

# Viterbi Algorithm: Background

Jean Mark Gawron

March 26, 2009

## 1 The Key property of an HMM

What is an HMM. Formally, it has the following ingredients:

1. a set of states:  $S$
2. a set of final states:  $F$
3. an initial state:  $s_0$
4. an input alphabet:  $\Sigma$
5. a transition function  $A$  which takes any pair of states and returns a probability. The transition probabilities LEAVING any state add up to 1.
6. An observation function  $B$ .

$$B(s_i, o_i)$$

is the probability of observing ‘emission’  $o_i$  (this is the way these guys talk) at state  $s_i$ .  $s_i$  is a state and  $o_i$  is a member of the input (observation) alphabet.

The key property:

The Markov Assumption: the probability of a transition to  $s_j$  from  $s_i$  depends only on  $s_i$  (Markov: Order 1).

### 1.1 Charniak’s Viterbi HMM example

Figure 1 shows a simple HMM due to Charniak with an alphabet “ab”. To discuss this example, all we need is the concept of a transition probability. There is a probability assigned to the transition from one state to another on a particular *observation* (or sometimes we say *emission*).

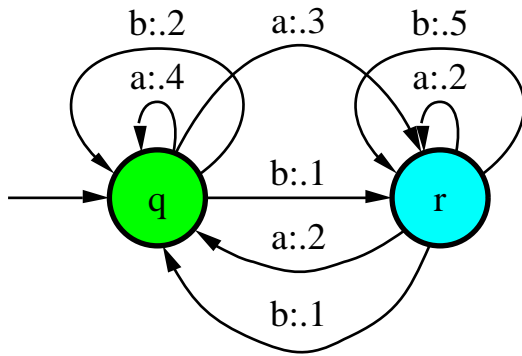


Figure 1: Charniak's simple example HMM

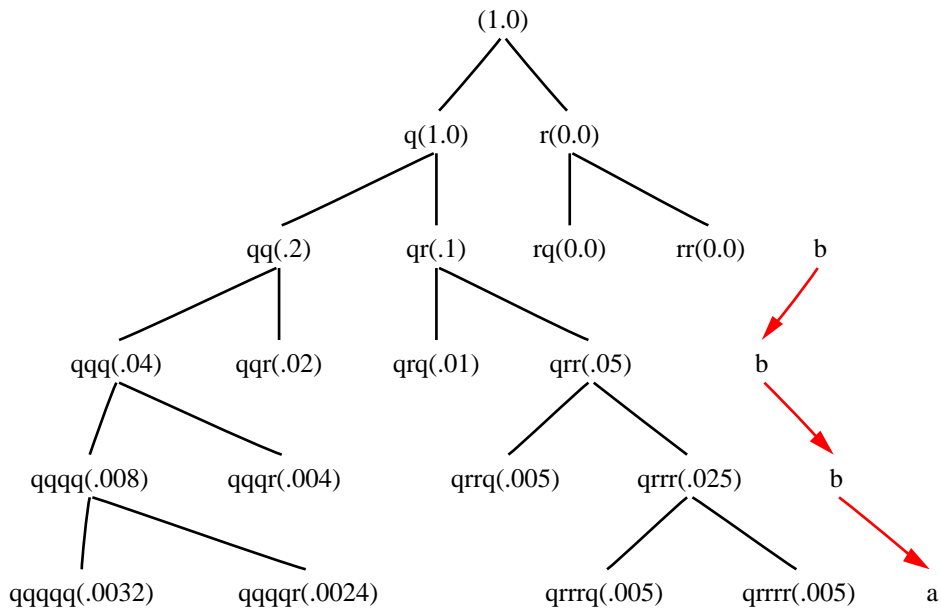


Figure 2: Computational Tree of Viterbi on Charniak's HMM

The key property is that the sum of all the transitions probabilities from a given state is 1.

The task of the Viterbi algorithm is to find the optimal path through the HMM for a given piece of input. In this case optimal path means *highest probability*.

Figure 2 shows the application of the Viterbi algorithm to the HMM. The view of the computation is a computational tree. Time is vertical. That is, each step down through the diagram reveals one more input symbol.

The HMM has two states that both accept a's and b's. From each state, for any input, there are two possible states one could go to next. Thus, if we were doing a naive search, computing the total cost of all paths covering the input, the number of active paths at each moment in time would double.

Instead, the number of active paths trees remains constant as we descend through the computation. Why? That's the trick of the algorithm.

## 2 Rob's coin-flipping HMM

Figure 3 shows the simple HMM used in the Viterbi implementation in the code model *viterbi* (see code model list in the assignment).

