11-751/ 18-781
Speech Recognition and Understanding

# Template Based Recognition

**Florian Metze**

September 9, 2013

Carnegie Mellon

## Agenda

- What do we have so far
- Problem motivation
- General principle

## What Do We Have So Far?

- Fundamental equation of speech recognition: $p(w|x)$ = … (Bayes formula)
  - Acoustic model $p(x|w)$
  - Language model $P(w)$
  - Pre-processing of audio signal $(x)$
- Parametric classification: likelihood $p(x|w)$
  - "Train" a model 1st, and "Search" brings it all together
- Let's start with something simpler
  - Single words/ complete utterances
  - Let's merge "model" and "features" ("patterns") – we can also compute a distance between two features

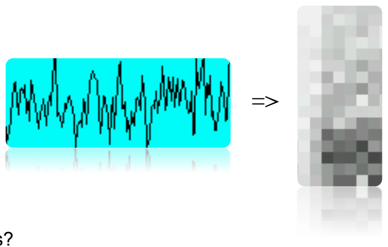
## Comparing Complete Utterances

So far

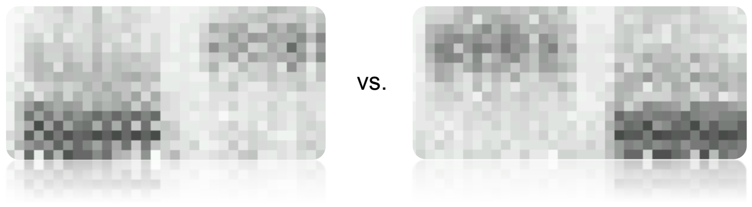
- Record sound signal (ADC)
- Compute frequency representation

We now have

- Sequence of pattern vectors
- We want similarity between two sequences
- Reduce problem to comparing individual patterns?



**Obviously:** order of vectors is important:



## Comparing Complete Utterances

Comparing speech vector sequences has to overcome three problems:

- Speaking rate: if the speaker is speaking faster, we get fewer vectors in the same time
- Changing speaking rate purposely: e.g. for disambiguation (said vs. sad)
- Changing speaking rate non-purposely: speaking dis-fluencies

vs.

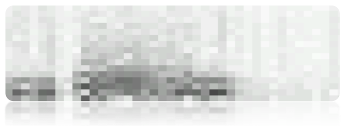
- So we have to find a way to decide which vector to compare to which
- Impose some (ordering, …) constraints (not every vector can be compared to every other) – using the Language Model (LM)

## Endpoint Detection

When comparing two recorded utterances there might be:

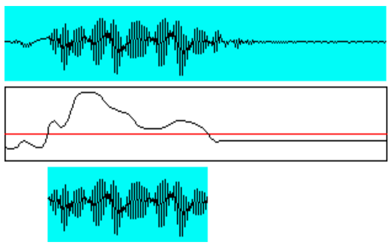
- utterances are of different length
- one or both utterances can be preceded or followed by a period of (possibly non-voluntarily recorded) silence

vs.



Typical Solution:

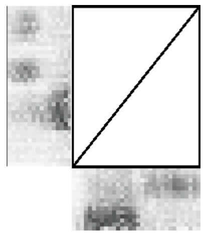
- compute signal power: $p[i...j] = \sum_{k=i...j} s[k]^2$
- then apply threshold to detect speech
- also: we might not have any mechanism to signalize the recognizer when it should listen



## Approaches to Sequence Alignment

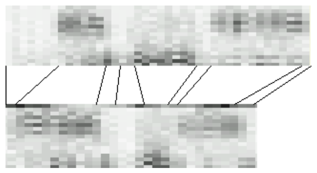
First idea:

- normalize length and
- make linear alignment



Linear alignment

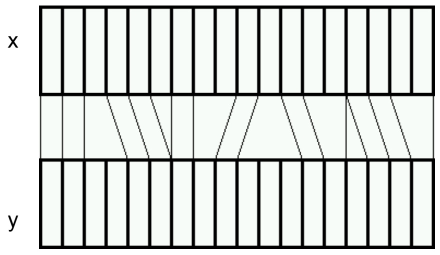
- can handle the problem of different speaking rates
- but it can not handle varying speaking rates within the same utterance (lengthening), or noises, insertions, etc …



