CS9544: Analysis of Algorithms Project. Survey Paper on Hidden Markov Model Algorithms.

Ljubisa Sehovac
250 690 611
lsehovac@uwo.ca

***Abstract*—The paper analyzes the forward, Viterbi, and Baum-Welch algorithms, each contributing in efficiently solving problems for Hidden Markov models. Mathematical material is provided in each section for better understanding of the algorithms. Critical analysis is done by walking through the algorithms and analyzing the time complexity at each step. It is shown that the forward and Viterbi algorithms take $O(N^2T)$ time, while the Baum-Welch algorithm takes O(#Iterations $\times$ N^2T) time.**

I. Introduction

This project focuses on surveying papers regarding Markov chains and, in further detail, Hidden Markov Models (HMMs). Firstly, we need to identify with Markov chains to be able to fully understand HMMs. Once the basics are covered, the project will introduce HMMs and give critical analysis of three well-known Hidden Markov Model (HMM) algorithms. Critical analysis of each algorithm will be given, while suggested improvements will be given to challenge better functionality of the algorithms.

Markov chains are stochastic discrete time processes, which are sometimes called observed Markov Models, and are valuable tools at predicting the sequence of events in discrete time problems. They are a unique case of weighted automation, such that the weights are probabilities, obtained from given probability distributions, and the input sequence distinctively governs which states the automation process will take [1]. Essentially, a Markov chain is a system that changes states over time. The changes between states are dependent on a probability distribution, usually already given before a Markov process takes place. Moving from one state to the next, and continuing this action, is known as a Markov process. Now, moving from one state to the next is entirely dependent on the current state the system is in [2]. Hence, the path that was taken to the current state has no impact in deciding the future state. This is known as the Markov property. The paper discusses Markov chains in further detail in Section 2.

A. *Hidden Markov Models*

Hidden Markov Models branch off of Markov chains. They too are stochastic discrete time processes, but the main difference between the two is that the states of HMMs are now hidden, instead of being observable. What we can observe from HMMs are the observations that each hidden state generates. HMMs are still governed by a Markov process, hence moving from one state to the next is only dependent on the current state, and this process still generates a **sequence** of states - except the states here are hidden. Hence, we can think of HMMs as a Markov chain system, but the main states are now hidden to us, while each hidden state produces a probability to T observations - here, T is arbitrary to each system [1]. Thus, a HMM is essentially a sequence modeler; they are probabilistic models that given a sequence of inputs, will compute the best sequence of outputs. The words input and output are interchangeable between observations, hidden states, and the parameters being passed (these are probability distribution matrices). What is meant by this, can be seen in the following algorithms and their uses:

- To determine the probability of a sequence of observations, if we are given the sequence of hidden states - This is known as **evaluation**, and we use the **Forward** algorithm
- To determine the probability of a sequence of observations taking the most probable sequence of hidden states - This is known as **decoding**, and we use the **Viterbi** Algorithm
- To determine the probability parameters - This is known as **learning**, and we train our model using the **Baum-Welch** algorithm

All of the algorithms can be more-or-less applied to each of the following fields, respectively [1]. To explain, we use each algorithm by way of what we need to determine with relation to what is given to us. Hence, if each problem contains the same hidden states and produces identical observations, then it is in the reader's discretion to use one of the three algorithms as needed. Thus, the following fields of study are, but not limited to:

- Speech and language processing
- Signal processing
- Molecular biology
- Computational biology
- Computer science
- Energy prediction levels
- Weather forecasting

For example, the forward algorithm is of importance in computational biology, especially in that of gene sequence modelling. We can use the forward algorithm to search for a specific sequence of genes that are grouped together, given the entire sequence of observations, hence the gene profile [3]. Hence, The forward algorithm will compute the probability of this specific sequence of interest being generated. In comparison, the Viterbi algorithm would be the ideal choice if one was interested in knowing which path gene profile takes to compute

the most probable sequence of specific genes [3]. Finally, the Baum-Welch algorithm can be applied to the exact same problem, to obtain the probability distribution of producing the gene profile, as well as the probability distribution of producing the specific sequence of genes [3]. Thus, each HMM algorithm can be applied to each of the stated fields above, the choice of which algorithm to use rests dependent on the problem needing to be solved.

B. *Main Papers Analyzed*

This section introduces the two main papers that were used to complete this project. Coincedentally, both papers are called *Hidden Markov Models*. The one that will be explained first, is the one that has been used as a supplement to the other. Phil Blunson wrote his paper in 2004, briefly describing Markov chains and HMMs [4]. The paper does not mathematically write out each HMM algorithm, but instead quickly analyzes them. It succeeds in providing the reader a basic version of each algorithm, adding easy-to-follow figures and less complicated mathematical notation. Blunson's paper is a little outdated, but it does well in simplifying the intensity of the work, thus it was a good choice as a supplement to the main paper used for the project.

The main paper used for this project is chapter 9 out of the textbook *Speech and Language Processing*, written by Daniel Jurafsky and James H. Martin, 2017 [1]. The chapter is 21 pages in length and firstly explains Markov chains, then goes into full detail regarding HMMs and the three algorithms listed above. The algorithms are written out clearly, with useful examples for a better understanding. Applications of the algorithms are given, but mainly for speech and language processing techniques. Ultimately, reasons for choosing this as my main survey paper is due to the style of writing, the clear explanations, and the material is understandable. The mathematical notation may seem complex, but is straightforward if the chapter is read from the beginning, as the material does build on itself.

