Viterbi Decoding of convolutional codes

Malyala Preethi Sravani
Raavi Gupta

Table of Contents

Overview

Understanding the problem

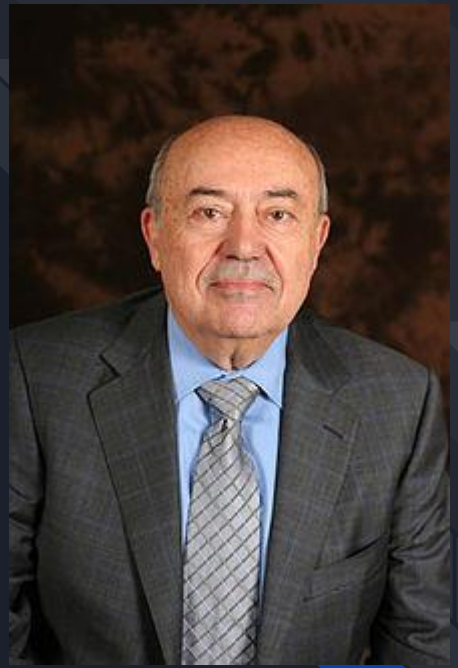
An informal example

The stages of Viterbi

The complete algorithm

Analysis of the decoding algorithm

Application of Viterbi Algorithm

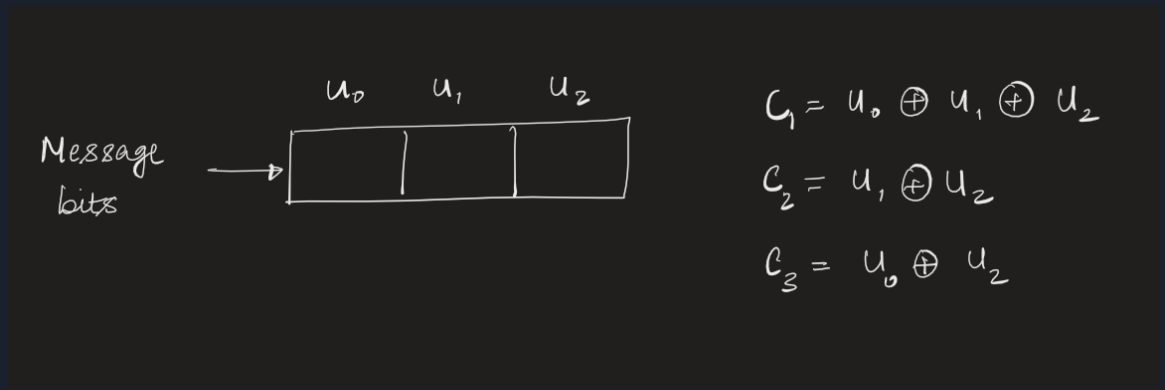


Overview

The Viterbi Algorithm is a dynamic programming algorithm for obtaining the maximum *a-posteriori* probability estimate of the most likely sequence of hidden states. These states denote all possibilities of $(n-1)$ values on which the present value of the system is dependent. The system is then called an n^{th} *order Markov process.*

In this presentation, we discuss a way to find the most probable order in which these states are traversed by the system. We start with an example, then state the algorithm formally, and finally discuss its complexity, and state some optimisations. We finally end with the applications of the algorithm.

Decoding of Convolutional codes - Why Viterbi?

