

Real-time Gesture Recognition with Minimal Training Requirements and On-line Learning

Stjepan Rajko, Gang Qian, Todd Ingalls and Jodi James
Arts, Media and Engineering Program
Arizona State University, Tempe, AZ
{srajko, gqian, testcase, jodi.james}@asu.edu

Abstract

In this paper, we introduce the semantic network model (SNM), a generalization of the hidden Markov model (HMM) that uses factorization of state transition probabilities to reduce training requirements, increase the efficiency of gesture recognition and on-line learning, and allow more precision in gesture modeling. We demonstrate the advantages both formally and experimentally, using examples such as full-body multimodal gesture recognition via optical motion capture and a pressure sensitive floor, as well as mouse / pen gesture recognition. Our results show that our algorithm performs much better than the traditional approach in situations where training samples are limited and/or the precision of the gesture model is high.

1. Introduction

Currently, hidden Markov models (HMMs) are the state of the art modeling scheme used in gesture recognition. Using state-based probabilistic modeling of the gestures, HMMs provide a robust and accurate framework for many kinds of pattern-based analysis. Still, there are several shortcomings of typical HMM approaches. Due to the time complexity of training and inference algorithms, the number of states that can be used in the model is limited, placing constraints on the number, length, and precision of gestures that can be modeled. Also, an HMM-based system requires a sizable amount of training data to work well.

For this reason, using gesture recognition is difficult or impossible in certain settings. In movement rehabilitation, it may be difficult for a patient to provide many samples of a gesture to train the system. In creative settings such as interactive dance performances, lengthy retraining of the system at every change of choreography can be tedious.

In this paper, we present a number of additions to the HMM framework with the specific goal of improving the time complexity of the algorithms, and relaxing the require-

ments for thorough training. We call the proposed model the semantic network model (SNM), after *semantic states* which it uses to pinpoint semantic meaning, such as the beginning and end of a gesture, in a sequence of observations.

In section 1.1, we review some existing work that is related to SNMs before presenting the model in Section 2. We formulate the gesture recognition problem in terms of SNMs in Section 3, and provide a treatment of the solution in Section 4. Finally, we provide comparative experimental results in Section 5, and conclude with Section 6.

1.1. Related Work

There are many bodies of work related to the semantic network model. Primarily, SNMs can be viewed as a generalization of hidden Markov models, and are related to many other similar probabilistic models. Murphy [8] provides an excellent treatment of various HMM-related models, all cast as special cases of dynamic Bayesian networks.

The hierarchical hidden Markov model (HHMM) [3] in particular is commonly used for gesture recognition. An HHMM can be viewed as a model that is itself composed of multiple HMMs. This is very useful in cases where the system being modeled is hierarchical, such as in speech recognition. SNMs extend the notion of hierarchy by introducing *semantic states*, which mark parts of the model that carry semantic meaning. Using semantic states to denote the beginning and end of a phoneme or a word, for example, can be used to represent hierarchy in the model much like is done in an HHMM. However, semantic states can also be used to represent non-hierarchical semantic structures. For example, they could mark the mid-point of a gesture, a variation, or other events carrying semantic meaning.

Before HMMs became the ubiquitous tool for gesture recognition, simpler dynamic programming alignment (DPA) methods were commonly used. These methods solve the problem by comparing a known sequence of observations (a training sample) with an unclassified sequence. This can be done, for example, using the edit distance algo-

rithm [7], or dynamic time warping [4, 2]. Our adaptation of the Viterbi algorithm (Section 4.1) is based on a joint treatment of HMMs and DPAs by Rajko and Qian [9].

The algorithms usually used for training and inference in both HMMs and DPAs can be described as instances of the sum-product algorithm in factor graphs [5], or the general distributive law [1]. Both of these frameworks increase computation efficiency by exploiting common factors in the expressions being calculated. For example, probabilities of state sequences in HMMs are computed using probabilities of shared partial state sequences. In SNMs, we take this concept a step further by exploiting common factors in the state transition probabilities themselves, resulting in further time complexity reduction.

2. The Semantic Network Model

We will first briefly review the basics of HMMs, and then define the proposed extension. We are mainly adapting the notation and nomenclature used by Murphy [8].

A hidden Markov model (HMM) $\lambda = (S, \pi, A, B)$ is defined by a set of states S , an initial state probability distribution π , a state transition probability distribution A , and the observation probability distribution B . The HMM begins its execution in a state chosen by π , i.e. it will begin in some state $s \in S$ with probability $\pi(s)$. At each *time step*, the HMM will make a state transition - if it is in a state $s \in S$ it will transition into state $s' \in S$ with probability $A(s, s')$. At each transition, an observation is generated according to the observation probability distribution B .

