In this lecture we complete the explanation of Dynamic Time Warping (DTW), and discuss some of its problems. We will then transition to Hidden Markov Models (HMMs), in effect presenting HMMs as a variant of DTW.

9.1 Review of Dynamic Time Warping

**Note:** Some of the material in this section may be redundant with that in Lecture 8, but is included here for the sake of completeness.

Dynamic Time Warping is a technique for aligning two sequences of symbols. For speech recognition, the obvious application is aligning a captured utterance with an idealized template utterance. In general the two sequences are not of equal length, and the expansion or compression needed to align the sequences may not be uniform. For example, a template utterance may be spoken at a uniform rate with no disfluencies, whereas an actual utterance may dwell on some phones, move rapidly through others, omit a few, insert some extraneous phones (e.g. “uhm”), and repeat certain sections, as in stuttering.

DTW addresses the problem by finding alignment functions \( \phi_x(k) \) and \( \phi_y(k) \) for the two sequences \( x \) and \( y \), where \( x(\phi_x(k)) \) and \( y(\phi_y(k)) \) are considered to be matching points in the two sequences. The goal of aligning the two sequences is equivalent to finding the alignment functions that minimize the difference between the (distorted) sequences \( x(\phi_x(k)) \) and \( y(\phi_y(k)) \). We will sometimes use \( \phi_x \) to refer to \( x(\phi_x(k)) \) and \( \phi_y \) to refer to \( y(\phi_y(k)) \), for notational convenience; the distinction should be clear from context.

The idea of finding the difference minimizing alignment is very intuitive, but suffers from a significant pitfall. Any two sequences that share at least one common symbol can be distorted into perfectly matching sequences. Consider the sequences \( x = \{a, b, c, d\} \) and \( y = \{b, i, g\} \). With the alignment functions \( \phi_x = \{2, 2, 2, 2\} \) and \( \phi_y = \{1, 1, 1, 1\} \), \( b \) is compared with itself four times, resulting in a perfect match. Similarly, the words “we” and “you”, which are phonetic time reversals of each other, could be matched with an alignment function that reversed one relative to the other.

This problem is solved in DTW by placing constraints on the alignment paths, so that only “reasonable” warpings are allowed. The constraints are based on intuition about what seems reasonable and experimentation. Path constraints also reduce the computational complexity of the problem substantially. In the unconstrained case, there are approximately \( T^2 \) possible paths, where \( T \leq T_x + T_y \). With certain constraints, this exponential problem can be reduced to nearly \( O(T) \) complexity.

The DTW problem can be solved efficiently using dynamic programming techniques. Dynamic programming requires that a problem possess two properties: optimal substructure, and overlapping subproblems. Consider the alignment problem as searching for the optimal path through a lattice, where the \( x \) and \( y \) axes are the indices into the two sequences. Optimal substructure means that the optimal solution to one problem contains the optimal solution to the subproblems of which the larger problem is composed. That is, if the optimal path, \( p \), from \( A \) to \( B \), goes through \( C \), then \( p \) contains the optimal path from \( A \) to \( C \), as well as from \( C \) to \( B \). Overlapping subproblems would mean that the optimal path from \( A \) to \( C \) would also be part of some other optimal paths between other points.

The properties of optimal substructure and overlapping subproblems allow the DTW problem to be efficiently solved...
by recursion. We define the problem to require that we start at the beginning and finish at the end. That is \( \phi_x(1) = \phi_y(1) = 1 \), \( \phi_x(T) = T_x \) and \( \phi_y(T) = T_y \). We can define the score for the entire sequence (or, rather, pair of sequences) as

\[
D(T_x, T_y) \triangleq d(X, Y) = \min_{\phi_x, \phi_y} d(\phi_x(X), \phi_y(Y))m(k)
\]

