended message contained therein). This is the Markov process. At any time t, the probability of being in any given state is the product of being in a state at $t-1$ and the probability of changing state.

To see how the Markov process functions, consider a system defined by the set of three possible states: R, S, W (to make the idea more concrete, let us assume the states represent the weather conditions on any given day:R=rainy, W=windy and S=sunny). The weather can transition from one state at time step $i-1$ to another at

Author can be reached at *bill.slade@ieee.org*

i with a probability $P(x_i|x_{i-1})$, where x_i is the system state S, R or W. ($P(x_i|x_{i-1})$) is read "the probability of changing to state x_i from state x_{i-1}.) Notice how the next state probability is conditional only on the previous state. This is the so-called first order Markov property that will be very important for the development of the Viterbi algorithm. Graphically, we represent the transitions using a state diagram as in Figure 1.

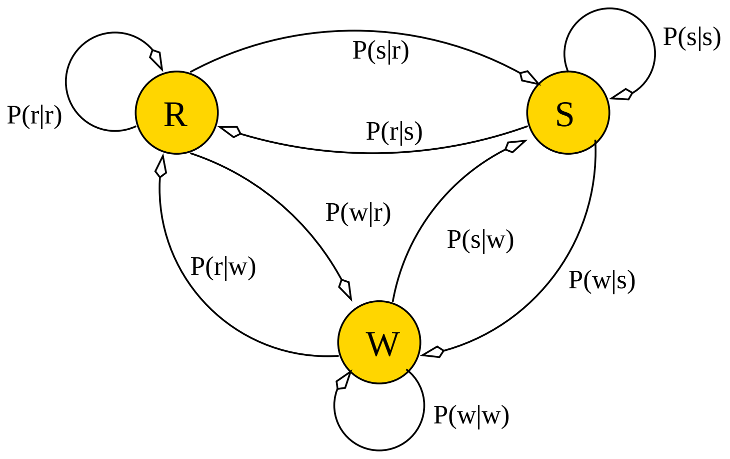


Fig. 1. Diagram showing the possible states of our hypothetical weather model and their transition probabilities. R=Rainy, W=Windy and S=Sunny.

Let us assume that it is summertime and the weather tends to be sunny most of the time. We shall define a state transition matrix $T_{ij} = P(x_i|x_j)$:

$$T_{ij} = \begin{array}{c|ccc} & \text{Rainy} & \text{Sunny} & \text{Windy} \\ \hline \text{Rainy} & 0.1 & 0.2 & 0.2 \\ \text{Sunny} & 0.1 & 0.7 & 0.7 \\ \text{Windy} & 0.8 & 0.1 & 0.1 \end{array} \qquad (1)$$

"Sunny" transitions to "Sunny" with an 70% chance, whereas "Rainy" stays "Rainy" only 10% of the time. Moreover, if it was rainy today, we see from the matrix in (1) have an 80% chance that it will be windy and a 10% chance of being either sunny or remain rainy tomorrow. We might also ask the question "If it is rainy today, what will the weather be like in three days' time?" This is answered by multiplicative chain of transition matrices to arrive at the relationship

$$\begin{pmatrix} P_{rain} \\ P_{sun} \\ P_{wind} \end{pmatrix} = \begin{pmatrix} 0.18 \\ 0.59 \\ 0.23 \end{pmatrix} = \begin{bmatrix} 0.1 & 0.2 & 0.2 \\ 0.1 & 0.7 & 0.7 \\ 0.8 & 0.1 & 0.1 \end{bmatrix} \begin{bmatrix} 0.1 & 0.2 & 0.2 \\ 0.1 & 0.7 & 0.7 \\ 0.8 & 0.1 & 0.1 \end{bmatrix} \begin{bmatrix} 0.1 & 0.2 & 0.2 \\ 0.1 & 0.7 & 0.7 \\ 0.8 & 0.1 & 0.1 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}. \qquad (2)$$

We see that we expect sunny weather with a 59% probability. However, there is a 23% probability of windy weather and an 18% chance that it will rain. Interestingly, we can extend far into the future and conclude that 18% of days will be rainy, 23% will be windy and 59% will be sunny. We have reached this steady-state probability after three days and we indeed see that the majority of days are likely to be sunny. We can also conclude that forecasts beyond a couple of days are of no use because the model loses all "memory" of the starting state after three days. We gain no forecasting benefit beyond two days from present knowledge. On the other hand, we can say what system behavior is most likely over the long term.

Note also that to be consistent probabilities, the elements of each column of T_{ij} must sum to one.

B. The observation model and the hidden Markov model

Of particular interest to us are systems where the state is not directly visible. We can only make "noisy" observations that are themselves random processes that depend on the (hidden) state of the Markov model.

Let us start off assuming our observations are discrete. Returning to the weather model, let us invent Mark, a character that has a very simple life. He likes to talk to his grandmother on the phone every day. Mark's grandmother knows nothing about the weather at Mark's place on any particular day, other than it follows the pattern set by our Markov model. She also knows Mark's habits. If it is sunny, he loves to walk in the park. If it is rainy, he hangs out at his girlfriend's apartment or cleans his own flat (which he really does not like doing). He also tends to go shopping when it is windy outside. Of course, there are extraordinary circumstances when he may decide not to follow his usual patterns. We codify these habits in the observation probability matrix O_{ij}:

$$O_{ij} = \begin{array}{r c} & \begin{array}{ccc} \text{Rainy} & \text{Sunny} & \text{Windy} \end{array} \\ \begin{array}{r} \text{Walk in park} \\ \text{Shopping} \\ \text{Clean flat} \\ \text{Visit girlfriend} \end{array} & \begin{bmatrix} 0.05 & 0.7 & 0.25 \\ 0.05 & 0.15 & 0.4 \\ 0.25 & 0.05 & 0.05 \\ 0.65 & 0.1 & 0.3 \end{bmatrix} \end{array} \tag{3}$$

