Outline of Today’s Lecture

- Recap: Structured Machine Learning Problems
- Dynamic Time Warping: A first solution to structured pattern recognition
books & sources

- Deller et. al. “Discrete-time Processing of speech signals”

Scribes so far

- Day 1: Jeff Bilmes
- Day 2: Xin Lie
- Day 3: Tom Mackel
- Day 4: Scott Philips
- Day 5: Kevin Duh
- Day 6: Travis Wilkins
- Day 7: Amar Subramanya
- Day 8: Andrei Alexandres
- Day 9: Dustin Lennon
- Day 10:
- Day 11:
- Day 12:
- Day 13:
- Day 14:
- Day 15:
- Day 16:
- Day 17:
- Day 18:
- Day 9:
- Day 10:
Dynamic Time Warping

- First approach to structured output domains
- Goal is to try to find a “distance” between strings, where in this case the strings are sequences of speech feature vectors (e.g., MFCCs)
- Strings can be arbitrary length, and arbitrary warpings of some “ideal” or template speech utterance.
- Given unknown utterance, template that is closest determines what was said.
- We try to “time-warp” each template to the unknown string.

Dynamic Time Warping

- Assume we have two sequences of speech vectors
  \[ X = \{x_1, x_2, \ldots, x_{T_x}\} \quad Y = \{y_1, y_2, \ldots, y_{T_y}\} \]
- There is a “distortion” measure (distance) between the two individual fixed length vectors.
  \[ d(x, y) = \frac{1}{d} \sum_{i=1}^{d} (x(i) - y(i))^2 w(i) \]

\[ d(x, y) = (x - y)^T A (x - y) \] Mahalanobis distance

- … there are many possibilities.
- Typically, \( T_x \neq T_y \) so can’t do: \( \sum_{i=1}^{T_x} d(x_i, y_i) \)
- Notation:
  \( i_x \in \{1, 2, \ldots, T_x\} \quad i_y \in \{1, 2, \ldots, T_y\} \)
- so we can write: \( d(i_x, i_y) \triangleq d(x_{i_x}, y_{i_y}) \)
Dynamic Time Warping

- Uniform warping isn’t good since different parts of speech are not accurately warped in this manner
- Example: hhheeeeeeeelo vs. hello (using orthography as phonetic spelling)
- Most general alignment & normalization using x and y specific alignment functions $\phi_x, \phi_y$

$$
\begin{align*}
  i_x &= \phi_x(k) \\
  i_y &= \phi_y(k)
\end{align*} \quad k = 1, \ldots, T \\
T &= \max(T_x, T_y)
$$

- Global distance defined as:

$$
d_\phi(X, Y) = \sum_{k=1}^{T} d(\phi_x(k), \phi_y(k))m(k)
$$
Dynamic Time Warping

• To form the overall distance, we do:
  \[ d(X, Y) = \min_{\phi} d_{\phi}(X, Y) \]

• Is this reasonable:
  – for same utterance, makes reasonable sense
  – for different utterances, choosing best alignment
    might not be ideal, but at least it still doesn’t like
    utterances of very different lengths

• Consider alternatives:
  \[ d(X, Y) = \max_{\phi} d_{\phi}(X, Y) \]

\[ d(X, Y) = \frac{1}{N} \sum_{\phi \in V(X, Y)} d_{\phi}(X, Y) \]

Dynamic Time Warping

• Still another alternative:
  \[ d(X, Y) = \sum_{\phi \in V(X, Y)} d_{\phi}(X, Y) Pr(\phi) \]

• … with a probabilistic interpretation, but where
  & how to get probabilities?? (later, HMMs)

• Note: very similar to Levenshtein edit distance
  measure between two strings.

• Note: there are many possible paths, all
  possible functions \( \phi_x \) and \( \phi_y \) that create an
  allignment (might even be greater length than
  \( T \)).

• Ex: \( \phi_x(k) \) has \( T_x \) possible values, so \( T_x^T \)
  possible functions!!
Dynamic Time Warping

• Total num. functions, ignoring constraints: \( T_x^T T_y^T = (T_x T_y)^T \)
• Computing this distance (by minimizing over all \( \phi \) functions) is sounding very expensive
• Dynamic programming is the solution.
• But first, does it make sense to start other than beginning and end other than the end?

Consider words: “you” vs. “we”, phonetically they are reversals of each other.

