Design and Implementation of Speech Recognition Systems

Spring 2014

Class 5: Dynamic Time Warping-Recognizing speech
10 Feb 2014
Speech Recognition by Template Matching

• Store “templates” for all words to be recognized
  – Template = example recording
    • Actually feature sequence from example recording

• Compute distance of input test data to all templates, select the closest

• Like spellchecking
• **Isolated word recognition scenario**

- **Spoken input word**
  - Recordings (templates)
  - Compare
    - Word1
    - Word2
    - Word3
    - Word-N
  - Best
Speech Recognition as Template Matching

- Problem: Input and template may be different lengths
- Worse – the change in length may not be uniform
- Must nevertheless be able to say that the distance between the two above examples is small
  - Like string matching
• Back to template matching for text: *dynamic time warping*
  – Input and templates are sequences of feature vectors instead of letters

• Intuitive understanding of why DP-like algorithm might work to find a best alignment of a template to the input:
  – We need to search for a path that finds the following alignment:

```
  template: s o me th i ng
  input:   s o me th i ng
```
  – The DP algorithm for text permits such alignments

• Consider the 2-D matrix of template-input frames of speech
DTW: DP for Speech Template Matching

Need to find something like this warped path
DTW: Adapting Concepts from DP

• Some concepts from string matching need to be adapted to this problem
  – What are the allowed set of transitions in the search trellis?
  – What are the edge and local node costs?
    • Nodes can also have costs

• Once these questions are answered, we can apply essentially the same DP algorithm to find a minimum cost match (path) through the search trellis
DTW: Adapting Concepts from DP

• What transitions are allowed..

• What is a “score”? 
DTW: Determining Transitions

• Transitions must account for *stretching* and *shrinking* of speech segments
  – To account for varying speech rates

• Unscored “Insertions” disallowed
  – Every input frame must be matched to *some* template frame
  – Different from Levenshtein distance computation where symbols were compared only at diagonal transitions

• For meaningful comparison of two different path costs, their lengths must be kept the same
  – So, every input frame is to be aligned to a template frame *exactly* once
  – Vertical transitions (mostly) disallowed
DTW: Transitions

• Typical transitions used in DTW for speech:

- The next input frame aligns to the same template frame as the previous one. (Allows a template segment to be arbitrarily stretched to match some input segment)

- The next input frame aligns to the next template frame. No stretching or shrinking occurs in this region

- The next input frame skips the next template frame and aligns to the one after that. Allows a template segment to be shrunk (by at most ½) to match some input segment

• Note that all transitions move one step to the right, ensuring that each input frame gets used exactly once along any path
Levenshtein vs. DTW: Transitions

• **LEVENSHTEIN**
  - Horizontal transition, no symbol comparison
  - Diagonal transition: Symbols are compared
  - Vertical transition: no symbol comparison

• **DTW**
  - Horizontal: symbol must be compared
  - Diagonal: Two varieties
    - Both require symbol comparison
  - Vertical: Disallowed
DTW: Use of Transition Types

- Short template, long input
- Approx. equal length template, input
- Long template, short input
DTW: Other Transition Choices

• Other transition choices are possible:
  – Skipping more than one template frame (greater shrink rate)
  – Vertical transitions: the same input frame matches more than one template frame
    • This is less often used, as it can lead to different path lengths, making their costs not easily comparable
DTW: Local Edge and Node Costs

• Typically, there are no edge costs; any edge can be taken with no cost
• Local node costs measure the dissimilarity or distance between the respective input and template frames
• Since the frame content is a multi-dimensional feature-vector, what dissimilarity measure can we use?
• A simple measure is Euclidean distance; i.e. geometrically how far one point is from the other in the multi-dimensional vector space
  – For two vectors \( X = (x_1, x_2, x_3 \ldots x_N) \), and \( Y = (y_1, y_2, y_3 \ldots y_N) \), the Euclidean distance between them is:

  \[
  \sqrt{\sum (x_i - y_i)^2}, \ i = 1 \ldots N
  \]

  – Thus, if \( X \) and \( Y \) are the same point, the Euclidean distance = 0
  – The farther apart \( X \) and \( Y \) are, the greater the distance
DTW: Local Edge and Node Costs

• Other distance measures could also be used:
  – Manhattan metric or the L1 norm: \( \Sigma |A_i - B_i| \)
  – Weighted Minkowski norms: \((\Sigma w_i |A_i - B_i|^n)^{1/n}\)
DTW: Overall algorithm

