The Viterbi Algorithm and Sign Language Recognition

Elana Margaret Perkoff

May 5, 2014

ABSTRACT

This paper analyzes the Viterbi algorithm and its application to Sign Language Recognition. The Viterbi algorithm is used as a maximum *a posteriori* approach to solving the decoding problem of Hidden Markov Models (HMM). This paper discusses the attributes of the HMM with an example. The theoretical time complexity of the algorithm is compared to the results of experiments on a Python implementation. The more general field of Gesture Recognition is briefly mentioned as foundation for the type of system necessary to facilitate Sign Language Recognition. Both word and subunit models for American Sign Language are detailed.

Table of Contents

1 Background and Motivation

For far too long Deaf and hearing communities have been separated by a significant communication barrier. In the past century, this barrier has been continuously broken down with advancements in technology including hearing aids, teletypewriters, captioning services, and video relay services. However, conveying information between a signing individual and a non-signer remains the main issue. The need for technology that could bridge this gap is great, as textual information is not sufficient for proper information comprehension (Gulliver and Ghinea). People whose first language is sign language learn best in their native language, just like anyone whose first language is a spoken one (Ladner). Thus, the ideal solution to this problem would be the creation of a robust real-time sign language recognition program. This program must have the ability to be used "on the go" with as few restrictions on the user as possible. Its response time must be extremely efficient in order to keep up with the pace of life of the consumer. However, analyzing visual input in such a short period of time remains a challenge.

One potential solution is the use of Hidden Markov Models (HMM) to model signs from American Sign Language and eventually extend the model for applications to other signed languages. This model can be used to map visual input of continuous sign language to textual representation of English words. The Viterbi algorithm can be used to translate from a sequence of signs to written or spoken English. This paper will give an introduction to HMMs, the Viterbi algorithm, and gesture recognition in order to demonstrate how they can be combined to translate from American Sign Language to English.

1.1 Prior Related Work

1.1.1 An Overview of Deaf Technology

For the past century, technology has done wonders to make the hearing world more accessible to people who are deaf. Richard Ladner takes into account the various Communication Technologies that have been developed to promote communication amongst the Deaf community as well as communication technologies that break down the language barriers between the Deaf community and the hearing community. (Ladner) This kind of technology is specifically referred to as "*Deaf technology*: which permits a person with a hearing loss to communicate using technology, without any hearing enhancement." (Ladner 959) People who are deaf can communicate primarily using either text or sign language. According to Ladner, sign language is preferred because it allows

for a quicker speed of communication. However, the majority of the Deaf technology currently available uses text. The Deaf community has access to numerous kinds of text-based communication software including e-mail, text messaging, instant messaging, and speech-to-text. A teletypewriter (TTY) was used prior to the invention of these. The TTY uses telegraph technology to convey textual messages from one person to another. Relay services were introduced along with a more portable version that included an operator who would convey messages between someone using a teletypewriter and someone without them. The advancements made in video calls have enabled deaf people to hold conversations with signers via software such as FaceTime, Google Hangouts, and Skype. Ladner also discusses hearing enhancement devices including cochlear implants and hearing aids.

Gulliver and Ghinea (2003) performed a study on the effects of captioning on user perception of multimedia clips. The participants were people with a range of hearing abilities from hearing to completely deaf. They were each shown a series of video clips alternating the presence of captions. After each clip, those involved had to answer a short questionnaire to gauge the amount of information they were able to assimilate from the video shown. The questions were geared to pertain to specific input types such as video, captioning, or audio. The study yielded some controversial results. "Hearing participants assimilated significantly more than deaf participants, across all QP-IA information sources…As quantity of captions is affected by audio content, it is clear that the difference between hearing and deaf QoP-IA is not due to the specific implementation and use of captions within the experiment." (Gulliver and Ghinea 384) This evidence supports the claim that a textual transcription of a clip is not sufficient to make it understood by someone who is deaf.

"Snapshots of Interactive Multimedia at Work Across the Curriculum in Deaf Education: Implications for Public Address Training" talks about a range of interactive multimedia available for deaf students and how it can benefit their academic careers. The benefits of communication software including English to sign or sign to English translators are mentioned. (Parton) This article emphasizes the academic benefits of having an efficient sign language recognition system.

1.1.2 The Viterbi Algorithm (Implementation and Efficiency)

An extremely useful resource for learning more about the Viterbi's main parameter the Hidden Markov Model is *Statistical Methods for Speech Recognition.* The HMM adds complexity to Markov chains by allowing the states to generate observations and making the states themselves "hidden". The book also explains the use of a trellis to represent an HMM. The Viterbi algorithm

itself is detailed in a formal structure. Later on in the book it is discussed how the Viterbi algorithm could be used to find the most likely sequence of words when given a set of observed acoustic data (Jelinek).

Probabilistic and Statistical Methods in Computer Science not only details HMMs and the Viterbi algorithm, it also details the modeling of Speech Recognition to HMMs. This book focuses on the use of second order HMMs, as opposed to first-order HMMs. The first-order HMM has a two dimensional transition matrix while the second-order ones are defined by a three dimensional transition matrix. This resources contains a concise description of the Viterbi algorithm as well as a discussion of how the HMM could be used in speech modeling. There are several ways that this could be done in spoken language, just like there are several ways that this could be done for sign language. (Mari and Schott)

There are two other sources I have found that provide further insight into HMMs and the Viterbi algorithm. By starting out with an example their explanation is complete and shows how the model would be used. This source also explains the three problems that are associated with HMMs. Namely, it explains the question to which the Viterbi algorithm is an answer: how do we find the optimal state sequence for a given observation sequence? (Ching et al.) In a separate article, Diego Hernando explains how to find the entropy for the Viterbi algorithm in order to evaluate its performance. Entropy in this case refers to the degree of uncertainty for a variable. Hernando examines the entropy of the possible state sequences that could potentially generate the given sequences of observations. By tracking entropy, it is possible to measure the accuracy of an HMM's parameters (2682) .

One of the most fundamental sources for this section is named for the algorithm itself and written by G. David Forney Jr. This article is meant to be a basic overview of the algorithm and how it can be implemented. It gives a very concise definition of the problem to which the Viterbi algorithm is a solution: "Given a sequence **z** of observations of a discrete-time finite-state Markov process in memoryless noise, find the state sequence **x** for which the *a posteriori* probability $P(\boldsymbol{x}|\boldsymbol{z})$ is maximum." (Forney Jr. 269) The article also mentions several examples of processes that can be modeled with Markov chains that the Viterbi algorithm could apply to including Convolutional Codes, Intersymbol Interference, Continuous-Phase FSK, and Text Recognition. The algorithm itself can be represented using images with either state diagrams or a trellis. For the state diagram, the actual states depend on the "hidden" states of the model and the branches are the transitions that can

occur between them. On the other hand, the trellis representation takes time into account. For every instant of time, there is a node for each state that could be present. The branches between nodes represent transitions from one state to another in a single time step. Forney states that "its most important property is that to every possible state sequence $\boldsymbol{x}$ there corresponds a unique path through the trellis, and vice versa" (Forney Jr. 271). Some of the shortcomings are mentioned including the fact that in the event the state sequence is extremely large then it may be necessary to shorten the state sequence up until this point to a more manageable length. Adjustments can also be made to account for large time values or starting the algorithm without knowing the first state. The overall complexity is simply stated as follows, "the amount of storage is proportional to the number of states, and the amount of computation to the number of transitions" (Forney Jr. 273).