As in the transition matrix, the column elements must sum to unity. Each element is the probability of observing activity $q_i = \{\text{walk in park, shopping, clean flat, visit girlfriend}\}$ given rainy, windy or sunny weather. Any observation depends conditionally only on the present state of the Markov model and is independent of all other states. The dependencies are illustrated graphically in Figure 2: the Bayes network graph [8], [9]. The dark

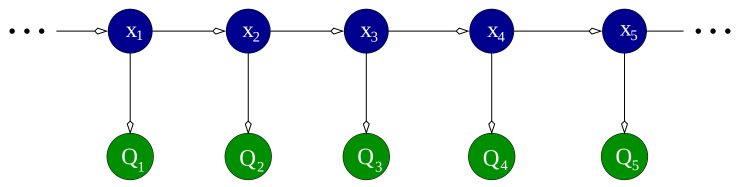


Fig. 2. Bayes network graph of the hidden Markov model. Observation probabilities depend exclusively on the present state of the hidden model and excludes all other states (Observational independence property). Likewise, the next model state depends exclusively on the preceding state (Markov property).

circles in the graph in Figure 2 represent the hidden weather states. The arrows indicate the dependence of later states on their immediate predecessors. The lighter circles represent the observation process. Each observation depends only on the present (hidden) state of the Markov model (Grandma's "noisy" observations of Mark's behavior).

The question we want to answer is: "using only the knowledge of the weather patterns (the Markov transition probability matrix), Mark's habits (known by Grandma) and the knowledge of what Mark did on a series of days (the telephone reports), can we construct the most likely weather sequence?" The answer is, of course, yes and the Viterbi algorithm provides the means!

C. *Basis of the Algorithm*

The solution lies in finding the sequence of weather states (represented by the weather state vector $\mathbf{x}$) that maximizes the posterior probability $P(\mathbf{x}|\mathbf{q})$, i.e. the probability of observing the vector containing the sequence of weather types over N days given that we have observed the sequence of N activities done by Mark (the observation set vector $\mathbf{q}$). Mathematically, we write this

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} P(\mathbf{x}|\mathbf{q}) \tag{4}$$

In general, solving this equation would involve an exhaustive search of all possible weather state sequences $\mathbf{x}$, which would rapidly become impossible for all but the smallest vector lengths N and state spaces N_S. In fact, for a general solution, the number of operations would be proportional to N_S^N, an exponential increase. Fortunately, it turns out that the hidden Markov model assumption brings a significant reduction in computational effort.

We know from Bayes' theorem that the posterior probability of observing vector $\mathbf{x}$ given we have made the series of observations $\mathbf{q}$ is expressed in terms of the observation likelihood $P(\mathbf{q}|\mathbf{x})$, the prior distribution of states $P(\mathbf{x})$ and the marginal distribution of observations $P(\mathbf{q})$ as

$$P(\mathbf{x}|\mathbf{q}) = \frac{P(\mathbf{q}|\mathbf{x})P(\mathbf{x})}{P(\mathbf{q})}. \tag{5}$$

The observation likelihood, which, by using the independence property of the observations is written using the *a-priori* defined observation probabilities (the elements of our observation probability matrix $P(q_i|x_j) = O_{ij}$):

$$\begin{aligned} P(\mathbf{q}|\mathbf{x}) = P(q_N, q_{N-1}, \cdots, q_2, q_1, q_0 | x_N, x_{N-1}, \cdots, x_2, x_1, x_0) = \\ P(q_N|x_N)\cdots P(q_2|x_2)P(q_1|x_1)P(q_0|x_0) = \prod_{n=0}^{N} P(q_n|x_n). \end{aligned} \tag{6}$$

This is possible because we have assumed that each observation q_n is conditionally dependent only on the present model state x_n.

Likewise, the probability of observing the vector of state sequences $\mathbf{x}$ is given by considering the Markov property, using only the model transition probabilities, viz.:

$$\begin{aligned} P(\mathbf{x}) &= P(x_N, \cdots, x_2, x_1, x_0) = \\ &P(x_N|x_{N-1}, \cdots, x_2, x_1, x_0)P(x_{N-1}|x_{N-2}, \cdots, x_2, x_1, x_0)\cdots P(x_1|x_0)P(x_0) = \\ &P(x_N|x_{N-1})P(x_{N-1}|x_{N-2})\cdots P(x_1|x_0)P(x_0) = \prod_{n=1}^{N} P(x_n|x_{n-1})P(x_0) \end{aligned} \tag{7}$$

Note that (6) and (7) are given entirely in terms of the products of the conditional probabilities. The *a-posteriori* probability is written using Bayes' rule as

$$P(\mathbf{x}|\mathbf{q}) = \frac{\prod_{n=1}^{N} P(q_n|x_n)P(x_n|x_{n-1})P(x_0)}{\sum_{x_N \in S}\cdots\sum_{x_1 \in S}\sum_{x_0 \in S}\prod_{m=1}^{N} P(q_n|x_n)P(x_n|x_{n-1})P(x_0)}, \tag{8}$$

where S is the set of states that x_n can assume (the random choices that we must search through). The double product collapses into a single product because each observation is conditionally linked to a single Markov model state. The nested sum in the denominator of (8) is, in general, very difficult to compute. It turns out that we do not need to compute it because it ends up being a constant that depends only on the model parameters and does not change the maximization process. In fact, we can arrive at the same desired solution by maximizing the joint probability instead of the posterior:

$$\arg\max_{\mathbf{x}} P(\mathbf{x}|\mathbf{q}) = \arg\max_{\mathbf{x}} P(\mathbf{x}, \mathbf{q}) \tag{9}$$

which we can expand using the observational conditional dependency and Markov properties to