C. *Purpose*

The main objective of this report was to obtain knowledge of Markov chains, HMMs, and the three main HMM algorithms. The author hopes to be able to convert this knowledge into a way that can be useful for his individual research goals. He is currently studying probabilistic forecasting of energy levels using HMMs. Hence, he aims to be able to implement all three algorithms to successfully predict energy usage levels from smart meter data, and this paper serves as solid foundation for his future goals.

II. Markov Chains

To be able to define HMMs properly, we first need to discuss Markov chains. Building off of the introduction, we define a Markov chain by the following components:

- A set $Q = \{q_1, q_2, ..., q_N\}$ of N **states**
- A **transition probability matrix** $A = (a_{ij})$ where $a_{ij} = P(q_j|q_i)$ represents the probability of moving from state i to state j, with $\sum_{j=1}^{N} a_{ij} = 1, \forall i$
- An **initial probability distribution** $\pi = \{\pi_1, \pi_2, ..., \pi_N\}$ where $\pi_i = P(q_i)$ and $\sum_{i=1}^{N} \pi_i = 1$. Hence, π_i is the probability that the system starts in state i

One assumption of Markov properties needs to be made and that is the Markov property, which was briefly discussed in the introduction, but here it can be formally defined. Note, let P denote the probability function. Thus, the probability of a occupying a particular state only depends on the previous state:

$$\textbf{Markov propety} : P(q_i|q_1, ..., q_{i-1}) = P(q_i|q_{i-1}) \quad (1)$$

Hence, deciding the future state, will only depend on the current state of the system. We should note that the future state can also be the current state. This means that the system can choose to stay in the current state for the next discrete time step, if the transition probabilities allow for this. Hence, if $a_{ij} \neq 0$ for $i = j$, then $P(q_j|q_i) = P(q_j|q_j)$. Also note that the future state can be numbered lower than the current state. The notation is not meant to restrict states from being able to stay stationary or move to lower numbered states, it is just meant to state the future state only depends on the current state. This will be simplified with an example shortly. Now, although the current state is the deciding factor in Markov chains, the sequence, or **Markov process**, that the system experiences is of importance to us. This process is used to generate the probability of a certain sequence of states occurring. Hence, if we are interested in computing $P(q_3, q_2, q_1, q_1)$, that is, if we are interested in computing the probability of starting in state 3, moving to state 2, moving to state 1 and staying in state 1, we simply multiply the associated probabilities of each occurrence. In general, this is given as:

$$P(Q) = \prod_{i}^{N} P(q_i|q_{i-1}) \quad (2)$$

Hence, for the sequence we are interested in, this can be written as:

$$P(q_3, q_2, q_1, q_1) = P(q_3|\pi_3) \times P(q_2|q_3) \times P(q_1|q_2) \times P(q_1|q_1)$$

Notice how the term $P(q_2|q_3)$ was not $P(q_2|q_1)$, this is because the notation given does not restrict states from moving only in an increasing state number, but rather, this term simply represents the probability that the next state will be a state numbered lower than the current state.

To give the reader a better understanding of Markov chains, refer to Figure 1 below [4], which represents a basic Markov system of a stock price.

The system above can be defined according to the necessary components as follows:

- Let q_1 be the Bull (up) state of a stock price, q_2 be the Bear (down) state, and q_3 be the Even (unchanged) state
- Let A be the transition probability matrix, such that A is given as:

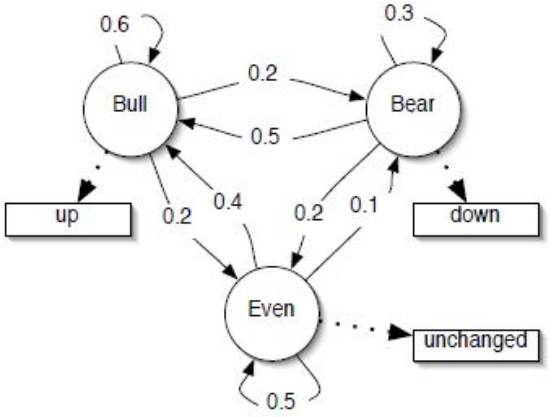

Fig. 1: Markov system for a stock price.

$$A = \begin{pmatrix} 0.6 & 0.2 & 0.2 \\ 0.5 & 0.3 & 0.2 \\ 0.4 & 0.1 & 0.5 \end{pmatrix}$$

- Let the initial probability distribution be given as $\pi = \{0.4, 0.3, 0.3\}$. That is, the probability that the system starts in state 1 is 0.4 (or 40%), as well as 0.3 and 0.3 to start in states 2 and 3 respectively. These probabilities have been generated by the author to simply explain the Markov process, while the actual initial probabilities were not given by Blunson in his paper.