- The transition structure and local edge and node costs are now defined
- The search trellis can be realized and the DP algorithm applied to search for the minimum cost path, as before
  - Example trellis using the transition types shown earlier:
DTW: Overall algorithm

- The best path score can be computed using DP as before
  - But the best path score must now consider both node and edge scores
  - Each node is a comparison of a vector from the data against a vector from the template
DTW: Overall Algorithm

- \(P_{i,j} = \text{best path cost from origin to node } [i,j]\)
  - \(i\)-th template frame aligns with \(j\)-th input frame
- \(C_{i,j} = \text{local node cost of aligning template frame } i \text{ to input frame } j\)

\[
P_{i,j} = \min (P_{i,j-1} + C_{i,j}, P_{i-1,j-1} + C_{i,j}, P_{i-2,j-1} + C_{i,j})
\]

\[
= \min (P_{i,j-1}, P_{i-1,j-1}, P_{i-2,j-1}) + C_{i,j}
\]

- Edge costs are 0 in above formulation
DTW: Overall Algorithm

- If the template is $m$ frames long and the input is $n$ frames long, the best alignment of the two has the cost $= P_{m,n}$

- The computational is proportional to:
  $M \times N \times 3$, where
  $M =$ No. of frames in the template
  $N =$ No. of frames in the input
  $3$ is the number of incoming edges per node
Handling Surrounding Silence

- The DTW algorithm automatically handles any silence region surrounding the actual speech, within limits:

- But, the transition structure does not allow a region of the template to be shrunk by more than ½!
  - Need to ensure silences included in recording are of generally consistent lengths, or allow other transitions to handle a greater “warp”
Isolated Word Recognition Using DTW

• We now have all ingredients to perform isolated word recognition of speech

• “TRAINING”: For each word in the vocabulary, pre-record a spoken example (its template)

• RECOGNITION of a given recording:
  – For each word in the vocabulary
    • Measure distance of recording to template using DTW
  – Select word whose template has smallest distance
• For each template:
  – Create a trellis against data
    • Figure above assumes 7 vectors in the data
  – Compute the cost of the best path through the trellis

• Select word corresponding to template with lowest best path cost
Time Synchronous Search

- Match all templates Synchronously
- STACK trellises for templates above one another
  - Every template match is started simultaneously and stepped through the input in lock-step fashion
    - Hence the term *time synchronous*

- Advantages
  - No need to store the entire input for matching with successive templates
  - Enables realtime: Matching can proceed as the input arrives
  - Enables *pruning* for computational efficiency
Example: Isolated Speech Based Dictation

• We could, in principle, almost build a large vocabulary isolated-word dictation application using the techniques learned so far

• Training: Record templates (i.e. record one or more instance) of each word in the vocabulary

• Recognition
  – Each word is spoken in isolation, *i.e.* silence after every word
  – Each isolated word compared to all templates
    • Accuracy would probably be terrible

• Problem: How to detect when a word is spoken?
  – Explicit “click-to-speak”, “click-to-stop” button clicks from user, for every word?
    • Obviously extremely tedious
  – Need a speech/silence detector!
Endpointing: A Revision

• Goal: automatically detect pauses between words
  – to segment the speech stream into isolated words?

• Such a speech/silence detector is called an endpointer
  – Detects speech/silence boundaries (shown by dotted lines)

• Most speech applications use such an endpointer to relieve
  the user of having to indicate start and end of speech
A Simple Endpointing Scheme

- Based on silence segments having low signal amplitude
  - Usually called *energy-based* endpointing

- Audio is processed as a short sequence of *frames*
  - Exactly as in feature extraction

- The signal *energy* in each frame is computed
  - Typically in *decibels* (dB): $10 \log (\Sigma x_i^2)$, where $x_i$ are the sample values in the frame

- A *threshold* is used to classify each frame as speech or silence
- The labels are *smoothed* to eliminate spurious labels due to noise
  - E.g. minimum silence and speech segment length limits may be imposed
  - A very short speech segment buried inside silence may be treated as silence

- The above should now make sense to you if you’ve completed the feature computation code
Speech-Silence Detection: Endpoint

- The computed “energy track” shows signal power as a function of time
- A simple threshold can show audio segments
  - Can make many errors though
- What is the optimal threshold?
Speech-Silence Detection: Endpoint

