Lecture 3
Dynamic Time Warping (DTW)

Yossi Keshet

October 29, 2012
A Very Simple Speech Recognizer

\[ w^* = \arg \min_{w \in \text{vocab}} \text{distance}(A_{\text{test}}', A_w') \]

- **signal processing** — Extracting features \( A' \) from audio \( A \).
  - *e.g.*, MFCC with deltas and double deltas.
  - *e.g.*, for 1s signal with 10ms frame rate \( \Rightarrow \sim 100 \times 40 \) values in \( A' \).

- **dynamic time warping** — Handling time/rate variation in the distance measure.
The Problem

$$\text{distance}(A'_{\text{test}}, A'_{w}) \equiv \sum_{t} \text{framedist}(A'_{\text{test},t}, A'_{w,t})$$

- In general, samples won’t even be same length.
Problem Formulation

- Have two audio samples; convert to feature vectors.
  - Each $x_t$, $y_t$ is $\sim$40-dim vector, say.

$$X = (x_1, x_2, \ldots, x_{T_x})$$
$$Y = (y_1, y_2, \ldots, y_{T_y})$$

- Compute distance($X$, $Y$).
Idea: omit/duplicate frames uniformly in $Y$ . . .

So same length as $X$. 

$$\text{distance}(X, Y) = \sum_{t=1}^{T_x} \text{framedist}(x_t, y_{t \times \frac{T_y}{T_x}})$$
What’s the Problem?

- Handling silence.

  \[ \text{silence} \quad \text{CAT} \quad \text{silence} \]

  \[ \text{silence} \quad \text{CAT} \quad \text{silence} \]

- Solution: *endpointing*.
- Do vowels and consonants stretch equally in time?
- Want *nonlinear* alignment scheme!
Alignments and Warping Functions

- Can specify alignment between times in $X$ and $Y$ using...
  - Warping functions $\tau_x(t)$, $\tau_y(t)$, $t = 1, \ldots, T$.
  - i.e., time $\tau_x(t)$ in $X$ aligns with time $\tau_y(t)$ in $Y$.
- Total distance is sum of distance between aligned vectors.

$$\text{distance}_{\tau_x, \tau_y}(X, Y) = \sum_{t=1}^{T} \text{framedist}(x_{\tau_x(t)}, y_{\tau_y(t)})$$
Let $x_d, y_d$ denote $d$th dimension (this slide only).

<table>
<thead>
<tr>
<th>Frame Distance</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean ($L^2$)</td>
<td>$\sqrt{\sum_d (x_d - y_d)^2}$</td>
</tr>
<tr>
<td>$L^p$ norm</td>
<td>$\sqrt[p]{\sum_d</td>
</tr>
<tr>
<td>weighted $L^p$ norm</td>
<td>$\sqrt[p]{\sum_d w_d</td>
</tr>
<tr>
<td>Itakura $d_i(X, Y)$</td>
<td>$\log(a^T R_p a / G^2)$</td>
</tr>
<tr>
<td>Symmetrized Itakura</td>
<td>$d_i(X, Y) + d_i(Y, X)$</td>
</tr>
</tbody>
</table>

Weighting each feature vector component differently.
- e.g., for variance normalization.
- Called *liftering* when applied to cepstra.
Another Example Alignment
Constraining Warping Functions

- Begin at the beginning; end at the end. (Any exceptions?)
  \[ \tau_x(1) = 1, \quad \tau_x(T) = T; \quad \tau_y(1) = 1, \quad \tau_y(T) = T. \]
- Don't move backwards (monotonicity).
  \[ \tau_x(t + 1) \geq \tau_x(t); \quad \tau_y(t + 1) \geq \tau_y(t). \]
- Don't move forwards too far (locality).
  \[ e.g., \quad \tau_x(t + 1) \leq \tau_x(t) + 1; \quad \tau_y(t + 1) \leq \tau_y(t) + 1. \]
Local Paths

- Can summarize/encode local alignment constraints . . .
  - By enumerating legal ways to extend an alignment.
- e.g., three possible extensions or *local paths*.
  - $\tau_1(t + 1) = \tau_1(t) + 1, \tau_2(t + 1) = \tau_2(t) + 1$.
  - $\tau_1(t + 1) = \tau_1(t) + 1, \tau_2(t + 1) = \tau_2(t)$.
  - $\tau_1(t + 1) = \tau_1(t), \tau_2(t + 1) = \tau_2(t) + 1$. 
Which Alignment?