## Alignment of Speech Vectors may be Non-Bijective

Task:

- Given: two sequences $x_1, x_2, ..., x_n$ and $y_1, y_2, ..., y_m$
- Wanted: alignment relation R (not a function), were (i,j) is in R iff $x_i$ is aligned with $y_j$

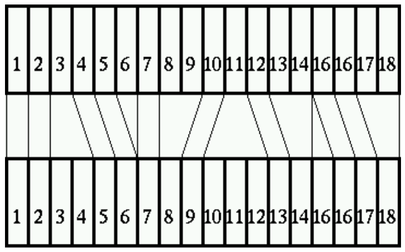


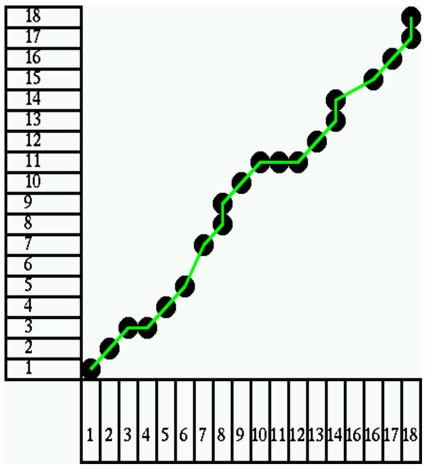
It is possible that more than one x is aligned to the same y (or vice versa)

It is possible that more than an x or a y has no alignment partner at all

## Time Warping

- Given: two sequences $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$
- Wanted: R (not a function), where (i,j) is in R iff $x_i$ is aligned with $y_j$
- We are effectively looking for a common time axis!





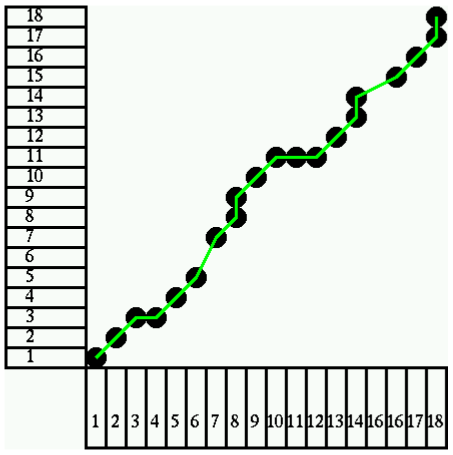
## Distance Measure Between Two Utterances

For a given **path** $R(i,j)$, the distance between $x$ and $y$ is the sum of all **local distances** $d(x_i,y_j)$.

In our example:
$d(x_1,y_1) + d(x_2,y_2) + d(x_3,y_3) + d(x_3,y_3) + d(x_5,y_4) + d(x_6,y_5) + d(x_7,y_7) + \ldots$

**Question:**
How can we find a path that gives the minimal overall distance?

Remember that "distances" are expressed as a "score"
d := -log(p)



## The Minimal Editing Distance Problem

**Given:** two character sequences (words) $x_1,x_2,...,x_n$ and $y_1,y_2,...,y_m$

**Wanted:** the minimal number (and sequence) of editing steps that are needed to convert x into y

The editing cursor starts at $x_0$, an editing step can be one of:

- Delete the character $x_i$ under the cursor
- Insert a character $x_i$ at the cursor position
- Replace character $x_i$ at the cursor position with $y_j$
- Moving the cursor to the next character is no editing step, and we can't go back

**Example:** convert x= "BAKERY" to y= "BRAKES"

- B = B, move cursor to next character
- Insert character $y_2$= R
- A = A, move cursor to next character
- K = K, move cursor to next character
- E = E, move cursor to next character
- Replace character $x_5$= R with character $y_5$= S
- Delete character $x_6$= Y (sequence not necessarily unique)

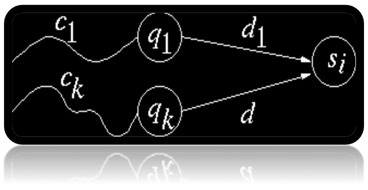
## Dynamic Programming

How can we find the minimal editing distance?

- "Greedy" algorithm? Always perform the step that is currently the cheapest. If there are more than one cheapest step, take any one of them.
- Obvious: can't guarantee to lead to the optimal solution.
- Solution: Dynamic Programming (DP). DP is frequently used in operations research, where consecutive decisions depend on each other and whose sequence must lead to optimal results.