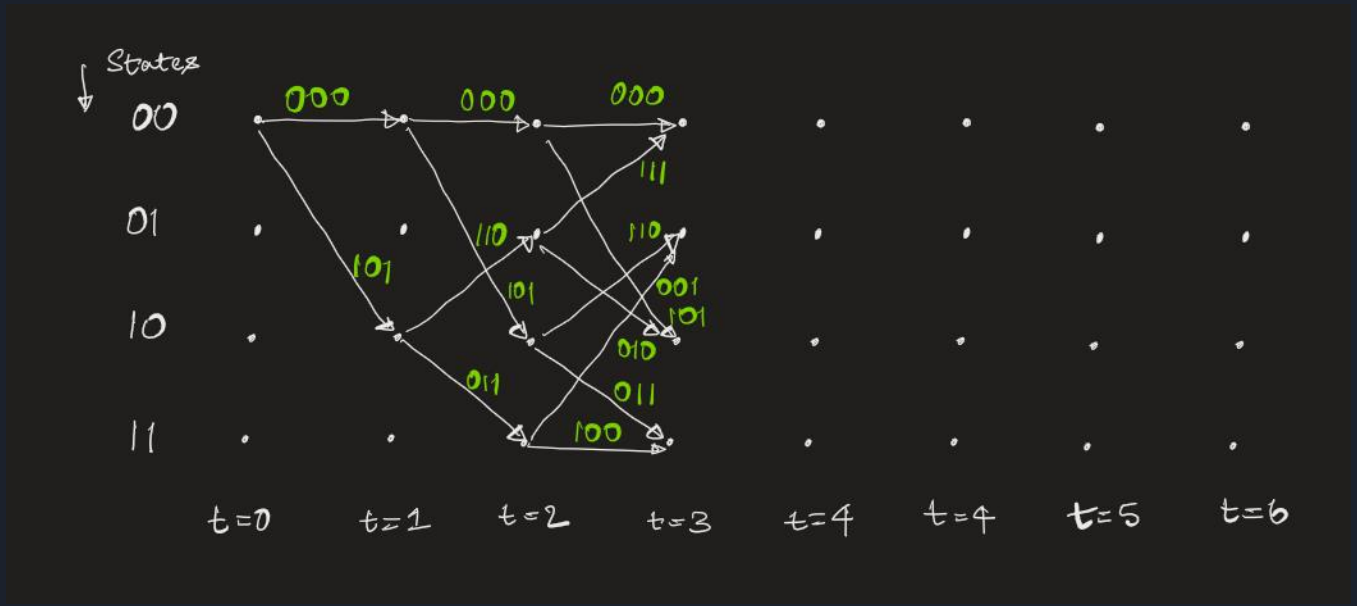


Rate = $\frac{1}{3}$

Let (u_1, u_2) denote the trellis states (trellis? Introduced in the next slide).

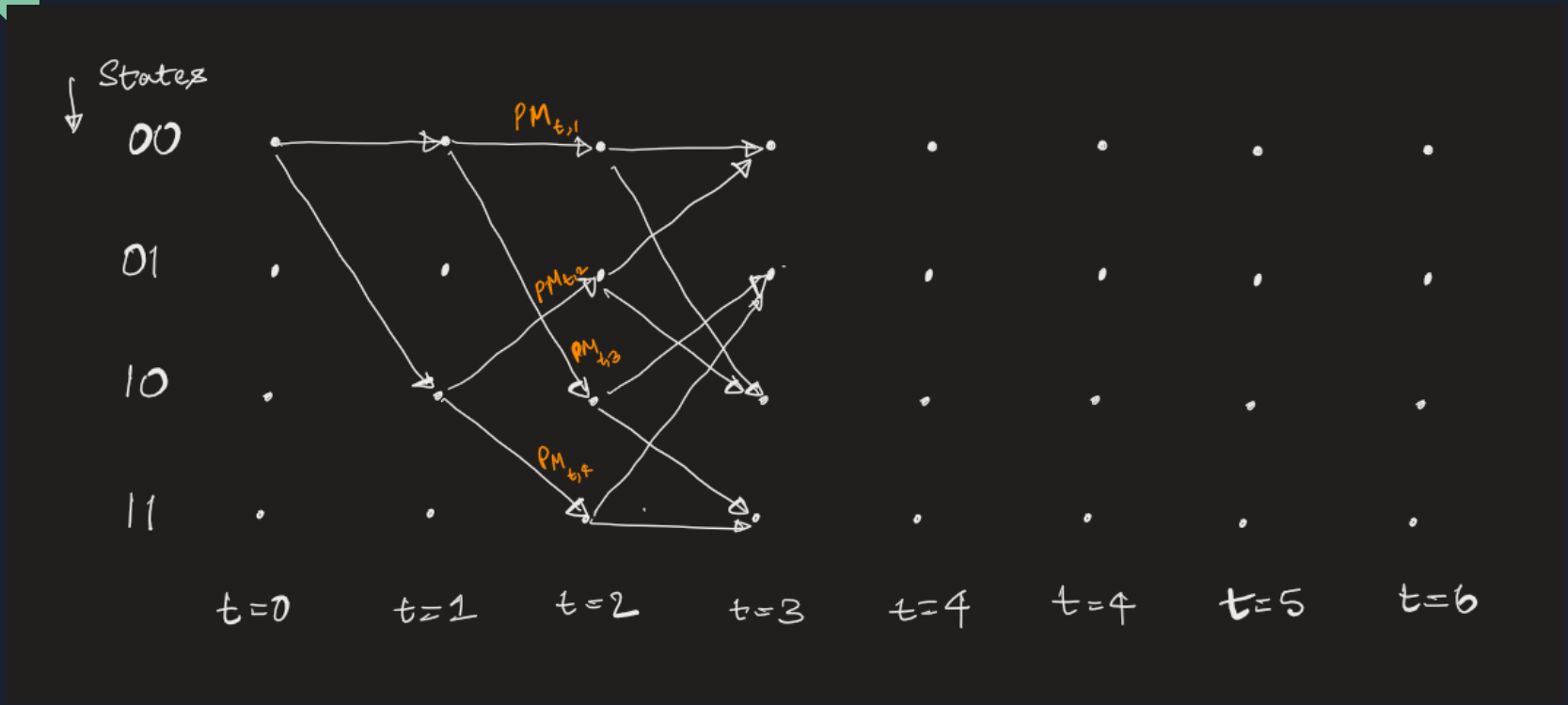
Due to convolution, the input data bits get spread out over multiple time slots, therefore it is not optimal to use just one time slot for decoding. That's where Viterbi decoding comes in!

