Augmented Markov Models

(Continuing Draft)

Dani Goldberg and Maja J Matarić
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
{dani,mataric}@usc.edu

Abstract

This technical report presents augmented Markov models (AMMs), and provides detailed descriptions of their structure and one model construction algorithm. Augmented Markov models are essentially probabilistic transition networks similar to hidden Markov models (HMMs), except that the hidden state assumption is removed. Additional statistics (augmentations) are maintained in the links and nodes of AMMs and may be employed in model construction and utilization. The model construction algorithm we present is designed to have relatively low computational and space overheads, and provide useful models on-line and in real-time.

1 Introduction

An augmented Markov model (AMM) is essentially a finite state automaton with probabilities associated with transitions from state to state, similar to a Markov chain. AMMs may also be viewed as a degenerate form of hidden Markov model (HMM) (Rabiner 1989) in which the observation symbol probability in each state is 1.0 for a particular symbol and 0.0 for all of the remaining symbols. In other words, each state in the AMM recognizes (or generates) exactly one symbol. The removal of the observation symbol probability distributions effectively removes the hidden state assumption in HMMs. AMMs differ from a degenerate HMM in that they incorporate additional statistics in model links and nodes which may be used for dynamic model construction, modification, and exploitation.

In this paper, we present a discrete first-order form of AMMs that is able to model processes that sufficiently adhere to the following Markov property:

1. the state q_t at any point in time t is dependent only on the state at the previous time step. Thus, given two states s_i and s_j, the transition probability p_{ij} is given by:

$$p_{ij} = P[q_t = s_i | q_{t-1} = sj] \qquad (1)$$

2. the value of $P[q_t = s_i | q_{t-1} = s_j]$ is independent of the time t.

In the next section we present the structural components of AMMs.

1.1 Structure of AMMs

An AMM model is defined by the following five elements:

1. S, a set of symbols $\{s_1, s_2, \ldots, s_M\}$ recognized by the network.
2. A, a set of states (or nodes) $\{a_1, a_2, \ldots, a_N\}$. Each state a_i has four attributes:
   - a_i^s, the symbol that the state recognizes, i.e., an element of S;
   - a_i^μ, the average number of time steps that the system remains in a_i whenever it enters that state;
   - $a_i^{\sigma^2}$, the variance associated with a_i^μ;
   - and a_i^p, the probability of remaining in a_i in the next time step.

   A special state, $a_\emptyset$, represents the initial state of the system and possesses none of the four attributes above.
3. B, an $N \times M$ transition matrix, where $b_i(k)$ contains the value of the state to transition to if the current state is a_i and symbol s_k is observed. If $a_i^s = s_k$, then $b_i(k) = a_i$, i.e., if the observed symbol is identical to the last symbol observed, then the system remains in the current state.
4. L, a set of directed links $\{l_1, l_2, \ldots, l_P\}$, connecting the states. Each link l_i has the following six attributes:
   - l_i^f, indicates the state from which the link begins, $l_i^f \in A$;
   - l_i^t, indicates the state to which the link connects, $l_i^t \in A$. The following constraints apply: a link can not start and end at the same state, $l_i^f \neq l_i^t$; and two links from the same state can not go to states that accept the same symbol, $\forall i, j$ s.t. $l_i^f = l_j^f$, $a_{l_i^t}^s \neq a_{l_j^t}^s$;
   - l_i^δ, stores the number of times the link has been traversed;
   - l_i^Σ, stores the total number of time steps that the system has been in l_i^t, after first having traversing the link;
   - $l_i^{\Sigma 2}$, contains the sum of squares of all the durations that comprise l_i^Σ;
   - and l_i^p is the probability of using the link at each time step, given the system is in state l_i^f.

Because no two links can have the same value for both their *from* and *to* attributes, they can not represent the same directed transition. Thus, $N-1 \leq P \leq N(N-1)$: at least $N-1$ links are needed to connect the non-initial states, and for a fully connected network there are $N(N-1)$ links between the non-initial states. The link from $a_\emptyset$ is a special link designated by $l_\emptyset$, with $l_\emptyset^f = a_\emptyset$. $l_\emptyset$ is traversed exactly one time, and thus $l_i^\delta = 1$ and we may set $l_i^p = 0$.