Similarly, for any point \((i_x, i_y)\) in the lattice, we define the “score,” or distance, for that point as the minimum total weighted distortion for any path from \((1,1)\) to that point. That is

\[
D(i_x, i_y) \triangleq \min_{\phi_x, \phi_y, T} \sum_{k=1}^{T} d(\phi_x(k), \phi_y(k))m(k)
\]

under the constraints that \(\phi_x(T') = i_x\) and \(\phi_y(T') = i_y\). Next, we define a score for a path \(p\), of length \(L_p\), from \(i'\) to \(i\):

\[
d((i'_x, i'_y), (i_x, i_y)) = \min_{\phi_x, \phi_y, T} \sum_{\ell=0}^{L_p} d(\phi_x(T' - \ell), \phi_y(T' - \ell))m(T' - \ell)
\]

These three definitions give rise to the recursion by which we will use dynamic programming to optimize the path.

\[
D(i_x, i_y) = \min_{i'_x, i'_y} [D(i'_x, i'_y) + d((i'_x, i'_y), (i_x, i_y))]
\]

This last formula tells us that the optimal path to a given destination point can be found from all of the intermediate points from which the destination point can be reached, their distances from the origin, and the minimum-distance paths between the intermediate points and the destination point. This fact is the optimal substructure of the DTW problem. Using this relationship, we can build the dynamic programming recursion:

**Start** \(D(1, 1) = d(1, 1)m(1)\)

**Recursion:** \(D(i_x, i_y) = \arg \min_{i'_x, i'_y} D(i'_x, i'_y) + d((i'_x, i'_y), (i_x, i_y))\)

**end** \(D(X, Y) = D(T_x, T_y)\)

At each step of the recursion, when we are searching for the minimum-distance path between two points \((i'_x, i'_y)\) and \((i_x, i_y)\), we must search all of the paths that can get from \((i'_x, i'_y)\) to \((i_x, i_y)\). This search space is limited by the path constraints, so we only consider paths that are allowed by the chosen constraint. The path constraints also penalize certain types of steps deemed to by unlikely. The \(m\) term contains the step penalty. Table 1 shows some common path constraints, and the corresponding distance equations.

### 9.1.1 DTW and String Edit Terminology

DTW can be thought of as analogous to string editing, wherein the task is to find the minimum set of edits that will transform one string, called the “source” to another, the “target,” taken from the alphabets \(S\) and \(T\), respectively. Each string is made up of symbols taken from its respective alphabet and the null symbol, \(\varepsilon\). When using string edit terminology, we will refer to the unknown utterance as the source, and the template as the target. However, the set of
Table 9.1: Some path constraints and corresponding distance equations.

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Distance Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Constraint" /></td>
<td>(a) [ D(i_x, i_y) = \min \begin{cases} \ D(i_x - 1, i_y) + d(i_x, i_y), \ D(i_x - 1, i_y - 1) + 2d(i_x, i_y), \ D(i_x, i_y - 1) + d(i_x, i_y) \end{cases} ]</td>
</tr>
<tr>
<td><img src="image" alt="Constraint" /></td>
<td>(b) [ D(i_x, i_y) = \min \begin{cases} \ D(i_x - 2, i_y - 1) + \frac{1}{2}[d(i_x - 1, i_y) + d(i_x, i_y)], \ D(i_x - 1, i_y - 1) + 2d(i_x, i_y), \ D(i_x - 1, i_y - 2) + \frac{1}{2}[d(i_x, i_y - 1) + d(i_x, i_y)] \end{cases} ]</td>
</tr>
<tr>
<td><img src="image" alt="Constraint" /></td>
<td>(c) [ D(i_x, i_y) = \min \begin{cases} \ D(i_x - 2, i_y - 1) + 3d(i_x, i_y), \ D(i_x - 1, i_y - 1) + 2d(i_x, i_y), \ D(i_x - 1, i_y - 2) + 3d(i_x, i_y) \end{cases} ]</td>
</tr>
</tbody>
</table>