The Progression of Trellis Diagram



Assuming the initial state is (0, 0), the next message bit could be either 0 or a 1. That is, the next state can be either (0, 0) or (1, 0). Based on this, the coded message bits can be either [0, 0, 0] or [1, 0, 1] respectively.

Path Metric - How do we actually decode?



First some definitions:

Path Metric: The distance between the survivor path (survivor path?) and the sequence of noisy symbols is called the path metric for that state.

Branch Metric: The branch metric is the distance between the received noisy symbol, y_n, and the ideal noiseless output symbol of that transition.

Let $M_{i,j}$ be the path metric for state j at recursion n, and $\{\mathrm{i}\}$ as the set of states that have transitions to state j, then the most likely path coming into state j at recursion n is the path that has minimum metric,

$$M_{j,n} = \min\{M_{i,n-1} + B_{i,j,n}\}$$

Where $B_{i,j,n}$ is the branch metric for the transition from state i to state j at recursion n.

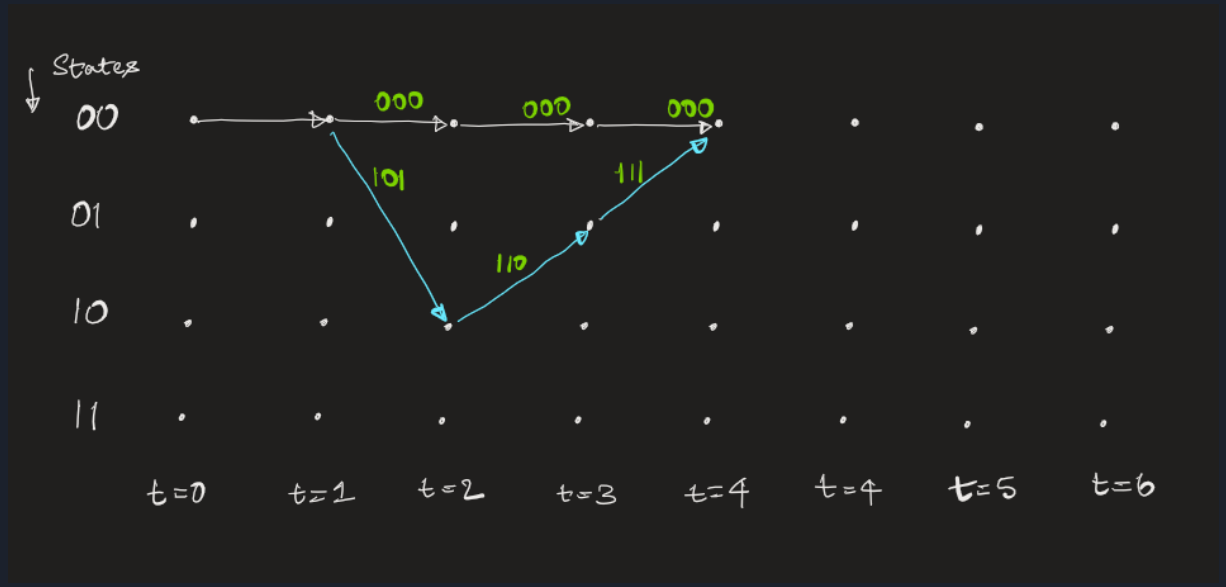
But how is a branch metric calculated?