Similar to an HMM, a semantic network model (SNM) is executed by transitioning through its states in a stochastic manner, and generating outputs (observations) when entering some of the states. Formally, we specify an SNM $\lambda = (S = (T \cup G \cup U), \pi, A, B)$ by a non-empty set of production states T , a set of semantic states G , a set of auxiliary states U , an initial state probability function $\pi : T \rightarrow [0, 1]$, a state transition function $A : S \times S \rightarrow [0, \infty)$, and an observation probability function $B : T \times \mathbb{D} \rightarrow [0, \infty)$, where \mathbb{D} is the set of observations. Intuitively,

- *Production states* (T) produce observations.
- *Semantic states* (G) do not produce an observation but entering them does carry some semantic meaning.
- *Auxiliary states* (U) are used to simplify computation but neither produce an observation or carry meaning.

We will often refer to semantic and production states together by *principal* states ($T \cup G$), and to semantic and auxiliary states by *non-production* states ($G \cup U$). While there are similar concepts in other modeling approaches (HMMs, grammar modeling, etc.), defining the states in this particular way facilitates both an intuitive understanding of our approach, and factorization of state transition probabilities.

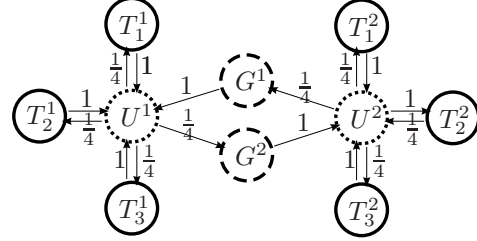


Figure 1. A sample SNM. Solid circles indicate production states, dashed circles semantic states, and dotted circles auxiliary states. Arrows indicate possible transitions between states, and are labeled by the state transition function values.

An SNM functions much like an HMM. It starts in a production state according to π , and then makes transitions according to A . However, as we explain later, A is not necessarily a probabilistic function. When entering a production state, an observation is produced according to B . Since only production states produce an observation, and we assume that observations are generated/collected at some consistent frame rate, entering a production state takes one time step, while all other transitions are executed instantaneously.

Figure 1 shows a sample SNM. The model has two parts, indicated by superscripts ¹ and ², with transitions between the parts going through semantic states G^1 and G^2 . Each part can generate observations through the production states, with the auxiliary states simplifying the transitions. Here, the semantic states indicate that we are interested in noting when the SNM transitions between the two parts.

Before we specify the behavior of the SNM formally, we first note a few definitions and restrictions. Let the (directed) graph of the SNM be $G(\lambda) = (S, E)$ with $(s, s') \in E$ if and only if $A(s, s') > 0$. We call a path (s_1, s_2, \dots, s_n) inside $G(\lambda)$ an *auxiliary path* if $s_1, s_n \in S$, and $s_2, \dots, s_{n-1} \in U$. If rather $s_2, \dots, s_{n-1} \in U \cup G$, we call the path a *non-production path*.

Given $s \in S$ and $t \in T \cup G$, define $C_{s,t}^{aux}$ (respectively, $C_{s,t}^{np}$) to be the set of all auxiliary paths (respectively, non-production paths) connecting s and t . For any such path $c = ((s_1 = s), s_2, \dots, (s_n = t)) \in C_{s,t}^{np}$, define $p(c) = \prod_{i=1}^{n-1} A(s_i, s_{i+1})$, i.e. the product of A values along the path. Finally, define $A_{s,t}^{aux} = \sum_{c \in C_{s,t}^{aux}} p(c)$, and $A_{s,t}^{np} = \sum_{c \in C_{s,t}^{np}} p(c)$. If $s \in T \cup G$, we call $A^{aux}(s, t)$ (respectively, $A^{np}(s, t)$) the probability of transitioning (respectively, stepping) from s to t .

Additionally, we assume the following:

1. For any states $s, s' \in S$, there is at most one auxiliary path from s to s' .
2. For every $t \in T \cup G$, $\sum_{t' \in T \cup G} A^{aux}(t, t') = 1$.
3. For every $g \in G$, there exists a production state $t \in T$ such that there is a non-production path from g to t .

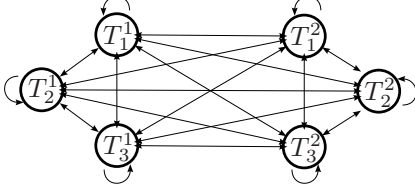


Figure 2. A HMM corresponding to the SNM in Figure 1. The probability of transitioning between states with the same superscript is $\frac{1}{4} \times \sum_{k=0}^{\infty} (\frac{1}{4} \times \frac{1}{4})^k$, or $\frac{1}{4} \times \frac{1}{4} \times \sum_{k=0}^{\infty} (\frac{1}{4} \times \frac{1}{4})^k$ if the superscripts are different.

The first rule simplifies the discussion by disallowing loops of auxiliary states, and alternate auxiliary paths between principal states. The second rule allows us to use products of A values along auxiliary paths between principal states as transition probabilities, and the third rule ensures that the SNM will never get “stuck” in states from which no production states are reachable.