1.1.3 Extensions and Modifications of the Viterbi

Lindsey Foreman adapts the algorithm in her article "Generalization of the Viterbi Algorithm". Her modifications increase the knowledge of the state-sequence structure that is underlying the HMM involved. The issue with the current algorithm is that it only gives the most likely state sequence and the probability of that sequence given the observations. For those interested in learning about the rest of the state-sequence space it is necessary to adjust the algorithm. Her version tries to find the L most likely state sequences for a series of observations. At each iteration of the algorithm (or at each time step) it will find the "top L most probable partial state sequences up to time t and ending in state i, for all states $i = 1, \ldots, N$." (Foreman 356) This could be a useful extension for sign language recognition as well. A program could potentially provide the top L transcriptions of a given sequence of signs and the user could then choose the one that best conveys their intentions.

Reza A. Soltan and Mehdi Ahmadian created another version of the algorithm. Their extension solves the same problem as the original but produces higher maximum likelihoods and has the ability to detect state transitions earlier. The new algorithm involves a two-step HMM. The first part runs the standard Viterbi; "In the second step, however, we introduce a combined state sequence and use an extended Viterbi to model the transient and steady probabilities" (Soltan and Ahmadian 2873). The combined state HMM is based on creating states that consist of a single state followed by one of the states that could directly follow it in the original HMM. The "steady states" are those in which the two states that have been combined are actually the same state in the original model. The "transient states" are those in which there are two different original states. One key thing to note is

that the result from the Viterbi on this combined state HMM actually has one less probability measure than the original because there is one less observation. This means that the total maximum likelihoods of the extended and the original versions cannot be directly compared. However, the authors propose that given that the result is shorter than the original it will have a probability since probabilities are less than one (Soltan and Ahmadian). One major downside of this extension is it has a significantly larger computational complexity than the original form. The 2-step HMM runs in $O(kN^4)$ with k being time. The additional complexity is because the algorithm is being run twice and the second run through involves N^2 states. However, what the extension lacks in time efficiency it makes up for in accuracy as demonstrated by one of their examples. Overall this algorithm has the potential to extend the accuracy of the standard Viterbi if used on a small set of items.

1.1.4 Gesture & Sign Language Recognition

"A Multi-Path Architecture for Machine Translation of English Text into American Sign Language Animation" does a good job of explaining how to fit English-ASL translation into the typical model of machine translation. This article explains why translating to and from a signed language is more difficult than translation with a spoken language. "English-to-ASL translation is as complex as translation between pairs of written languages, and in fact, the difference in modality (from a written/spoken to a visual/spatial manually performed system) adds new complexities to the traditional MT problem." (Huenerfauth 1) In order to accurately interpret the full semantic meaning of a signed phrase or sentence, you need to track hand placement, where the palm is facing, the path of motion, facial expressions, and other non-manual markers. Sign Language also happens to be very concerned with spatial awareness. Signers can convey different meanings by making the same sign in different locations relative to the body. That means that a sign language recognition system would have to be able to judge relative spatial difference for each signer (since each person's body will be different) and integrate this information into the overall translation. The system would be even more complex if it allowed signers to use classifier predicates. There is some discussion in the article of the three kinds of machine translation architectural designs: direct, transfer, and interlingua systems. An interlingua system allows the most flexibility, the source language is first translated into a semantic representation known as an interlingua and then the interlingua is translated to the target language with the appropriate syntactic structure. Direct translation refers to source to target language translation for individual words without syntactic analysis. Transfer languages incorporate some sort of syntactic or semantic analysis into the translation from source to target language. For

sign language, a transfer or interlingua system would be the best option. An interlingua system would be preferable if trying to translate from sign language to multiple target languages (Huenerfauth).

The article "A Machine Translation System from English to American Sign Language" lays out a prototype for a system that would take in an English sentence and produce a visual model signing the proper translation in ASL. They discuss the importance of having sign-based translation systems for spoken language instead of just text-based ones due to the fact that many deaf individuals cannot adequately read and write in English. Additionally, there is a concern that there is a large amount of semantic information conveyed in the tones and timing of spoken language that is lost in a textual representation. Sign language could properly account for these nuances and convey all of the intended meaning of the sentence. The system described would be an "interlingua" system according to the terms defined above as it translates the spoken English into an intermediate representation before converting it into sign language. There is discussion of the Stokoe notation which is a written representation of American Sign Language centered around three phonological features: handshape, location, and movement. Others have argued that the orientation of the palm is also a necessary feature element (Zhao et al.). The morphology of sign language plays a significant role in its complexity. There are different inflections that can be used to convey temporal aspect as well as other important characteristics. These inflections are subtle but crucial to conveying overall meaning. It is also necessary to analyze the "paired noun-verbs" that share similar features and meanings such as "FLY" and "AIRPLANE". When analyzing sentences, this system turns the ASL into various glosses. However, more than one sentence or type of sentence (declarative, question, etc.) can be mapped to the same gloss. The distinction between these sentences typical lies in the non-manual features such as speed of delivery or facial expression. This article is a useful resource for the various structures in ASL that make the language unique and more complex for machine translation.

The researchers involved in the machine translation of sign language have recognized a major obstacle in achieving this goal: the lack of a large parallel sign language corpus. "A parallel corpus contains large and structured texts aligned between source and target languages. They are used to do statistical analysis and hypothesis testing, checking occurrences or validating linguistic rules on a specific universe" (Othman, Tmar, and Jemni 192). The article this quote comes from describes the methods used to achieve such a parallel corpus. The purpose of this corpus in

particular is for their project Websign, an online interpreter of Sign Language to be distributed on a non-profit basis. To be more specific, the corpus they discuss uses English as the source language and ASL gloss as the target language. ASL gloss is a system that is used to transcribe sign language since it does not have a one-to-one translation with English words.

At the heart of machine translation of ASL or any signed language we have gesture recognition. As described by the authors of "Vision-Based Hand-Gesture Applications", the three major benefits of these systems are "accessing information while maintaining sterility", "exploring big data", and "overcoming physical handicaps" (Wachs et al. 62). The article discusses the components that are essential to any gesture recognition system. One of these is price, which can vary dramatically based on the hardware (such as video cameras and sensors) that are being used. Speed is also crucial to having a successful recognition system given that regardless of application, users want results instantly. The systems can vary in their ability to accept gestures. Some can only handle a certain defined set of motions while others have the capability to learn new ones. Ease of use is also critical to creating a successful product; the acceptable motions should not be overly complex or unnatural. Finally, the accuracy of the recognition process has a major impact on overall effectiveness.

It may be of interest to mention the discussion of lexicon size of a recognition system. The lexicon is the set of signals that the system has the capability to recognize. "For sign languages (such as American Sign Language), hand-gesture-recognition systems must be able to recognize a large lexicon of both single-handed and two-handed gestures" (Wachs et al. 63). The major challenge here is incorporating a large lexicon without compromising performance. This becomes even more difficult to manage with two-handed gestures. Systems can also vary based on user restrictions. There are systems that require the user to wear a glove or have a particular background in order to function properly. These do not enhance the portability of the system and limit its uses. For example, someone could not use the system on the go because they would not be able to take out their gloves and the background might be a crowded street. This leads the authors to conclude that computer-vision-based-interfaces would be the most user-friendly form of gesture recognition systems. However, "deploying them in everyday environments is a challenge, particularly for achieving the robustness necessary for user-interface acceptability: robustness for camera sensor and lens characteristics, scene and background details, lighting conditions, and user differences" (Wachs et al. 64). There is then some discussion of the various methods used for feature extraction and the

classification stage of recognition models. The article focuses more on feature extraction methods, but these are not relevant to my work. Finally, the applications of such systems are discussed such as their role in entertainment, assistive technology, and human-robot interaction.