As defined earlier, branch metric is the “distance” between the received noisy symbol and the noiseless output.
In Viterbi decoder, we can use either Hamming distance or Euclidean distance.

Hard decision decoding: For hard decision decoding, each noisy symbol in the received sequence of noisy symbols is quantized to one bit before it is compared to the corresponding coded bit in the coded bit sequence of the given transition. The number of bits in which the two sequences differ is called the Hamming distance, and it is used as the branch metric for that transition.

Soft decision decoding: As opposed to hard decision decoding in which the noisy symbols are quantized to bits before the Hamming distances are computed, for each noisy symbol in the sequence, it calculates the squared distance between each noisy symbol and the corresponding coded bit in the coded bit sequence of the given transition, and then adds these distances together to form the branch metric of that transition.

What if the path arises from a common state?



Let us suppose that our trellis diagram looks like the one above. It can be clearly seen that the path selection at $t = 4$ depends completely on the branch metric on the previous 3 time steps and not on the Path metric previous that.

This leads us to the idea of survivor length in which after a delay of L recursions, we can say that the survivor paths for all states have converged at L stages ago. Thus, one can start from any state and trace its survivor path L stages backward to find the most likely symbol at $(L - 1)$ stages ago.

Viterbi Stages

We can now see that the algorithm can be broken into 3 stages:

1 Branch Metric Generation.
2. Survivor Path Update
3. Optimum Path traceback

Now that we have developed a gist of what the algorithm does, let us now formally define the parameters.

Problem Statement

The VA is viewed as a solution to the problem of maximum a posteriori probability (MAP) estimation of the state sequence of a finite-state discrete-time Markov process observed in memoryless noise.

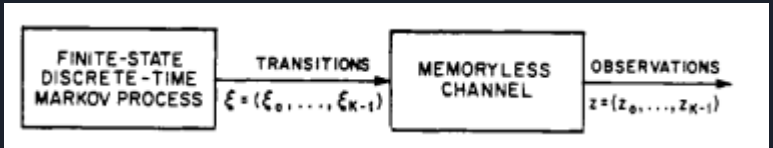
Underlying Markov process is characterized as follows:

State $x := (x_0, x_1 \dots x_k)$ where $x_i \in \{1,2,3 \dots M\}$

Since the process is Markov,

$$\mathbb{P}(x_{k+1}|x_0, x_1 \dots x_k) = \mathbb{P}(x_{k+1}|x_k)$$

Transition $\xi_k \triangleq (x_{k+1}, x_k)$
Number of states := $|X|$
Number of transitions := $|\Xi|$



Given a sequence z of observations of a discrete-time, finite-state Markov process in memoryless noise, find the state sequence x for which *a-posteriori* probability, $\mathbb{P}(x|z)$ is maximum. This is equivalent to finding the most probable input sequence, state transitions or most probable sequence of outputs.

Algorithm

Trellis: Each node corresponds to a distinct state at a given time, and each branch represents a transition to some new state at the next instant of time. For every possible state sequence x, there corresponds a unique path to the trellis and vice versa.

Every path is assigned with a length proportional to $\lambda(\xi_k) = -\ln\big(\mathbb{P}(x,z)\big)$

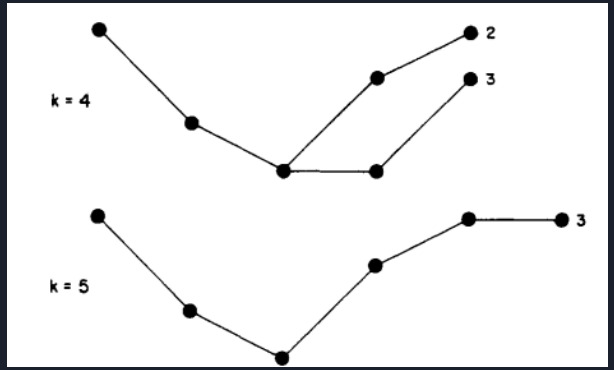
Algorithm:

- Let $\hat{x}_0$ denote a path starting at node x_0 and ending at x_k.
- At any time, there are a number of such paths, each with some length $\lambda(\hat{x}_0)$
  - The shortest of these is called the ***survivor path*** corresponding to node x_k and is denoted by $x(x_k)$, and its length is denoted by $\Gamma(x_k)$.
- For any time instance, k>0, there are M survivors in all, one for each x_k.
- The shortest complete path must begin with one of these survivors.
- At any time, we just need to remember M survivors and their lengths.
- To get to time k+1, we extend all time -k survivors by one unit, compute lengths of extended path segments, and for each node, x_{k+1}, select the shortest path terminating in it.

Optimization

The following are the few optimizations and modifications suggested for the Viterbi algorithm:

1) When dealing with sequences that are very large or infinite, truncate survivors to some manageable length δ. That is, take decisions for nodes up to time *k*-δ by time *k*. In general, when δ is large enough, all time-*k* survivors go through the same nodes up to time *k*-δ and can be put out as the algorithm's firm decision.



1) If *k* becomes large enough, $\Gamma(x_k)$ is normalised from time to time by subtracting a constant from all of them.
2) The algorithm might be required to start without knowledge of the initial state x_0. Initialise the node lengths as $\Gamma(m)=0$ or all =m or $\Gamma(m) = - \ln(\pi_m)$ where π_m are the ***a priori*** probabilities of the states.

Complexity and Performance Analysis

The most relevant metric for performance analysis is generally the probability of error (denoted by $P(\xi_k)$. This is the probability that an error sequence starts at time *k*.
As the trellis evolves, the actual message bit and the predicted bit sequence diverge and re-emerge multiple times as shown.

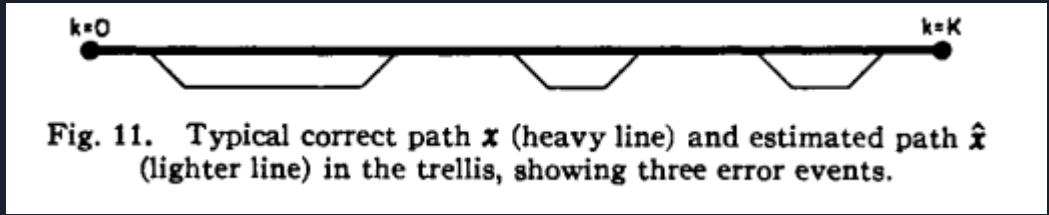


Fig. 11. Typical correct path $\boldsymbol{x}$ (heavy line) and estimated path $\hat{\boldsymbol{x}}$ (lighter line) in the trellis, showing three error events.

This probability is upper bounded by the sum of probabilities of all sequences where error messages occur at time *k*.
Memory Complexity: |X| storage locations, one for each state; each location stores the survivor length and the node list of δ symbols.
Computation Complexity: |Ξ| calculations and |X| comparisons among the results.

Applications of Viterbi Algorithm

Convolutional Codes: Viterbi algorithm is generally used for rate 1/n , but can also be extended to rate k/n, where inputs may be non-binary, or where feedback to the shift registers is implemented.

Inter-symbol Interference: Message bits getting perturbed by other neighbouring bits is called ISI. This can be modelled using a finite impulse response. We then obtain our shift register model: $y_k = \sum_i h_i u_{k-i}$

Text Recognition: In Optical-Character-Recognition (OCR) readers, individual characters are scanned and decision is made as to what character it is. English language is treated as a discrete time Markov process and contextual information is used to assist reader in resolving ambiguities.

References:

- The Viterbi Algorithm, G. David Forney Jr., https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1450960&tag=1
- Coding Viterbi Algorithm for HMM From Scratch, Zfeng, https://medium.com/@zhe.feng0018/coding-viterbi-algorithm-for-hmm-from-scratch-ca59c9203964
- Viterbi Algorithm, https://en.wikipedia.org/wiki/Viterbi_algorithm
- Hidden Markov Model, https://medium.com/analytics-vidhya/hidden-markov-model-part-1-of-the-hmm-series-3f7fea28a08
- Decoding Convolutional Codes: The Viterbi Algorithm Explained, https://youtu.be/IJE94FhyygM

Thank you!