5. T, a set of elements $\{t_1, t_2, \ldots, t_Q\}$, each storing information on a particular two-link traversal sequence entering and leaving a state. Each element t_i has five attributes:

   - t_i^e, the link $\underline{e}$ntering the state, where $t_i^e \in L$, except that there is exactly one t with $t^e = \emptyset$;
   - t_i^l, the link $\underline{l}$eaving the state, where $t_i^l \in L$ and $t_i^e \neq t_i^l$;
   - t_i^δ, the number of times the two-link combination has been traversed;
   - t_i^Σ, the total number of time steps that the system has been in the state that link t_i^l connects to, after first having traversed t_i;
   - $t_i^{\Sigma^2}$, the sum of the squares of all the durations that comprise t_i^Σ.

   The bounds of Q are given by:

$$\left.\begin{array}{ll} 0, & \text{if } P < 2 \\ P-1, & \text{if } P \geq 2 \end{array}\right\} \leq Q \leq N(N-1)^2.$$

   In order for a two-link transition to exist there must be at least two links. If more than two links exist, the fewest two-link transitions ($P-1$ of them) are created when an Euler path exists and is followed through the network. In a fully connected network, each of the $P = N(N-1)$ links has a transition to $N-1$ other links, giving us the upper bound. Additionally, there is one special two-link transition, designated $t_\emptyset$, with $t_\emptyset^e = l_\emptyset$ and $t_\emptyset^\delta = 1$.

Given this notation, a degenerate HMM of the type mentioned in the Introduction would be represented by S, a_i^s, a_i^p, B, and l_i^p. The other elements provide the augmentation necessary for model construction and utilization.

1.2 Generation of AMMs: Overview

The data used for constructing an AMM consist of a stream of symbols belonging to S. The structure of AMMs naturally lends itself to the low cost incremental construction algorithm we present now.

- Initialize the system by creating the initial state, $a_\emptyset$.
- If the current input symbol has never been seen before, add it to S and create a state that recognizes this symbol. Create a link from the current state to the new state and make the transition. Add this transition to B and create the corresponding new entry for T.

- If the current input symbol is the same as the last input symbol, then remain in the current state. Update the appropriate values in A, L, and T for mean values and length of time in the state. Recalculate the transition probabilities associated with that state and its links.

- If the current input symbol has been seen before, but is different from the last symbol, transition to a state that accepts the new symbol. If the link for this transition does not exist, create it.

- When transitioning from one state to another, update the variance $a_i^{\Sigma^2}$ for the state being transitioned from, and the appropriate sum of squares values in L and T.

- When about to transition from one state to another, calculate the binomial mean and variance for transitions in T with the same in-link. If the mean falls outside c times the standard deviation, where usually $2 \leq c \leq 3$, then *split* the current state, and attach the current in-link (with its associated out-links, as indicated in T) to the new state. Using T, make all appropriate changes to the two states and their related links, in order to keep all global probabilities consistent. Update T appropriately.

The above rules do not provide the complete semantics, but capture the general flavor of the model construction process. The final rule of the algorithm describes *node splitting* and deserves further explanation. Since an AMM is constructed incrementally as training data become available, it is important that there be some mechanism for model modification when new data invalidate the current structure of the model. This mechanism is provided by node splitting which utilizes data from T and many of the statistics maintained in the AMM, ensuring that the model remains consistent with the training data.

Note that there is relatively little computation involved in model construction. The computational complexity per input symbol is at most $O(N^2)$ (for a fully connected network) when a state is being split, and at most $O(N)$ otherwise. In addition, the space complexity is $O(N^3)$ for N fully connected states. In practice, the space complexity would probably tend to fall between $O(N)$ and $O(N^2)$ for something less than a fully connected graph. In the next section, we present the details of the model construction algorithm.

