F

Viterbi algorithm

F.1. Introduction

The Viterbi algorithm (VA) was originally proposed in 1967 for decoding convolutional codes. Shortly after its discovery, it was observed that the VA was based on the principles of *dynamic programming*, a general technique for solving extremum (that is, maximization or minimization) problems.

Our application of the VA consists of finding, among the paths traversing a *trellis* from left to right, the one with the maximum or minimum *metric*. Specifically, we define a trellis as a diagram representing all the allowable trajectories of a Markov chain with N_σ states $\{S_i\}_{i=1}^{N_\sigma}$ from time $k = 0$ to time $k = K$. The trellis begin and ends at two known states, and there is a one-to-one correspondence between the sequences of $K + 1$ states and the paths through the trellis. Fig. F.1 shows an example of a four-state trellis with $K = 6$. A *branch metric* is associated with each branch (or edge) of the trellis, in the form of a label. The branch metrics are additive, i.e., the metric associated with a pair of adjoining branches is the sum of the two metrics. Consequently, the total metric associated with a path traversing the whole trellis from left to right is the sum of the labels of the branches forming the path. The problem here is to find the path traversing the trellis with the maximum (or minimum) total metric (the choice between maximum or minimum depends of course on the problem being solved). Formally, if σ_k denotes the state at time k, taking values $\{S_i\}_{i=1}^{N_\sigma}$, and $m(\sigma_k, \sigma_{k+1})$ denotes the metric associated with the branch emanating from node σ_k and joining node σ_{k+1}, we want to maximize (or minimize) the function

$$\lambda(\sigma_0, \sigma_1, \ldots, \sigma_K) \triangleq \sum_{k=0}^{K-1} m(\sigma_k, \sigma_{k+1}) \tag{F.1}$$

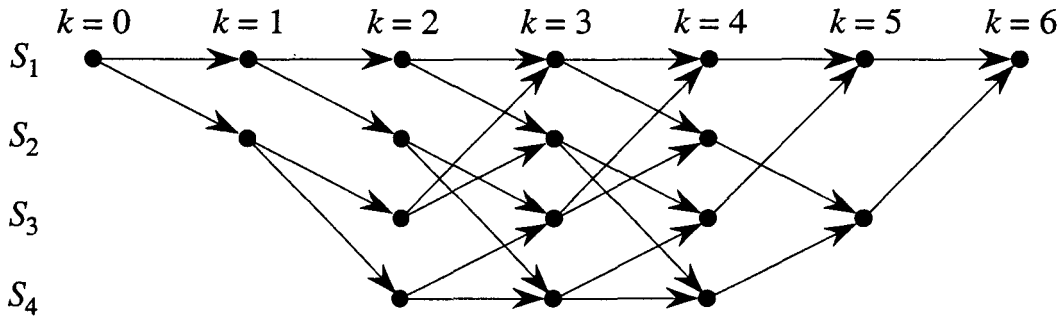

Figure F.1: Trellis of a four-state Markov process.

over all the possible choices of the state sequences $(\sigma_0, \sigma_1, \ldots, \sigma_K)$ compatible with the trellis structure.

Clearly, the problem above could be solved in principle by a brute-force approach, consisting of evaluating all the possible values of the function $\lambda(\cdot)$ in (F.1), and choosing the largest (or the smallest). However, this algorithm would suffer from two main drawbacks, viz.,

Complexity: The number of computations required, and the storage needed, grow exponentially with the length K of the sequence.

Delay: If the branch labels are computed sequentially from time $k = 0$ to time $k = K$ (as it occurs in the applications considered in this book), then the decision on the best path must be deferred until the whole sequence of labels is computed, which entails a delay K.

As we shall see, the Viterbi algorithm solves the maximization problem without suffering from exponential complexity: actually, its computational complexity (and storage requirements) grow only *linearly* with K. Moreover, the *truncated* version of the VA has a delay which may be much smaller than K, at the price of a minor loss of optimality.