1.1.5 Incorporating the Viterbi Algorithm into Sign Language Recognition

The backbone article behind this entire section happens to be a broad survey of all the recent developments that have been made in visual sign language recognition. The article discusses all of the stages involved in sign language recognition. The classification stage of a sign language recognition system "requires that, for each sign of the vocabulary to be recognized, a reference model must be built beforehand. Depending on the linguistic concept, a reference model represents a single sign either as a whole or as a composition of smaller subunits --- similar to phonemes in spoken languages. The corresponding models are therefore called *word models* and *subunit models*" (von Agris et al. 344). The system described in this article implements the Viterbi algorithm for the classification stage based on these models. One important thing to mention is that this approach would employ HMMs to represent every sign in the vocabulary and thus use the Viterbi algorithm for training (von Agris et al.).

1.2 Paper Overview

The paper will begin with an explanation of the Hidden Markov Model. I will detail the components of the model using a written description as well as diagrams. It will then be implemented in a small example. Next I will focus on the Viterbi algorithm. I will begin with a description of its parameters and output. There will be a step-by-step explanation of the algorithm as well as a pseudocode prototype. I will analyze the time complexity of the algorithm and compare it to experiments run on a Python implementation. Finally, I will explain the concept of gesture recognition and its subfield, sign language recognition. In that final section, I will explain how the nuances of sign language can be modeled with the Hidden Markov Model. Additionally, I will show how the Viterbi algorithm fits into the sign language recognition process in an efficient and accurate manner.

2 An Introduction to Hidden Markov Models

2.1 Hidden Markov Models

A Hidden Markov Model (HMM) is a mathematical model to track stochastic processes, or systems in which there is not a distinct pattern that explains the relationship between two sequences, but there are defined probabilities that can be used to connect them. By definition: "there is no one-to-one correspondence between the hidden states and the observed symbols. It is therefore no longer possible to tell what hidden state the model is just by looking at the observation symbol generated" (Ching et al. 35). The purpose of the HMM is to track both the hidden and observable sets, as well as the probabilities that define the connection between their elements.

Formally, a HMM is defined as:

1. An output alphabet $\mathcal{Y} = \{0, 1, \dots b-1\}$
2. A state space $\mathcal{S} = \{1, 2, \dots, c\}$
3. A probability distribution of transitions between states $p(s'|s)$, and
4. An output probability distribution $q(y|s, s')$ associated with transitions from state *s* to state *s'*.

Then the probability of observing an HMM output string
$y_1, y_2, \dots, y_k = \sum_{s1,\dots,sk} \prod_{i=1}^{k} p(s_i|s_{i-1}) q(y_i|s_{i-1}, s_i)$
(Jelinek 17)

Sets *Y* and *S* represent the potential values of our hidden and observable states. The output alphabet *Y* contains a finite number of symbols that include all of the possible tangible events that could be produced by the model; its elements are the *observable states*. These elements can be used to represent a number of processes, including, but not limited to, amino acids, stock prices, or words in a sentence. *S* describes the list of hidden states that the model could be in at a given time. The hidden states are related to the output symbols, but, as mentioned above, not by a one-to-one function.

The last two pieces of the model are the two probability matrices. The first of these matrices gives the probability of being in a given state while seeing a certain output. If every row *i* in the matrix corresponds to an output symbol from $\mathcal{Y}$ and every column *j* in the matrix corresponds to a state from $\mathcal{S}$, then every square *i,j* contains the probability that the model would be in state *j* when it has produced the output *i*. The sum of the values in any column will be the total probability of being

in any state given this symbol. If a cell *i,j* has the value zero then there is no chance of being in state *i* at the same time as observing *j*.

The second matrix is the transition probability matrix. In this matrix, both the rows and the columns represent the states in $\mathcal{S}$. Each cell *x*, *y* corresponds to the probability of being in state *x* at time *t* and having been in state *y* at time *t -1*. With this information, we can safely say that the probability of being in a state *x* at a time *t* while seeing the output $m \in Y$ is based on the probability of being in state *x* given the output *m* and the probability of being in state *x* following being in state *y* at the previous time step. Mathematically, this can be stated as:

$$P(x, t\,|m) = (x|m) \cdot P(x, t|y, t-1).$$

HMMs can also contain a fifth parameter, an initial state distribution. This distribution has the same number of entries as the number of states in the model. Each member of the list denotes the probability of being in state *s* at time $t = 0$, or the probability that the process would start in this state. The initial state distribution is relevant for many applications, such as part-of-speech tagging. In part-of-speech tagging, the observable states are the words in the sentence and the hidden states are the parts-of-speech (*e.g*. nouns, verbs, articles, etc.) to which they correspond. In the English language, it is much more common for a sentence to begin with an article such as *the* or *a* instead of a verb. For other stochastic processes, these probabilities can be considered irrelevant, because they do not have to start in a particular place. The following example does not require an initial state distribution because our weather pattern sequence does not start at a particular time of the year. This has been done for the sake of simplicity.

2.2 Representing Seasons Using the Model

We will take a look at a simple example to demonstrate the usefulness of the HMM. On a daily basis, we can observe weather patterns that are occurring such as rain, snow, sun, or clouds. These patterns tend to correspond to the four seasons: spring, summer, fall, and winter. The weather patterns are observable events, or the outputs of our HMM. Seasons, on the other hand, are the hidden states in which the world can be at a given time. We can then generate a matrix representing the probabilities of being in a certain season given a weather pattern. It is also easy to discern the probabilities of changing from one season to another. Please note, the probabilities in this example

are completely arbitrary and not based on any scientific data. Therefore, we can represent these patterns with a HMM as follows:

1. An output alphabet $\mathcal{Y} = \{snow, rain, hail, sun, thunderstorm\}$
2. A state space $\mathcal{S} = \{winter, spring, summer, fall\}$
3. A probability distribution of transitions between seasons:

Table 1: Transition Probabilities between Seasons

	Winter	Spring	Summer	Fall
Winter	0.75	0.25	0	0
Spring	0	0.75	0.25	0
Summer	0	0	0.75	0.25
Fall	0.25	0	0	0.75

4. An output probability distribution of being in a certain season given an observable weather event:

Table 2: Output Probability Distribution from Season to Weather

	Snow	Rain	Hail	Sun	Thunderstorm
Winter	0.9	0.1	0.9	0.15	0
Spring	0.05	0.5	0.05	0.15	0.1
Summer	0	0.25	0	0.5	0.8
Fall	0.05	0.15	0.05	0.2	0.1

It is important to fully understand what each cell, row, and column represent in the two tables above. In Table 1, each row represents a season *z* that the world could be in at a given point in time *t*. Every column represents a season *s* that the world could be in at the next point in time *t* +1. Thus, every cell in Table 1 represents the probability of moving from being in season *z* at time *t* to season *s* at time *t + 1*. For example, the cell (spring, summer) in this table tells us that the probability of transitioning from spring into summer in a single time step is 25%. If a cell is equal to zero, then there is no possibility of transitioning to season *s* after being in season *z*. The sum of the probabilities of each row must be equal to one because they represent the entire state space of possible state transitions from the corresponding starting state. Essentially, each of these rows gives us the full list of states that it is possible to move to in a single time step from the season corresponding to the row. Although the columns in this particular table appear to follow a pattern, the sum of their values does not have significant meaning in relation to the process being modeled.