From the above properties, we can derive that $\forall t \in T$, $\sum_{t' \in T} A^{np}(t, t') = 1$, which allows us to map an SNM to an HMM. In this case, however, we can only model transitions between production states, because that is the only state supported by the (non-hierarchical) HMM. We thereby define the HMM-equivalent of an SNM $\lambda = (S = (T \cup G \cup U), \pi, A, B)$ to be the HMM $\lambda' = (T, \pi, A^{np}, B)$ (with A^{np} restricted to the domain T).

For example, in the SNM given in Figure 1, the probability of stepping from T_1^1 to T_2^1 is identical to the probability of stepping from T_3^1 to T_2^1 . It is equal to the sum of products of A values over all non-production paths from one state to the other, i.e. $\frac{1}{4} \sum_{k=0}^{\infty} (\frac{1}{4} \times \frac{1}{4})^k$. The infinite summation is due to the non-production loop consisting of states G^1, U^1, G^2 and U_2 . Figure 2 shows the HMM obtained through this conversion. You can see how the regularity in the state transition probabilities, exploited in the SNM through auxiliary states, is lost in the HMM.

Describing the behavior of an SNM $\lambda = (S = (T \cup G \cup U), \pi, A, B)$ directly is simple if $\sum_{s' \in S} A(s, s') = 1$ for all $s \in S$. In this case, we call the SNM *normalized*, and Figure 3(b) provides an example. Here, the execution starts in a state $t \in T$ with probability $\pi(t)$. The SNM will then transition from state s to state s' with probability $A(s, s')$. Whenever the SNM enters a production state $t \in T$, it generates an observation $o \in \mathcal{O}$ as determined by $B(t, o)$.

If A is not normalized, the SNM functions the same except when it comes to state transitions. In this case, we make use of the following proposition:

Proposition 2.1 *If $A^*(s, s') = A(s, s') \frac{\sum_{t \in T \cup G} A^{aux}(s', t)}{\sum_{t \in T \cup G} A^{aux}(s, t)}$, then for any two principal states t and t' , any path $c = (s_1, s_2, \dots, s_n) \in C^{np}(t, t')$ satisfies $\prod_{i=1}^{n-1} A(s_i, s_{i+1}) = \prod_{i=1}^{n-1} A^*(s_i, s_{i+1})$. Also, for any state $s \in S$, $\sum_{s' \in S} A^*(s, s') = 1$. (Proof omitted for brevity).*

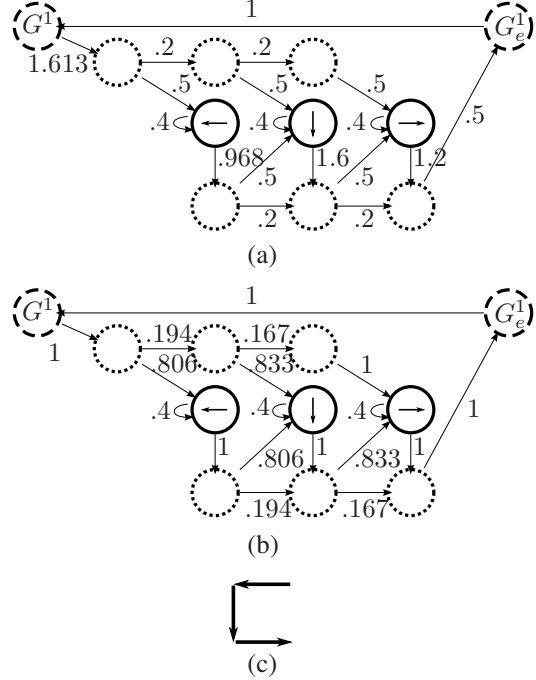


Figure 3. (a) An example SNM used for modeling of a single mouse gesture. The arrows within the production states indicate the mouse movement represented by the state. Semantic states G^1 and G_e^1 mark the beginning and end of the gesture, respectively. (b) The normalized version of the SNM. (c) The gesture that is modeled by the SNMs in a) and b).

Proposition 2.1 implies that the behavior of λ is equivalent to that of the normalized SNM $\lambda^* = (S, \pi, A^*, B)$. Figure 3(a) shows an example SNM that (very roughly) models the execution of a mouse gesture resembling the letter C (see Figure 3(c)). The consistent values in Figure 3(a), such as .5, .4, and .2, help specify how likely it is for the SNM to advance to the next state (.5), stay in the same state (.4), or skip a state (.2). The remaining values are used to ensure $\sum_{t' \in T \cup G} A^{aux}(t, t') = 1$ for every $t \in T \cup G$. Figure 3(b) shows the normalized version, in which A values have been replaced by A^* .