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \prod_{n=1}^{N} P(q_n|x_n)P(x_n|x_{n-1})P(x_0). \tag{10}$$

From a practical point of view (avoiding numerical underflow in computer or hardware implementations), it is often easier to maximize the log probability:

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} \sum_{n=1}^{N} (\log P(q_n|x_n) + \log P(x_n|x_{n-1})) + \log P(x_0). \tag{11}$$

The search can now be reduced to exploring a data set for **x** of size $N_S^2 \cdot N$ using the Viterbi algorithm, because now we do not need to explore all the permutations of **q** and **x**. We only explore the state changes that follow the Markov chain in Figure 2. The sum in (11) lends itself very well to iterative evaluation. At each time step, we explore the paths that lead to each state from the possible old states. For each possible new state, we keep only the path coming from the previous state that has the largest probability, that is:

$$V_x^i = \max(V_{x'}^{i-1} + logP(x|x') + logP(q|x^i), \tag{12}$$

for each time step. V_x^i at each time step corresponds to the maximum cumulative log-probability achieved so far in transitioning from state x' to x. By recursively maximizing the joint probability for each possible new state, we maximize the final posterior probability of the entire sequence of states. Figure 3 depicts a single step

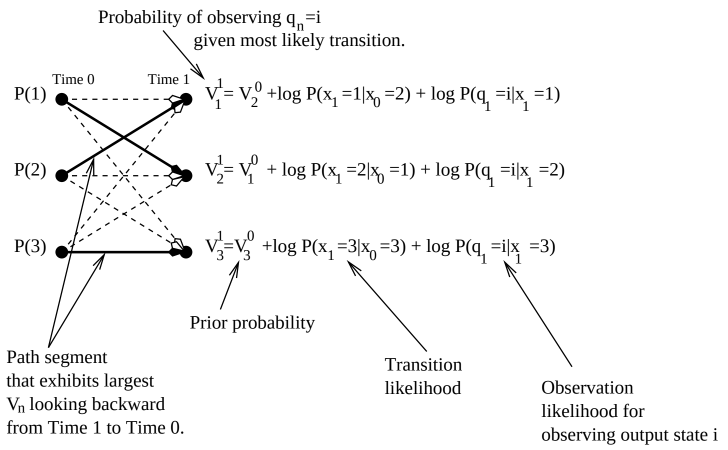

Probability of observing q_n=i
given most likely transition.
Time 0
Time 1
P(1)
P(2)
P(3)
V_1^1= V_2^0 +log P(x_1 =1|x_0 =2) + log P(q_1 =i|x_1 =1)
V_2^1= V_1^0 + log P(x_1 =2|x_0 =1) + log P(q_1 =i|x_1 =2)
V_3^1=V_3^0 +log P(x_1 =3|x_0 =3) + log P(q_1 =i|x_1 =3)
Prior probability
Path segment that exhibits largest V_n looking backward from Time 1 to Time 0.
Transition likelihood
Observation likelihood for observing output state i

Fig. 3. A single step in the Viterbi algorithm. Explore all possible paths between time n-1 and time n. Keep only the path segments that have the highest cumulative probability, V_n.

of the Viterbi algorithm. At any time step n we use the running sum of log-probabilities as the prior probability $V_{1,2,3}^{n-1}$. We then explore all possible transitions that lead to the state we are now sitting on, keeping only the transition that yields the highest new log-probabilities $V_{1,2,3}^{n}$. We keep a list of the most likely transitions (for the backtracking operation) and move on to the next step. If we put it all together, we build a series of paths

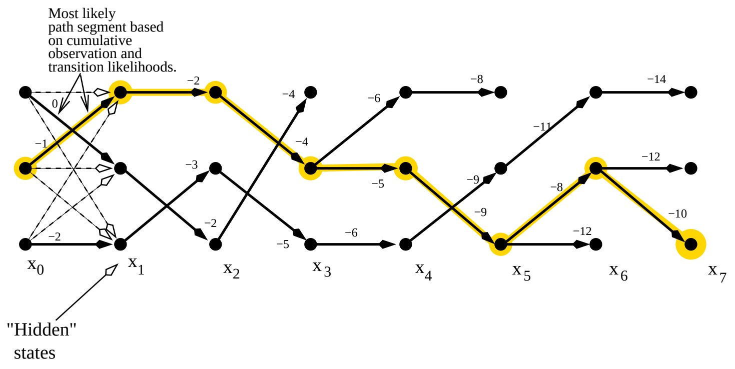


Fig. 4. The development of the full paths from beginning to end. Keep only the path with highest probability (highlighted in yellow). Trace backwards to get sequence of hidden states **x**.

from the beginning to the end (Figure 4). When we reach the end of the observation sequence, we select the final sequence that yields the highest log-probability and move backwards, following the transitions with highest probability until the beginning is reached. The path will trace out the most likely sequence of hidden states traversed by the Markov model. This is the path highlighted in yellow in Figure 4.