This is a simple coin flipping HMM. The state H does not expect a tail. The state T does not expect a head. However, when in H, you can accept an "h" either by remaining in H or, with equal probability, by moving to T. When in T you can accept a "t" either by moving to H or by remaining in T.

Figure 4 shows the same HMM with zero-probability transitions removed. This makes the overall picture clearer. Only the start state 0 is flexible about what it accepts, But when it accepts an "h" it can do so either by moving into an h-accepting or t-accepting state. Thus, each time input is accepted, a "prediction" is made about what will come next.

The transition probabilities are computed using the following probability model:

$$\Pr(o_1, \dots, o_n, s_1, \dots, s_n) = \prod_{i=1}^n \Pr(o_i | s_{i-1}, s_i) * \Pr(s_i | s_{i-1}) \quad (1)$$

We call:

$$\Pr(s_i | s_{i-1}) \quad (2)$$

the *state transition probability*. These probabilities are kept in the array of arrays \$A in the implementation.

We call

$$\Pr(o_i | s_i) \quad (3)$$

the *observation probability* These probabilities are kept in the hash of arrays \$B in the implementation.

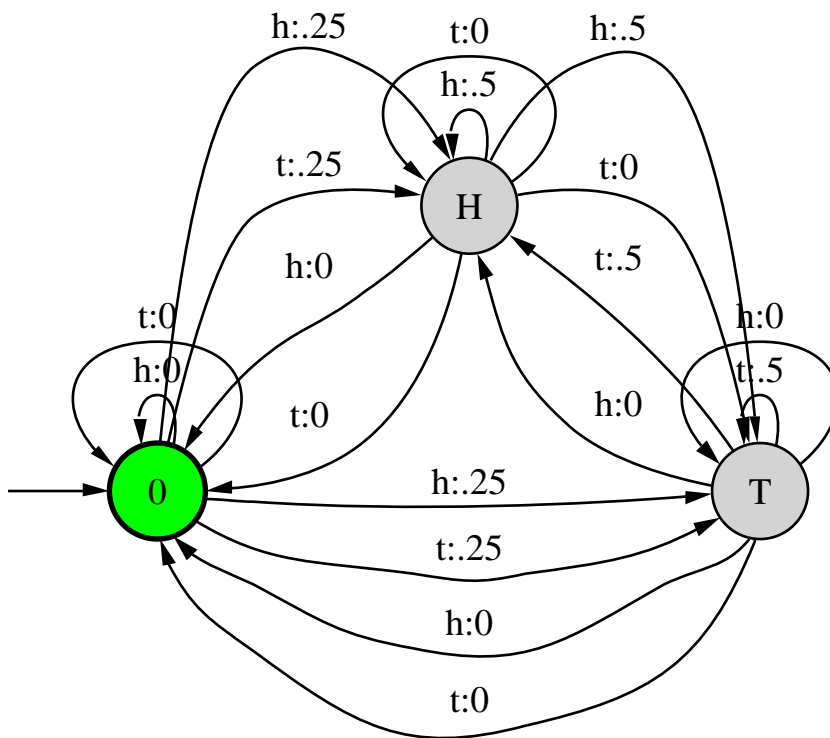


Figure 3: Rob Malouf' Viterbi Implementation Example

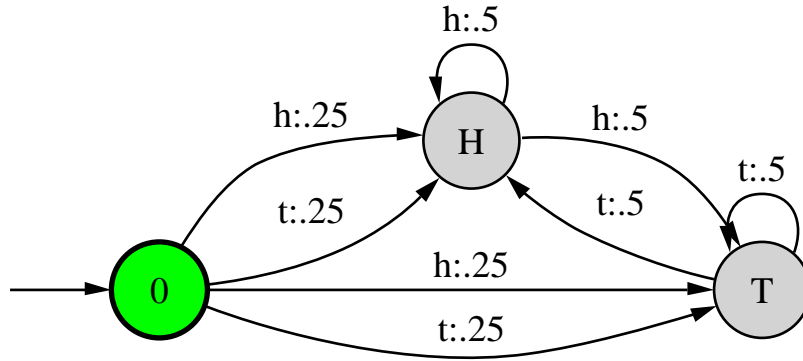


Figure 4: Rob Malouf' Viterbi Implementation Example

We call the product of these two probabilities the *transition probability*:

$$\Pr(o_i | s_i) * \Pr(s_i | s_{i-1}) \tag{4}$$

At each stage in the Viterbi algorithm this is the cost of moving from a previous state to the current state given the output at the current moment in time.