Although a normalized SNM is more convenient when simulating the execution of the model, keeping the SNM unnormalized can allow more efficient computation in training and inference. In particular, unnormalized SNMs can elegantly represent factorizations of state transition probabilities, as will be the case in the following section.

3. Problem Formulation

Our ultimate goal is to recognize gestures in a continuous stream of real-time observations. The term “gesture” can be interpreted as any underlying phenomenon that results in a relatively consistent sequence of observations. Depending on the scenario, it could be a particular movement of a per-

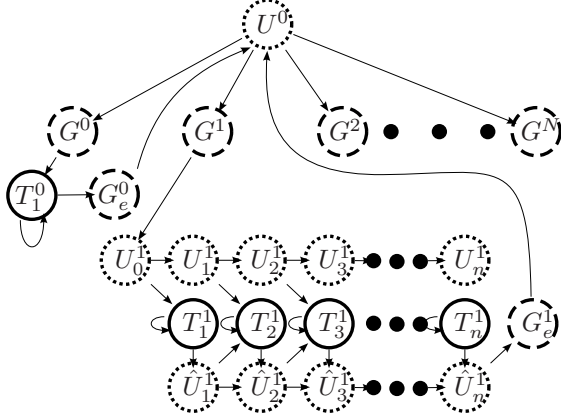


Figure 4. The Gesture Recognition SNM. Only states corresponding to non-gesture observations (G^0), and the first gesture (G^1) are shown in detail. n is the number of production states in gesture 1.

son’s body (a baseball umpire’s safe signal), the inscription of a symbol using a pen (a letter written in cursive), or a melody (the Liberty Bell March). The kinds of gestures we can recognize are determined by the observation set \mathcal{O} .

Suppose we want to recognize N different gestures, numbered 1 through N . We assume we are given examples of each of the gestures as training data. Each example is given as a tuple (L^i, M^i) , where $L^i \in 1 \dots N$ is the gesture number, and $M^i = M_1^i, M_2^i, \dots, M_{|M^i|}^i$, with each $M_j^i \in \mathcal{O}$, is the sequence of observations for that example. The goal is to find continuous parts of the real-time observation sequence O that correspond to the gestures. That is, we want to separate parts of O that are similar to an example gesture (instances of a gesture) from parts that are not.

We use a SNM $\lambda = (S = (T \cup G \cup U), \pi, A, B)$ to model the gestures as a gesture producing mechanism. To accomplish gesture recognition, we focus on two tasks. First, we train the SNM so that gestures it is likely to produce correspond to gestures we are trying to recognize. Then, we try to find the most likely sequence of states to have produced the observed observations. Isolating semantic states in this sequence corresponds to gestures most likely executed.

Figure 4 displays the general structure of the SNM. For $i = 1, \dots, N$, we define a semantic state G^i as the entry point into gesture i . All underlying states U_j^i , \hat{U}_j^i and T_j^i represent the execution of that gesture, while G_e^i marks its end. G^0 models non-gesture behavior - i.e. observations observed when no gesture is being expressed. This closely follows the structure of a HHMM, in which the starting and ending semantic states would correspond to a single abstract state containing the underlying production states. Although this arrangement does not exploit the non-hierarchical possibilities of semantic states, additional semantic states could be used for any points of interest in a particular application.

We simulate the SNM beginning in state U^0 by setting

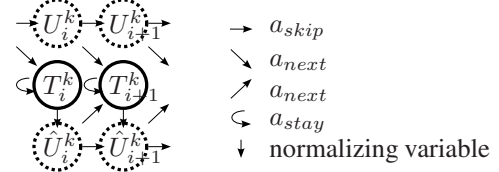


Figure 5. The transition function for gesture states of the Gesture Recognition SNM.

$\pi(t) = A^{aux}(U^0, t)$ for all $t \in T$. Transitions into each of the gesture entry states G^k have equal values, i.e. we consider each gesture to be equally likely. As Figure 4 indicates, each gesture is executed by going through its production states in a left to right fashion, which we model using a simplified form of the transition probabilities. These are determined completely by three constants, a_{stay} , a_{next} , and a_{skip} , which are illustrated in Figure 5. These constants, which we use to factorize state transitions, have intuitive meanings which are very appropriate for gestures.

a_{stay} determines the probability of staying in the same production state from one time step to another (directly, i.e. without going through any semantic states). A high value will allow recognition of gestures executed “slower” than the model (i.e., lasting more observation than the model has states). The probability of stepping to the next production state in the left-to-right state sequence is proportional to a_{next} , while the probability of stepping to the s^{th} next state is proportional to $a_{skip}^{s-1} a_{next}$. Intuitively, a_{skip} dictates the probability of skipping states, so a high value will allow gestures that have parts of the model missing to be recognized. This can occur either by omitting parts of the gesture, or executing the gesture “faster” than the model.

