Dynamic Time Warping (DTW) for Single Word and Sentence Recognizers

Reference: Huang et al. Chapter 8.2.1; Waibel/Lee, Chapter 4
Overview (I)

- Simplified Decoding
- Representation
- Simplified Classifier Training
- Pattern Recognition
- Supervised – Unsupervised Training
- Supervised Classification
- Classifier Design in Practice
- Gaussian Densities
  - Mixtures of Gaussian Densities
- Finding Codebooks of Reference Vectors - The $k$-Means Algorithm
- Problems of Classifier Design
- Curse of Dimensionality
- Trainability
- Simplified Decoding and Training
Overview (II)

- Dynamic Programming and Single Word Recognition
  - Comparing Complete Utterances
  - Endpoint Detection
  - Speech Detection
  - Approaches to Alignment of Vector Sequences
  - Alignment of Speech Vectors May Be Non-Bijective
  - Solution: Time Warping
  - What is the best alignment relation R?
- Dynamic Programming
  - Key Idea of Dynamic Programming
  - The Dynamic Programming Matrix
  - The Minimal Editing Distance Problem
  - Levinshtein
- Utterance Comparison by Dynamic Time Warping (DTW)
  - What we could do already
  - Compare Complete Utterances
  - DTW Summary
Overview (III)

• Dynamic Programming and Single Word Recognition
  – Isolated Word Recognition with Template Matching
  – From Isolated to Continuous Speech
    • Plan: Cut Continuous Speech Into Single Words
    • Compare Complete Utterances / Words
    • Compare Smaller Units
    • What we can’t do yet OR Problems with Pattern Matching
    • Make a Wish
  – Speech Production as Stochastic Process
    – What’s different?
    – Keep in Mind for HMM Session
Overview

- Simplified Decoding
- Representation
- Simplified Classifier Training
- Pattern Recognition
- Supervised – Unsupervised Training
- Supervised Classification
- Classifier Design in Practice
- Gaussian Densities
  - Mixtures of Gaussian Densities
- Finding Codebooks of Reference Vectors - The \( k \)-Means Algorithm
- Problems of Classifier Design
- Curse of Dimensionality
- Trainability
- Simplified Decoding and Training
Simplified Decoding

Recognition Units: Phonemes

Phoneme Classifier or any kind of Reference Vectors

Speech

Feature extraction

Speech features

Decision

Hypotheses (phonemes)

/h/ /e/ /l/ /o/ /w/ /o/ /r/ /l/ /d/

/o/

/0 ... t ... T
Representation could be a database of stored example samples
Or a (statistical) model → train a classifier

This is a plot of measured formants for different vowels from different speakers:

The so-called vowel-triangle expresses which vowels have which formants in average:

- **F1**: major resonance of the pharyngeal cavity
- **F2**: major resonance of the oral cavity
Simplified Classifier Training

Train Classifier
Use aligned speech vectors (e.g. all frames of phoneme /e/) to train the reference vectors of /e/ (= Codebook)
Pattern Recognition – Review

- **Static Patterns,**
  i.e. no dependency on time or sequential order

- **Approaches:**
  - **Knowledge-based approaches:**
    1. Compile knowledge
    2. Build decision trees
  - **Connectionist approaches:**
    1. Automatic knowledge acquisition, "black-box" behavior
    2. Simulation of biological processes
  - **Statistical approaches:**
    1. Build a statistical **model** of the "real world"
    2. Compute probabilities according to the models
Pattern Recognition

• Important Notions:
  – Supervised - Unsupervised Classifiers
  – Parametric - Non-Parametric Classifiers
  – Linear - Non-linear Classifiers

• Classical Statistical Methods:
  – Bayes Classifier
  – K-Means

• Connectionist Methods:
  – Perceptron
  – Multilayer Perceptrons
Supervised – Unsupervised Training

- **Supervised training:**
  - Class to be recognized is known for each sample in training data.
  - Requires a priori knowledge of useful features and knowledge.
  - Labeling of each training token (→ cost).