Table 9.2: Comparison of string edit operations with DTW operations

<table>
<thead>
<tr>
<th>String Edit Operation</th>
<th>Conditions</th>
<th>Step Vector ((d_x, d_y))</th>
<th>Lattice Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deletion</td>
<td>(s \neq \varepsilon, t = \varepsilon)</td>
<td>((d_x, d_y) = (1, 0))</td>
<td>right</td>
</tr>
<tr>
<td>Insertion</td>
<td>(s = \varepsilon, t \neq \varepsilon)</td>
<td>((d_x, d_y) = (0, 1))</td>
<td>up</td>
</tr>
<tr>
<td>Substitution</td>
<td>(s \neq t, s \neq \varepsilon, t \neq \varepsilon)</td>
<td>((d_x, d_y) = (1, 1))</td>
<td>diagonal</td>
</tr>
<tr>
<td>Match</td>
<td>(s = t \neq \varepsilon)</td>
<td>((d_x, d_y) = (1, 1))</td>
<td>diagonal</td>
</tr>
</tbody>
</table>

If we define a vector \((d_x, d_y)\) as the distance moved in the lattice for one step in the alignment functions \(\phi\),

\[
(d_x, d_y) = (\phi_x(k + 1) - \phi_x(k), \phi_y(k + 1) - \phi_y(k))
\]

then we can see how certain values of \((d_x, d_y)\) correspond to specific operations in string editing, as shown in table 9.2. The string editing task is similar to what is accomplished by the UNIX program \textit{diff}. Diff compares two files and outputs the differences and, optionally, a set of commands that can be used to create one from the other. It is frequently used as part of version control systems to store the incremental changes made to text files, e.g. source code. Diff also uses dynamic programming to minimize the number of required edits.

### 9.1.2 Comments on Dynamic Time Warping

Dynamic Time Warping is a reasonably efficient way to solve a difficult problem. Until the early 1980s it was the primary method used by speech recognition systems to match captured utterances to known ones. Its implementation is well-studied and relatively easy. DTW-based recognition systems get reasonable results for small vocabularies, e.g. in voice-activated phone dialers. Even with large differences in speaking rates, DTW attains reasonably accurate alignments.

However, DTW also suffers from some substantial shortcomings. In the simplest implementation, there would be only one template per utterance. Since a given phrase can differ in many ways other than speech rate, many utterances could not be matched to the correct template with any alignment. It would be possible to use multiple templates...
for each utterance, but having one template for every possible variation of a given utterance would start to resemble memorizing the entire training set.

The path constraints and weights are somewhat arbitrary, and are based on intuition, and lack a firm theoretical basis. It would be preferable to have a system where path weights and constraints were a function of the distribution of actual paths in the training data.

Additionally, it would be difficult to generalize the DTW system to continuous speech. As we saw in the first homework, it is not trivial to build a silence detector. Thus a speech recognition system should not require small, segmented pieces of speech.

Finally there is no mechanism in DTW to deal with uncertainty, or to utilize knowledge of the probabilistic structure of speech. Bayes decision theory is a well-developed body of theory applicable to speech recognition problems. It would be nice to have a formal probabilistic structure, which would enable us to leverage Bayes theory.

As we will see, Hidden Markov Models (HMMs) addresses these shortcomings.

9.2 From DTW to HMMs

9.2.1 History

In the early 1980s, the research of Jelinek, Baker, Rabiner, Poritz and others, led to the replacement of DTW with HMMs. None of these people invented HMMs, but they were the first to apply them to the speech recognition problem. Exactly who invented HMMs is not completely clear, but it appears that Claude Shannon was the first to mention the idea, in a paper discussing the “identifiability problem” in Markov chains.

9.2.2 Notation

In these section we will use $x$ to denote the source, or unknown utterance, and $y$ to denote a target template, as we make the necessary modifications to DTW to transition to HMMs. We will refer to the length of the template as $N$, instead of $T_y$. We will refer to the template frames as states, labeled $q_1, q_2, \ldots, q_N$, so we are aligning the source (unknown utterance) with a sequence of states.