This operation is encapsulated in the following algorithm.

```
procedure VITERBI($T_{ij}, Q_{ij}, p_i, \mathbf{y}, N_S, N$)    ▷ transition, observation probabilities, priors, observations
    for $i = 0 : N_S - 1$ do
        $V(0, i) \leftarrow \log p_i + \log Q_{y_0 i}$    ▷ Initialize log-likelihood sum
        $Bs(0, i) \leftarrow i$    ▷ Initialize state memory (for backtrace)
    end for
                                                        ▷ Do forward iterations and find most probable state transitions
    for $t = 1 : N - 1$ do    ▷ Loop over number of observations
        for $i = 0 : N_S - 1$ do    ▷ Loop over possible new states
            $P_{max} \leftarrow -1e100$    ▷ Initialize max test variable
            for $j = 0 : N_S - 1$ do    ▷ Loop over old states
                $p \leftarrow V(i - 1, j) + \log T_{ij} + \log Q_{y_i j}$    ▷ Try new path cost
                if $p > P_{max}$ then
                    $P_{max} \leftarrow p$    ▷ Search for most prob. transition
                    $S_{max} \leftarrow j$
                end if
            end for
            $V(t, i) \leftarrow P_{max}$    ▷ Store accumulated max log prob.
            $Bs(t, i) \leftarrow S_{max}$    ▷ Store corresponding path step.
        end for
    end for
            ▷ Now do backtracking to recover most probable path    ▷ and corresponding probability.
    $Path(N-1) \leftarrow S_{max}$                  ▷ Final point (of most prob. path)
    **for** $t = N-2 : 0 : -1$ **do**                 ▷ backtrack through incremental paths
        $Path(t) \leftarrow Bs(t+1, Path(t+1))$
    **end for**
**end procedure**
```

On completion, P_{max} will contain a numerical value of the estimate of the joint probability of the observations and hidden states $\log P(\mathbf{q}, \mathbf{x})$. $Path(t)$ contains the most likely sequence of states given the Markov model transition probabilities, observation probabilities and the actual observation sequence.

II. Example calculations

A. Discrete event observations

Let us return to the weather model presented in the introduction. We run a series of "experiments" where we observe Mark's activities and try to deduce the sequence of weather types from the observed activities. Using our knowledge of the weather and Mark's habits, we come up with the table of observations and most likely weather sequence:

Observation	Naive prediction $\max_{x_i} P(q_i\|x_i)$	Viterbi sequence	Actual sequence	Error?
Walk	Sunny	Sunny	Sunny	N
Shop	Windy	Sunny	Sunny	N
Walk	Sunny	Sunny	Sunny	N
Clean	Rainy	Rainy	Rainy	N
Shop	Windy	Windy	Windy	N
Walk	Sunny	Sunny	Sunny	N
VisitGirlfriend	Rainy	Sunny	Rainy	Y
Walk	Sunny	Sunny	Windy	Y
Shop	Windy	Sunny	Rainy	Y
Shop	Windy	Sunny	Windy	Y
VisitGirlfriend	Rainy	Rainy	Rainy	N
VisitGirlfriend	Rainy	Windy	Windy	N
Walk	Sunny	Sunny	Sunny	N
⋮		⋮		⋮

The observations are generated by numerically synthesizing the progression of the Markov state "machine" (using a random number generator in a Monte-Carlo algorithm [10]) and using another Monte-Carlo scheme to generate the noisy observations that are consistent with our chosen probabilities. These observations are visible in the first column. The second column contains the "naive" guesses one might make based on a maximum observation likelihood (like Grandma making a snap guess of the weather state based only on Mark's behavior that day). The third column holds the more intelligent "guess;" the one based on the Viterbi scheme. Here, we take into account the weather model as well as Mark's past behavior. The actual weather state is in the fourth column, for comparison with the inferred states (so we can spot when the Viterbi algorithm returns an erroneous result). The incidence of errors is shown in the final column.

Running sequences of different length, we can establish the advantage conferred by using the Viterbi algorithm in place of a naive guess based on maximizing the observation likelihood $P(q_i|x_i)$. Table I summarizes the results. It is interesting to note that the naive guess based on the observation likelihood maintains an error rate

TABLE I

SUMMARY OF ERRORS IN WEATHER MODEL.

Sequence length	Number of errors, Viterbi	Number of errors, naive
10	2 (20%)	3 (30%)
20	3 (15%)	7 (35%)
50	12 (24%)	16 (32%)
100	18 (18%)	35 (35%)
200	53 (27%)	67 (34%)
500	116 (23%)	156 (33%)
1000	287 (24%)	348 (35%)
2000	562 (28%)	639 (32%)
5000	1419 (28%)	1654 (33%)

of about 33%. Using the Viterbi algorithm allows us to “beat the house” by a nearly 10% improvement over the naive decision. This occurs because we have added “evidence” to the observation data in the form of the weather model.

The reader may recall that the weather model loses practically all memory of previous states after three iterations. As a consequence, no new information is added to the present decision by observation data older than three time steps before. If we change the “weather” model such that memory of previous states is preserved over a long time period, the errors in the Viterbi estimates can be reduced. By using the weather transition matrix

$$T_{ij} = \begin{bmatrix} 0.1 & 0.1 & 1.0 \\ 0.1 & 0.8 & 0.0 \\ 0.8 & 0.1 & 0.0 \end{bmatrix}, \tag{13}$$

the steady-state is not reached until after approximately 30 time steps. Significant memory of previous states is maintained beyond 20 time steps, hence we expect performance to improve (because the model provides us with more knowledge about the likely state of the weather over a longer time). Table II shows a marked improvement in estimating the sequence of states. There are still some “bad guesses” generated by the Viterbi algorithm, but we have clearly reduced the error rate with the new weather model. There is always a finite probability of error that depends on the probability of error in observation as well as the uncertainty in the weather state model. The algorithm merely provides us with an estimate of the state sequence that may be optimal based on the information that we have on hand. We see for a weather state model with longer “memory”, the state estimation was significantly improved. This is an important property that has implications for error correction algorithms.

TABLE II

SUMMARY OF ERRORS IN WEATHER MODEL.