Note in this example, that all sums of probability distributions (rows of A) add up to 1 as needed. With the given probability distributions, we can now substitute numbers for terms and compute the probability of any desired sequence of states occurring. In general, Analyzing the sequence above, we see that the probability for the sequence (q_3, q_2, q_1, q_1) to occur is:

$$P(q_3, q_2, q_1, q_1) = 0.3 \times 0.1 \times 0.5 \times 0.6 = 0.009$$

Hence, the probability of that sequence occurring in that order is 0.009, or 0.9%. This is the power that Markov chains have, they allow calculations of certain events happening in a specific order, for some discrete time process. If a specific order is not of interest, a **Markov simulation** can be executed to generate arbitrary probabilities of many different sequences occurring. The simulations can be done in a number of ways, all depending on the corresponding algorithms that go along with them. Unfortunately, these algorithms are not the scope of this paper and will respectfully be left out.

III. Hidden Markov Models

Now that Markov chains have been explained in more detail, we can begin to define HMMs. A Markov chain is useful when we need to compute a probability for a sequence of events that we can observe in the world [1]. However, this is usually not the case, and many events we are interested in may not be directly observable in the world. The critical components of a HMM build off of the components needed to specify a Markov system. That is, the necessary components of a HMM $\lambda = (A, B, \pi)$ are:

- A set $Q = \{q_1, q_2, ..., q_N\}$ of N **hidden states**
- A **transition probability matrix** $A = (a_{ij})$ where $a_{ij} = P(q_j|q_i)$ represents the probability of moving from state i to state j, with $\sum_{j=1}^{N} a_{ij} = 1, \forall i$
- A set $O = \{o_1, o_2, ..., o_T\}$ of T **observations**
- The **observation (emission) probability matrix** $B = (b_j(o_k))$ where $b_j(o_k) = P(o_k|q_j)$ is the probability of the current state j generating observation k. Note: The sums of the rows of B add up to one
- An **initial probability distribution** $\pi = \{\pi_1, \pi_2, ..., \pi_N\}$ where $\pi_i = P(q_i)$ and $\sum_{i=1}^{N} \pi_i = 1$. Hence, π_i is the probability that the system starts in state i

Here, two assumptions have to be stated, rather than just the one as for Markov chains. The first assumption is the exact same, which is the Markov property, defined as in (1). As before, the future state can also be the current state. For further explanation of this statement, we refer the reader to the paragraph under the Markov property in section 2. The second assumption is similar to the first, but regards the observations rather than the states. It is known as the **Output Independence (OI)**, and states that the probability of an output observation o_i depends only on the state q_i that produced the observation, and not any of the other states or observations. In mathematical notation:

$$\textbf{OI} : P(o_i|o_1, ..., o_{i-1}, q_i) = P(o_i|q_i) \tag{3}$$

Figure 2 shows how the previous Markov system can be extended into a HMM [4]. The new model now shows all observation probabilities that are emitted from each state. This straightforward change now makes the system much more expensive.

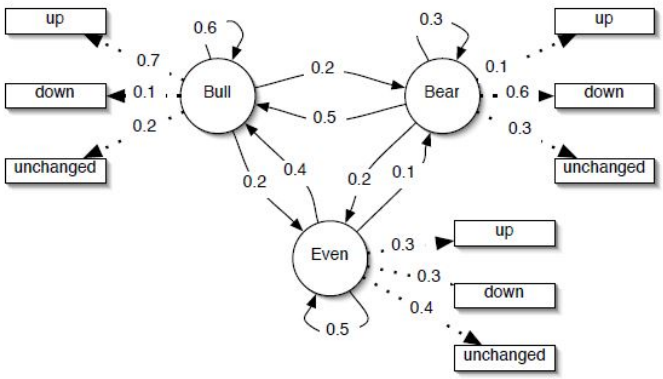

Fig. 2: HMM extension to Markov system of stock price.

From Figure 2, we can observe that the states and transition probability matrix stay the same as the Markov system from Figure 1. The new additions are the observations *up, down* and *unchanged* that each hidden state can observe. The matrix A is the same as before, while the matrix B is defined as:

$$B = \begin{pmatrix} 0.7 & 0.1 & 0.1 \\ 0.1 & 0.6 & 0.3 \\ 0.3 & 0.3 & 0.4 \end{pmatrix}$$

Assuming the same initial probability distribution as before, we can now begin to examine our HMM and how it can help us generate the probability of events occurring. The main difference from Markov chains is that if we are given the observation sequence of *down, up, up*, then we cannot say exactly which sequence of hidden states produced these observations. Thus, we have to analyze all of them. Which means if we are interested in finding $P(down, up, up)$, we need to sum over all possible hidden state combinations. In general, computing this probability can be written as:

$$P(O, Q) = P(O|Q) \times P(Q) = \prod_{i}^{T} P(o_i|q_i) + \prod_{i}^{T} P(q_i|q_{i-1}) \tag{4}$$

Hence, to compute the probability of our example observation sequence, it would look something like this:

$$P(down, up, up|Q) = P(down, up, up| \text{ bull, bull, bull}) + P(down, up, up| \text{ bull, bull, bear}) + P(down, up, up| \text{ bull, bear, bull}) + ...$$