Table 2 represents the output probability distribution for the season model. Each cell *i, j* contains the probability of being in a given season *i* while seeing the weather pattern *j*. The cell that intersects the row *winter* and the column snow tells us that there is a 90% chance that if it is snowing, the current season is winter. It is important to remember that this is different from saying that if the current state is winter there is a 90% chance that is snowing. The sum of the values in all of the columns in Table 2 totals one because they represent the entire set of possible seasons that the world could be in while seeing the weather pattern *j*. The rows in Table 2 represent the weather patterns that can be observed during a particular season. However, each of these probabilities is independent from one another (at least in this example). This means that the probability of seeing snow during the winter is not related to the probability of seeing rain. For this reason, the sum of the values of the rows does not need to be a particular number. Although these tables are good for visualizing the probabilities, it is easier to see that they denote a flowing process when the HMM is in a different form.

A HMM can also be represented by a transition diagram. Nodes represent each of the hidden states in the model, while arcs represent state transitions. The probabilities on each arc represent the probability of moving from one state to another. If there is no arc between two states, then the probability of changing between them is zero. Similarly to the sum of the values in the rows of Table 1, for any state, the sum of the probabilities on all of the arcs leaving the state will be equal to one.

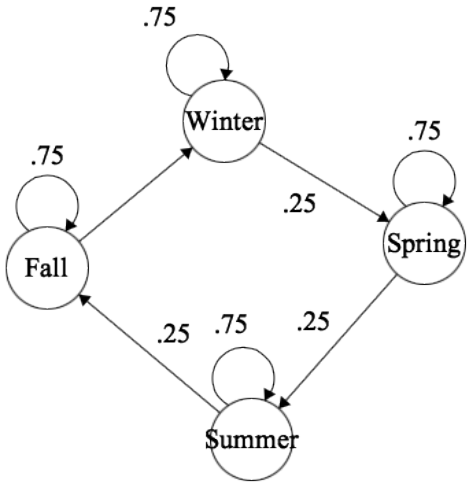

Figure 1: This is the state transition diagram with probabilities included for the Seasons. The probabilities on the arrows correspond to the probability of transition from one state to another.

For each season, it is more likely to continue to be in that season than to change to a different one. I have assigned a zero percent chance of going backwards in the seasons or from skipping a season such as going from winter to summer. All of the values in the above tables and diagram are completely arbitrary and not based on actual meteorological data.

2.3 The Decoding Problem

The usefulness of the HMM becomes more apparent when solving certain problems relating to stochastic processes. There are three problems in particular that are referred to when discussing the model:

> *Problem 1:* Given the observation sequence $O = \{O_1O_2...O_T\}$ and a HMM, how to efficiently compute the probability of the observation sequence?
>
> *Problem 2:* Given the observation sequence $O = \{O_1O_2...O_T\}$ and a HMM, how to choose a corresponding state sequence $Q = \{Q_1Q_2...Q_T\}$ which is optimal in a certain context?
>
> *Problem 3:* Given the observation sequence $O = \{O_1O_2...O_T\}$, how to choose the model parameters in a HMM? (Ching et al. 37)

The Viterbi algorithm is used to solve the second of these three problems, which is also known as the *decoding problem*. The goal is to find the optimal hidden state sequence for a given set of observations over time. This sequence is optimal in the sense that it has the highest probability of corresponding to these events. Referring to our optimal states, as Q, our model as Λ, the initial state distribution as π, time as t, and our set of observations of O, this optimality criterion can be given by the formula:

$$P(Q, O|\Lambda) = P(Q_0, O_0) \cdot P(Q_0|\pi) \prod_{i=1}^{t} P(Q_i, Q_{i-1}) \cdot P(Q_i, O_i)$$

The Viterbi algorithm is used to find the state sequence $Q = \{Q_1Q_2...Q_T\}$ that maximizes this probability for a given observation sequence $O = \{O_1O_2...O_T\}$. (Ching et al.)

The decoding problem is relevant to many processes found in different fields. By solving for the maximum value of the formula above, we can attempt to find the hidden states associated with the observable states for any HMM. In terms of the season example, this means that for any given sequence of weather patterns, we can calculate the single best sequence of seasons with which it corresponds. As I will discuss later in this paper, we can also solve the decoding problem in the

context of real-time sign language recognition. This means that given a video sequence over a period time T, if we refer to chunks of video as observable states, we can deduce the hidden states that correspond to their meaning.

3 Understanding the Viterbi Algorithm

3.1 The Viterbi Algorithm

The Viterbi algorithm is given two inputs: a HMM and a sequence of observed events. The characteristics that make up the model can be used to refer to it in the form $\Lambda = \{Y, Q, A, B, \pi\}$. In this notation, Y is the output alphabet or the observable states, Q is the set of hidden states, A is the state transition matrix, B is the output probability distribution, and π refers to the initial state probability matrix. The sequence of observed events will be referred to as O over time T in the form $O = \{o_0, o_1, o_2, \ldots o_T\}$.

For every observation o at time t, the algorithm finds the state s that the process is most likely to be in based on the values stored in Λ. While considering the computational implementation of the algorithm, all of the matrices are referred to as arrays. To determine the most likely state to be in at time $t = 0$, the probability of starting in s is multiplied by the probability of being in s while seeing the output o_0. Therefore, in order to find the first hidden state in the sequence, the maximum over all s is found over the equation: $P(s_0, o_0| \Lambda) = \pi[s] \cdot B[s][o_0]$. This equation iterates over all n states, storing the current best state as well as its probability. These values are replaced whenever a state with a better probability is found until all n states have been checked. The algorithm proceeds similarly for values of $t > 0$. However, at this point the probability of being in a given state s_t given the observation o_t is actually dependent on the state at time $t-1$. As such, the probability of being in s_n given o_t is found by the equation $P(s_t , o_t | \Lambda) = A[s_t][s_{t-1}] \cdot B[s_t][o_t]$. Once the single best state s^* at time t has been found, the program repeats itself for $t+1$. The probability of being in s^* given o_t is multiplied by the probability of the sequence thus far in order to keep track of the total probability. Additionally, s^* is added to the optimal sequence S. Once the algorithm has iterated over all $o_t \in O$, both S and the total probability of S, $P(S, O| \Lambda)$ are returned.

3.2 Efficiency of the Viterbi Algorithm: Time Complexity

```
#HMM is the Hidden Markov Model represented by a class
#O is the observation sequence
#t is the point of time that we are starting at
#S is the optimal state sequence so far, with each entry being an integer value
#P is the total probability of seeing S so far
Set S equal to an empty list
Set t equal to 0
Set P equal to 1
def Viterbi(HMM, O, t, S, P):
        if (t == length(O)):
                return (S, P)
        else:
                c = index of O_t in the set of outputs for HMM
                if t == 0:
                        for  every hidden state in the HMM:
                                -get probability of this state being the first state
                                -get probability of this state given the output
                                        -if the combined probability of these is greater than that of the
                        current best state, this becomes the best state
                else:
                        for every hidden state in the HMM:
                                -get probability of this state following the one before it
                                -get probability of this state given the output
                                        -if the combined probability of these is greater than that of the
                        current best state, this becomes the best state


                append the best state to S and multiply its probability by P
                return Viterbi(HMM, O, t + 1, S, P)
```

Figure 1: Sample Pseudocode Implementation of the Viterbi Algorithm

Suppose the algorithm is being run on a set of T observations O where T is the total number of observations and a HMM Λ with $|N|$ hidden states. The complexity of the algorithm is linearly dependent over time T, because the program must iterate once for every observation o at a time $t \in T$. At every time step t, the algorithm then needs to loop through all of the $|N|$ possible hidden states that the HMM, this takes time $O(|N|)$. When finding the optimal state corresponding to the observation o_t the algorithm has to find the state transition probability as found in the state transition distribution

defined in Λ for every hidden state $n \in N$ to the optimal hidden state s_{t-1}. It also needs to reference the probability of being in state n given o_t as found in the output probability matrix in Λ. When $t = 0$, we need the latter of these values, but instead of the state transition probability, we find the probability of this state being the starting state as given by the initial state distribution. In the Figure 1 above, all of the probability matrices of Λ are denoted by arrays. We assume that accessing an item in an array takes constant time O(1). However, in order to reference the appropriate output probability, we need the lookup of the observation o_t as found in the array representation of Σ. This needs to be done once for each iteration t thus adding the time to lookup an element of an array, O(*loookup*). Therefore, finding the optimal hidden state s_t corresponding to o_t takes O($|N|$ + *O(lookup*) time. Knowing that for every observation o_t the algorithm finds s_t, the total time taken by the algorithm can be computed by O($T \cdot (|N| + O(lookup))$).