Sequence length	Number of errors, Viterbi	Number of errors, naive
10	0 (0%)	3 (30%)
20	2 (10%)	6 (30%)
50	3 (6%)	14 (28%)
100	10 (10%)	30 (30%)
200	29 (15%)	71 (35%)
500	87 (18%)	158 (32%)
1000	141 (14%)	315 (32%)
2000	307 (15%)	625 (32%)
5000	790 (16%)	1630 (32%)

B. Continuous event observation example: forward error correction in a noisy communication channel

The Viterbi algorithm finds extensive use in forward error correction schemes using convolutional coding [11] for reducing power requirements of reliably transmitting information over noisy channels. The signals involved assume continuous values (e.g. voltages), so it is natural to assign continuously varying probabilities to these observations. This is the basis of the "soft decision" Viterbi scheme.

1) The signal model: coding: Coding provides us with a way to introduce redundancy into a signal that we can exploit to counteract the deleterious effects of noise on the signal as it passes through a communication channel. For the illustrative example, we shall use a simple convolutional coder described in Figure 5.

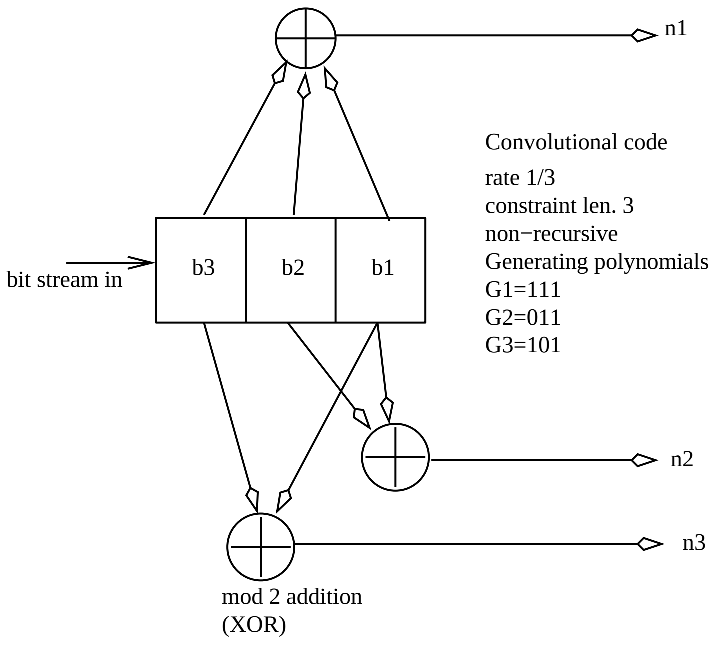


Fig. 5. System diagram of k=3, rate=1/3 convolutional encoder consisting of a 3 bit shift register and modulo 2 adders (XOR).

The message bits are shifted in on the left hand side and the modulo-2 adders produce the coded message. The message bits can assume a "1" or "0" state with equal probability. The state of the encoder can be conveniently be modelled as a Markov process with the state diagram as in Figure 6.

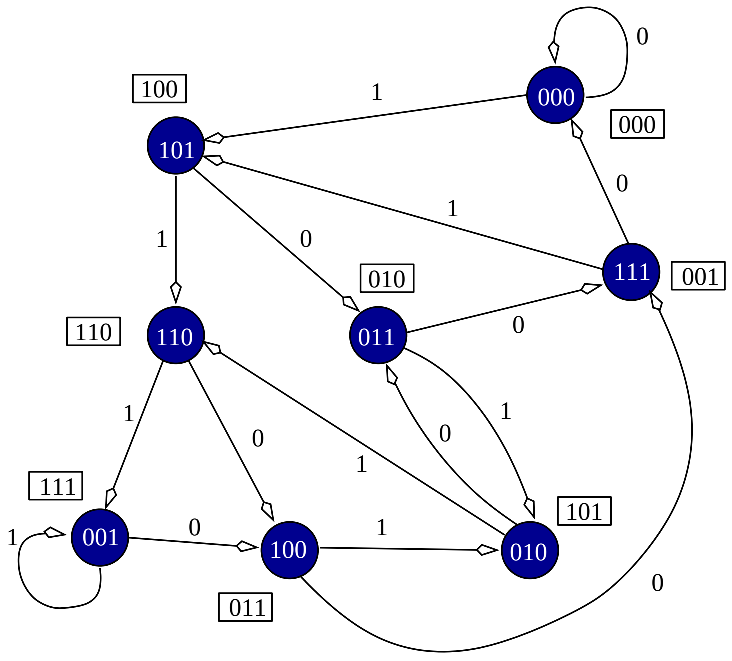


Fig. 6. The state diagram of transmit the encoder. Each permitted transition occurs with probability 0.5 since bits "1" and "0" are equally likely. The boxed bits represent the contents of the shift register. The bits in the circles are the mod-2 sum sequences that are to be transmitted through the noisy channel.

Since "0s" and "1s" are equally likely, the Markov transition probability matrix for the encoder states is

$$T_{ij} = \begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 \end{bmatrix} \tag{14}$$

As in the weather model, our task is to estimate the sequence of states of the encoder based on a set of noisy observations. The states of the encoder are "hidden" because they lie at the transmitter end and we can only "observe" them at the receiver through the noisy channel.

An important feature of the signal model is that only certain transitions are permitted. Others are strictly forbidden (i.e. the transition probability vanishes). As in a Sudoku puzzle, only certain sequences will fit when we try to decode the original signal given that we are using only corrupted (partial) information, which the receiver provides to us. This property allows us to fill in what we believe to be the most likely symbol sequence given the corrupted received information.

2) The observation model: a noisy communication channel: Listening at a receiver, we can have no direct knowledge of the encoder state. However, we know that a channel, which produces signals contaminated with

Gaussian noise provides us with an observation likelihood for any one bit observation as

$$p(q_i|x_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(q_i - x_i)^2}{2\sigma^2}\right), \tag{15}$$

where q_i is the observed (random) variable, x_i is the possible transmitted bit ("1" = 1V and "0" = -1V, for example). This is a model of BPSK modulation. The variable σ is the standard deviation that is a function of the noise power in the received signal. Specifically,

$$\sigma = \sqrt{\frac{N_0 B}{2}}, \tag{16}$$

where N_0 is the Gaussian noise spectral density (a constant, approximated by $kT(1+F)R_n$; k=Boltzmann's constant, T=290K, F=receiver noise factor, R_n= equivalent noise resistance and B is the channel bandwidth).