We start our description of the VA with the illustration of its key step, commonly called ACS (for Add, Compare, and Select). Consider Fig. F.2, where (and from now on) we shall consider a *maximum* problem. It shows the trellis states at time k (denoted σ_k) and at time $k + 1$ (denoted σ_{k+1}). The branches joining pairs of paths are labeled by the corresponding branch metrics, while the states σ_k are labeled by the *accumulated state metrics*, to be defined soon. The ACS step consists of the following: For each state σ_{k+1}, examine the branches

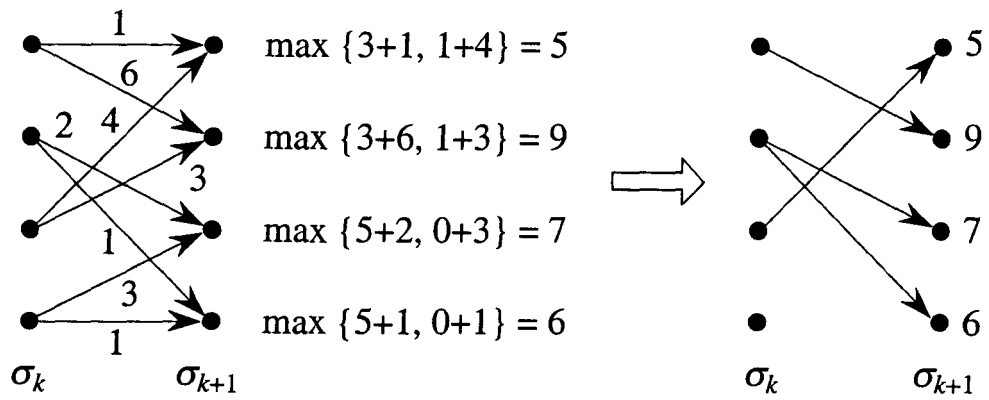

Figure F.2: The ACS step of Viterbi algorithm.

stemming from states σ_k and leading to it (there are two such branches in Fig. F.2). For these branches, ADD the metric accumulated at the state from which it stems to the metric of the branch itself. Then COMPARE the results of these sums, and SELECT the branch associated with the maximum value (and consequently discard, for each state, all the other branches entering it; and if two or more of the quantities being compared are equal, choose one at random). The maximum value is associated with the state, and forms its accumulated metric. This value is retained only for the next ACS step and is then discarded.

The VA consists of repeating the ACS step from the starting state to the ending state of the trellis. After each ACS step the VA retains, for each state, one value of accumulated metric and one path, usually called the *survivor* corresponding to the state. Thus, at any time k we are left, for each σ_k, with a single survivor path traversing the trellis from the initial state to σ_k, and with one value of accumulated metric. This survivor path is the maximum-metric path to the corresponding state. After K ACS steps, at the termination of the trellis we obtain a single K-branch path and a single accumulated metric. These are the maximum-metric path and the maximum-metric value, respectively. Fig. F.3 illustrates the determination of a maximum-metric path through a four-state trellis via the VA.

To prove the optimality of the VA it suffices to observe the following. Assume that the optimum path passes through state S_i (say) at time k. Then, *its first k branches must be the same as for the survivor corresponding to S_i*. In fact, if they did not, the optimum path would begin with a path passing through S_i and having a metric lower than the survivor of S_i, which is a contradiction. In other words, no path discarded in favor of a survivor can provide a contribution to the total metric larger than the survivors.

The computational complexity of the VA is the same at each time instant (we disregard initial and final transients). Hence, it grows only linearly with K. More specifically, the VA requires N_σ storage locations, one for each state, with each location storing an accumulated metric and a surviving path. In terms of the number of computations, at each time instant the VA must make Q additions, where Q is the number of transitions in a trellis section (for example, $Q = 8$ in Fig. F.1), and N_σ comparisons. Thus, the amount of storage is proportional to the number of states, and the amount of computation per time unit is proportional to the number of transitions.

F.1.1. The truncated Viterbi algorithm

The VA as described above leaves the delay problem unsolved. In fact, the algorithm cannot reach a decision about the maximum-metric sequence before time K. On the other hand, it is obvious that a decision about the best sequence cannot be reached before scanning all the states from $k = 0$ to $k = K$, so that reducing the delay would necessarily entail a loss of optimality of the algorithm.

When the delay of K time instants cannot be tolerated, the *truncated* Viterbi algorithm may be used. This consists of forcing decisions at stage k on all paths prior to stage $k - D$, for some *truncation depth* (or *decision depth*) D. The approach consists of comparing the partial path metrics for the paths at stage k, and noting which one is the largest. The branch chosen at this time is the one belonging to this path at time $k - D$. Thus, after a latency of D time instants, the truncated VA outputs one branch at a time. Doing this entails also a reduction in the storage needed, since only the last D branches of the survivor paths must be kept in memory. The loss of optimality is reduced when D is increased, because when D is large there is a high probability that all the surviving paths leading to any node have an initial part in common: so this initial path will be a part of the optimum one, and we say that a *merge* has taken place.