- Given alignment, easy to compute distance:

\[
\text{distance}_{\tau_x, \tau_y}(X, Y) = \sum_{t=1}^{T} \text{framedist}(x_{\tau_x(t)}, y_{\tau_y(t)})
\]

- Lots of possible alignments given \(X, Y\).
- Which one to use to calculate \(\text{distance}(X, Y)\)?
- The best one!

\[
\text{distance}(X, Y) = \min_{\text{valid} \tau_x, \tau_y} \{ \text{distance}_{\tau_x, \tau_y}(X, Y) \} 
\]
Which Alignment?

$$\text{distance}(X, Y) = \min_{\text{valid } \tau_x, \tau_y} \{ \text{distance}_{\tau_x, \tau_y}(X, Y) \}$$

- Hey, there are many, many possible $\tau_x$, $\tau_y$.
- Exponentially many in $T_x$ and $T_y$, in fact.
- How in blazes are we going to compute this?
Dynamic Programming

- Solve problem by caching subproblem solutions . . .
  - Rather than recomputing them.
- Why the mysterious name?
  - The inventor (Richard Bellman) was trying to hide . . .
  - What he was working on from his boss.
- For our purposes, we focus on *shortest path* problems.
Let’s consider small example: $T_x = 3; T_y = 2$.

- Allow steps that advance $t_x$ and $t_y$ by at most 1.

Matrix of frame distances: $\text{framedist}(x_t, y_{t'})$.

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_2$</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$y_1$</td>
<td>3</td>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

Let’s make a graph.

- Label each arc with $\text{framedist}(x_t, y_{t'})$ at destination.
- Need dummy arc to get distance at starting point.
Each path from point A to B corresponds to alignment.

- Distance for alignment = sum of scores along path.
- Goal: find alignment with smallest total distance.
Let’s take a break and look at something totally unrelated. Here’s a map with distances labeled between cities. Let’s say we want to find the shortest route . . . From point A to point B.
I’m having déjà vu!

In fact, length of shortest route in 2nd problem . . .

Is same as distance for best alignment in 1st problem.

i.e., problem of finding best alignment is equivalent to . . .

*Single-pair shortest path* problem for . . .

*Directed acyclic graphs.*

Has well-known dynamic programming solution.

Will come up again for HMM’s, finite-state machines.
Key Observation 1

- Shortest distance $d(S)$ from start to state $S$ is...
  - $d(S') + \text{distance}(S', S)$ for some predecessor $S'$ of $S$.
- If know $d(S')$ for all predecessors of $S$ ...
- Easy to compute $d(S)$.

$$d(S) = \min_{S'\rightarrow S} \{d(S') + \text{distance}(S', S)\}$$
Proposed Algorithm

- Loop through all states $S$ in some order.
  - For each state, compute distance to start state $d(S)$.
  - $d(S')$ must already be known for all predecessors $S'$.
- Is there always an ordering on states such that . . .
  - If there is an arc $S' \rightarrow S$, . . .
  - $d(S')$ is computed before $d(S)$?
Key Observation 2

- For all acyclic graphs, there is a *topological sorting* . . .
  - Such that all arcs go from earlier to later states.
- In many cases, topological sorting is obvious by inspection.
- Otherwise, can be computed in time $O(\text{states} + \text{arcs})$. 
The Shortest Path Algorithm

- Sort states topologically: number from 1, \ldots, N.
  - Start state = state 1; final start = state N.
- \( d(1) = 0 \).