If each observed symbol consists of groups of $N = 3$ bits, the observation likelihood can be written as

$$p(\mathbf{q}|\mathbf{x}) = \frac{1}{\sigma^N (2\pi)^{N/2}} \prod_{n=1}^{N} \exp\left(-\frac{(q_n - x_n)^2}{2\sigma^2}\right). \tag{17}$$

Of course, in (17), we assume that each observation is independent of every other observation. This simplifies the analysis. In real systems, however, effects like filtering, channel fading and inter-symbol interference will produce correlations between observations that complicate detection. We will not consider these effects in this essay.

Each symbol in our coded signal consists of three bits, so we can write the log-likelihood as

$$\log p(\mathbf{q}|\mathbf{x}) = -N \log\left(\sigma\sqrt{2\pi}\right) - \sum_{n=1}^{N} \frac{(q_n - x_n)^2}{2\sigma^2} \tag{18}$$

We use a message bit stream that satisfies the statistical properties of the Markov model: a pseudorandom noise sequence

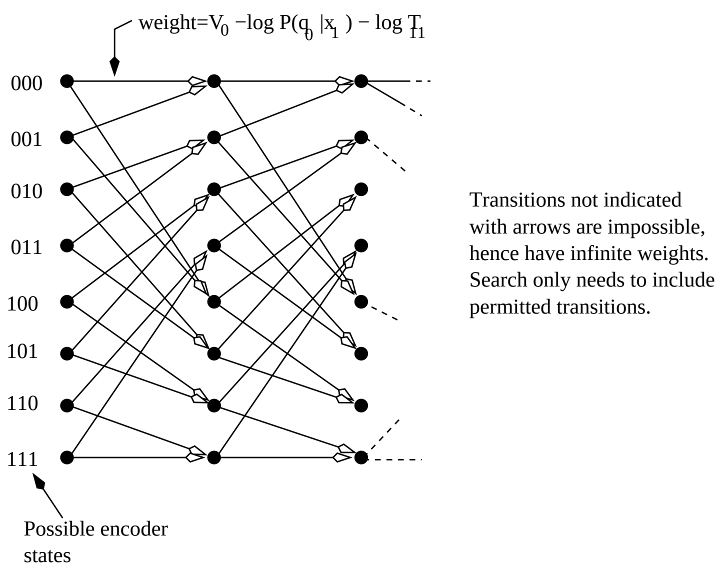
```
Bits 1-32:  10000110001010011110100011100100
Bits 33-64: 10110111011001101010111111000000
```

At each symbol period, we calculate the recursive sum of the log-probabilities for each possible encoder state

$$V_s^n = V_{s'}^{n-1} + \log T_{ss'} - \sum_{n=1}^{3} \frac{|\mathbf{q} - \mathbf{x}_s|^2}{2\sigma^2}, \tag{19}$$

where $\mathbf{q}$ is the vector containing the observation voltages of the three bits in the symbol, $\mathbf{x}_s$ is the vector containing the encoder state s we are exploring. Where $T_{ss'}$ is zero (a forbidden state transition), we ignore that contribution because the log-probability approaches negative infinity and will never contribute to a path. Hence, we only need to explore two transitions at each new state, depicted in Figure 7. This provides us with a savings in computation.

We encode our message stream using the simple encoder and "contaminate" the resulting bitstream (now 3 times the length) with additive noise. Figure 8 depicts a segment of the bit stream and the corresponding symbol sequence. Remember that the symbol bit rate is three times the data bit rate, because we transmit three code-word (symbol) bits for each data bit. Figure 9 shows the symbol stream corrupted by additive Gaussian white noise. As we will see, the Viterbi algorithm is highly successful at "coaxing" the original symbol sequence out of this very noisy representation.



weight=V_0 −log P(q_0 |x_1) − log T_{11}
000
001
010
011
100
101
110
111
Possible encoder states
Transitions not indicated with arrows are impossible, hence have infinite weights. Search only needs to include permitted transitions.

Fig. 7. The reduced search that excludes forbidden encoder states, thereby reducing computational needs.