The transition probabilities of Figure 3 are computed as follows. (For each transition, the observation probability is given first and the transition probability second).

```

0 :
  h
    0=>0: 0 * 0 = 0
    0=>H: 0.5 * 0.5 = 0.25
    0=>T: 0.5 * 0.5 = 0.25
  t
    0=>0: 0 * 0 = 0
    0=>H: 0.5 * 0.5 = 0.25
    0=>T: 0.5 * 0.5 = 0.25
  -----
0 (Total Probs) = 1

H :
  h

```

```

H=>0: 0 * 0 = 0
H=>H: 1 * 0.5 = 0.5
H=>T: 1 * 0.5 = 0.5
t
H=>0: 0 * 0 = 0
H=>H: 0 * 0.5 = 0
H=>T: 0 * 0.5 = 0
-----
H (Total Probs) = 1

T :
h
T=>0: 0 * 0 = 0
T=>H: 0 * 0.5 = 0
T=>T: 0 * 0.5 = 0
t
T=>0: 0 * 0 = 0
T=>H: 1 * 0.5 = 0.5
T=>T: 1 * 0.5 = 0.5
-----
T (Total Probs) = 1

```

Here is the result of running the Viterbi implementation on this for the output “htttt”:

```

Output: htttt
Time  State    Path Prob
0:    0         1
1:    T         0.25
2:    T         0.125
3:    T         0.0625
4:    T         0.03125
5:    H         0.015625

```

States: OTTTTH

The HMM accepts an “h” by going to the T state, then remains there and accepts the final “t” by transitioning to the h-predicting state H. (The final state is always an arbitrary choice since we have not seen the “predicted” output yet.). The path probability after the first transition is .25, as required by Figure 4, and it is halved by each succeeding transition.

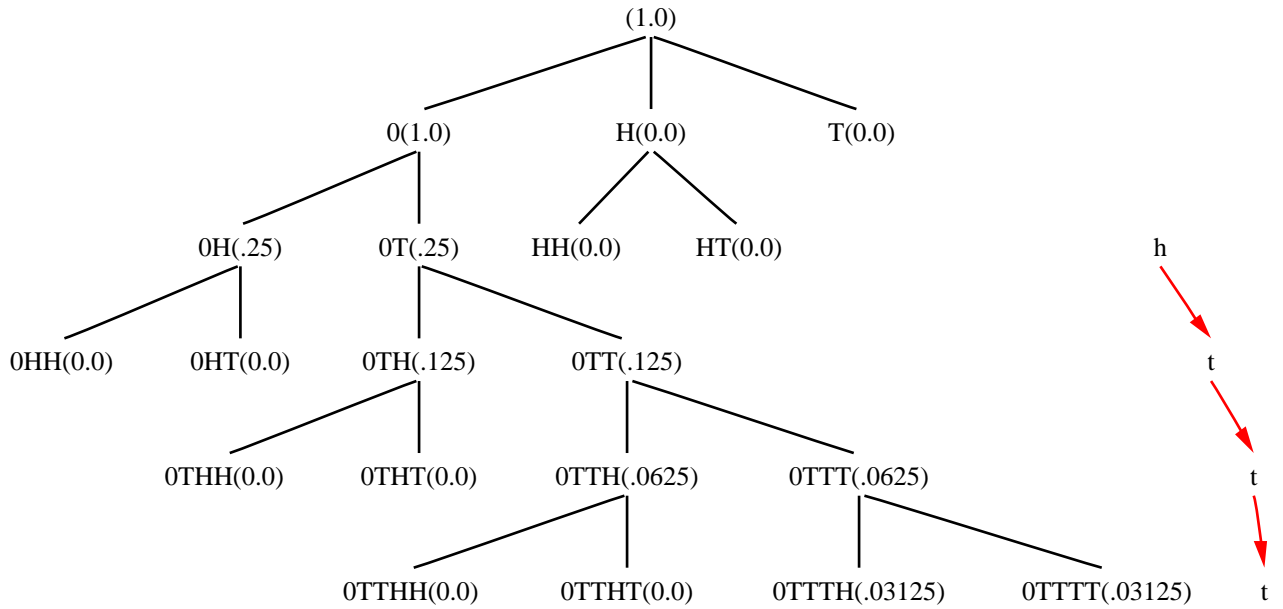


Figure 5: Rob Malouf's Viterbi Implementation Example

The application of Viterbi, using a computational tree diagram, is shown in Figure 5.

### 3 The Markov Assumption