We will also modify the path constraints. Purely vertical movements through the lattice will no longer be allowed, with the result that a symbol from the unknown utterance must be consumed at every step. This constraint also has the effect of requiring the unknown utterance to be at least as long as the template. Thus, many shorter templates can be disqualified early, saving computation. Additionally, all steps move exactly one space in the x direction, so no symbols from the unknown utterance are skipped. It should also be noted that movement in the y direction is not constrained to be upwards; that is, we can move backwards through the template. Figure 1 shows two valid HMM constraints, as well as one invalid constraint.
9.2.3 DTW,FSA,HMM,etc

With the modifications to terminology and to the path constraints, we can now see DTW as a type of Finite State Automata (FSA). The path constraints correspond to allowable state transitions. Note that the constraints need not be the same at each point in the lattice. Correspondingly, each state has its own set of allowable “next-states”, to which the FSA can transition. Horizontal movement in the lattice corresponds to a self-transition, where the FSA stays in the same state. Figure 2 shows the relationship between FSA and DTW.

---

Figure 9.1: Two valid HMM path constraints (a),(b), and one invalid path, (c). The constraint in (c) would allow vertical steps, which would consume no symbols from the unknown utterance.

Figure 9.2: On the left is a lattice representation of DTW. On the right is an FSA representation of the same DTW problem. The solid lines indicate the possible state transitions under the path constraint shown. The dotted line in the FSA diagram shows one possible state sequence, corresponding to the dotted line in the lattice.
The weights in the path constraints correspond to a penalty for each possible state transition. Transition penalties can be assembled into a transition cost matrix, which need not necessarily be constant in time. The cost matrix representation of allowable and preferable transitions is a much more flexible way to represent the possible transitions than the simple path constraints, because each state $q$ (corresponding to a symbol in the template) can have its own set of allowable “next-states.” Figure 3 shows an FSA diagram, annotated with transition costs. The corresponding transition cost matrix would be:

$$ W = \begin{bmatrix}
1 & 3 & \infty & \infty & 5 \\
\infty & 2 & \infty & 4 & \infty \\
\infty & 6 & 1 & \infty & \infty \\
\infty & \infty & 7 & 2 & 3 \\
\infty & \infty & \infty & \infty & 1 
\end{bmatrix} $$

If we think of states as representing phones, then the transition matrix can be used to encode typical flow of speech. For example, vowels tend to last longer than consonants, so vowel states could be given a stronger preference for staying in the same state.

We have been trying to establish equivalencies between DTW, string editing, and FSA. Given that the FSA model allows downward transitions in the lattice (i.e., moving backwards in the template), one might wonder what string edit operation would correspond to downward motion. Here, the equivalence begins to break down. There really is no “move-backward” operation in string editing.

It is also worth noting that the allowance for backward motion means that the alignment function (plotted as $\phi_y$ vs $\phi_x$) could be non-monotonic. It would also be possible to skip a state. While somewhat counter-intuitive, this does not violate the rules of FSAs (or HMMs) and does not hurt accuracy. The weight matrix, if correctly calculated, will apply the appropriate penalty to backwards movement or skipped states.

In figure 3, there are some states with no connection between them, corresponding to transitions that are not allowed. These are represented as infinite entries in the transition matrix. However it is not necessarily impossible to move between such states in multiple steps. The Chapman-Kolmogorov algorithm determines the cost of eventually getting from one state to another, even when there is no allowable direct transition, using transition probabilities instead of transition costs. (We will discuss the relationship between transition cost matrices and transition probability matrices in the next section.) Given a transition probability matrix $A$, one can compute a matrix $A_N = A^N$, where $A_N(i, j)$ is the probability of getting from state $i$ to state $j$ in $N$ steps. The Chapman-Kolmogorov algorithm finds the limit as $N$
approaches infinity of the transition probability matrix $A_N$. For more details, see [P84].

### 9.2.4 Probabilistic Interpretation of DTW

As a measure of distortion between the source sequence $x$ and the template sequence $y$, we can use the Mahalanobis distance $d_M(x, y) = (x - y)^T A (x - y)$. The Mahalanobis distance can be thought of as a weighted version of the Euclidean distance, and, in fact, when $A$ is the identity matrix, it reduces to the squared Euclidean distance. More interestingly, if we consider $x$ to be a multivariate Gaussian with mean $y$ and covariance matrix $A^{-1}$, then we can relate the Mahalanobis distance to the probability of $x$.

$$ p(x) = \frac{1}{|2\pi A|^\frac{1}{2} e^{\frac{1}{2} (x-y)^T A (x-y)}} $$

(9.1)

$$ d_M(x, y) = -\log(p(x)) + c $$

(9.3)

The constant $c$ is unrelated to either $x$ or $y$, and, since we are typically more interested in the comparative likelihood of two strings than in the absolute likelihood of an individual string, a constant offset is not important. The significance of this relationship is that we have formalized a way to incorporate state-conditional gaussians into our transition cost matrix.