• Begin/end issues (as previous plot shows)
• Clearly, there needs to be constraints.
  • Endpoint constraints:
    \[
    \begin{align*}
    \phi_x(1) &= 1 \\
    \phi_y(1) &= 1 \\
    \phi_x(T) &= T_x \\
    \phi_y(T) &= T_y
    \end{align*}
    \]

• Monotonicity constraints (don’t go backwards)
  \[
  \begin{align*}
  \phi_x(k + 1) &\geq \phi_x(k) \\
  \phi_y(k + 1) &\geq \phi_y(k)
  \end{align*}
  \]

• Local Continuity constraints:
Dynamic Time Warping

- Local Continuity constraints (never jump by more than 1):
  \[ \phi_x(k+1) - \phi_x(k) \leq 1 \]
  \[ \phi_y(k+1) - \phi_y(k) \leq 1 \]

- Can specify allowable paths:
  \( P_1 \rightarrow (1, 0) \quad P_2 \rightarrow (1, 1) \quad P_3 \rightarrow (0, 1) \)

- Paths can be specified by increments (deltas):
  \( P \rightarrow (p_1, q_1)(p_2, q_2)(p_3, q_3) \ldots (p_T, q_T) \)

---

Dynamic Time Warping

- Local constraints specified using lists of paths.
- There exists standard types:
  - I.
  - II.
  - III.

and so on. There are some types that are well known, such as the "ITAKURA" path constraint.
Dynamic Time Warping

• Path constraints are only heuristic beliefs about the reasonable differences between a template and unknown utterance
  – too much flexibility: different phrases will be warped together (e.g., “you”, “we”)
  – too little flexibility, different tokens of same word that are highly warped will fail to be aligned.

• Global path constraints
  – either implied by local constraints or can either be explicitly stated,
  – ex: region of possible points (with start and end constraints), other points can’t be reached by local path constraints

Dynamic Time Warping

• Constraint characteristics: max/min slope of set of N paths

\[ P^\ell = \{(p_1^\ell, q_1^\ell), (p_2^\ell, q_2^\ell), \ldots, (p_{T^\ell}^\ell, q_{T^\ell}^\ell)\} \]

\[ P = \{P^\ell; \ell = 1 \ldots N\} \]

\[ Q_{\text{max}} = \max_{\ell} \left[ \frac{\sum_{i=1}^{T_\ell} q_i^\ell}{\sum_{i=1}^{T_\ell} p_i^\ell} \right] \quad Q_{\text{min}} = \min_{\ell} \left[ \frac{\sum_{i=1}^{T_\ell} q_i^\ell}{\sum_{i=1}^{T_\ell} p_i^\ell} \right] \]

• Often, \( Q_{\text{max}} = \frac{1}{Q_{\text{min}}} \).
• Example where \( Q_{\text{max}} = 2, Q_{\text{min}} = 1/2 \)
Dynamic Time Warping

• Points reachable from (1,1)
\[ 1 + Q_{\min}[\phi_x(k) - 1] \leq \phi_y(k) \leq 1 + Q_{\max}[\phi_x(k) - 1] \]

Dynamic Time Warping

• Points that can reach \((T_x, T_y)\)
\[ T_y + Q_{\max}[\phi_x(k) - T_x] \leq \phi_y(k) \leq T_y + Q_{\min}[\phi_x(k) - T_x] \]
Dynamic Time Warping

- Maximum asynchrony constraint.

\[ |\phi_x(k) - \phi_y(k)| \leq T_0 \]

Putting it all together: global constraints:

Intersection ( )
Dynamic Time Warping

- Slope weighting: might want to penalize slopes differently
  \[ d_\phi(X, Y) = \sum_{k=1}^{T} d(\phi_x(k), \phi_y(k)) m(k) \]
- Many possible options (\(m(k) = 1\) is no weighting)
  \[ m(k) = \min[\phi_x(k) - \phi_x(k-1), \phi_y(k) - \phi_y(k-1), 1] \]
  \[ m(k) = \max[\phi_x(k) - \phi_x(k-1), \phi_y(k) - \phi_y(k-1)] \]

Dynamic Time Warping

- But still need to solve efficiently: use dynamic programming
  \[ d(X, Y) = \min_\phi d_\phi(X, Y) \]

- For DP need two things: optimal substructure and overlapping subproblems.
Dynamic Time Warping

• **Optimal Substructure**

  • Given an optimal solution to a problem, that solution contains optimal solutions to subproblems.

  • Ex:

    Ex: given optimal way to get from inside EE building to red square, that includes a solution to get from EE building to go north/west.

Dynamic Time Warping

• **Optimal Substructure**

  Ex: Suppose optimal solution to point A in the grid (so we have the optimal path to A). Suppose this goes through C (using EC). If the path to C (EC) wasn’t optimal (say EC’ was), then nor would the path to A be optimal. Therefore, we must have optimal path (EC) to C as well.

  When we compute optimal paths to A, we first compute optimal paths to all the things that could directly link to A (e.g., B, C, and D), and then choose the one (B, C, or D) that when connected to A is optimal. This is then done recursively.
Dynamic Time Warping

- **Common Sub-problems**
- The optimal sub-problems are used over and over.
- **Ex:**

Ex: given optimal way to get from inside EE building to red square, that includes a solution to get from EE building to go north/west.