3.3 Evaluating the Runtime Further

I have experimented with different input sizes using my implementation as found in Appendix A. First, I worked on increasing the observation sequence size. I used a script to generate random sequences of increasing lengths. For each sequence size, I have generated ten unique random sequences and taken the average runtime over these implementations. I used the python random module in order to ensure that the sequences were selecting random weather patterns and the time module to calculate the amount of time taken by each instance of the program.

An additional script was created to modify the size of the HMM itself. My program produced a given number n hidden states named $\{s_0, s_1, \ldots s_{n-1}\}$ and a given number s observable states $\{a_0, a_1, \ldots a_{n-1}\}$. To simulate probability matrices for the model, I generated an $n \cdot n$ and an $n \cdot s$ matrix. Both of these were populated with values between zero and one as provided by python's random number generator. Given that the states, the alphabet, and the matrices of these models were created arbitrarily, the actual output of the program did not convey a significant meaning. However, given that everything was appropriately sized, it is possible to evaluate the time taken on runs for each of the generated models and sequences. The results can be seen in table 3 below:

Table 3: Runtime Increase for Parameters			
Observation Sequence	**Hidden States**	**Observable States**	**Average Runtime (seconds)**
10	10	10	1.491E-04
100	10	10	1.567E-03
10	100	10	1.172E-03
10	10	100	1.570E-04
100	100	100	1.227E-02
1000	100	100	1.278E-01
100	1000	100	1.177E-01
100	100	1000	1.228E-02
1000	1000	1000	1.248E+00
10000	1000	1000	1.239E+01
1000	10000	1000	1.227E+01
1000	1000	10000	1.253E+00
10000	10000	10000	1.279E+02

The findings in this chart are consistent with the theoretical time complexity that I have proposed above. The runtime increases linearly in correspondence with both the number of hidden states and the length of the observation sequence. This means the longer the length of the observation sequence and the higher the number of hidden states in the HMM, the longer the runtime. However, given a model with 10^4 hidden states, 10^4 observable states, and an observation sequence of length 10^4, the program only takes approximately 2 minutes to run. The algorithm's parameters will most likely be lower based on its application.

The graph below details the effect of changing a single parameter of the algorithm on the runtime. While changing a single parameter, the other two were fixed at one hundred. The diamond points correspond to the runtime given different observation sequence sizes. There is also a line of best fit marked on the graph to demonstrate the linearity of the relationship. Squares are used to relate change in the number of hidden states in the model to runtime. Once again there is a line of best fit to show the linear relationship. The slope of this line is slightly lower than that of the one corresponding to observation sequence length suggesting that an increase in number of observable states has less of an impact on runtime than observation sequence length. The third set of points on

the graph demonstrates the effect of increasing the count of observable states in the model. It is clear that the impact of increasing this parameter is minimal. This suggests that increasing the output alphabet of the HMM for the system will not cause significant delays.

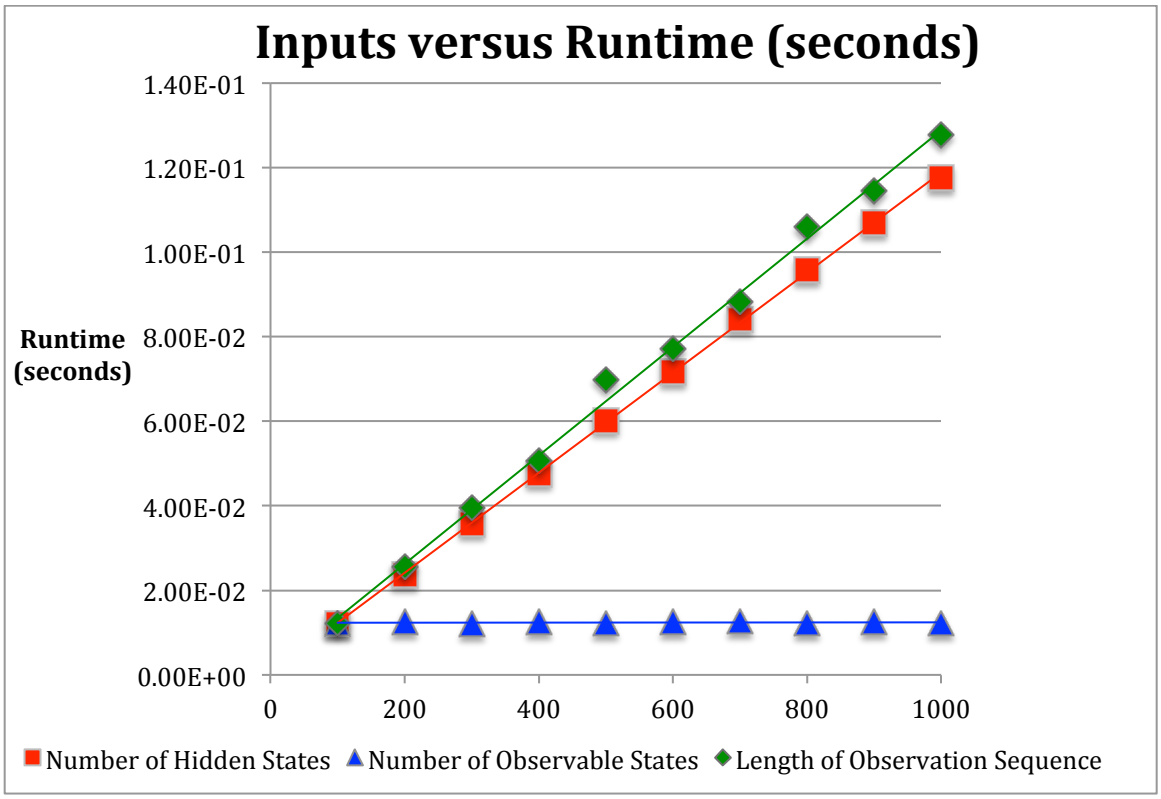

Figure 2: This graph compares the effect of changing each of the parameters of the algorithm on the runtime of the program. The points correspond to the runtime given the appropriate value of the changing input while keeping the other two fixed at 100.

The full table of results used to plot these graphs can be found in Appendix B.