Where, according to (4), the second term $P(down, up, up|$ bull, bull, bear) would be given as:

$$P(down, up, up| \text{ bull, bull, bear}) = P(down|\text{bull}) \times P(up|\text{bull}) \times P(up|\text{bear}) \times P(\text{bull}|\pi_1) \times P(\text{bull}|\text{bull}) \times P(\text{bear}|\text{bull})$$

This is just for one term in this particular example, which contains 3 observations and 3 hidden states. Thus, in order to compute the probability of a 3-sequence observation, we would need to compute the probability of 3^3 hidden state sequences, and then sum over all of them. Hence, in general, for a HMM with N hidden states and T observations, there are N^T possible hidden state sequences. For tasks where N and T are both large numbers, N^T is a very large number. The time complexity is greatly increased compared to that of Markov chains, where $T = 0$ and the time complexity is $O(1)$ for computing the probability of a certain sequence occurring. This of course, is not acceptable and hence the need for HMM algorithms.

A. *Forward Algorithm*

We use the forward algorithm in evaluation problems. This was stated before as determining the probability of a sequence of observations, if we are given the sequence of hidden states. Here, we give a more formal definition [1]:

Evaluation: Given a HMM $\lambda = (A, B, \pi)$ and an observation sequence O, determine $P(O|\lambda)$.

As seen in the previous section, analyzing all possible combinations of hidden state sequences is not of optimal time complexity. The forward algorithm takes the approach of removing redundant calculations given by (4), hence caching these calculations will lead to a reduced time complexity. The algorithm will store calculations and build on them to produce the final desired probability. The desired probability is generated by summing over the probabilities of all possible hidden state paths that could generate the observation sequence, but it does this by implicitly folding each of these paths into a single **forward trellis** [1]. Figure 4 shows the first two steps of the forward trellis for our observation sequence of *down, up, up*. See that Each "cell" of the forward trellis is defined as $\alpha_t(j)$, and this denotes the probability of being in state j after experiencing the first t observations. The value of $\alpha_t(j)$ is computed by summing over the probabilities of every path that could lead us to that specific cell. Formally, this value of probability is expressed as:

$$\alpha_t(j) = P(o_1, o_2, ..., o_t, q_t = j|\lambda) \tag{5}$$

Here, $q_t = j$ means "in the sequence of hidden states, the tth state is state j." The formula to compute the value $\alpha_t(j)$ is given as:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{6}$$

Where the factors in (6) are:

- $\alpha_{t-1}(i)$: the **previous forward path probability** from the time step before
- a_{ij} : the **transition probability** from previous state q_i to current state q_j
- $b_j(o_t)$: the **state observation probability** of seeing the observation o_t given the current state is state j

To see how this all comes together, observe Figure 3, and how the coloured cells $\alpha_{t-1}(i)$, arrows a_{ij}, and coloured box $b_j(o_t)$ are used to produce the value of $\alpha_t(j)$ at one state cell.

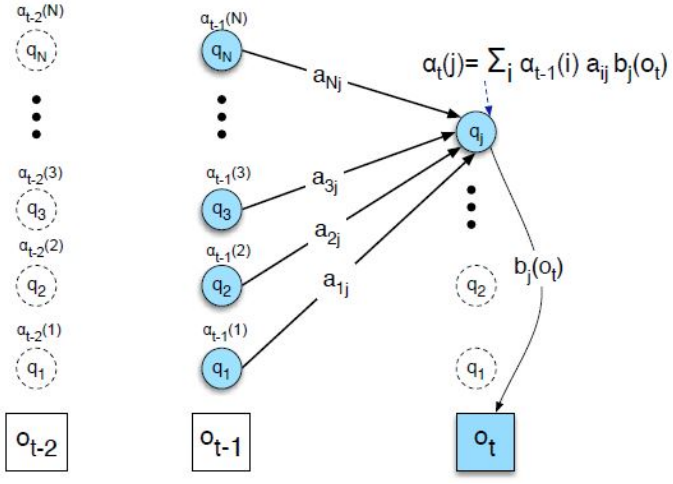

Fig. 3: Visualization of one state cell value being computed.

Figure 4 shows the first two time steps of computing the probability of the observation sequence $down, up, up$, the same example we have previously worked with. Thus, see that in order to compute $\alpha_2(1)$, we need to compute it as:

$$\alpha_2(1) = \alpha_1(1)a_{11}b_1(o_2) + \alpha_1(2)a_{21}b_1(o_2) + \alpha_1(3)a_{31}b_1(o_2)$$

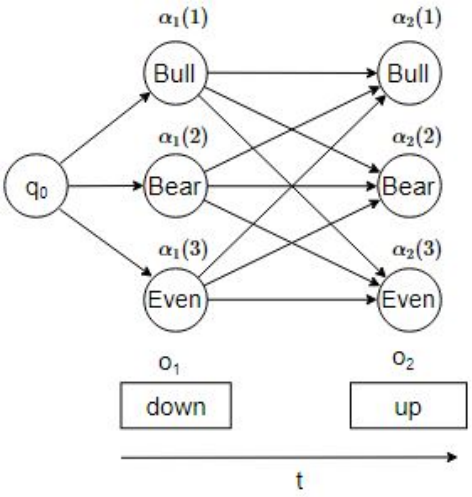

Fig. 4: A forward trellis for computing the observation probability for down, up, up, after the second time step.