Using the state transition function determined by these constants, we can write for all k and $j \leq |G^k|$, where $|G^k|$ is the number of production states used to model gesture k ,

$$A^{aux}(T_i^k, T_j^k) = \begin{cases} 0, & \text{if } j < i \\ a_{stay}, & \text{if } j = i \\ \nu a_{skip}^{j-i-1} a_{next}, & \text{if } j > i \end{cases} \quad (1)$$

Here, ν is a normalizing variable ensuring that $\sum_{t' \in T \cup G} A^{aux}(T_i^k, t') = 1$. Similarly, $A^{aux}(T_i^k, G_e^k) = \nu a_{skip}^{|G^k|-i} a_{next}$.

For simplicity, we will use the above scheme in the discussions throughout this paper. There is, however, a slightly improved modeling scheme which assigns a different cost to a series of skipped states. The only difference is the introduction of another constant, a_{skip_one} , which determines the probability of skipping the first of a series of states. This changes the value for $A^{aux}(T_i^k, T_j^k)$ for the case $j > i$ to $\nu a_{skip_one} a_{skip}^{j-i-2} a_{next}$, with an appropriately modified ν . The transitions in Figure 5 can be easily revised to reflect this change. Although the altered scheme is only slightly more complicated, it leads to far better recognition.

4. Training and Inference

In this section, we will present the algorithms used to initialize the SNM from training data, recognize gestures in the observation sequence, and perform on-line learning.

4.1. SNM Viterbi Inference

One commonly addressed problem in the context of HMM is one of state estimation: given a sequence of observations O_1, O_2, \dots, O_n , what is the most likely sequence of states to have produced it? In our case, this problem is closely related to gesture recognition. Specifically, we will be finding the most likely sequence of both semantic and production states to have produced a sequence. If the most likely sequence of states includes the semantic states marking the beginning and end of a particular gesture, we can conclude that the gesture has been executed. Furthermore, we know when the gesture was started (going through G^i), and when it was ended (going through G_e^i).

For this purpose, we adapt the Viterbi algorithm, which is the usual way of finding the most likely state sequence in traditional HMMs. Formally, given a particular sequence of n observations O_1, O_2, \dots, O_n , we seek to find the most likely state sequence s_1, s_2, \dots, s_m of principal states $s_i \in T \cup G$ that would have produced it (since only production states produce observations, $m \geq n$ and there are exactly n production states in the state sequence).

We do so using a dynamic programming variable $\delta_i(s)$, which captures the probability of partial state sequence generating a partial observation sequence:

$$\delta_i(s) = \begin{cases} \max p(s_1, \dots, s, O_{1..i} | \lambda) & \text{if } s \in (T \cup G) \\ \max_{s_v, c \in C_{s_v, s}^{aux}} \delta_i(s_v) p(c) & \text{if } s \in U \end{cases} \quad (2)$$

For a semantic or production state s , $\delta_i(s)$ represents the maximum probability state sequence ending in s to have generated the observations. For auxiliary states, it is the highest possible value that can be obtained by starting with $\delta_i(s_v)$ at a principal state s_v , and multiplying the values along an auxiliary path in $C_{s_v, s}^{aux}$.

Even though the definitions are slightly different for principal and auxiliary states, the δ values are calculated quite similarly. It is always a function of the maximum δ value among all states that we can transition from:

$$\delta_i(s) = \begin{cases} \max \delta_{i-1}(s') A(s', s) B(O_i, s) & \text{if } s \in T \\ \max \delta_i(s') A(s', s) & \text{otherwise} \end{cases} \quad (3)$$

For brevity, we will sometimes omit the subscripts i and $i-1$, and replace $B(O_i, s)$ with $B^*(O_i, s)$ where $B^*(O_i, s) = B(O_i, s)$ if $s \in T$ and 1 otherwise. This yields

$$\delta(s) = \max \delta(s') A(s', s) B^*(O_i, s). \quad (4)$$

From (3), we can see that the δ values should be calculated in a certain order. Specifically, before calculating

$\delta_i(t)$ for production states $t \in T$ we should have calculated $\delta_{i-1}(s)$ for all states $s \in S$. Also, before calculating $\delta_i(s')$ for some $s' \in G \cup U$, we should have calculated $\delta_i(s)$ for each $s \in S$ such that $A(s, s') > 0$.

To determine the order of calculation, we define a sequence S_1, S_2, \dots, S_n of the non-production states in S to be a *propagation ordering* if the following are satisfied:

- for every $s \in G \cup U$, $s = S_i$ for exactly one i
- $(S_i, S_j) \in E \implies i < j$ for all $S_i, S_j \in G \cup U$.

In other words, the sequence is an ordering of non-production states in which the order is determined by the edges in the graph of the SNM.