The key idea of DP is:

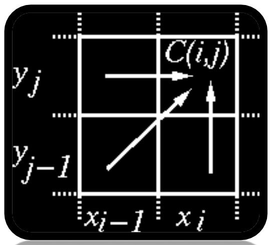
- If we would like to take our system into a state $s_i$, and we know the costs $c_1, ..., c_k$ for the optimal ways to get from the start to all states $q_1, ..., q_k$ from which we can go to $s$,
- then the optimal way to $s$ goes over the state $q_l$ where $l = \text{argmin}_j\ c_j$



## The Dynamic Programming Matrix

For finding the minimal editing distance from $x_1, x_2, ..., x_n$ to $y_1, y_2, ..., y_m$ we can define an inductive algorithm. Let $C(i, j)$ denote the minimal editing distance from $x_1, x_2, ..., x_i$ to $y_1, y_2, ..., y_j$. We then get:

- $C(0,0) = 0$ (no characters/ no editing)
- $C(i, j)$ is either (whichever is smallest)
  - $C(i\text{-}1, j\text{-}1)$ plus the cost for replacing $x_i$ with $y_j$ or
  - $C(i\text{-}1, j)$ plus the cost for deleting $x_i$ or
  - $C(i, j\text{-}1)$ plus the cost for inserting $y_j$



Usually:

- The cost for deleting or inserting a character is $1$
- The cost for replacing $x_i$ with $y_j$ is $0$ (if $x_i = y_j$) or $1$ (else)
- It might be useful to define other costs for special purposes

Eventually:

- Remember for each state $(i\text{-}1, j\text{-}1)$ which one was the best predecessor ("back-pointer")
- To find the sequence of editing steps: backtrace the predecessor pointers from final state

## Utterance Comparison by Dynamic Time Warping

How can we apply the DP algorithm for the minimal editing distance to the utterance comparison problem?

**Differences and Questions:**

- What do editing steps correspond to?
- We "never" really get two identical vectors.
- We are dealing with continuous and not discrete signals here.

**Answers:**

- We can delete/ insert/ substitute vectors. Define cost for deleting/ inserting as we wish, define cost for substituting = distance between vectors
- No two vectors are the same? So what
- Continuous signals $\rightarrow$ so we get continuous distances (no big deal)

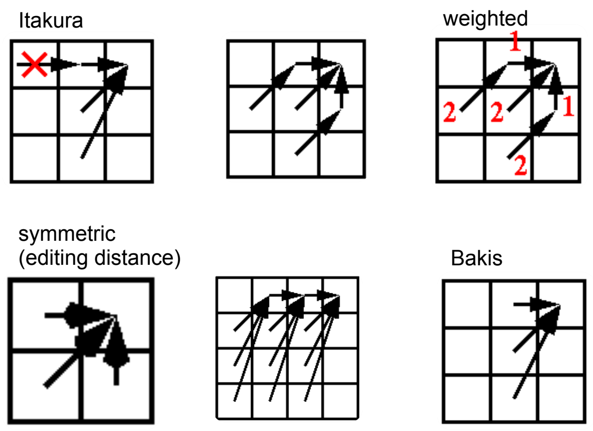
**The DTW-Algorithm**

- Works like the minimal editing distance algorithm with minor modifications:
- Allow different kinds of steps (different predecessors of a state).
- Use vector-vector distance measure as cost function.

## DTW Steps

Many different warping steps are possible and have been used. Examples:

General rule is: Cumulative cost of destination = best-of (cumulative cost of source + cost of step + distance in destination)



## DTW Steps (2)

- Admissible warping steps influence optimal alignment
- Influence overall shape of path
- Should be chosen “reasonably”
- Can be “weighted” and “un-weighted”

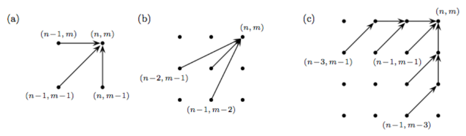


**Fig. 4.5.** Illustration of three different step size conditions, which express different local constraints on the admissible warping paths. **(a)** corresponds to the step size condition (iii) of Definition 4.1