- **Unsupervised training:**
  - Class is not known and structure is to be discovered automatically.
  - Feature-space-reduction
  - E.g.: Clustering, auto-associative nets
Supervised Classification

- **Classification:**
  - Classes Known: Phonemes /i/, /a/, /u/
  - Features: F1 and F2 (Hz)
  - Classifiers
Classifier Design in Practice

- Need:
  - a priori probability $P(\omega_i)$ (not too bad)
  - class conditional probability density function (PDF) $p(x / \omega_i)$

- Problems:
  - limited training data
  - limited computation
  - class-labeling potentially costly and prone to error
  - classes may not be known
  - good features not known

- Parametric Solution:
  - Assume that $p(x / \omega_i)$ has a particular parametric form
  - Most common representative: multivariate normal density
Gaussian Densities (1)

- The most often used model for (preprocessed) speech signals are **Gaussian densities**.
- Often the "size" of the parameter spaces is measured in "number of densities"
- A Gaussian density of a random variable \( x \) looks like this:

\[
N(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(x - \mu)^2\right]
\]

Its parameters are:
- the **mean** \( \mu \)
- the **variance** \( \sigma^2 \)
Gaussian Densities (2)

- A multivariate Gaussian density with $D$ dims. looks like this:

$$N(x | \mu, \Sigma^2) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp \left[ -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right]$$

Its parameters are:

- the mean vector $\mu$  
  (a vector with $D$ coefficients)

- the covariance matrix $\Sigma$  
  (a symmetric $DxD$ matrix), if $x$ independent, $\Sigma$ is diagonal

- the determinant of the covariance matrix $|\Sigma|$
Mixtures of Gaussian Densities

- Often the shape of the set of vectors that belong to one class does not look like what can be modeled by a single Gaussian.
- A (weighted) sum of Gaussians can *approximate* many more densities:

\[ p_{Mix}(x) = \sum_{m=1}^{M} w_m N(x \mid \mu_m, \Sigma_m) = \sum_{m=1}^{M} w_m \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp \left[ -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right] \]

- Usually, each Gaussian has a weight \( w_m \) \((m = 1, \ldots, M; M \text{ number of Gaussians})\).
- In general, a class can be modeled as a **mixture of Gaussians**:

Left: Approximation with a single Gaussian (distribution)

Right: Approximation with 2 Gaussians.
Mixtures of Gaussian Densities

Other pros and cons:

- **Flexibility**
  - We can adapt our models to existing training data, e.g. by selecting the number of distributions dependent on the amount of training data.
  - This allows a flexible adjustment of the set of parameters: The more parameters \((\mu_i, \Sigma_i)\) we want to train in a system, the more training data we need – but the better the classification!
  - There are algorithms to find the optimum between amount of training data and modeling accuracy.

- **Parameter Tying**
  - We can also save parameters if there are not enough training data available.
  - We can use identical Gaussians for different classes but assign different mixture weights to the individual distributions.
  - E.g. begin, middle and end of phonemes are often modeled with identical Gaussians but different mixture weights.
Finding Codebooks of Reference Vectors – The $k$-Means Algorithm (1)

- **Goal**: Partition $n$ samples (observations) into $k$ classes (clusters) in which each sample belongs to the class with the nearest *mean*.

- Given a set of samples $(x_1, x_2, ..., x_n)$, where each sample is a $D$-dimensional real vector, $k$-means clustering aims to partition the $n$ samples into $k$ classes ($k \leq n$) $S = \{S_1, S_2, ..., S_k\}$ so as to minimize the within-cluster sum of squares (WCSS):

$$J = \sum_{k=1}^{K} \sum_{n=1}^{N_k} \left\| x_n^{(k)} - \mu^{(k)} \right\|^2$$

where $\mu_i$ is the mean of points in $S_i$.