Now we can replace the transition cost matrix, $W$, of DTW, with a probability matrix, also known as the stochastic transition matrix (STM), $A$. The relationship between $W$ and $A$ is straightforward and intuitive. Because we want to choose likely paths over unlikely paths, high costs correspond to low probabilities, with the extreme cases being infinite cost, or zero probability. In both cases, the row corresponds to the state at a given time, and the column corresponds to the state at the next time step. So each row in $A$ is a discrete probability distribution of the possible next states. As such, each row must sum to unity. The relationship between $W$ and $A$ is expressed in these equations:

$$ A = [a_{ij}] \quad W = [w_{ij}] $$

(9.5)

$$ a_{ij} = e^{-w_{ij}} \sum_{j'} e^{-w_{ij'}} $$

(9.6)

$$ \forall i, \sum_j a_{ij} = 1 $$

(9.7)

$$ w_{ij} = \inf \Leftrightarrow a_{ij} = 0 $$

(9.8)

### 9.2.5 State Duration

Remember that the whole purpose of DTW was to address the fact that equivalent utterances spend different amounts of time on each phone. So we should look at how well our probabilistic model models phoneme duration, or more generally state duration. The diagonal entries $a_{ii}$ give the probabilities that the SFSA will stay in a given state. The probability that the SFSA will stay in a given state $i$ for $N$ time steps is given by:


\[ P(Dur_i = N) = a_{ii}^{N-1}(1 - a_{ii}) \]

which is a geometric distribution. The geometric distribution has the “memoryless” property. That is, the probability of a state change is independent of how long a given state has been active. Actually, since HMMs only keep track of the current state (as opposed to a history of states), any distribution that arose from an HMM would have to have the memoryless property. It is worth asking if the geometric distribution is a good model of phone duration. The answer, unfortunately, is no. Probability as a function of duration decreases monotonically under the geometric distribution, whereas phones have been empirically shown to have a non-zero most-likely duration.

The negative binomial distribution is a better model of phone duration. It models the sum of a fixed number of independent random variables drawn from identical geometric distributions. So a phone represented by a sequence of \( N \) states, each with a self-transition probability of \( s \), then the total duration of that phone would be distributed as \( NB(N, s) \). See [JL96] for more details on the negative binomial distribution.

Another potential criticism of geometric distributions for phone duration is that there is no absolute maximum duration, since arbitrarily large numbers have non-zero probability under a geometric distribution. So there is a real (albeit small) probability that a phone could have an exorbitantly long duration. In practice, however, this is not a problem. Geometric distributions die off quickly, so the non-zero probability in the tails is not enough to effect decisions.

Q: How long could HMMs be? Ex: Hello vs. Helsinki A: Templates are not really used, but similar parts of words do in effect share pseudo-templates.

### 9.2.6 State Sharing

We have been discussing word recognition in terms of matching an unknown utterance with a template. Such a framework would imply the need for a separate template for every known word, or potentially even for every possible word sequence, which would quickly become infeasible for unrestricted vocabularies. HMMs do not actually require a template, per se. Words that share common phones share states. For example, the words “hello” and “Helsinki” would have the same first few states.

Another benefit of HMMs is that multiple words can share common states.