This second solution (getting out of EE building) could be useful for another optimal problem (e.g., leaving EE building and going to Kane Hall)

---

Dynamic Time Warping

- **Common Subproblems**

Ex: Suppose we have optimal solution to point A in the grid (so we have the optimal path to A). This means that we have computed and considered solutions to A via both B and C, and therefore must have computed optimal solutions to B and C.

When we start computing the optimal solutions to D, we need not re-compute the optimal solutions to B and C since they were already computed.

This implies the DP recursion, or another form is often called “memoization”
Dynamic Time Warping

- So how to solve this:
  \[ D(T_x, T_y) \triangleq d(X, Y) = \min_{\phi} d_{\phi}(X, Y) \]
  \[ = \min_{\phi_x, \phi_y} \sum_{k=1}^{T} d(\phi_x(k), \phi_y(k))m(k) \]

- Define:
  \[ D(i_x, i_y) \triangleq \min_{\phi_x, \phi_y, T'} \sum_{k=1}^{T'} d(\phi_x(k), \phi_y(k))m(k) \]
  \[ \phi_x(T') = i_x \quad \phi_y(T') = i_y \]

- Gives DP recursion:
  \[ D(i_x, i_y) = \min_{i'_x, i'_y} \left[ D(i'_x, i'_y) + d((i'_x, i'_y), (i_x, i_y)) \right] \]

- All valid paths \( \mathcal{P} \) that take \( i'_x, i'_y \) to \( i_x, i_y \), and \( d \) is the distance according to those paths.

Dynamic Time Warping

- where \( d() \) is weighted distortion for path from \( i' \) to \( i \).
  \[ d((i'_x, i'_y), (i_x, i_y)) = \sum_{\ell=0}^{L_p} d(\phi_x(T' - \ell), \phi_y(T' - \ell))m(T' - \ell) \]

- We only consider allowable paths, according to current constraints. Examples follow:

\[
D(i_x, i_y) = \min \begin{cases} 
D(i_x - 1, i_y) + d(i_x, i_y), \\
D(i_x - 1, i_y - 1) + 2d(i_x, i_y), \\
D(i_x, i_y - 1) + d(i_x, i_y) 
\end{cases}
\]

\[
D(i_x, i_y) = \min \begin{cases} 
D(i_x - 2, i_y - 1) + \frac{1}{2}[d(i_x - 1, i_y) + d(i_x, i_y)], \\
D(i_x - 1, i_y - 1) + d(i_x, i_y), \\
D(i_x - 1, i_y - 2) + \frac{1}{2}[d(i_x, i_y - 1) + d(i_x, i_y)] 
\end{cases}
\]

\[
D(i_x, i_y) = \min \begin{cases} 
D(i_x - 2, i_y - 1) + 3d(i_x, i_y), \\
D(i_x - 1, i_y - 1) + 2d(i_x, i_y), \\
D(i_x - 1, i_y - 2) + 3d(i_x, i_y) 
\end{cases}
\]
Dynamic Time Warping

- Recursion to get optimal (minimum) score.
- Start:
  \[ D(1, 1) = d(1, 1)m(1) \]
- Recursion:
  \[ D(i_x, i_y) = \arg\min_{i_x', i_y'} \left[ D(i_x', i_y') + d((i_x', i_y'), (i_x, i_y)) \right] \]
- End:
  \[ D(X, Y) = D(T_x, T_y) \]

Dynamic Time Warping

- How to get optimal path (rather than just score?)
- Keep back-trace when going forward (i.e., at each point, store the most previous location that yielded the current optimal path)
- Recursion (in addition to previous):
  \[ \psi(i_x, i_y) = \arg\min_{i_x', i_y'} \left[ D(i_x', i_y') + d((i_x', i_y'), (i_x, i_y)) \right] \]
- Back trace computation:
  - start:
    \[ (i_x^T, i_y^T) = (T_x, T_y) \]
  - repeat: \( j = T \) downto 2
    \[ (i_x^{j-1}, i_y^{j-1}) = \psi(i_x^j, i_y^j) \]
Dynamic Time Warping

• Each point keeps track of previous location that best leads to it.

Dynamic Time Warping

• Advantages of DTW
  – easy to implement
  – gets reasonable results for limited vocabulary speech recognition
  – adequate job aligning templates with different speaking rates

• Problems with DTW
  – Only one template used per utterance
    • possible to use multiple templates per utterance, and use the score from the lowest template to determine the unknown utterance
    • Ideally, we might like a template that is an “average” of utterances.
  – path constraints are somewhat arbitrary
  – path weights are somewhat arbitrary, ideally it should be automatic
  – How do we do continuous speech (connected words)? Not so simple to generalize this system
  – Not a formal system for dealing with uncertainty. Ideally, we’d like a formal probabilistic schema & use Bayes decision theory.

• Solution: From DTW to Hidden Markov Models (HMMs)
• Next time: a “morph” from DTW to HMMs.