2 Generation of AMMs: Details

In the pseudo-code that follows, the five elements of the AMM being constructed (i.e., S, A, B, L, T) are global variables. The three variables 'Alast,' 'Llast,' and 'Tlast' are also global and function as indices to the last valid element of A, L, and T, respectively.

2.1 Main Loop

Below we present the pseudo-code for the main loop of the algorithm that is executed for every input symbol. At the start of the code, the variable 'sym' already holds the current input symbol, and 'oldsym' holds the last input symbol. In addition, 'numsym' contains the number of symbols observed in the current state, 'oldnode' stores the index of the last state the system was in, 'currnode' stores the index of the current state, 'inlink' holds the index of the link traversed to enter the current state, and 'outlink' holds the index of the link to be traversed in leaving the current state.

```
1.  if (oldsym == sym)
2.      numsym = numsym + 1;
3.      $l^{\Sigma}_{\text{inlink}} = l^{\Sigma}_{\text{inlink}} + 1$;
4.      $t^{\Sigma}_{\text{Told}} = t^{\Sigma}_{\text{Told}} + 1$;
5.      traversal_prob(currnode);
6.  else
7.      $l^{\Sigma^2}_{\text{inlink}} = l^{\Sigma^2}_{\text{inlink}} + \text{numsym}^2$;
8.      node_prob(currnode);
9.      $t^{\Sigma^2}_{\text{Told}} = t^{\Sigma^2}_{\text{Told}} + \text{numsym}^2$;
10.     if (sym $\notin S$)
11.         $S = S \cup \{\text{sym}\}$;
12.         Alast = Alast + 1;
13.         $a^s_{\text{Alast}}$ = sym;
14.         for $i$ = 1 to Alast
15.             $b^s_i(\text{sym})$ = Alast;
16.         end
17.         $\forall i \mid i \in S, b_{\text{Alast}}(i) = b_1(i)$;
18.     end
19.     oldnode = currnode;
20.     currnode = $b_{\text{oldnode}}(\text{sym})$;
21.     outlink = $i \mid (l^f_i$ == oldnode & $l^t_i$ == currnode);
22.     if ($\neg\exists$ outlink)
23.         Llast = Llast + 1;
24.         $l^f_{\text{Llast}}$ = oldnode;
25.         $l^t_{\text{Llast}}$ = currnode;
26.         outlink = Llast;
27.     end
28.     numsym = 1;
29.     $l^{\delta}_{\text{outlink}} = l^{\delta}_{\text{outlink}} + 1$;
30.     $l^{\Sigma}_{\text{outlink}} = l^{\Sigma}_{\text{outlink}} + 1$;
31.     traversal_prob(oldnode);
32.     $x_1 = i \mid (t^e_i$ == inlink & $t^l_i$ == outlink);
33.     if ($\neg\exists\ x_1$)
34.         Tlast = Tlast + 1;
35.         $t^e_{\text{Tlast}}$ = inlink;
36.         $t^l_{\text{Tlast}}$ = outlink;
37.         $x_1$ = Tlast;
38.     end
39.     $t^{\delta}_{x_1} = t^{\delta}_{x_1} + 1$;
40.     Told = $x_1$;
```

```
41.     $t^{\Sigma}_{\text{Told}} = t^{\Sigma}_{\text{Told}} + 1$;
42.     outlink = do_node_split(oldnode, inlink, sym, outlink);
43.     inlink = outlink;
44.  end
```

2.2 Calculating Traversal Probabilities

The following pseudo-code function calculates the traversal probabilities associated with a particular node and updates the appropriate statistics in the node and its links.

```
function ans = traversal_prob(node)
1.  $x_1 = \{i \mid l^t_i == \text{node}\}$;
2.  $n_1 = \sum_{\text{all } i \in x_1} (l^{\Sigma}_i - l^{\delta}_i)$;
3.  $x_2 = \{i \mid l^f_i == \text{node}\}$;
4.  $n_2 = \sum_{\text{all } i \in x_2} l^{\delta}_i$;
5.  if $(n_1 + n_2 \neq 0)$
6.      $a^p_{\text{node}} = \frac{n_1}{n_1+n_2}$;
7.      $l^p_{x_2} = \frac{l^{\delta}_{x_2}}{n_1+n_2}$;
8.  end
```