### 9.2.7 Stochastic Outputs of SFSAs

What we have described so far is known as a Markov chain. Markov chains model processes where the probability of transition from one state to another depends only on the current state, and where the states are directly observable. For the problem of speech recognition, we can think of the state as a phoneme, and the state sequence as a word or phrase. One might point out that if we could directly observe the phonemes, the speech recognition problem would be nearly solved. Unfortunately, we cannot. So we will make another modification to the model. We will create a variable \( X \), to denote a vector of observable output variables, e.g. MFCCs. Then each state will be associated with a different distribution for \( X \). Figure 4 shows a depiction of this arrangement.
Now we have two components of uncertainty in our model: the stochastic nature of the state transitions, and the
probability distribution of outputs for each state. The question “Given what we know about the likelihood of various
state transitions, and what we know about state-specific output distributions, and what we have observed, what is the
most likely state sequence?” can be captured mathematically like this:

\[ P(x_1:7, \text{path}) \triangleq P(x_1, x_2, x_3, x_4, x_5, x_6, x_7, \text{path}) = P_{q_1}(x_1)P_{q_1}(x_2)P_{q_2}(x_3)P_{q_2}(x_4)P_{q_3}(x_5)P_{q_3}(x_6)P_{q_3}(x_7)a_{11}a_{12}a_{22}a_{23}a_{34}a_{45} \]  

(9.10)

(9.11)

Actually, not. Fix this. And this is specific to the picture on page 4 slide 3.

In DTW, there was a symmetry in that the source and target strings were interchangeable. We no longer have that
symmetry. Frames of the unknown utterance are used exactly once each, whereas template “frames” (states) are used
multiple times (or possibly none at all). Recalling the two relations,

\[ a_{ij} = \frac{e^{-w_{ij}}}{\sum_{j} e^{-w_{ij}'}}, \]

\[ P_{q_i}(x) = e^{-d(x,x_{q_i})} + c \]

we can compute the joint probability of a path and our observations as

\[ -\log P(x_1:T_X, \text{path}) = \sum_{k=1}^{T_X} [d(x_i, y_{q_{i+1}}) + w_{q_i,q_{i+1}}] + C \]

Compare this with the path distance from DTW:

\[ d_{\phi}(X, Y) = \sum_{k=1}^{T} [d(\phi(k), \phi_y(k))m(k)] \]

The \( d(\phi(k), \phi_y(k)) \) term from DTW and the \( d(x_i, y_{q_i}) \) term from HMM both correspond to the difference between
the “ideal” output for a given state, or template frame, and the observed utterance. (Remember that \( y_{q_i} \) is the expected
output for the state $q_i$.) The transition probability/cost is encoded in the $m(k)$ term for DTW, and in the $w_q,q_{i+1}$ term for HMMs. Note that we used the transition cost, $w_{ij}$ here, not the transition probability $a_{ij}$ here, so we have a bit of a mixed representation, but it shows the correspondence between HMMs and DTW nicely. The additivity of transition probabilities in HMMs comes from combining probabilities in the log domain, where multiplication is transformed into addition.

The structure we have described thus far is a Hidden Markov Model. Some of the differences between HMMs and DTW are summarized in Table 9.3.

<table>
<thead>
<tr>
<th>Feature</th>
<th>DTW</th>
<th>HMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition weights</td>
<td>Multiplicative</td>
<td>Additive</td>
</tr>
<tr>
<td>Transition weights</td>
<td>Slope Penalty</td>
<td>Transition Probability</td>
</tr>
<tr>
<td>Path through unknown utterance</td>
<td>Can skip unknown frames</td>
<td>Must absorb every frame*</td>
</tr>
<tr>
<td></td>
<td>with two-stage moves.</td>
<td></td>
</tr>
<tr>
<td>Transition weight representation</td>
<td>Path constraints, typically uniform.</td>
<td>Transition probability matrix, entries specific to each possible transition.</td>
</tr>
<tr>
<td>Sequence Length Constraints</td>
<td>Imposed by path constraints.</td>
<td>Template cannot be shorter than the source. Might not allow utterance to be shorter than the number of states.</td>
</tr>
<tr>
<td>Probabilistic Interpretation</td>
<td>Poorly modeled through path constraints</td>
<td>Transition and output probabilities provide good interpretation.</td>
</tr>
</tbody>
</table>

*The requirement that HMMs absorb every frame, like some of the other HMM “rules,” can be circumvented with the clever/kludgy addition of extra states.