- Optimal threshold: Find average value of latest contiguous non-speech segment of minimum length
- Find average energy value in the segment
  - \( \text{Avgnoiseegy} = \frac{1}{N_{\text{contiguous frames}}} \times \Sigma(\text{energy of frames}) \)
- Average noise energy plus threshold = speech threshold
  - \( \text{Egy} > \alpha \times \text{Avgnoiseegy} \)
  - Alpha typically > 6dB
Speech-Silence Detection: Endpoint

- Alternative strategy: TWO thresholds
  - Onset of speech shows sudden increase in energy

- Onset threshold: avgnoiseegy * alpha
  - Speech detected if frame energy > onset threshold
  - Alpha > 12dB

- Offset threshold: avgnoiseegy * beta
  - Beta > 6dB

- Speech detected between onset and offset
  - Additional smoothing of labels is still required
  - Typically, detected speech boundaries are shifted to include 200ms of silence either side
Isolated Speech Based Dictation (Again)

• With such an endpointer, we have all the tools to build a complete, isolated word recognition based dictation system, or any other application

• However, as mentioned earlier, accuracy is a primary issue when going beyond simple, small vocabulary situations
Dealing with Recognition Errors

• Applications can use several approaches to deal with speech recognition errors

• Primary method: improve performance by using better models in place of simple templates
  – We will consider this later

• However, most systems also provide other, orthogonal mechanisms for applications to deal with errors
  – Confidence estimation
  – Alternative hypotheses generation (N-best lists)

• We now consider these two mechanisms, briefly
Confidence Scoring

- **Observation**: DP or DTW will *always* deliver a minimum cost path, *even if it makes no sense*

- Consider string matching:
  
<table>
<thead>
<tr>
<th>templates</th>
<th>min. edit distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yesterday</td>
<td>7</td>
</tr>
<tr>
<td>Today</td>
<td>5</td>
</tr>
<tr>
<td>Tomorrow</td>
<td>7</td>
</tr>
<tr>
<td>input: January</td>
<td></td>
</tr>
</tbody>
</table>

- The template with minimum edit distance will be chosen, even though it is “obviously” incorrect
  - How can the application discover that it is “obviously” wrong?
- **Confidence scoring** is the problem of determining how confident one can be that the recognition is “correct”
Confidence Scoring for String Match

• A simple confidence scoring scheme: Accept the matched template string only if the cost $\leq$ some threshold
  – We encountered its use in the spell checking example!

• Accept if no. of errors is below some fixed threshold
• Or: Accept if cost $\leq 1 +$ some fraction (e.g. $0.1$) of template string length
  – Templates of 1-9 characters tolerate 1 error
  – Templates of 10-19 characters tolerate 2 errors, etc.

• Easy to think of other possibilities, depending on the application

• Confidence scoring is one of the more application-dependent functions in speech recognition
Confidence Scoring for DTW

• Similar thresholding technique for template matching by DTW?
  – Unlike in string matching, the cost measures are not immediately, meaningfully “accessible” values
  – Need to know range of minimum cost when correctly matched and when incorrectly matched
    • If the ranges do not overlap, one could pick a threshold

[Diagram showing the distribution of DTW costs of correctly identified templates and the distribution for incorrectly identified templates. The overlap region is susceptible to classification errors.]
**Confidence: Procedure**

- "Recognize" many many "development" recordings
  - Several will be recognized correctly
  - Others will be recognized wrongly

- Training confidence classifier
  - Distribution of scores of all wrongly recognized utterances
  - Distribution of scores of all correctly recognized utterances

- Confidence on test recording:
  - Option 1: Find optimal threshold for correct vs. wrong
  - Option 2: Compute confidence score = P(test | correct) / P(test | error)
Confidence Scoring for DTW

• As with string matching, DTW cost must be *normalized*
  – Use DTW cost / frame of input speech, instead of total DTW cost, before determining threshold

• Cost distributions and threshold have to be determined *empirically*, based on a sufficient collection of test data

• Unfortunately, confidence scores based on such distance measures are not very reliable
  – Too great an overlap between distribution of scores for correct and incorrect templates
  – We will see other, more reliable methods later on
N-best List Generation

• *Example*: Powerpoint catches spelling errors and offers several alternatives as possible corrections

• *Example*: In *Dragon Dictate*, one can select a recognized word and obtain alternatives
  – Useful if the original recognition was incorrect