For \( S = 2, \ldots, N \) do

\[
d(S) = \min_{S' \rightarrow S} \{ d(S') + \text{distance}(S', S) \}
\]

- Final answer: \( d(N) \).
- Total time: \( O(\text{states} + \text{arcs}) \).
What are the states of the graph?
What is a topological sorting for the states?
What are the predecessors for each state?
$d(1, 1) = \text{framedist}(x_1, y_1)$.

For $t_1 = 1, \ldots, 3$ do.
  
  For $t_2 = 1, \ldots, 2$ do.
    
    $d(t_1, t_2) = \min\{$
    
    $d(t_1 - 1, t_2 - 1) + \text{framedist}(x_{t_1}, y_{t_2}),$
    
    $d(t_1 - 1, t_2) + \text{framedist}(x_{t_1}, y_{t_2}),$
    
    $d(t_1, t_2 - 1) + \text{framedist}(x_{t_1}, y_{t_2}) \}$
  
  Final answer: $d(3, 2)$. 

DTW: Example

\[
\begin{array}{c|ccc}
   & x_1 & x_2 & x_3 \\ 
 y_2 & 4   & 1   & 2   \\ 
 y_1 & 3   & 0   & 5   \\
\end{array}
\quad
\begin{array}{c|ccc}
   d(\cdot, \cdot) & 1 & 2 & 3 \\ 
 2 & 7 & 4 & 5 \\ 
 1 & 3 & 3 & 8 \\
\end{array}
\]
Two-step path incurs two frame distances, while . . .
- One-step path incurs single frame distance.
- Biases against two-step path.

Idea: use weights to correct for these types of biases.
Weighted Local Paths (Sakoe and Chiba)

\[
d(t_1, t_2) = \min \left\{ 
    d(t_1 - 1, t_2 - 1) + 2 \times \text{framedist}(x_{t_1}, y_{t_2}), \\
    d(t_1 - 2, t_2 - 1) + 2 \times \text{framedist}(x_{t_1-1}, y_{t_2}) + \text{framedist}(x_{t_1}, y_{t_2}), \\
    d(t_1 - 1, t_2 - 2) + 2 \times \text{framedist}(x_{t_1}, y_{t_2-1}) + \text{framedist}(x_{t_1}, y_{t_2}) \right\}
\]
Speeding Things Up

- What is time complexity of DTW?

\[ O(\text{states } + \text{arcs}) \]

- For long utterances, can be expensive.
- Idea: put ceiling on maximum amount of warping, e.g.,

\[ |\tau_x(t) - \tau_y(t)| \leq T_0 \]

- Another idea: beam pruning or rank pruning.
- Can make computation linear.
A DTW Recognizer: The Whole Damn Thing

\[ w^* = \arg \min_{w \in \text{vocab}} \text{distance}(A'_\text{test}, A'_w) \]

- **Training**: collect audio \( A_w \) for each word \( w \) in vocab.
  - Apply signal processing \( \Rightarrow A'_w \).
  - Called *template* for word \( w \).
  - (Can collect multiple templates for each word.)

- **Test time**: given audio \( A'_\text{test} \), convert to \( A'_\text{test} \).
  - For each \( w \), compute \( \text{distance}(A'_\text{test}, A'_w) \) using DTW.
  - Return \( w \) with smallest distance.
Recap

- DTW is effective for computing distance between signals . . .
  - Using nonlinear time alignment.
- Ad hoc selection of frame distance and local paths.
- Finds best path in exponential space in quadratic time . . .
  - Using dynamic programming.
- Signal processing, DTW: all you need for simple ASR.
  - e.g., old-school cell phone name dialer.
Some recent work has revisited DTW.
- Can extend algorithm to connected speech.
- Word models (as opposed to *phonetic* models).
- Don’t average word instances together; keep separate!
- Can model longer-distance acoustic dependencies . . .
  - As compared to conventional GMM/HMM systems.
- Hasn’t gone anywhere yet.