2.3 Calculating Node Probabilities

The following function updates the mean and variance for the particular node passed as a parameter.

```
function ans = node_prob(currnode)
1.  $x_1 = \{i \mid l^t_i == \text{currnode}\}$;
2.  $n_1 = \sum_{\text{all } i \in x_1} l^{\delta}_i$;
3.  $n_2 = \sum_{\text{all } i \in x_1} l^{\Sigma}_i$;
4.  mean = $n_2/n_1$;
5.  $a^{\mu}_{\text{currnode}}$ = mean;
6.  $n_3 = \sum_{\text{all } i \in x_1} l^{\Sigma^2}_i$;
7.  if $(n_1 > 1)$
8.      $a^{\sigma^2}_{\text{currnode}} = (n_3 - 2 * \text{mean} * n_2 + n1 * \text{mean}^2)/(n_1 - 1)$;
9.  end
```

2.4 Node Splitting

This function determines whether node splitting is necessary, and if it it, splits the nodes and creates new links and two-link transitions as appropriate.

```
function ol = do_node_split(oldnode,inlink,sym,outlink)
1.  $x = \{i \mid l_i^f == \text{oldnode}\}$;
2.  $\forall i \mid i \in x,\ p_i = l_i^p$;
3.  $\forall i \mid i \in x,\ p_i = p_i / \left( \sum_{\text{all } i \in x} p_i \right)$;
4.  $y = \{i \mid t_*^e == \text{inlink}\}$;
5.  flag = 0;
6.  for j = 1 to $|y|$
7.      $m = p_{t_{y_j}^l} * l_{\text{inlink}}^\delta$;
8.      $s = \sqrt{m * (1 - p_{t_{y_j}^l})}$;
9.      if $((|t_{y_j}^\delta| - m) > 2 * s)$
10.         flag = 1;
11.     end
12. end
13. if (flag == 1)
14.     Alast = Alast + 1;
15.     $a_{\text{Alast}}^s = a_{l_{\text{inlink}}^t}^s$;
16.     $\forall i \mid s_i \in S,\ b_{\text{Alast}}(i) = b_{l_{\text{inlink}}^t}(i)$;
17.     $l_{\text{inlink}}^t = \text{Alast}$;
18.     $b_{l_{\text{inlink}}^f}(a_{\text{Alast}}^s) = \text{Alast}$;
19.     $b_{\text{Alast}}(a_{\text{Alast}}^s) = \text{Alast}$;
20.     for j = 1 to $|y|$
21.         Llast = Llast + 1;
22.         $l_{\text{Llast}}^f = \text{Alast}$;
23.         $n = t_{y_j}^l$;
24.         $l_{\text{Llast}}^t = l_n^t$;
25.         $l_{\text{Llast}}^\delta = t_{y_j}^\delta$;
26.         $l_n^\delta = l_n^\delta - l_{\text{Llast}}^\delta$;
27.         $l_{\text{Llast}}^\Sigma = t_{y_j}^\Sigma$;
28.         $l_n^\Sigma = l_n^\Sigma - l_{\text{Llast}}^\Sigma$;
29.         $l_{\text{Llast}}^{\Sigma^2} = t_{y_j}^{\Sigma^2}$;
30.         $l_n^{\Sigma^2} = l_n^{\Sigma^2} - l_{\text{Llast}}^{\Sigma^2}$;
31.         $x = \{i \mid t_i^e == n\}$;
32.         $z = \{i \mid t_i^l == n\}$;
33.         $r = t_{y_j}^\delta / \left( \sum_{\text{all } i \in z} t_i^\delta \right)$;
34.         $t_{y_j}^l = Llast$;
```

```
35.         for $k = 1$ to $|x|$
36.             Tlast = Tlast + 1;
37.             $t^{e}_{\mathrm{Tlast}}$ = Llast;
38.             $t^{l}_{\mathrm{Tlast}} = t^{l}_{x_k}$;
39.             $t^{\delta}_{\mathrm{Tlast}} = \mathrm{round}(t^{\delta}_{x_k} * r)$;
40.             $t^{\delta}_{x_k} = t^{\delta}_{x_k} - t^{\delta}_{\mathrm{Tlast}}$;
41.             $t^{\Sigma}_{\mathrm{Tlast}} = \mathrm{round}(t^{\Sigma}_{x_k} * r)$;
42.             $t^{\Sigma}_{x_k} = t^{\Sigma}_{x_k} - t^{\Sigma}_{\mathrm{Tlast}}$;
43.             $t^{\Sigma^2}_{\mathrm{Tlast}} = \mathrm{round}(t^{\Sigma^2}_{x_k} * r)$;
44.             $t^{\Sigma^2}_{x_k} = t^{\Sigma^2}_{x_k} - t^{\Sigma^2}_{\mathrm{Tlast}}$;
45.         end
46.     end
47.     node_prob(Alast);
48.     traversal_prob(Alast);
49.     node_prob(oldnode);
50.     traversal_prob(oldnode);
51.     outlink = $i \mid (l^{f}_{i}$ == Alast & $l^{t}_{i}$ == $b_{\mathrm{Alast}}(sym))$;
52. end
53. ol = outlink;
```