Algorithm 1 Forward($G = (Q_N, A), O_T, B$)

1: **Input** : Graph G of length N, with states Q_N as vertices
2: and transition probability matrix A as edges. Set O_T
3: of observation sequence of lenght T, with emission
4: probability matrix B.
5: **Output** : Returns forward probability $\alpha_T(q_F)$. Where
6: q_F is the final state, or termination state.
7: **Build** : Probability matrix *forward*[N+2,T].
8:
9: **for** each state s **from 1 to** N **do**
10: *forward*[s,1] $\leftarrow \pi_s \times b_s(o_1)$
11:
12: **for** each time step t **from 2 to** T **do**
13: **for** each state s **from 1 to** N **do**
14: *forward*[s,t] $\leftarrow \sum_{s'=1}^{N}$ *forward*[s',t-1] $\times a_{s',s} \times b_s(o_t)$
15:
16: *forward*[q_F,T] $\leftarrow \sum_{s=1}^{N}$ *forward*[s,T]
17:
18: **Return** : *forward*[q_F,T] = $\alpha_T(q_F)$.

The calculations involving real numbers will be omitted, since their is no analytical use for them. Thus, refer to Algorithm 1, to observe how the HMM forward algorithm can be formally written.

1) ***Critical Analysis****:* To begin, note that the algorithm was not copied directly as seen in the literature referenced by [1]. Instead, it was written using the notation learned in class. Proving correctness of the algorithm will be omitted, since copying out a proof does not generate much knowledge.

Instead, we can examine the time complexity of the algorithm, which was not given by [1]. See that building the matrix *forward* takes $O(NT$+2), which is $O(NT)$, assuming $T < N$. We add the two extra states to N to make it easier to visualize the graph. We can think of these as the source and sink states. The source state provides the initial probabilities to the first observation sequence of states. The sink state, denoted as q_F in our algorithm (not shown in either figure 3 or figure 4), is used to produce the final probability. It should be noted that the algorithm given in [1] uses a final state probability, or an ending probability distribution. This is similar to our initial probability distribution π, except applied to our ending state. The author did not feel this was necessary, hence we can think of the edges incident on q_F as having probabilities = 1. That is, before the algorithm is run, $a_{iF} = 1$. That is, summing all the probabilities that are incident on our sink produces the output probability *forward*[q_F,T] = $\alpha_T(q_F)$. The first for loop, is the initialization step. This step is essentially computing the initial $\alpha_0(s)$ for each of the states s. Since there are N states, there will need to be N probabilities computed, hence, this will take $O(N)$ time. Computing the major for loop in the algorithm is what fills our matrix we built at the start. The first for loop sums over all the observations, while the second sums over all the states. Computing one coloumn, or t, of our matrix *forward* takes $O(N^2)$ time. This is because, in order to compute one cell probability value, this takes us $O(N)$ time, since there are N states producing probabilities involved in calculating one cell value. To account for all the cells, this is again, another N states. Hence, of order $O(N^2)$. See that, the algorithm runs for our length of observation sequence T, hence these N^2 calculations continue for lenght T, so the total time complexity of the main for loop is $O(N^2T)$. Returning the final probability takes $O(N)$ time, since like the initialize step, there are N states contributing to this calculation of summing the final probabilities together. Thus, the time complexity of the forward algorithm takes $O(N^2T)$ time.

A few remarks need to be made regarding the algorithm. The fact it uses a matrix makes it easy to follow. Each calculation is cached in as an element of a matrix, and then built off this calculation, which saves redundant calculations and greatly decreasing the time complexity. Note that for loop on line 12 starts from $t = 2$, this is because the initialization step, or the for loop on line 9 , takes care of the first observation $t = 1$. Also, The algorithm in [1] did not have a descriptive input, which made it more complicated to understand. The input was written by the author of this report, using knowledge obtained from class.

B. *Viterbi Algorithm*

The Viterbi Algorithm is used for determining which sequence of hidden states is the probable path for a given sequence of observations. This task is known as decoding, and can be formally written as [1]:

Decoding: Given a HMM $\lambda = (A, B)$ and an observations sequence O, find the most probable sequence of hidden states Q.

The Viterbi algorithm makes use of a dynamic programming trellis, as can be seen in Figure 5. This trellis is very similar to the forward trellis, except now the process assigns values to each cell, $v_t(j)$, as the probability that the HMM is in state j after seeing the t observations, having seen the most probable sequence $q_1, ..., q_{t-1}$ [1]. The value $v_t(j)$ is calculated by

recursively taking the most likely path that could lead to that cell. We can formally write this as:

$$v_t(j) = \max_{q_1,...,q_{t-1}} P(q_1, ..., q_{t-1}, o_1, ..., o_t, q_t = j|\lambda) \quad (7)$$

We represent the most likely path by taking the maximum over all possible state sequences. Hence, we can express (7) as:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t) \quad (8)$$

Where the three factors being multiplied in (8) are:

- $v_{t-1}(i)$: the **previous Viterbi path probability** from the previous time step
- a_{ij}: the **transition probability** from previous state q_i to current state q_j
- $b_j(o_t)$: the **state observation probability** of seeing the observation o_t given the current state is state j

The Viterbi algorithm can now be formally given. The author wrote it slightly different than the algorithm given in [1], so as to apply algorithm writing techniques learned in class.