- **Problem**: $\mu^{(k)}$ itself is dependent on the class assignment
  - Optimal assignment is computationally difficult (NP-hard)
  - Use iterative algorithm!
Finding Codebooks of Reference Vectors – The $k$-Means Algorithm (2)

- The algorithm uses an iterative refinement technique:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
</table>
| Step 1 | **Initialize:**
| | Given a value of $k$ and sample vectors $v_1, \ldots, v_T$, initialize any $k$ means (e.g. $\mu_i = v_i$) |
| Step 2 | **Nearest-Neighbor classification:**
| | Assign every vector $v_i$ to its class' centroid $\mu_{f(i)}$ |
| Step 3 | **Codebook update:**
| | Replace every mean $\mu_i$ by the mean of all sample vectors that have been assigned to it |
| Step 4 | **Iteration:** If not satisfied, yet, then go to Step 2 |

- Possible stop-criteria:
  - A fixed number of iterations
  - The average (maximum) distance $|v_i - \mu_{f(i)}|$ is below a fixed value
  - The derivative of the distance is below a fixed value
    (nothing happens any more)
Finding Codebooks of Reference Vectors – The \( k \)-Means Algorithm (3)

Initial scatter diagram without any additional information

After \( k \)-means clustering (with \( k=3 \)):
(The class' centroids are shown as squares.)
Finding Codebooks of Reference Vectors – The $k$-Means Algorithm (4)

• Typical issues:
  - Theoretically $k$-means can only converge to a local optimum
  - Initialization is often critical
    → Repeat $k$-means for several codebook sets
    OR:
    → Linde-Buzo-Gray (LBG) Algorithm:
      I.e. start with a 1-vector codebook and use splitting algorithm to obtain 2-vector, ..., $M$-vector codebook
Problems of Classifier Design

• Features:

  – What and how many features should be selected?

  – Any features?

  – The more the better?

  – If additional features not useful (same mean and covariance), classifier will automatically ignore them?
Curse of Dimensionality

• Adding more features
  – Adding independent features may help
  – BUT: Adding indiscriminant features may lead to worse performance!

• Reason:
  – Training Data vs. Number of Parameters
  – Limited training data

• Solution:
  – Select features carefully
  – Reduce dimensionality
  – Principle Component Analysis (PCA)
Trainability

- The number of distributions must be well chosen and depending on the amount of training data.

Example:
- Two-phoneme classification example (Huang et al.)
- Phonemes modeled by Gaussian mixtures
- Parameters are trained with a varied set of training samples
Simplified Decoding and Training

1. **Speech**
   - Feature extraction

2. **Speech features**
   - Decision (apply trained classifiers)

3. **Hypotheses** (phonemes)
   - /h/ /e/ /l/ /o/ /w/ /o/ /r/ /l/ /d/

---

**Aligned Speech**

1. **Speech**
   - Feature extraction

2. **Speech features**

3. **Train Classifier**
   - Train codebook - kmeans
Overview

- Dynamic Programming and Single Word Recognition
  - Comparing Complete Utterances
  - Endpoint Detection
  - Speech Detection
  - Approaches to Alignment of Vector Sequences
  - Alignment of Speech Vectors May Be Non-Bijective
  - Solution: Time Warping
  - What is the best alignment relation R?
  - Dynamic Programming
    - Key Idea of Dynamic Programming
    - The Dynamic Programming Matrix
    - The Minimal Editing Distance Problem
    - Levenshtein
  - Utterance Comparison by Dynamic Time Warping (DTW)
    - What we could do already
    - Compare Complete Utterances
    - DTW Summary
Comparing Complete Utterances

- **What we had so far:**
  - Record a sound signal
  - Compute frequency representation
  - Quantize/classify vectors

- **We now have:**
  - A sequence of pattern vectors

- **What we want:**
  - The similarity between two such sequences

Obviously: The order of vectors is important!
Comparing Complete Utterances

Comparing speech vector sequences has to overcome 3 problems:

1) Speaking rate characterizes speakers (speaker dependent!)
   if the speaker is speaking faster, we get fewer vectors

2) Changing speaking rate by purpose: e.g. talking to a foreign person

3) Changing speaking rate non-purposely: speaking disfluencies

→ So we have to find a way to decide which vectors to compare to another
   - Impose some constraints (compare every vector to all others is too costly)
Endpoint Detection

When comparing two recorded utterances we face 2 problems:

1) *When does the speech begin?*
   - We might not have any mechanism to signal the recognizer when it should listen.