• Basic idea: identifying not just the best match, but the top so many matches; *i.e.*, the *N-best list*

• Not hard to guess how this might be done, either for string matching or isolated word DTW!
  – (How?)
N-best List

- Match all templates
- RANK the words (templates) by the minimum-cost-path score for the template/trellis
- Return top-N words in order of minimum cost
Improving Accuracy: Multiple Templates

• Problems with using a single exemplar as a template
  – A single template will not capture all variations in the manner of saying a word
    • Works poorly even for a single speaker
    • Works very poorly across different speakers

• Use multiple templates for each word to handle the variations
  – Preferably collected from several speakers

• Template matching algorithm is easily modified
  – Simply match against all available templates and pick the best

• However, computational cost of matching increases linearly with the number of available templates
Reducing Search Cost: Pruning

- Reducing search cost implies reducing the size of the lattice that has to be evaluated

- As in string matching, there are several ways to accomplish this
  - Reducing the size of the models (templates)
    - *E.g.* replacing the multiple templates for a word by a single, *average* one
    - Reducing allowed transitions
  - Eliminating parts of the lattice from consideration altogether
    - *search pruning*, or just *pruning*
    - We consider search pruning first

- Basic consideration in pruning: *As long as the best cost path is not eliminated by pruning, we obtain the same result*
Pruning by Limiting Search Paths

- Assume that the input and the best matching template do not differ significantly from each other
  - For speech, equivalent to assuming the speaking rate is similar for the template and the input
  - The best path matching the two will lie close to the “diagonal”
- Thus, we need not search far off the diagonal. If the search-space “width” is kept constant, cost of search is linear in utterance length instead of quadratic
- However, errors occur if the speaking rate assumption is violated
  - i.e. if the template needs to be warped more than allowed by the width
Pruning by Limiting Search Paths

- What are problems with this approach?
Pruning by Limiting Search Paths

• What are problems with this approach?
  – Text: With lexical tree models, the notion of “diagonal” becomes difficult
  – For speech too there is no clear notion of a diagonal in most cases
    • As we shall see later
Pruning by Limiting Path Cost

- **Observation**: Partial paths that have “very high” costs will rarely recover to win
- Hence, poor partial paths can be eliminated from the search:
  - For each frame $j$, after computing all the trellis nodes path costs, determine which nodes have too high costs
  - Eliminate them from further exploration
  - *(Assumption: In any frame, the best partial path has low cost)*
- **$Q$**: How do we define “high cost”?

![Diagram showing partial best paths and high cost partial paths](image-url)
Pruning by Limiting Path Cost

• As with confidence scoring, one *could* define high path cost as a value worse than some fixed threshold
  – But, as already noted, absolute costs are unreliable indicators of correctness
  – Moreover, path costs keep increasing monotonically as search proceeds
    • Recall the path cost equation

\[
P_{i,j} = \min (P_{i,j-1}, P_{i-1,j-1}, P_{i-2,j-1}) + C_{i,j}
\]

• Fixed threshold will not work
Pruning : Fixed Width Pruning

- Retain only the $K$ best nodes in any column
  - $K$ is the “fixed” beam width

With $K = 2$

The two best scoring nodes are retained
Fixed Width Pruning

• Advantages
  – Very predictable computation
    • Only K nodes expand out into the future at each time.

• Disadvantage
  – Will often prune out correct path when there are many similar scoring paths
  – In time-synchronous search, will often prune out correct template
Pruning: Relative Fixed Beam

• Solution: In each frame $j$, retain only the best $K$ nodes relative to the best cost node in that frame
  – Note that time synchronous search is very efficient for implementing the above

• Advantages:
  – Unreliability of absolute path costs is eliminated
  – Monotonic growth of path costs with time is also irrelevant
Pruning: *Beam Search*

- In each frame $j$, set the pruning threshold by a fixed amount $T$ *relative to the best cost in that frame*
  - *I.e.* if the best partial path cost achieved in the frame is $X$, prune away all nodes with partial path cost $> X+T$
  - Note that *time synchronous* search is very efficient for implementing the above

- Advantages:
  - Unreliability of absolute path costs is eliminated
  - Monotonic growth of path costs with time is also irrelevant

- Search that uses such pruning is called *beam search*
  - This is the most widely used search optimization strategy

- The relative threshold $T$ is usually called “*relative beam width*” or just *beam width* or *beam*
Beam Search Visualization