Algorithm 2 Viterbi$(G = (Q_N, A), O_T, B)$

1: **Input** : Graph G of lenght N, with states Q_N as vertices
2: and transition probability matrix A as edges. Set O_T
3: of observation sequence of lenght T, with emission
4: probability matrix B.
5: **Output** : Return *best-path*
6: **Build** : Path probability matrix *viterbi*[N+2,T], and back
7: pointing matrix *backpointer*[N+2,T].
8:
9: **for** each state s **from 1 to** N **do**
10: *viterbi*[s,1] $\leftarrow \pi_s \times b_s(o_1)$
11: *backpointer*[s,1] $\leftarrow 0$
12:
13: **for** each time step t **from 2 to** T **do**
14: **for** each state s **from 1 to** N **do**
15: *viterbi*[s,t] $\leftarrow \max_{s'=1}^{N}$ *viterbi*[s',t-1] $\times a_{s',s} \times b_s(o_t)$
16: *backpointer*[s,t] $\leftarrow \arg\max_{s'=1}^{N}$ *viterbi*[s',t-1] $\times a_{s',s}$
17:
18: *viterbi*[q_F,T] $\leftarrow \max_{s=1}^{N}$ *viterbi*[s,T]
19: *backpointer*[q_F,T] $\leftarrow \arg\max_{s=1}^{N}$ *viterbi*[s,T]
20:
21: **Return** : back trace path by following backpointers to
22: states from *backpointer*[q_F,T].

1) ***Critical Analysis:*** We can take the reader through the algorithm step by step, while analyzing the time complexity of each step. Before we do that, let it be known that the Viterbi algorithm is very similar to the forward algorithm. The main difference is that instead of taking the **sum** over the previous path probabilities, such as in the forward algorithm, it takes the **max** over this path. The second main difference is that the Viterbi algorithm uses backpointers, while the forward algorithm does not. Otherwise, the input is identical to the input for our forward algorithm. This time, however, the output will be a sequence of states that generate the most probable path it took to see the given observation sequence. We build two matrices of same size, *viterbi* and *backpointer*. The *viterbi* matrix is used to store the values of $v_t(j)$ at each cell, while the *backpointer* matrix is used to store which state the path came from. Hence, we can think of the *backpointer* matrix as each element is a state that points back to which state it came from, and we store that state there. The time complexity to build these two matrices is $O(2(NT+2))$, not minding the 2 since it is a constant, this can be seen as the same time complexity for the building procedure in the forward algorithm. Hence, the total time it takes for the building procedure of the Viterbi algorithm is $O(NT)$, again, assuming $N > T$. The for loop on line 9 is the initialization step, similar to that of the forward algorithm, except the first coloum *backpointer* matrix is assigned 0, since it points to q_0, which is essentially just the beginning. The second for loop starting on line 13 moves through the Viterbi trellis, calculating $v_t(j)$ at each cell and storing it in *viterbi*, while the *backpointer* matrix essentially stores the state that lead to that current state. This procedure can be seen in Figure 5, where the bold arrows indicate the most probable path chosen, by use of the back pointers from each state, denoted as the dotted red lined arrows. Now, calculating one cell value takes $O(N)$ time, while calculating one coloumn takes $O(N^2)$, and since the observation sequence is of length T, the total time complexity of this main for loop is $O(N^2T)$. Then, the last coloumn is computed for both the *viterbi* and *backpointer* matrices. Note again here, as discussed in the critical analysis section for the forward algorithm, that the edges incident on the sink state q_F have probabilities $a_{iF} = 1$. This takes $O(N^2)$ times, essentially like the main for loop, but multiplied by $T = 1$, representing the last coloumn. Finally, the back trace path is returned by following the backpointers from the *backpointer* matrix. Thus, the total time complexity is of $O(N^2T)$, the same as it was for the forward algorithm.

A few remarks on the algorithm. The main recommendation on how the algorithm might perform better is to avoid another matrix, such as the one used named *backpointer*, and instead use a vector. It would not save a drastic amount of time, but instead of filling each spot of the matrix, and then looking back through it, the recommendation is to use an if loop, and store the greater value into a vector. Once the algorithm is complete, we simply return this vector which contains the most probable path for the states to occur.

C. *Baum-Welch Algorithm*

Lastly, we analyze the problem of training our HMM to produce the transition probability matrices A and B. Formally, this is given as [1]:

Learning: Given an observation sequence O and the set of possible states in the HMM, learn the HMM paramters A and B.

For this problem, we use the Baum-Welch algorithm, which is a special case of the **Expectation-Maximization**, or **EM**

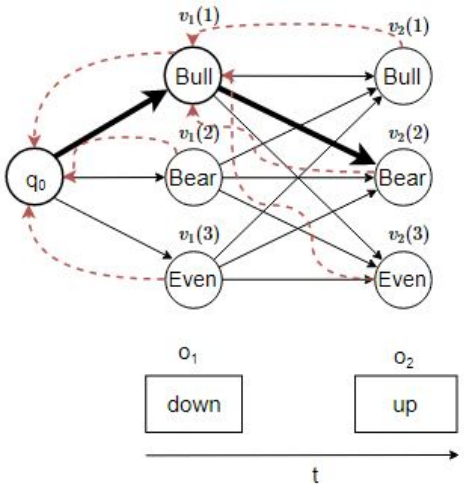

Fig. 5: A Viterbi trellis for computing the most probable sequence of states for the observation sequence down, up, up, after the second time step.