Given a propagation ordering, the algorithm is simple. To initialize, we set $\delta_0(s) = \pi(s)$ for all $s \in T$. Given each O_i , $i = 1 \dots n$, we can first calculate the δ_{i-1} values for non-production states according to the propagation ordering, and finally the δ_i values for production states. In our gesture recognition SNM, one propagation ordering can be obtained by listing all \hat{U}_j^k ordered by k and j , followed by U^0 , G^k for all k , U_j^k ordered by k and j , and finally all T_j^k .

Proposition 4.1 *The Viterbi algorithm for the gesture recognition SNM requires $O(|S|)$ time for each time step.*

Sketch of proof: *There is only a small, constant number of states that satisfy $A(s, s') > 0$ for any state s' . Hence, $O(1)$ computation is required per state per observation, resulting in $O(|S|)$ time complexity for each time step. ■*

Please note that the general Viterbi algorithm for an HMM in which each state can transition to $O(|S|)$ states, which is the case in the gesture recognition SNM, would require $O(|S|^2)$ time to calculate the corresponding partial probabilities for each time step.

4.2. Training, Recognition, and On-Line Learning

The training of our gesture recognition SNM is greatly simplified because the state transitions are completely determined by a small number of parameters. Also, because of the increased precision of the modeling, a state's observation probability distribution can be modeled effectively by a single Gaussian rather than a mixture of Gaussians.

The training needs to focus on determining three things: the number of states used to model a particular gesture; the parameters for the observation probability distribution of each state; and the values for a_{stay} , a_{skip} , a_{next} . In HMMs, the number of states is usually given a priori. Although that would work in our case, we usually set it to the size of the smallest training sample ($\min_{M_i=g} |L^i|$) for each gesture g .

Since the other parameters can easily be obtained through the expectation-maximization (EM) algorithm, we will not go into detail except to mention that some care

needs to be taken when the number of training samples is very small. In this case, the variance of the observation probability distributions might not be reliably calculated, so we supplement the results of the EM algorithm with domain knowledge (e.g., a prior probability on the variances).

Once we have trained the SNM, we process the observation sequence O by invoking an iteration of the adapted Viterbi algorithm at each time step. Whenever $\delta_i(U^0) = G_e^k$ for some gesture $k = 1 \dots N$, we know that the sequence of observations O_1, \dots, O_i has most likely been generated by a sequence of states that ends with gesture k . We can backtrack through the calculation of $\delta(G_e^k)$, i.e. find the state that preceded it in the optimal state sequence, by noting the state s that was yielded the maximum value in the expression for $\delta(G_e^k)$. Applying this recursively, we eventually reach G^k , i.e. the start of the gesture, and determine the exact sequence of production states in between.

Once it has been detected that a gesture has been completed in the observation sequence, the traceback method tells which observation was generated by which state. This induces a mapping from the observations to the states of the model, which allows us to easily update the expectation of the parameters obtained in the training phase. This process of on-line learning using the recently recognized gesture is similar to the approach taken by Lee and Xu[6].

5. Experimental Results

We focus our experimental results on demonstrating two things. First, we compare the performance of our algorithm with the widely used hidden Markov model toolkit (HTK), and show that in scenarios that involve little training our algorithm has superior performance. Second, we measure the performance of our algorithm in distinguishing between two similar gestures with various amount of training data.

5.1. Comparison with HTK

For the comparison, we used the SNM model involving *a_skip_one* (see Section 4). In the HTK scenarios, we made the evaluation as fair as we could by translating affordances given to our algorithm into the HTK model. We modeled each gesture with a separate HMM, and connected the HMMs into a hierarchical structure that allowed the gestures to be recognized in any sequence. We also added a single state non-gesture model, which was trained using non-gesture data. Without these affordances, the performance of HTK was significantly worse.

The performance was compared in two scenarios - one scenario applies gesture recognition to a point in a plane, and the second scenario involves multimodal recognition of body gestures via a marker-based optical motion capture system and a pressure sensitive floor.

Gesture recognition of a point moving in a plane is in-

spired by many real-world scenarios. For example, consider cursive writing in which the pen never leaves the paper - the point of contact moves over the paper (plane), and individual letters can be regarded as gestures. Similar scenarios arise in novel interactive table systems in which the system can track objects moved across the surface of the table (e.g., using a video camera), as well as existing systems in which commands are executed using a mouse or a pen (e.g., [10]).

In our case, the gestures are expressed continuously using a Wacom tablet, where we tracked the pen’s direction. The direction is captured by the observation set defined by $\mathbb{D} = \{(x, y) | x^2 + y^2 = 1\} \cup \{(0, 0)\}$, with (x, y) being the unit vector in the direction of pen movement, or $(0, 0)$ if the pen is stationary. In line with our goal to detect gestures in a continuous sequence of observations, there is nothing to help determine the beginning and end of a gesture.

We used a set of six gestures, which are depicted in Figure 6. We chose gestures representative of several symbol sets - abstract symbols (first symbol depicted), a cursive approximation of a Chinese character, a symbol corresponding to a Latin character, and several gestures that could possibly correspond to various commands in an application (for example, “undo”, “save”, or “next document”).