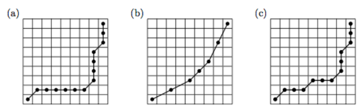
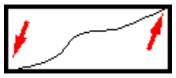


**Fig. 4.6.** Three warping paths with respect to the different step size conditions indicated by Fig. 4.5. **(a)** Step size condition of Fig. 4.5a may result in degenerations of the warping path. **(b)** Step size condition of Fig. 4.5b may result in the omission of elements in the alignment of $X$ and $Y$. **(c)** Warping path with respect to the step size condition of Fig. 4.5c

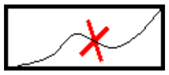
## Constraints for the DTW Path

interACT

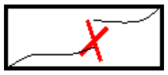
Endpoint constraints: we want the path to not skip a part at the beginning or end of the utterance



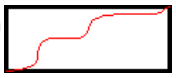
Monotonicity conditions: we can't go back in time (for any utterance)



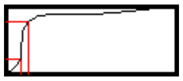
Local continuity: no jumps etc



Global path constraints: path should be close to diagonal

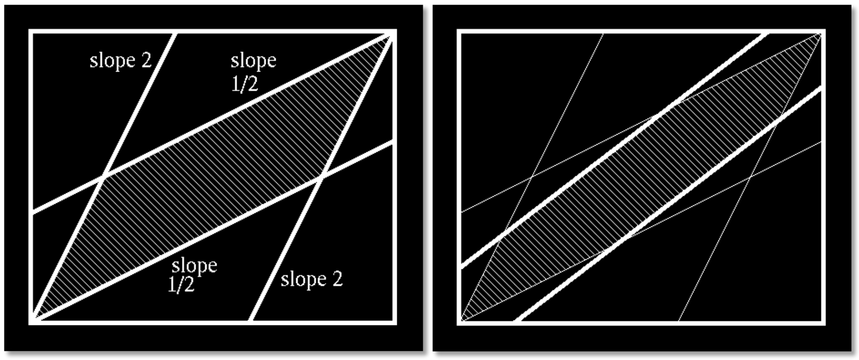


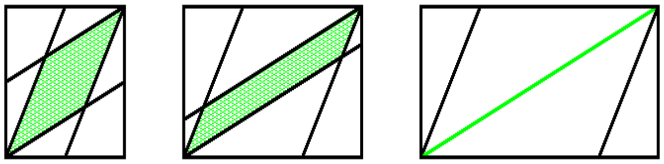
Slope weighting: we believe the DTW path sjould be somewhat „smooth"



## Global Constraints for the DTW Path

interACT





← only one path

## Java Source Code for DTW

```
public double dtw() { for (int r=0; r<rowN; r++) {
accu[0][r] = 9.9e99; accu[1][r] = 9.9e99; }
accu[0][0] = distance(0,0); for (int c=1; c<colN;
c++) { int curr = (c-1)%2, next = c%2; for (int r=0;
r<rowN; r++) { accu[next][r] = 9.9e99; double d =
distance(c,r); if (r>1) if (accu[curr][r-2] + d <
accu[next][r]) { accu[next][r] = accu[curr][r-2] +d;
back[c][r]=r-2; } if (r>0) if (accu[curr][r-1] + d <
accu[next][r]) { accu[next][r] = accu[curr][r-1] +d;
back[c][r]=r-1; } if (accu[curr][r ] + d <
accu[next][r]) { accu[next][r] = accu[curr][r ] +d;
back[c][r]=r; } } } return accu[curr][rowN-1]; }
```

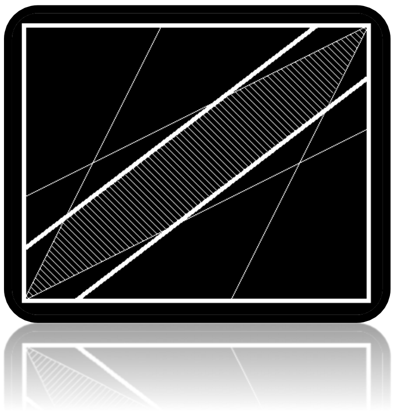
## The DTW Search Space

Already suggested: restrict search space by window around diagonal. Caveats:

- Silence period in one utterance can cause "edgy" path
- Search area becomes too restricted when utterance durations differ a lot