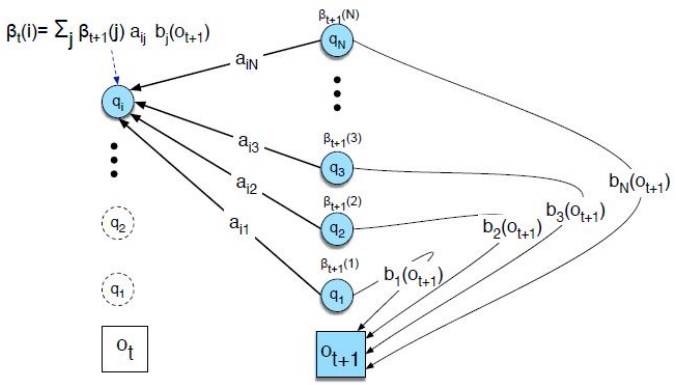

Fig. 6: The calculation of $\beta_t(i)$, using factors a_{ij}, $b_j(o_{t+1})$, and $\beta_{t+1}(j)$.

algorithm. Crucially, Baum-Welch is an iterative algorithm. It works by computing an initial estimate for the probabilities, then uses these initial estimates to compute better and better estimates, iteratively improving the probabilities as it learns [1]. To understand the algorithm, we need to define the **backward probability**. The backward probability β is the probability of seeing the observations from time $t+1$ to the end, given we are in state i at time t, and given the automation λ. Formally, this is:

$$\beta_t(i) = P(o_{t+1}, ..., o_T | q_t = i, \lambda \tag{9}$$

The backward probability is computed inductively, such as in the forward probability. There is an algorithm, called the **Backward algorithm** which essentially does what the forward algorithm does, but computes the probabilities backwards through the trellis. The algorithm won't be given, but the three main steps will be. Initialization, recursion, and termination are as follows:

$$\begin{aligned} \textbf{Initialization} : \beta_T(i) = a_{iF} = 1, \\ 1 <= i <= N \end{aligned} \tag{10}$$

$$\begin{aligned} \textbf{Recursion} : \beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \beta_{t+1}(j), \\ 1 <= i <= N, 1 <= t <= T \end{aligned} \tag{11}$$

$$\begin{aligned} \textbf{Termination} : P(O|\lambda) = \alpha_T(q_F) = \\ \beta_1(q_o) = \sum_{j=1}^{N} a_{0j} b_j(o_1) \beta_1(j) \end{aligned} \tag{12}$$

Figure 6, obtained from literature [1], shows the calculation of $\beta_t(i)$ and the factors involved. In general, it is essentially the forward algorithm, just working backwards.

Now that we have the forward and backward probabilities, we can begin to estimate a_{ij} and $b_i(o_t)$. Let $\hat{a}_{ij}$ denote the estimate. Informally, we can give this as:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i} \tag{13}$$

We introduce a new probability, and will denote it δ_t, defined as the probability of being in state i at time t and state j at time $t+1$, given an observation sequence. Formally, this is:

$$\delta_t(i,j) = P(q_t = i, q_{t+1} = j | O, \lambda) \tag{14}$$

And this is computed as:

$$\delta_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(o_{t+1} \beta_{t+1}(j)}{\alpha_T(q_F)} \tag{15}$$

How (15) is obtained from (14) is time and space consuming, thus we urge the reader to view the literature in [1] for a further, in-depth, breakdown. For now, we accept this formula to continue towards the algorithm. Now, the accepted number of transitions from state i to state j, or the numerator in (13), is the sum over all t for δ. To obtain the denominator, the total expected number of transitions from state i, we sum over all transitions out of state i. Hence, (13) can now be formally written as:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \delta_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \delta_t(i,k)} \tag{16}$$

We also need to compute an estimate for the observation probabilities, denoted as $\hat{b}_j(v_k)$, informally defined as:

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and seeing } v_k}{\text{expected number of times in state } j} \tag{17}$$

Now, define $\gamma_t(j)$ as the probability of being in state j at time t, as:

$$\begin{aligned} \gamma_t(j) &= P(q_t = j | O, \lambda) \\ &= \frac{P(q_t = j, O|\lambda)}{P(O|\lambda)} \end{aligned} \tag{18}$$

Where in (18), we used the probability law of:

$$P(X|Y,Z) = \frac{P(X,Y|Z)}{P(Y|Z)} \tag{19}$$

Where again, we accept the following equation and leave the in-depth breakdown for the reader as given in [1]. Thus, $\gamma_t(j)$ is computed as:

$$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda) = \alpha_T(q_F)} \tag{20}$$

It should be noted that the law (19) was also used in computing (15), but the notation is lengthy for that equation. Hence, (19) is the key in producing both (15) and (20), although further steps and lengthy notation is omitted to save time and space. We can now formulate (17). The numerator is computed by summing $\gamma_t(j)$ for all time steps for seeing v_k, the observation of interest. Hence, the notation $\sum_{t=1 s.t. O_t=v_k}^{T}$ means to sum over all t, at t, when the observation is v_k. The denominator is computed by summing $\gamma_t(j)$ over all t. Thus, we can now compute the estimate for the observation probabilities as:

$$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{21}$$

Thus, we can now use (16) and (21) to keep re-estimating the transition probabilities in A and B. These estimations are the main key in the Baum-Welch Algorithm, sometimes called the Expectation-Maximization algorithm, because the **expectation** step, or **E-step** computes γ and δ, while the **maximization** step, or **M-step**, computes the estimates $\hat{a}$ and $\hat{b}$. The Baum-Welch algorithm can now be formally given.