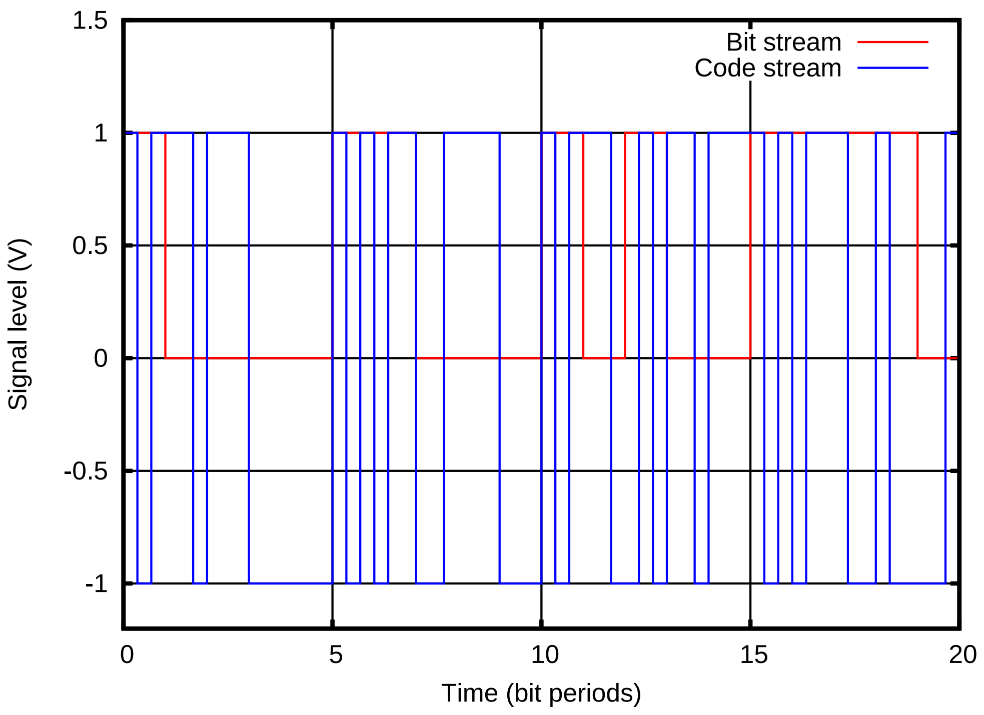


Fig. 8. Illustration of a segment of the bit stream and code symbol stream. Notice that the bits in the code stream occur at three times the rate of the data bits.

In our simulation example, we keep the signal level constant and vary the noise such that the signal "power" ranges from 9 dB below the mean noise power to 1 dB above the noise. We then count the errors in received symbols and plot this result against signal-to-noise ratio to generate symbol error rate curves.

In Figure 10 we can readily discern from the trend that the probability that a symbol is in error appears to follow a smooth relationship. In fact, it is straightforward to show that the probability of symbol error (i.e. that

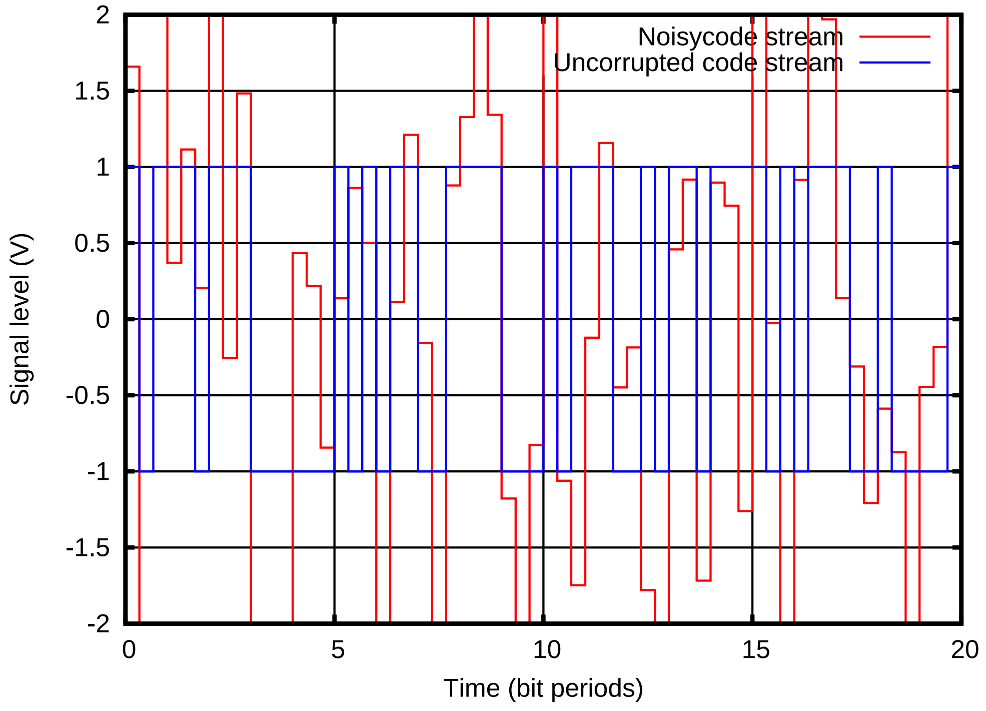


Fig. 9. The code symbol stream and its noisy version (corrupted with additive Gaussian white noise).

at least 1 bit in a symbol is wrong) is theoretically

$$P_{se} = 1 - (1 - P_e)^3, \tag{20}$$

where the probability of error in a single bit is

$$P_e = \frac{1}{2}\,\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right). \tag{21}$$

The ratio E_b/N_0 is a measure of the signal energy received per bit over the noise spectral density and $\mathrm{erfc}()$ is the complementary error function given by

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-u^2/2}\, du. \tag{22}$$

The energy/spectral density ratio is related to the signal to noise ratio SNR and the bit time T_b by

$$\frac{E_b}{N_0} = SNR \cdot T_b. \tag{23}$$

The difference between the observed error rates for the Viterbi decoded stream and the "naive" maximum-observation likelihood (ML) symbols is striking. Without using the Viterbi algorithm, the symbol error rate is above 10% for E_b/N_0=1dB. With Viterbi decoding, this error rate is achieved for E_b/N_0=-6dB or $E_s/N_0 \approx -1$dB. (Symbol energy E_s is 4.88 dB higher than symbol bit energy. This is because there are three bits per symbol.) This is an astonishing gain in performance using this very simple coding scheme. More advanced coding schemes allow for even lower error rates, approaching the theoretical limits of capacity in a noisy channel [11]. The practical applications in communications are ubiquitous: cellular telephony, deep space communication links, satellite communications, disk drive data reads, etc.