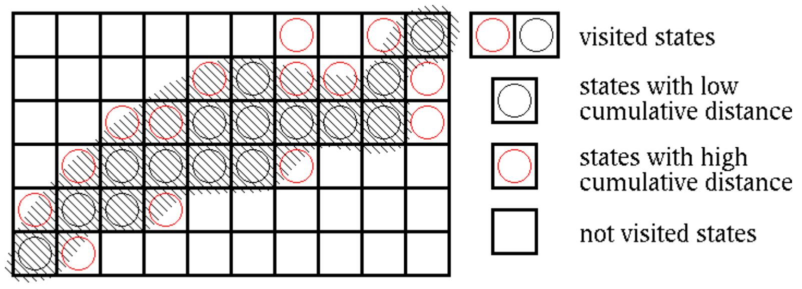
Other reason (besides global path constraints) for restricting search space:

- **Save time:** A window that has a constant width, reduces the search effort from O($n^2$) to O(n)
- To overcome caveats of "diagonal window" restriction, use: **beam search.**



## DTW with Beam Search

Idea: do not consider steps to be possible out of states that have “too high” cumulative distances.


visited states
states with low cumulative distance
states with high cumulative distance
not visited states

Approaches:

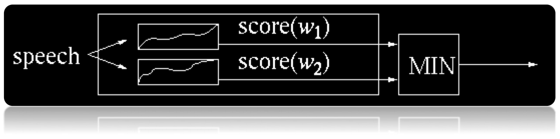
- “expand” only a fixed number of states per column of DTW matrix
- expand only states that have a cumulative distance less than a factor (the “beam”) times the best distance so far

## The Principles of Designing Speech Recognizers

- The following approach is generally considered to be good research habit:
- Task specification (what must be recognized)
- Data collection
- Split up collected data into
  - Train set (used for estimating the recognizers parameters)
  - Development set (used for evaluating the recognizer during optimization)
  - Evaluation set (used only once and never again to report results)
- Estimate parameters of recognizer with training data
- Evaluate and optimize recognizer with development data
- Run test on evaluation data and report results

## What Are We Trying to Do Now?

We will build a first simple isolated-word recognizer using DTW. The recognizer will record speech and print the score for each of its reference patterns:



- Build recognizer that can recognize two words $w_1$ and $w_2$
- Collect training examples (one per word in demo, in real life: a lot more)
- Skip the optimization phase (don't need development set)
- Collect evaluation data (a few examples per word)
- Run tests on evaluation data and report results

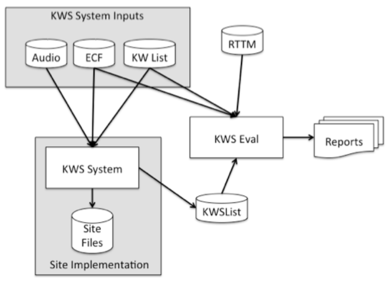
## Isolated Word Recognition with Template Matching

- For each word in the vocabulary, store at least one reference pattern
- When multiple reference patterns are available, either use all of them or compute an average
- During recognition
  - Record a spoken word
  - Perform pattern matching with all stored patterns (or at least with those that can be used in the current context)
  - Compute a DTW score for every vocabulary word (when using multiple references, compute one score out of many, e.g. average or max)
  - Recognize the word with the best DTW score
- This approach works only for very small vocabularies and/ or for speaker-dependent recognition
- Forms the basis of Hidden Markov Model based recognizers

## The Next Step: Keyword Spotting

- Still “Easier than Speech to Text”?
- What can we do with DTW? Find “matches” of “examples”
  - We need to extend DTW so that it can cope with arbitrary begin and end points
  - But that’s not a big problem (“segmental DTW”) and helps to improve efficiency
  - We don’t need a language model or other information: “zero resource” technique
- Then we can do “grep -F” in a corpus
  - Find occurrences, repetitions, … of words (or word sequences) in audio data
  - If we don’t have audio examples, but phone strings or similar information:
    … OW **HH EH L OW W ER L T** H AW ER .. -> “HELLO WORLD” detected at 3 seconds
- Keyword Search/ Spotting: do not convert speech to text, but detect the presence or absence of specific keywords (at points in time)