• The set of lattice nodes actually evaluated is the active set
• Here is a typical “map” of the active region, aka beam (confusingly)

• Presumably, the best path lies somewhere in the active region
Unlike the fixed width approach, the computation reduction with beam search is unpredictable

- The set of *active nodes* at frames $j$ and $k$ is shown by the black lines

However, since the active region can follow any *warping*, it is likely to be relatively more efficient than the fixed width approach.
Determining the Optimal Beam Width

• Determining the optimal beam width to use is crucial
  – Using too narrow or tight a beam (too low $T$) can prune the best path and result in too high a match cost, and errors
  – Using too large a beam results in unnecessary computation in searching unlikely paths
  – One may also wish to set the beam to limit the computation (e.g. for real-time operation), regardless of recognition errors
• Unfortunately, there is no mathematical solution to determining an optimal beam width

• Common method: Try a wide range of beams on some test data until the desired operating point is found
  – Need to ensure that the test data are somehow representative of actual speech that will be encountered by the application
  – The operating point may be determined by some combination of recognition accuracy and computational efficiency
Determining the Optimal Beam Width

- Any value around the point marked $T$ is a reasonable beam for minimizing word error rate (WER)
- A similar analysis may be performed based on average CPU usage (instead of WER)
Beam Search Applied to Recognition

• Thus far, we considered beam search to prune search paths within a single template

• However, its strength really becomes clear in actual recognition (i.e. time synchronous search through all templates simultaneously)
  – In each frame, the beam pruning threshold is determined from the *globally* best node in that frame (from all templates)
  – Pruning is performed globally, based on this threshold
Beam Search Applied to Recognition

• Advantage of simultaneous time-synchronous matching of multiple templates:
  – Beams can be globally applied to all templates
  – We use the best score of all template frames (trellis nodes at that instant) to determine the beam at any instant
  – Several templates may in fact exit early from contention

• In the ideal case, the computational cost will be independent of the number of templates
  – All competing templates will exit early
  – Ideal cases don’t often occur
Pruning and Dynamic Trellis Allocation

• Since any form of pruning eliminates many trellis nodes from being expanded, there is no need to keep them in memory
  – Trellis nodes and associated data structures can be allocated on demand (i.e. whenever they become active)
  – This of course requires some book-keeping overhead

• May not make a big difference in small vocabulary systems
• But pruning is an essential part of all medium and large vocabulary systems
  – The search trellis structures in 20k word applications take up about 10MB with pruning
  – Without pruning, it could require more than 10 times as much!
Recognition Errors Due to Pruning

• Speech recognition invariably contains errors
• Major causes of errors:
  – Inadequate or inaccurate models
    • Templates may not be representative of all the variabilities in speech
  – Search errors
    • Even if the models are accurate, search may have failed because it found a *sub-optimal* path
• How can our DP/DTW algorithm find a sub-optimal path?
  – Because of pruning: it eliminates paths from consideration based on *local* information (the pruning threshold)
• Let $W$ be the best cost word for some utterance, and $W'$ the recognized word (with pruning)
  – In a *full* search, the path cost for $W$ is better than for $W'$
  – But if $W$ is not recognized when pruning is enabled, then we have a *pruning error* or *search error*
Measuring Search Errors

- How much of recognition errors is caused by search errors?
- We can estimate this from a sample test data, for which the correct answer is known, as follows:
  - For each utterance $j$ in the test set, run recognition using pruning and note the best cost $C_j'$ obtained for the result
  - For each utterance $j$, also match the correct word to the input without pruning, and note its cost $C_j$
  - If $C_j$ is better than $C_j'$ we have a pruning error or search error for utterance $j$
- Pruning errors can be reduced by lowering the pruning threshold (i.e. making it less aggressive)
- Note, however, this does not guarantee that the correct word is recognized!
  - The new pruning threshold may uncover other incorrect paths that perform better than the correct one
Summary So Far

- Dynamic programming for finding minimum cost paths
- Trellis as realization of DP, capturing the search dynamics
  - Essential components of trellis
- DP applied to string matching
- Adaptation of DP to template matching of speech
  - Dynamic Time Warping, to deal with varying rates of speech
- Isolated word speech recognition based on template matching
- Time synchronous search
- Isolated word recognition using automatic endpointing
- Dealing with errors using confidence estimation and N-best lists
- Improving recognition accuracy through multiple templates
- Beam search and beam pruning