For the multimodal body gesture recognition test cases, we used a marker-based optical motion capture system and a pressure sensitive floor to record the movement of a dancer executing 6 different gestures. The motion capture system was used to capture the movement of the upper body, while the pressure sensitive floor provided information about the weight distribution between the feet. Correspondingly, the gesture used for testing emphasised both the use of upper body movement and weight shifts.

Please note that each of these test sets presents its own challenges. In the point gestures, the dimensionality of the observation set is very small (1), making it more difficult to distinguish gestures. Even though they look significantly different drawn on paper, they are substantially more similar in the observation set. In the body gestures, the dimensionality of the observation set is larger (18), and each individual gesture is performed with much more intrinsic variability. A full body gesture with weight transfers tends to include variation in the sequencing of individual movements each time it is executed, making it difficult to capture the full range of possible gesture executions during training.

For the training phase in both sets, we collected a number of examples of each gesture. We trained the SNM gesture recognition systems using only two examples of each gesture. For HTK, we used either two or ten examples of each gesture. The number of states that each system used was also manipulated. The SNM systems were each trained with a *full* number of states, i.e. the number of states for a gesture was determined by the shortest training sequence for that gesture (as proposed in Section 4.2). This ranged from

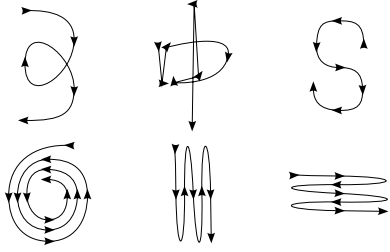


Figure 6. The six pen gestures used in system testing.

25 to 55 states for the point data, and 64 to 186 states for the body data. Half of the HTK scenarios were tested with the full number of states, while the other half was tested with each gesture given a fixed number of states (20).

Finally, each gesture recognition system was given three long, unsegmented test sequences. The first sequence contained 6 carefully executed instances of each gesture, for a total of 36 instances. The second sequence contained the same gestures, but executed quickly and carelessly (often much quicker than in the training sequences). In these two test sequences, the gestures were executed immediately one after the other, connected by only brief non-gesture data. The third test sequence contained only non-gesture data.

The results of the experiment are shown in Table 1. They represent a summary of the output of each of the systems, which was essentially a segmentation of each test sequence into gesture and non-gesture segments, with the gesture segments classified. The true recognition rate (TRR) was determined by the percentage of gestures that were correctly detected and classified. The number of false positives (FP) represents the number of detections that did not correspond to an intended gesture in the test sequence.

We notice that the performance of the HMM-based HTK drops significantly as we increase the number of states. This is presumably because the number of training observations per state decreases with an increased number of states. For the body gesture recognition, HTK was not able to train the models for most of the gestures when a full number of states was used, so we were not able to run the gesture recognition on any of the data sets for those scenarios.

This demonstrates a difficulty with typical HMM approaches in cases where a large number of states is necessary, which can happen with long or very detailed gestures. To illustrate this, consider our test gesture consisting of three counterclockwise circles. When executed with only 10 states per gesture, the HTK algorithm was unable to recognize the gesture correctly. Instead of one instance of the gesture, it would recognize three, because its model could only adequately represent one of the circles. If we wanted to model a gesture involving a long sequence of circles or other movements, even twenty states would not be enough.

Finally, we point out that the SNM results are on par with or better than the best HTK results (which required

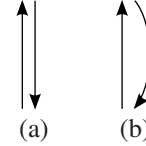


Figure 7. Two similar reaching gestures. (a) Reaching and retracting the arm in a straight trajectory. (b) Reaching in a straight trajectory, retracting in a curved trajectory.

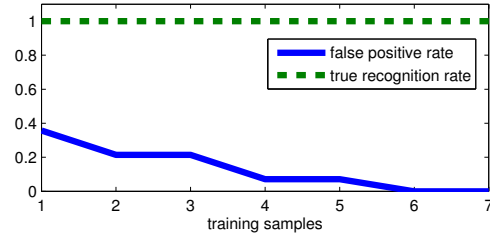


Figure 8. The impact of training on distinguishing similar gestures.

more training samples). This indicates that by simplifying the training phase, and taking advantage of an increased number of states made possible by the reduction of time complexity, we can achieve the same results as the typical HMM-based method and with less training. Furthermore, our approach greatly outperforms the HMM method when a large number of states is used, even when the HMM method is provided with a larger number of training sequences.