The structure of HMMs imposes a number of constraints on how sequences may be matched. Most notable among these are the requirements that the unknown utterance may not be longer than the template, and that utterance frames cannot be skipped. The flexible nature of HMMs allows the designer to “cheat”, or circumvent these requirements through the addition of extra states. For example, to deal with unknown frames in the utterance (perhaps corresponding to noise), one could add a state with a high-variance output distribution (so that widely variant outputs are interpreted as equally probably), that can be reached from any other state. Then the unknown frame would not technically be skipped, but it would also not be forced into a bad match with a meaningful template frame. Similarly, a template shorter than the unknown utterance could be allowed to fit by adding some dummy states.

### 9.2.8 Probability, Bayes and HMMs

HMMs provide a nice way to interface with Bayes decision theory. The joint probability of the path and the outputs sum over all possible paths and all possible outputs to unity, as required of a legitimate probability distribution.

$$\int_X \sum_{AllPaths} P(X, Path) = 1$$

Additionally, the probability of an output sequence can be considered to be the joint probability of the output sequence and the path, marginalized over all paths:

$$P(X) = \sum_{allpaths} P(X, path)$$
This fact allows us to use Bayes’ rule to find the best, or most probable, path:

\[ W^* = \arg\max_W \, P(X|W)P(W) \]

For comparison, remember the best, or minimum cost, path in DTW was defined as:

\[ d(X, Y) = \min_{\phi} \, d_\phi(X, Y) \]

Under the HMM model, the best path, is defined as the most probably path, also known as the Viterbi path.

\[ P^*(X, \text{path}) = \max_{\text{allpaths}} \, P(X, \text{path}) \]

\[ \text{path}^* = \arg\max_{\text{allpaths}} \, P(X, \text{path}) \]

The Viterbi score of a path is the probability of that path, and the Viterbi path is the path with the highest Viterbi score. We have so far been treating the transition probabilities as constant over time, known as the time-homogeneous case. HMMs need not necessarily be time-homogeneous. In the time-inhomogeneous case, the transition probability matrix \( A \) would be a function of time.

### 9.2.9 Representations of HMMs

We have already seen two ways of graphically depicting an HMM: as a lattice, similar to a DTW representation, and as a Markov chain with stochastic outputs. The key elements of an HMM are:

- Stochastic state variable, \( q \).
- Outputs \( x_i \) are a stochastic function of the state \( q \).
- The next state is a stochastic function of the current state.
- Given the current state, the next state is independent of everything else.

The fourth point is one of the most important aspects of an HMM, for it greatly simplifies the computation. It is sometimes expressed as “Given the present, the future is independent of the past.” Another way to express it would be, “Everything you need to know about the future is contained in the present.” It is worth emphasizing that this does not mean that the future (state sequence) is completely independent of the past. The future depends on the present, which depends on the immediate past, which in turn depends on the past one step farther back, and so on. In this way, the future is dependent on the entire past, but that dependence is completely contained in the current state. The dependence relationship among states is clarified by a third representation: the Bayesian network, shown in figure 5. While the Bayesian network and the SFSA (or Markov chain) representations both have circles and arrows, one should be careful not to confuse them; the pictorial elements in the two representations do not have a consistent meaning across representations. For example, there can be no self-loops in a Bayesian network, because that would simply indicate that a variable depended on itself. In the Bayesian network, each circle represents a variable, in this case, the state or output at a given time. Each arrow represents a dependence. So a variable is directly dependent only on those variables which point to it. However, a variable is indirectly dependent on any variable indirectly connected to it. In the figure, \( X_{13} \) is dependent on \( Q_1 \), as shown by the flow of arrows, but that dependence is contained in \( X_3 \)'s dependence on \( Q_3 \), so we can say that given \( Q_3 \), \( X_3 \) is independent of everything else. Note that a variable is not independent of other variables “downstream” from it. For example, in the figure, knowledge of \( Q_3 \) would provide information about what values of \( Q_1 \) could have led to the known state at time \( t = 3 \), so \( Q_1 \) is not independent of \( Q_3 \).
Each of the three representations have advantages and disadvantages. The SFSA, or Markov chain, representation shows the underlying model well, and clearly indicates any impossible transitions by the absence of an arrow, as well as showing which transitions are likely and which states tend to have longer durations (assuming the transition probabilities are annotated on the graph). However, SFSA graphs are poor for indicating time-inhomogeneity. They also show little about the unknown utterance.