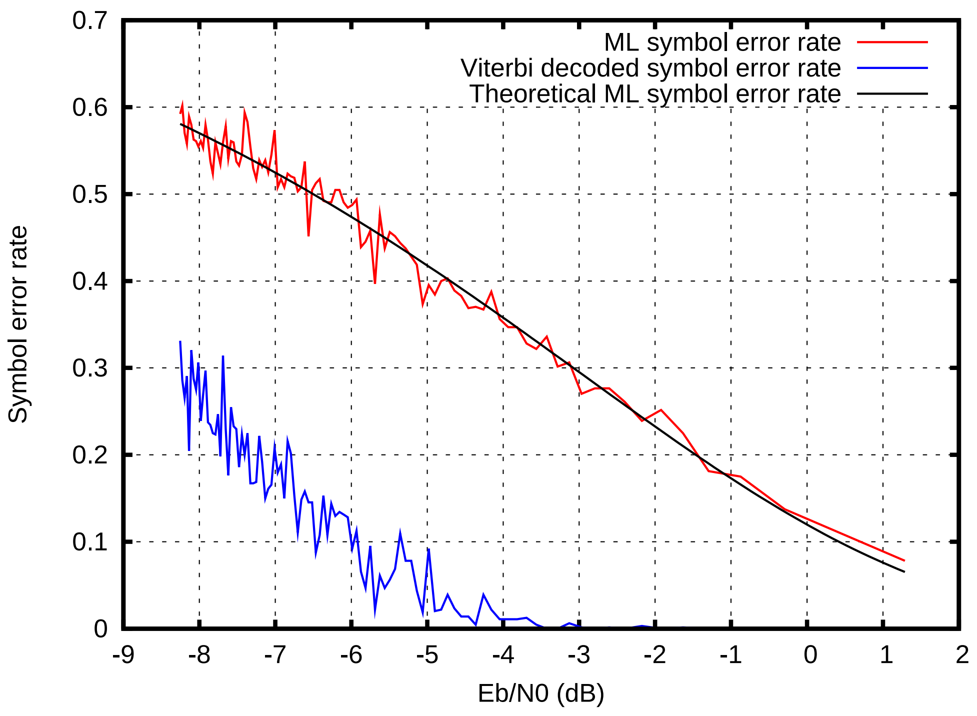


Fig. 10. Detected symbol errors using 1) Maximum-likelihood detection (based only on observation likelihood) and 2) Viterbi decoded encoder state sequence (includes encoder Markov model to generate most probable sequence of states). Horizontal axis is ratio of bit energy to noise spectral density in dB. Add 4.8dB to get symbol energy/noise spectral density ratio (there are three bits per symbol).

For our case, the Shannon limit on maximum possible transmission rate $R_{max} = B\log_2(1 + E_s/N_0)$ would be near $E_s/N_0 = -2.3$dB to achieve less than 10^{-5} symbol error rate. With our simple code, we can achieve this only with E_s/N_0 well above 0dB. Longer codes will bring us closer to the Shannon limit. Modern codes (like Turbo codes) can bring us within a fraction of a dB of this theoretical limit [12].

The reader may have noticed that the bits of each symbol must be aligned properly, i.e. the symbol boundaries must be known to the decoder. In a practical sense, this means we need a bit clock at the receiver that is well synchronized to the transmitter bit clock. If this is not the case, decoding will be impossible. In practical systems, this is accomplished by either transmitting a bit clock signal that provides a reference to a receiver phase locked loop or by transmitting a known bit sequence in a frame preamble that provides “training” to a receiver bit clock so the bit and frame timing can be properly synchronized.

III. Summary

The Hidden Markov model provides a convenient platform for simulating probabilistic events that are described by a time series. Whether these events form a set of non-numeric states (like our weather model) or are described in terms of numeric values (like our received signal voltage), as long as we can describe the events in terms of observation and transition probabilities, a Viterbi scheme can be devised to find the most probable sequence of states.

Qualitatively speaking, the examples presented here show how the Viterbi algorithm is, in effect, an “evidence gathering” machine. The transition matrix (or encoder state model) forms a set or prior knowledge that is used to improve on the data gathered during the observations. Then, all the observations are considered as a whole to decide the best fit of the path with the observed data. This is a powerful method based on a class of algorithms that perform what can be called “Bayesian reasoning.” [8]. These techniques reach far beyond signal processing into areas of artificial intelligence, machine learning, molecular biology, law and the theory of knowledge [9].

We hope that you, the reader, have come away, after reading this article, with a different perspective on one of the most important algorithms of the last 50 years.

REFERENCES

[1] A. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *IEEE Trans. Information Theory*, vol. 13, no. 2, Apr. 1967, pp. 260-269.

[2] A. Viterbi, “Convolutional codes and their performance in communication systems,” *IEEE Trans. Comm. Tech.*, vol. 19, no. 5, Oct. 1971, pp751-772.

[3] J. Omura, “On the Viterbi decoding algorithm,” *IEEE Trans. Information Theory*, vol. 15, no. 1, Jan. 1969, pp. 177-179.

[4] http://en.wikipedia.org/wiki/Hidden_Markov_model

[5] A. M. Fraser, *Hidden Markov Models and Dynamical Systems*, SIAM, 2008.

[6] S. Meyn and R. Tweedie, *Markov Chains and Stochastic Stability*, Cambridge University Press, 2009.

[7] http://en.wikipedia.org/wiki/Random_walk

[8] D. Barber, *Bayesian Reasoning and Machine Learning*, Cambridge University Press, 2012.

[9] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Pearson, 2010.

[10] W. Press, S. Teukolsky, W. Vetterling and B. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, 2002.

[11] W. Huffman and V. Pless, *Fundamentals of Error Correcting Codes*, Cambridge University Press, 2003.

[12] C. Berrou, A. Glavieux and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” *Encyclopedia of Telecommunications*, Wiley, Apr. 15, 2003, pp. 1064-1070. Available online at: http://www.cp.eng.chula.ac.th/ piak/teaching/ice/intro2007/com-theory/turbocode.pdf