2.5 Operations on AMMs

Aside from construction, there are two general operations that may be performed on a given AMM: generating data, and calculating the probability of a symbol sequence.

2.5.1 Data Generation

Data generation using a given AMM is analogous to simulating the data generating process that the AMM is modeling. If the AMM is a good model of that process, the original data and the generated data will "look" alike. The algorithm for data generation is as follows:

1. Begin at some initial state.
2. Generate the symbol that the current state accepts.
3. Select a transition to the next state according the probability distribution for the current state and all links leading from that state. Make the transition.
4. Return to step 2.

2.5.2 Probability of a Symbol Sequence

Given an AMM M and a sequence of symbols $O = \{O_1, O_2, \ldots, O_k\}$, where $O_i \in S$, we wish to calculate $P(O|M)$. This may be done in a straightforward manner using the following algorithm:

1. Repeat the following for each state $a_i \in A$ that accepts the symbol O_1, considering it the initial state:
   (a) Initialize the probability $p_i = 1$.
   (b) For each symbol from $O_2, ..., O_k$ in turn: make a transition from the current state to the state that accepts that symbol, multiply p_i by the probability of that transition, and set the current state to be the new state. If no such transition exists, then set $p_i = 0$.
2. $P(O|M) = \max_i p_i$.

3 Continuing Work

This technical report is a work in progress. As such, there are several sections yet to be added as well as possible elaborations of sections already present. Most of the issues that we are currently exploring involve the theoretical properties of our model construction algorithm. We are in the process of characterizing the subset of AMMs that the algorithm captures. This involves understanding the effects of maintaining statistics solely on two-link transitions, as opposed to, or in addition to, higher order transitions. There is also the issue of whether two AMMs generated using data from the same underlying process will converge to be functionally equivalent, and if so, under what conditions they will also be topologically equivalent. Updates of this technical report will periodically become available.

Acknowledgments The research reported here was conducted at the Interaction Lab, part of the Robotics Research Lab at the University of Southern California Computer Science Department. The work is supported by the Office of Naval Research Grant N00014-95-1-0759 and the National Science Foundation Infrastructure Grant CDA-9512448.

References

Gat, E. (1998), On Three-Layer Architectures, *in* D. Kortenkamp, R. P. Bonnasso & R. Murphy, eds, 'Artificial Intelligence and Mobile Robotics', AAAI Press.

Matarić, M. J. (1997), 'Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior', *Journal of Experimental and Theoretical Artificial Intelligence* **9**(2–3), 323–336.

Rabiner, L. R. (1989), 'A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition', *Proceedings of the IEEE* **77**(2), 257–285.