Algorithm 3 Baum-Welch(Q_N, O_T)

Input : Set Q_N of hidden states, of lenght N. Set O_T of observation sequence of lenght T.
Output : Transition probability matrices A and B.

Initialize : Matrices A and B.

Iterate : Until convergence
E-step
$\gamma_t(j) = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)=\alpha_T(q_F)}$

$\delta_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1}\beta_{t+1}(j)}{\alpha_T(q_F)}$

M-step
$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1}\delta_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N}\delta_t(i,k)}$

$\hat{b}_j(v_k) = \frac{\sum_{t=1 s.t. O_t=v_k}^{T}\gamma_t(j)}{\sum_{t=1}^{T}\gamma_t(j)}$

Return : Matrices A and B.

*1) **Critical Analysis**:* In this section, we take the reader through the algorithm step by step, explaining the process it takes, while analyzing the time complexity of the steps. It should be noted that this algorithm was the most difficult of the three algorithms analyzed in this paper. Thus, it was the most difficult when trying to write it with algorithm techniques learned from class.

The algorithm takes only the hidden states of size N and the observation sequence of length T, and returns the transition probabilit matrices A and B. Initializing A is of order $O(N^2)$ time, since the last term of the matrix will essentially be $a_N N$. Initializing B is of order $O(NT)$ time, since there are N states and T observations. Now, let us analyze one iteration. During the E-step, the γ denotes the expected state occupancy count, while the δ denotes the expected state transition count, from previous A and B probabilities. During the M-step, we use the γ and δ, computed in the E-step, to compute new probability matrices A and B. See that, in order for us to compute these probabilities, we need to use forward and backward probabilities. Using these probabilities, we use the forward and backward algorithms. The backward algorithm was not discussed in this paper since it is very similar to the forward algorithm, but rather goes backward through the trellis. Nonetheless, the time complexity of the backward algorithm is the same of the forward algorithm. This time complexity for the forward algorithm was calculated in its respective critical analysis section, and was stated as having time complexity $O(N^2T)$. Hence, the backward algorithm has the same time complexity. Thus, to compute one iteration of the Baum-Welch algorithm, takes $O(N^2T)$ time. Denote the number of iterations as Which means that the total time complexity of the Baum-Welch algorithm is O(#Iterations $\times$ N^2T). Lastly, the recommendations on how to make the algorithm more efficient are minimal, in fact, none. Mainly since it is quite complex, and extensive analysis would be necessary to make any sort of significant suggestion.

IV. Conclusion

The main purpose of this paper was to obtain knowledge of three hidden Markov Model algorithms; the foward algorithm, the Viterbi algorithm, and the Baum-Welch algorithm. Each algorithm is used for a specific problem regarding hidden Markov Models. We use the forward algorithm to determine the probability of a specific sequence of observations occurring. The Viterbi algorithm is used to determine the most probable path of hidden states that occurred for a given sequence of observations. The Baum-Welch algorithm determines the probabilities of transitioning from from state i to state j, and the probabilities of seeing observation k being in current state i. The forward and Viterbi algorithms take $O(N^2T)$ time, while the Baum-Welch algorithm is of order O(#Iterations $\times$ N^2T) time complexity. The paper describes the necessary components needed to perform each algorithm, and then gives a critical analysis for each.

V. Final Remarks

It should be noted that the algorithms and critical analysis were written with techniques learned from class, to the best of the authors abilities. Also note that Figures 4 and 5 do not have a reference, since the author made these diagrams himself, to show understanding of the material. Lastly, the paper is approximately 8 pages in length, but the format was

written to practice writing journal articles. Hence, the amount of work is more than what it seems to be. To conclude, the purpose of the paper was a success. The author is grateful for having done this work, since this knowledge will be of great use to him.

VI. REFERENCES

[1] Jurafsky, D., & H,Martin, J. (2016). Hidden Markov Models. *Speech and Language Processing*, (Chapter 9). Retrieved from https://web.stanford.edu/ jurafsky/slp3/9.pdf

[2] Ricci, L. (2012). Dipartimento di Informatica. PhD in Computer Science, Lecture slides.

[3] Yoon, B. (2009). Hidden Markov Models and their Applications in Biological Sequence Analysis. *Current Genomics*, Vol. 10, pages 402-415.

[4] Blunsom, P. (2004). Hidden Markov Models. Retrieved from http://digital.cs.usu.edu/ cyan/CS7960/hmm-tutorial.pdf