## Spoken Term Detection (STD) and Keyword Spotting/ Search (KWS)



- NIST 2006 STD evaluation
- ECF & RTTM define the regions to process
- “Site files” is like an index, generated by the system
- Index can be generated with knowledge of the keywords (better, less flexible) or without (harder, more flexible)

## STD and KWS

- If the index is the audio, we can use DTW to search and don't need to have a separate index
- Otherwise we can use the output of a word or phone recognizer, and search for the string
- Practical importance!
  - Media monitoring, government applications, …
  - Large archives and very fast search (just as in text)
- Biggest differentiator: are the keywords known when you create the index?
  - Yes: just run speech to text
  - No: do something more complex

## Evaluation

- KWS/ STD is essentially an Information Retrieval Problem
- Precision and Recall are good metrics
- Should take into account length of speech and expected number of correct hits somehow
- Should also be computed on a per-term basis (i.e. may want to get some insight into which terms are detected well, which ones are not)
- "Detection" will typically be based on some (term specific) threshhold

## Actual Term Weighted Value (ATWV)

$$ATWV = \frac{1}{\mathrm{N_{terms}}} \sum \mathrm{Value}(term)$$

$$\mathrm{Value}(term) = 1 - \mathrm{P_{Miss}}(term) - \beta \cdot \mathrm{P_{FA}}(term)$$

$$\mathrm{P_{Miss}}(term) = 1 - \frac{\mathrm{N_{correct}}(term)}{\mathrm{N_{true}}(term)} \qquad \mathrm{P_{FA}}(term) = \frac{\mathrm{N_{spurious}}(term)}{T_{speech} - \mathrm{N_{true}}(term)}$$

where :

$$T_{speech} = \text{duration of search corpus in seconds} \qquad \beta \approx 1{,}000$$

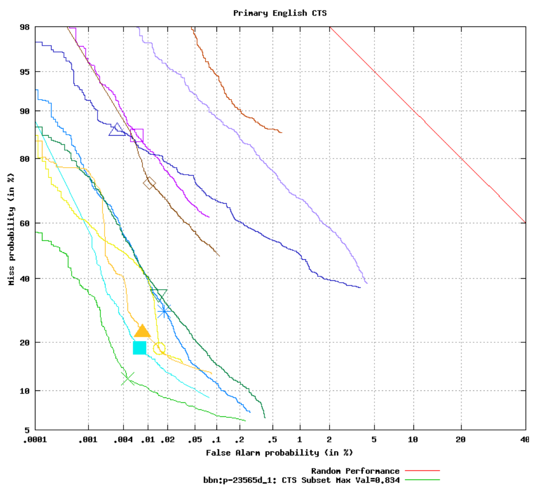
## Understanding ATWV

- Perfect ATWV = 1.0
- Mute detector has ATWV = 0.0
- Negative ATWV is possible.
- Motivated by application-based costs:

$$Value \approx \frac{V \cdot \mathrm{N_{correct}} - C \cdot \mathrm{N_{spurious}}}{V \cdot \mathrm{N_{true}}}$$

- All search terms are weighted equally
- False alarm cost is almost constant, but miss cost varies by term.
    - Missing an instance of a rare term is expensive.
    - Missing an instance of a frequent term cheap.

## Evaluation with DET Curves



## Backup

## Where are we

- Have discussed very basics of speech to text (evaluation, nature of speech)
- Next Wednesday, we'll present term projects
- Then: signal processing (what is "$x$"?) and phonetics and phonology ("what are good target units $s$ for classification", what is a "dictionary" $W \rightarrow s$)
- Then we will be ready to talk more about $p(W)$ and $p(x|W)$

## Literature/ Reading

- Chapter 12.2 "Search Algorithms for Speech Recognition" (and 12.1) in Huang/ Acero/ Hon
- "Rapid and Accurate Spoken Term Detection", David R. H. Miller, Michael Kleber, Chia-lin Kao, Owen Kimball Thomas Colthurst, Stephen A. Lowe, Richard M. Schwartz, and Herbert Gish
- "Results of the 2006 Spoken Term Detection Evaluation", Jonathan G. Fiscus, Jerome Ajot, John S. Garofolo, and George Doddington
- "Dynamic Programming Algorithm Optimization for Spoken Word Recognition", Hiroaki Sakoe and Seibi Chiba