F.1.2. An example of application

Here we describe a simple example of application of the Viterbi algorithm to a decoding problem. Assume that a symbol sequence $\mathbf{x}$, consisting of K equally likely binary symbols taking values 0 or 1, is transmitted over a memoryless channel. Assume also that the symbols, rather than being independent, are correlated with each other, and that their correlation can be described in the form of a trellis as defined before. Specifically, all the admissible symbol sequences are in one-to-one correspondence with the the paths traversing the trellis from $k = 0$ to $k = K$, with one symbol associated with each branch. This occurs for example when the symbol sequence can be thought of as the output of a finite-state

machine driven by an independent, identically distributed sequence of random variables, so that the sequence of states forms a Markov chain.

Let y_k denote the components of the received signal sequence $\mathbf{y}$, and $p(y_k \mid x_k)$ the probability density function of the received samples given that x_k was transmitted. Maximum-likelihood detection of the transmitted sequence consists of maximizing the conditional pdf

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{k=1}^{K} p(y_k \mid x_k) \tag{F.2}$$

over all the admissible sequences $\mathbf{x}$. Here the assumption of a memoryless channel has been used to factorize the pdf.

By taking the logarithm of (F.2), we obtain the additive form

$$\ln p(\mathbf{y} \mid \mathbf{x}) = \sum_{k=1}^{K} \ln p(y_k \mid x_k) \tag{F.3}$$

We can then use $\ln p(y_k \mid x_k)$ as the metric that labels the trellis branches associated at time k with the symbol x_k when the observed channel output is y_k. Maximization of the sum (F.3) leads to choosing the most likely sequence of transmitted symbols. As a special case, for the additive white Gaussian noise channel the above leads to a problem equivalent to the minimization of a Euclidean distance, or, for equal-energy signals, to the maximization of a scalar product.

F.2. Maximum a posteriori detection. The BCJR algorithm

It is known (Section 2.6) that maximum-likelihood detection minimizes the probability that the whole detected sequence be in error. Assume instead that in the example of Section F.1.2 we are interested in minimizing the symbol error probability for the detected symbols (motivation for this choice is provided in Section 11.3). To do this, for each k we should choose the value of x_k that leads to the greater between the two quantities (a posteriori probabilities) $P(x_k = 0 \mid \mathbf{y})$ and $P(x_k = 1 \mid \mathbf{y})$. This is tantamount to comparing with a unit threshold the a posteriori probability ratio

$$\Lambda_k = \frac{P(x_k = 1 \mid \mathbf{y})}{P(x_k = 0 \mid \mathbf{y})} \tag{F.4}$$

Now, observe that the transmitted symbol x_k is associated with one or more branches of the trellis stage at time k, and that each one of these branches can

be characterized by the pair of states, say (σ_k, σ_{k+1}), that it joins. Thus, we can write

$$\Lambda_k = \frac{\sum_{(\sigma_k,\sigma_{k+1}):x_k=1} p(\mathbf{y}, \sigma_k, \sigma_{k+1})}{\sum_{(\sigma_k,\sigma_{k+1}):x_k=0} p(\mathbf{y}, \sigma_k, \sigma_{k+1})} \tag{F.5}$$

where the two summations are over those pairs of states for which $x_k = 1$ and $x_k = 0$, respectively, and the conditional probabilities of (F.4) are replaced by joint probabilities after using Bayes' rule and cancelling out the pdf of $\mathbf{y}$, common to numerator and denominator.

We proceed now to the computation of the pdf $p(\mathbf{y}, \sigma_k, \sigma_{k+1})$. By defining $\mathbf{y}_k^-$, the components of the received vector before time k, and $\mathbf{y}_k^+$, the components of the received vector after time k, we can write

$$\mathbf{y} = (\mathbf{y}_k^-, y_k, \mathbf{y}_k^+)$$

and consequently

$$\begin{aligned} p(\mathbf{y}, \sigma_k, \sigma_{k+1}) &= p(\mathbf{y}_k^-, y_k, \mathbf{y}_k^+, \sigma_k, \sigma_{k+1}) \\ &= p(\mathbf{y}_k^-, y_k, \sigma_k, \sigma_{k+1})\, p(\mathbf{y}_k^+ \mid \mathbf{y}_k^-, y_k, \sigma_k, \sigma_{k+1}) \\ &= p(\mathbf{y}_k^-, \sigma_k)\, p(y_k, \sigma_{k+1} \mid \mathbf{y}_k^-, \sigma_k) p(\mathbf{y}_k^+ \mid \mathbf{y}_k^-, y_k, \sigma_k, \sigma_{k+1}) \end{aligned}$$