3.4 The Seasons Model: Finding the Optimal Season Sequence

Now that we have formally defined the Viterbi algorithm, we can apply it to the HMM for the seasons that was presented in section 2.2. Suppose that we are given the following observation sequence: {*snow, rain, rain, sun*}. In the first iteration the program will do the following:

1.) Check if $t >$ *length of* O → false, we continue

2.) Find the index of observation o_0 (*snow)* in the array Σ
3.) For each season in *S*:
    a. Calculate the probability of being in this season given *snow*
    b. If this value is greater than the current maximum probability replace the current best season with this season and the current best probability with this value
4.) Call the algorithm again on *t=1*, with the optimal state (*winter*) added to the optimal state sequence and the probability thus far is set to the probability of being in *winter* given *snow*

For the proceeding three calls to the program (on observations {*rain, rain, sun*}), steps 1,2, and 4 will be performed in the same manner as they are for the first call to the function. At each step the appropriate indices will be found as they correspond to the current observation and the appropriate optimal states will be added to the sequence. The probability of the optimal state for each time step will be multiplied by the probability that was fed into that call of the function. The true difference between the calls at time $t = 0$ and $t > 0$ is in the third step of the program. For $t > 0$, it will proceed as follows:

3.) For each season *s* in *S*:
    a. Calculate the probability of being in this season given o_t
    b. Calculate the probability of being in *s* at time *t* given the probability of being in *s'* at time *t -1*
    c. If the product of these two values is greater than the current maximum probability, then *s* becomes the best state so far and the product becomes the maximum probability seen so far

During the final iteration of the program, the algorithm checks to see if *t* is equal to the length of *O*. We know that there is not an infinite amount of recursive loops because the value of *t* is increased by 1 during every recursive call of the function. In this example, the algorithm stops once *t =4*. At this point, the optimal sequence of seasons is returned along with the total probability of this sequence of seasons. My implementation of this example as found in **Appendix A** produces the optimal state sequence: {*winter, spring, spring, summer*} with the probability 0.00527 (rounded to five decimal places). Given that this is such a small sequence of states, the probability can actually be calculated by hand. To do this we do the following calculations:

First iteration: P(*winter*|*snow*) = 0.9
    Max probability = 0.9
    Total probability so far = 0.9
Second iteration: *p* = P(*spring*|*rain*) = 0.5
    q = P(*spring*|*winter*) = 0.25

Max probability $p \cdot q = 0.125$
Total probability so far = 0.1125
Third iteration: p = P(*spring*|*rain)* = 0.5
q = P(*spring*|*spring)* = 0.75
Max probability $p \cdot q = 0.375$
Total probability so far = 0.04219*
Fourth iteration: p = P(*summer*|*sun*) = 0.5
q = P(*summer*|*spring*) = 0.25
Max probability $p \cdot q = 0.125$
Total probability so far = 0.00527*

*These values were truncated to five decimal places. The calculations were performed on the full float values.

The probability in bold is the total probability of the sequence as calculated by hand, which is identical to the value produced by my implementation of the algorithm. This value is the probability that the sequence of hidden states returned by the algorithm {*winter, spring, spring, summer*} is the sequence corresponding to the sequence of observations {*snow, rain, rain, sun*}.

4 Applying the Viterbi Algorithm to Sign Language Recognition

4.1 From Gesture Recognition to Sign Recognition

The term *gesture recognition* is used to describe technology that takes a physical gesture as its input. Although this concept is still relatively new, its applications are already widespread and continue to grow rapidly. This form of human-computer interaction can be implemented in a number of ways. However, we find that accuracy suffers when trying to accommodate the needs of everyday users. Most consumers want a recognition design that will be non-intrusive as well as extremely fast. Gesture recognition is what allows you to interact with your Wii using a controller, and it lets you play Dance Dance Revolution on a Kinect without one. People can now use gesture recognition to explore the human body and wave their hands through immensely large sets of data. These systems seem incredibly simple and work almost instantaneously, so why is sign language recognition not yet implemented by this "magical" technology?

The key difference between standard gesture recognition systems and a visual sign language recognition system is the complexity of their lexicons. The lexicon of a recognition system is the collection of all the motions that it is capable of recognizing. Gaming systems tend to focus on

motions of the legs and arms. They then associate certain movements with a meaning, which is then interpreted to perform an action on the device. Other systems tend to use simple gestures such as a wave of the hand in one direction or the other, clapping of the hands, or pointing to certain locations. Lexicon complexity is the key difference between voice commands and voice recognition. With voice commands, you can lock a house, turn on lights, or maybe play some music by saying a few simple words. A voice recognition system, such as Siri, can handle full sentences or questions and interpret them properly.

A sign language lexicon would have to be significantly larger due to the nature of the language. In American Sign Language (ASL), there are signs that involve one hand, signs involving both hands moving together, and signs that require the left and the right hand to perform different actions simultaneously. Additionally, facial expressions play a significant semantic role in sign language, so they should be analyzed along with the appropriate gestures (Liddell and Johnson). These nuances of ASL increase the size and complexity of the program's lexicon immensely. The diagram below depicts the phases of a prototype for a sign language recognition system.

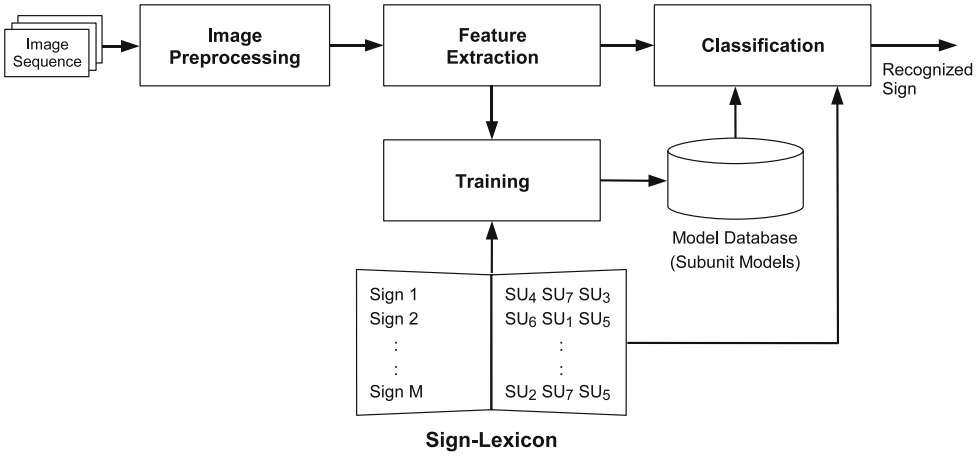

Figure 3: This diagram simulates the phases of a sign language recognition system.

(von Agris et al. 349)

The first step in the recognition process is to take in the visual input from a camera. Next, feature extraction is performed. This will remove background elements that are distracting to the program in order to isolate the parts of the image that will provide it with the proper input for the algorithm to interpret. The hands and face will be isolated so the system can focus on their movements instead of the movements of other objects in the screen. These features are then sent to

the classification stage where the motions are officially 'recognized' or associated with a semantic meaning. Once a semantic meaning has been assigned to the entire video sequence then they can be put in the desired translated output. This output can be in the form of written text, which could later be interpreted as spoken output. Von Agris' and his colleagues focus on outputting English, but the system could easily be adapted to accommodate other output languages. Hypothetically, the system could also be adapted to input other signed languages. The Viterbi algorithm has potential to apply to two stages of this process. The first is in the feature extraction stage and the second is in the classification phase.

4.2 Where the Viterbi Algorithm Fits

In this section, I will focus on the implementation of the Viterbi algorithm in the classification stage of a visual sign language recognition application. At this phase of the recognition process, the gestures themselves have been isolated from the background. The classification stage of a sign language recognition system "requires that, for each sign of the vocabulary to be recognized, a reference model must be built beforehand. Depending on the linguistic concept, a reference model represents a single sign either as a whole or as a composition of smaller subunits --- similar to phonemes in spoken languages. The corresponding models are therefore called *word models* and *subunit models*" (von Agris et al. 344). These reference models can be built using HMMs. The difference between a subunit model and a word model depends on our potential set of outputs. In both cases, the system would be taking visual input as interpreted by a camera, and associate this input with a meaning from a predetermined lexicon. For both systems, it is also possible to integrate semantic information from facial expressions into the results by using parallel HMMs (von Agris et al. 344).