5.2. Distinguishing two similar gestures

Our second experiment uses two similar gestures involving reaching and retraction of the arm (see Figure 7). In gesture (a), the subject reaches forward with a straight trajectory and retracts with a straight trajectory. In gesture (b), the reach is straight but the retraction is curved. We recorded a sequence for each gesture, in which the gesture was executed 14 times. We then trained the system on gesture (a) by providing one instance of the gesture, and used on-line learning a variable number of times to improve the model. Finally, we tested the recognition (with further on-line learning disabled) on both the gesture (a) sequence to measure the true response rate, and the gesture (b) sequence to measure the false positive rate.

Figure 8 shows the results of the analysis. In all cases, the system correctly identified all 14 instances of gesture (a). However, after training only on the one provided training instance, the system gave 5 false positives when processing the gesture (b) sequence. When the system is allowed to see 5 additional instances of gesture (a) on which it uses on-line learning after recognizing them correctly, the number of false positives goes down to 0.

This shows that even when dealing with similar gestures and using only one training sample, the model can quickly learn to distinguish the gestures using on-line learning.

| | SNM | | HTK | | HTK | | HTK | | HTK | |
|---------------------|---------------|----|--------|----|--------|----|---------------|----|---------------|----|
| # of States | full (25-186) | | 20 | | 20 | | full (25-186) | | full (25-186) | |
| Training Samples | 2 | | 2 | | 10 | | 2 | | 10 | |
| Test Case | TRR | FP | TRR | FP | TRR | FP | TRR | FP | TRR | FP |
| point - Easy | 100.0% | 0 | 100.0% | 0 | 97.9% | 0 | 100.0% | 0 | 100.0% | 0 |
| point - Hard | 100.0% | 0 | 97.9% | 0 | 100.0% | 0 | 50.0% | 0 | 58.3% | 0 |
| point - Non-gesture | | 1 | | 4 | | 1 | | 0 | | 1 |
| body - Easy | 100.0% | 0 | 97.9% | 1 | 100.0% | 0 | 0.0% | 0 | 0.0% | 0 |
| body - Hard | 81.3% | 3 | 70.8% | 2 | 75.0% | 3 | 0.0% | 0 | 0.0% | 0 |
| body - Non-gesture | | 2 | | 1 | | 1 | | 0 | | 0 |

Table I. Results of point-based and full body gesture recognition, showing the true recognition rate (TRR) and number of false positives (FP) in each scenario. The performance of HTK degrades when using a large number of states.

6. Conclusions and Future Work

In this paper, we presented an initial treatment of semantic network models with an application in gesture recognition. SNMs allow a flexible way of incorporating semantic meaning through semantic states, and allow for a reduction in computational complexity through auxiliary states.

In the model we proposed for gesture recognition, both of these concepts prove helpful. Particularly, we were able to represent the model's state transitions using a handful of parameters, which reduces the time complexity of inference algorithms from $O(|S|^2)$ per observation to $O(|S|)$. The reduced number of modeling parameters greatly simplifies the training requirements, and the reduced computational complexity allows the use of much larger number of states to model a gesture. We experimentally showed that our approach is superior in scenarios that require many / long / detailed gestures, and involve minimal training.

In our future work and presentation of this material, our goal is to focus on experimental results in much more detail. For example, we plan to present detailed comparisons with traditional HMM-based methods in cases with much larger dimensionality of observation vectors, and more on-line learning scenarios. We would also like to further investigate and discuss the formal relationships between SNMs and other related approaches.

7. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No.0403428. and Grant No. 0504647. Additionally, we would like to thank the CVPR anonymous reviewers for their thoughts and suggestions. Their comments have helped us significantly improve our discussion, and are helping guide us in our future research. We would also like to thank the people of the Arts, Media and Engineering Program for making this research possible.

References

- [1] S. Aji and R. McEliece. The generalized distributive law. *IEEE Trans. on Information Theory*, 46(2):325–343, March 2000.
- [2] A. Corradini. Dynamic time warping for off-line recognition of a small gesture vocabulary. In *Proceedings of the IEEE ICCV/RATFG-RTS'01*, page 82, Washington, DC, USA, 2001. IEEE.
- [3] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.
- [4] J. B. Kruskal and M. Liberman. The symmetric time warping algorithm: From continuous to discrete. In *Time Warps, String Edits and Macromolecules*. Addison-Wesley, 1983.
- [5] F. Kschischang, B. J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47, 2001.
- [6] C. Lee and Y. Xu. Online, interactive learning of gestures for human/robot interfaces. In *1996 IEEE International Conference on Robotics and Automation*, volume 4, pages 2982–2987, 1996.
- [7] W. J. Masek and M. S. Paterson. A faster algorithm for computing string edit distances. *Journal of Computer and System Sciences*, 20:18–31, 1980.
- [8] K. P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division, 2001.
- [9] S. Rajko and G. Qian. A hybrid hmm/dpa adaptive gesture recognition method. In *ISVC*, pages 227–234, 2005.
- [10] Sensiva. Symbol commander [computer software]. <http://www.sensiva.com/products/index.html>, 2005.