Now, observe that due to the dependences among observed variables and trellis states, reflected by the trellis structure or, equivalently, by the Markov-chain property of the trellis states, $\mathbf{y}_k^+$ depends on σ_k, σ_{k+1}, $\mathbf{y}_k^-$, and y_k only through σ_{k+1}, and, similarly, the pair y_k, σ_{k+1} depends on σ_k, $\mathbf{y}_k^-$ only through σ_k. Thus, by defining the functions

$$\alpha_k(\sigma_k) \triangleq p(\mathbf{y}_k^-, \sigma_k) \tag{F.6}$$

$$\beta_{k+1}(\sigma_{k+1}) \triangleq p(\mathbf{y}_k^+ \mid \sigma_{k+1}) \tag{F.7}$$

$$\gamma_{k,k+1}(\sigma_k, \sigma_{k+1}) \triangleq p(y_k, \sigma_{k+1} \mid \sigma_k) = p(y_k \mid \sigma_k, \sigma_{k+1})\, p(\sigma_{k+1} \mid \sigma_k) \tag{F.8}$$

we may write

$$p(\mathbf{y}, \sigma_k, \sigma_{k+1}) = \alpha_k(\sigma_k)\, \gamma_{k,k+1}(\sigma_k, \sigma_{k+1})\, \beta_{k+1}(\sigma_{k+1}) \tag{F.9}$$

In conclusion, the a posteriori probability ratio (F.5) can be rewritten in the form

$$\Lambda_k = \frac{\sum_{\sigma_k,\sigma_{k+1}:x_k=1} \alpha_k(\sigma_k)\gamma_{k,k+1}(\sigma_k, \sigma_{k+1})\beta_{k+1}(\sigma_{k+1})}{\sum_{\sigma_k,\sigma_{k+1}:x_k=0} \alpha_k(\sigma_k)\, \gamma_{k,k+1}(\sigma_k, \sigma_{k+1})\, \beta_{k+1}(\sigma_{k+1})} \tag{F.10}$$

To complete our calculations, we now describe how the functions $\alpha_k(\sigma_k)$ and $\beta_{k+1}(\sigma_{k+1})$ can be evaluated recursively. We note the forward recursion

$$\begin{aligned}
\alpha_{k+1}(\sigma_{k+1}) &= p(\mathbf{y}_{k+1}^-, \sigma_{k+1}) \\
&= p(\mathbf{y}_k^-, y_k, \sigma_{k+1}) \\
&= \sum_{\sigma_k} p(\mathbf{y}_k^-, y_k, \sigma_k, \sigma_{k+1}) \\
&= \sum_{\sigma_k} p(\mathbf{y}_k^-, \sigma_k) p(y_k, \sigma_{k+1} \mid \sigma_k) \\
&= \sum_{\sigma_k} \alpha_k(\sigma_k)\, \gamma_{k,k+1}(\sigma_k, \sigma_{k+1})
\end{aligned}$$

with the initial condition $\alpha_0(s_1) = 1$ (s_1 denotes the initial state of the trellis). Similarly, we note the backward recursion

$$\begin{aligned}
\beta_k(\sigma_k) &= p(\mathbf{y}_{k-1}^+ \mid \sigma_k) \\
&= \sum_{\sigma_{k+1}} p(y_k, \mathbf{y}_k^+, \sigma_{k+1} \mid \sigma_k) \\
&= \sum_{\sigma_{k+1}} p(y_k, \sigma_{k+1} \mid \sigma_k) p(\mathbf{y}_k^+ \mid \sigma_{k+1}) \\
&= \sum_{\sigma_{k+1}} \gamma_{k,k+1}(\sigma_k, \sigma_{k+1})\, \beta_{k+1}(\sigma_{k+1})
\end{aligned}$$

with the final value $\beta_K(s_K) = 1$.

The combination of the latter two recursions with (F.10) forms the BCJR algorithm, named after the authors who first derived it (Bahl, Cocke, Jelinek, and Raviv, 1974). This algorithm, that was derived here with the aim of maximizing the a posteriori probabilities of the symbols, is used in Section 11.3 to the purpose of computing a posteriori probabilities.

Roughly speaking, we can state that the complexity of the BCJR algorithm is about three times that of Viterbi algorithm. A truncated version of this algorithm (the "sliding-window" algorithm) is described in Section 11.3.