The representation of a gesture in the output alphabet of the model is crucial to its ability function. The system needs to feed a transcription of the gestures identified in the feature extraction stage that can be stored within the model. For every recognizable gesture, or every element in the output alphabet of the model, the system will require a detailed description. The description must be complete enough to recognize this element as unique from the others, while being basic enough that it can be immediately discerned from a video clip, even with variation over individual signers.

Multiple attempts have been made at creating such a system, although there is not currently a universally accepted option. William Stokoe proposed a notation that "analyze[d] each sign into three phonological components: *handshape, location* (place of articulation), and *movement*"*(*Zhao et al. 3). Scott K. Liddell and Robert E. Johnson have also proposed their own notation for sign transcription. Their system is based on separating signs into sequences of holds and movements. Characteristics of these segments including hand configuration, contour planes, and orientation are then assigned values to be incorporated into a matrix for the sign. It would be easy to incorporate these feature matrices as the elements of an output alphabet in an HMM for ASL (Liddell and Johnson).

4.2.1 Word Model

A word model representation of sign language would process every complete sign individually. This means that in a HMM representation, the output alphabet would consist of one element for every individual sign in the lexicon. The set of hidden states would include possible semantic meanings for each of the signs. These meanings would be in the form of an ASL gloss. In a gloss notation, "signs are represented in their natural order by uppercase words taken from their nearest spoken counterparts" (Zhao et al. 2). In turn, the probability matrices would relate these two sets of states together. Each entry in transition probability matrix would give the likelihood of having a certain ASL gloss following the prior one. The output probability matrix would be composed of the probabilities associating each sign with each state. By using probability matrices, some of the ambiguity in sign language translation can be removed. In certain contexts, it will be easy to discern what the proper meaning is since the likelihoods are given based on the surrounding translations as well as the meaning of the current sign.

One of the main issues focused on this interpretation is when to decide that a video sequence has comprised an entire sign. Sign language is extremely fluid with one motion flowing into the next. This means that the system must be able to identify sign boundaries to properly analyze individual signs. Determining these boundaries without restricting the nature of the user's signing proves to be quite difficult. It is not natural for a signer to return to a neutral position between words so that would not be a useful indicator for a truly robust recognition system. One potential solution to this problem is to attempt to pursue several different sign boundaries. With the Viterbi algorithm, we can return the likelihood of a total path and determine which of the sign boundaries produces the best result, or the one with the highest total probability. Although this solves the problem of

determining what video sections are an entire sign, this does not leave much room for expansion of the lexicon. The system will only be able to recognize words that were in the training set of the application under a word model.

4.2.2 Subunit Model

The subunit model for sign language representation solves some of the problems associated with the word model. Instead of analyzing a complete sign, a subunit model would break every individual sign into a number of parts. Each of these parts would be represented as an element in the set of output symbols in the HMM. In turn, the transition probability matrix for such a model would give the probabilities of transitioning from one signed subunit to another signed subunit. Similarly, the probabilities of associating a certain signed subunit with a specific semantic meaning would be depicted in the output probability distribution. Although this version would significantly increase the number of observable states in the model, I have already proven in Section 3.3 that this will not significantly impact the speed of the program. The subunit model has many advantages including the possibility for learning within the model. Since the sequence is recognized based on subunits and not entire words, new words could potentially be added to the lexicon of the system. These words would be analyzed by simply associating two or more subunits together in a context in which they were not originally changed. This would limit the amount of preprocessing that would be required to design the lexicon for the system without limiting the number of words recognized. Additionally, there is no longer a concern of what qualifies as a word boundary for signs. Instead of analyzing for word boundaries, the sequence would be split into its subunits.

The major concern with this system is how to define subunits in ASL or any signed language. For spoken languages, subunits can be defined by phonemes, which are the smallest units of spoken sound. However, defining a comparable subunit in sign language is more difficult. One potential solution is to use the hold and movement system defined by Liddell and Johnson. Units can be categorized in one of two ways: “a movement (M) segment is characterized by a change in one or more of its articulatory features and hold (H) segments are not” (Liddell and Johnson 280). As described in Section 4.2, each of these segments is then classified according to a feature matrix. Each unique feature matrix could be included as an element in the feature matrix. Multiple movement and hold matrices could be linked together to form a sign, or a single hidden state.

5 Conclusion

5.1 Implementation Challenges

In the past fifty years, Deaf technology has made incredible strides towards closing the gap between the hearing and non-hearing communities. The goal of this field of computing is to enhance the ability of Deaf people to communicate with others without actually changing their hearing ability. Early examples of products from this field include closed captioning, the teletypewriter, and video relay services. Closed captioning has made videos with sound accessible to those with some degree of hearing loss, while video relay services allow for real-time communication between someone who is signing and someone who is speaking. Video conferencing technology also falls under the umbrella of Deaf technology because it allows signing individuals to communicate long-distance in real time. However, real time communication between a signer and a non-signer still remains a dream. A truly robust real-time sign language recognition system would be able to make this dream a reality.

There are many challenges still ahead in developing the optimal sign language recognition system. To promote usability, the system must be non-intrusive without reducing its accuracy. This means that recognition should be based on visual information alone; sensors or markers on the face or hands would reduce the appeal of these systems. Yet, there is a flaw in this system: "deploying them in everyday environments is a challenge, particularly for achieving the robustness necessary for user-interface acceptability: robustness for camera sensor and lens characteristics, scene and background details, lighting conditions, and user differences" (Wachs et al. 64). Ideally, the system would be portable, available on a tablet or other mobile device to allow its users to communicate with non-signing people while outside of their own homes.

The semantic nature of sign language introduces many issues to such a system as well. One of the key issues is the amount of information that is conveyed through facial expressions. For instance, raising the eyebrows is used to indicate that a sentence is a question instead of a statement. Additionally, there are many noun-verb pairs for which the signs are the same. One example for this is the noun-verb pair, chair-sit. In the images below, a man is forming the sign that represents both of the words in this pair. When gesturing for *chair*, the signer would make this motion twice, while when gesturing to sit it is only performed once.

Figure 4: This sequence of motions can be used to indicate *chair* or *sit*. To signify *sit*, the signer makes this gesture once, while it is done twice to indicate *chair*. (Viccars)

A truly accommodating lexicon would have to include both of these words and accommodate their similarity appropriately. Luckily, based on context it is easier to decipher whether a signer in using a verb or a noun based on the preceding and following words. This means that the HMM representation would be very helpful for keeping track of these likelihoods.

Another issue is the lack of a large sign language corpus. One of the difficulties with statistical classification that von Agris' group discusses is the fact that "in order to adequately train a set of word models, each word in the vocabulary must appear several times in different contexts" (von Agris et al. 344). For speech recognition, there already exist unfathomably large corpora including recordings of different spoken languages. However this is not the case with sign language (Othman, Tmar, and Jemni).

5.2 Future Directions

The full potential of the Viterbi algorithm, along with HMMs, to facilitate translation between ASL and English has not yet been reached. A program based on this framework could make real-time communication between signing individuals and non-signers are reality. This could have a huge impact in the field of universal accessibility in terms of healthcare, equal employment and education opportunities, and cultural diversity.