The lattice depiction, like we used for DTW, shows less about the underlying model but more about the match between a template and an unknown utterance. We can see what sort of distortion was applied from the shape of the path. Additionally since time is represented as increasing along the x axis, lattice representations can more conveniently be annotated with information about time-inhomogeneity. Lattice depictions lack a convenient way to show the stochastic nature of the outputs.

The Bayesian network depiction shows little about a specific model, but it does clearly show some very important aspects of HMMs in general. The dependence relationship is shown. Network graphs also include the outputs, unlike lattice depictions. Network graphs, like SFSA graphs, do not show time-inhomogeneity well. Bayesian network graphs also show the factorization or probabilities. For example, the graph in figure 5 would correspond to the probability equation:

\[
p(x_1:T, q_1:T) = p(x_1|q_1)p(q_1) \prod_{t=2}^{T} p(x_t|q_t)p(q_t|q_{t-1})
\]

Note that this equation assumes time-homogeneous transition probabilities, since the distributions are not a function of time. We could rewrite the equation under a time-inhomogeneous assumption as:

\[
p(x_1:T, q_1:T) = p_t(x_1|q_1)p(q_1) \prod_{t=2}^{T} p_t(x_t|q_t)p_t(q_t|q_{t-1})
\]

The fundamental assumption underlying HMMs, that given the current state, the current and future outputs, and future states are independent of everything else can be formalized as follows:

\[
\{Q_1:t-1, X_1:t-1\} \perp \perp \{Q_{t+1:T}, X_{t:T}\}|Q_t
\]
\[
\text{and } X_t \perp \perp \{Q_{t}, X_{\neq t}\}|Q_t
\]

An HMM can be defined as a set of \( T \) discrete random variables (the states), and another set of \( T \) random variables (the outputs, which may be discrete or continuous), such that the above two independence relations are true. “The past is independent of the future given the present”.

Figure 9.5: A Bayesian Network representation of an HMM clearly displays dependence relationships. Each circle represents a variable, and each arrow a dependence relationship, unlike SFSA depictions where the arrows represent state transitions. The shaded circles are outputs associated with particular states.
One should ask if the Markov assumption is valid. One can easily come up with examples of speech structures that seem to violate the assumption. For example, sequences of two consonants are very common, and three consecutive consonants are not uncommon (e.g. the m-p-l in amplify), but it is difficult to find examples of four or five consecutive consonants. So it might seem that we should look a little farther back to evaluate the probability of a state sequence. However, in practice, HMMs in general, and the conditional independence assumption in particular, have proven immensely powerful. Additionally, as we mentioned earlier, HMMs are rather amenable to cheating, which generally takes the form of adding extra states. We can effectively relax the Markov assumption by making duplicates of a given state for the various states which may have preceded it. In this way, the advantage of a longer history can be gained where it is beneficial without the cost of storing long histories for every state, and without needlessly complicating the probabilistic computations.

An HMM model consists of three sets of probability distributions. We have discussed the state transition probability matrix $A$ in detail. Additionally, each state $i$ has an associated output distribution $b_i(x_t) = P(X_t = x_t|Q_t = q) = p(x_t|q)$. The output distributions from all states form the set of distributions $B \triangleq \{b_1(x), \ldots, b_{|Q|}(x)\}$, where $|Q|$ is the number of states. Finally, we need to know where to start. The initial state distribution $\Pi = [\pi_i]$, where $\pi_i = P(Q_1 = i)$ gives the probability of starting in any given state. The model $\lambda$ comprises these three parameters.

Some HMM representations forgo the $\Pi$ parameter, and instead use start-states and end-states. These are extra states in which the HMM is constrained to start and end, respectively. The $\Pi$ parameter would simply be absorbed as another row in the transition matrix $A$. Start-states and other states not associated with outputs are collectively referred to as non-emitting states.

References