There are still many pieces that need to fall into place in order to allow for such systems to realize their full potential. There is a great need for a large sign language corpus that would present a wide array of signed words in many contexts. The corpus would have to be created by native signers, preferably with several different signers. The purpose of having multiple signers for the corpora is to ensure that none of the models are being created based on the nature of a single signer. This will minimize the amount of training needed for each user when they begin using the device. In addition to the creation of a comprehensive corpus, significant testing must be done to ensure that the Viterbi algorithm is as efficient with visual input as it is with textual input. Such experimentation would also help determine what kind of process is best to determine boundaries between signs when analyzing continuous sign language sequences. I hope that this paper will inspire more people to dive into such experiments and motivate people worldwide to begin assembling the corpora necessary to reach the full potential of this technology.

Acknowledgements

I would like to sincerely thank my advisor John Dougherty, as well as the entire Computer Science Department at Haverford College. They have all been extremely supportive throughout this process in donating their time and providing me with invaluable guidance. Additionally, I would like to thank Dora Wong without whom I would not have been able to find many of the sources that were vital to this paper.

References

Ching, Wai-Ki et al. *Markov Chains : Models, Algorithms and Applications*. 2nd ed. Dordrecht: Springer, 2013. Print.

Foreman, Lindsey. "Generalization of the Viterbi Algorithm." *IMA Journal of Mathematics Applied in Business and Industry* 4.4 (1992): 351–367. Print.

Forney Jr., G. David. "The Viterbi Algorithm." *Proceedings of the IEEE*. 61, No. 3. N. p. 268–278.

Gulliver, S.R., and G. Ghinea. "How Level and Type of Deafness Affect User Perception of Multimedia Video Clips." *Universal Access in the Information Society* 2.4 (2003): 374–386. *CrossRef*. Web. 19 June 2013.

Hernando, Diego. "Efficent Computation of the Hidden Markov Model Entropy for a Given Observation Sequence." *IEEE Transactions on Information Theory* 51.7 (2005): 2681–2685. Print.

Huenerfauth, Matt. "A Multi-path Architecture for Machine Translation of English Text into American Sign Language Animation." *Proceedings of the Student Research Workshop at HLT-NAACL 2004*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2004. 25–30. *ACM Digital Library*. Web. 30 July 2013. HLT-SRWS '04.

Jelinek, Frederick. *Statistical Methods for Speech Recognition*. Cambridge, Massachussetts: The MIT Press, 1997. Print.

Ladner, Richard E. "Communication Technologies for People With Sensory Disabilities." *Proceedings of the IEEE*. 100, No. 4. N. p., 2012. 957–973. Print.

Liddell, Scott K., and Robert E. Johnson. “American Sign Language: The Phonological Base.” *Sign Language Studies* 64 (1989): 195–277. Print.
Mari, Jean-François, and René Schott. *Probabilstic and Statistical Methods in Computer Science*. Boston: Kluwer Academic Publishers, 2001. Print.
Othman, Achraf, Zouhour Tmar, and Mohamed Jemni. “Toward Developing a Very Big Sign Language Parallel Corpus.” *Computers Helping People with Special Needs*. Ed. Klaus Miesenberger et al. Springer Berlin Heidelberg, 2012. 192–199. *link.springer.com*. Web. 30 July 2013. Lecture Notes in Computer Science 7383.
Parton, Becky Sue. “Snapshots of Interactive Multimedia at Work Across the Curriculum in Deaf Education: Implications for Public Address Training.” *Journal of Educational Multimedia and Hypermedia* 15.2 (2006): 159–173. Print.
Soltan, Reza A., and Mehdi Ahmadian. “Extended Viterbi Algorithm for Hidden Markov Process: A Transient/Steady Probabilities Approach.” *International Mathematical Forum*. Vol. 7. N. p., 2012. 2871–2883. *Google Scholar*. Web. 12 July 2013.
Viccars, William. “Noun Verb Pairs in American Sign Language (ASL).” *ASLU*. N. p., n.d. Web. 7 Mar. 2014.
Von Agris, Ulrich et al. “Recent Developments in Visual Sign Language Recognition.” *Universal Access in the Information Society* 6.4 (2008): 323–362. *ProQuest*. Web. 9 July 2013.
Wachs, Juan Pablo et al. “Vision-Based Hand-Gesture Applications.” *Communications of the ACM* Feb. 2011 : 60–71. Print.
Zhao, Liwei et al. “A Machine Translation System from English to American Sign Language.” *Proceedings of the 4th Conference of the Association for Machine Translation in the Americas on Envisioning Machine Translation in the Information Future*. London, UK, UK: Springer-Verlag, 2000. 54–67. *ACM Digital Library*. Web. 30 July 2013. AMTA ’00.

Appendix

A. Python Implementation of the Viterbi Algorithm

```
class HMM:
    def __init__(self, Y, S, A, B):
      #the output matrix for the hMM
      self.outputs = Y

      #the state space S for the hMM
      self.states = S

      #the probability distribution of transitions betweeen states s and s'
      self.transitions = A

      #the output probability distribution between a state s and an observation y
      self.outputprob = B

  def Viterbi(model, observations):
    #FS will be the final state sequence
    FS = []

    #FP will be the final probability of this state sequence
    FP = 1
```

```
    #counter will be the current observation that we are observing, we will start at 0
    counter = 0
    return Viterbi_helper(model, observations, FS, FP, counter)

def Viterbi_helper(m, o, FS, FP, c):
    if c == len(o):
        return (FS, FP)
    else:

        #current gives us the observation we currently have
        current = o[c]

        #this will find us the index of the observation in the model's outuput matrix
        current_i = m.outputs.index(current)

        #best_prob and best_state will be used to keep track of the best probability so far, as well as
the state corresponding to
        #that probability
        best_prob = 0
        best_state = 0
        if c==0:
            #this is the first element in the list, we assume the state transition probabilty is 1
            x = 0
            while x < len(m.states):
                #temp is the probability that we would be in state x given observation current
                temp = m.outputprob[x][current_i]
                if temp > best_prob:
                    best_prob = temp
                    ##note best_state is stored as integer, to make indexing the list easier
                    best_state = x
                x = x+1
        else:
            #this is not the first observation in the sequence
            x = 0
            while x < len(m.states):
                #tempo is the probability that we would be in state x given observation current
                tempo = m.outputprob[x][current_i]

                #tempt is the transition probability that we would be in state x given state s-1
                tempt = m.transitions[FS[c-1]][x]

                if (tempo * tempt) > best_prob:
                    best_prob = tempo * tempt
                    best_state = x
                x = x+ 1
```

```
    FS.append(best_state)

    return Viterbi_helper(m, o, FS, (FP*best_prob), c+1)
```

B. Full Runtime Comparison Results

Length of Observation Sequence	Number of Hidden States	Number of Observable States	Runtime (seconds)
100	100	100	0.0122704029083
200	100	100	0.0255000829697
300	100	100	0.0394173145294
400	100	100	0.0506064414978
500	100	100	0.0699458599091
600	100	100	0.0771495103836
700	100	100	0.0882344961166
800	100	100	0.1059865474700
900	100	100	0.1145632982250
1000	100	100	0.1277986764910
100	200	100	0.0237019062042
100	300	100	0.0357755184174
100	400	100	0.0475756168365
100	500	100	0.0601468801498
100	600	100	0.0716802120209
100	700	100	0.0842028856277
100	800	100	0.0958550214767
100	900	100	0.1070508480070
100	1000	100	0.1176595926280
100	100	200	0.0125206232071
100	100	300	0.0121295452118
100	100	400	0.0124610662460
100	100	500	0.0122648000717
100	100	600	0.0125157594681
100	100	700	0.0126628398895
100	100	800	0.0122137308121
100	100	900	0.0125510931015
100	100	1000	0.0122840166092