2) *Varying length of utterance*
   - Utterances might be of different length (speaking rate, …)
   - One or both utterances can be preceded or followed by a period of (possibly non-voluntarily recorded) silence
Speech Detection

Solution to Problem 1) - When does speech begin?

A: **Push-to-talk scenario:** Only listen when user pushes button
B: **Always on scenario:** Always listen, only consider speech regions

**Select Speech Regions:**

- Use signal-to-noise ratio (**SNR**):
  works well if SNR > 30dB, otherwise problematic
- Compute **signal power**:
  \[ p[i..j] = \sum_{k=i..j} s[k]^2, \text{ then apply a threshold } t \text{ to detect speech} \]
First idea to overcome the varying length of Utterances (Problem 2):

1. Normalize sequence length
2. Make a **linear alignment** between the two sequences

Linear alignment can handle the problem of *different* speaking rates
But ..... 
What about *varying* speaking rates?
• Linear alignment can handle the problem of different speaking rates

• **But:** It can not handle the problem of *varying* speaking rates during the same utterance.
Alignment of Speech Vectors May Be Non-Bijective

Given: Two sequences $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$

Wanted: Alignment relation $R$ (not function), where $(i, j)$ is in $R$ iff $x_i$ is aligned with $y_j$.

It is possible that …

... more than one $x$ is aligned to the same $y$ (e.g. $x_3, x_4$)

... more than one $y$ is aligned to the same $x$ (e.g. $y_8, y_9$)

... more than an $x$ or a $y$ has no alignment partner at all (e.g. $y_6$)
Solution: Time Warping

Given: Two sequences $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$

Wanted: Alignment relation $R$, where $(i,j)$ is in $R$ iff $x_i$ is aligned with $y_j$ i.e. we are looking for a common time-axis:

$y$

18
17
16
15
14
13
12
11
10

$x$

1
2
5
6
7
9
10
11
12
13
14
16
16
17
18

$y_8, y_9$ align to the same $x$

$y_6$ has no partner

$x_3, x_4$ align to the same $y$
What is the best alignment relation $R$?

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:

$$d(x_1, y_1) + d(x_2, y_2) + d(x_3, y_3) + d(x_4, y_3) + d(x_5, y_4) + d(x_6, y_5) + d(x_7, y_7) + ...$$

Question:
How can we find a path that gives the minimal overall distance?
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 $s_i$,

→ then the optimal way to $s_i$ goes over the state $q_l$ where $l = \text{argmin}_j c_j$
The Dynamic Programming Matrix (1)

- To find the minimal editing distance from
  \(x_1, x_2, \ldots, x_n\) to \(y_1, y_2, \ldots, y_m\),
  we can define an algorithm inductively:

  - Let \(C(i, j)\) denote the *minimal editing distance*
    from \(x_1, x_2, \ldots, x_i\) to \(y_1, y_2, \ldots, y_j\).

  - Then we get:
    - \(C(0,0) = 0\) (no characters no editing)
    - \(C(i, j)\) is either (whichever is smallest):
      - \(C(i-1, j-1)\) plus the cost for replacing \(x_i\) with \(y_j\)
      - \(C(i-1, j)\) plus the cost for deleting \(x_i\)
      - \(C(i, j-1)\) plus the cost for inserting \(y_j\)
The Dynamic Programming Matrix (2)

- Usually for the minimal editing distance:
  - 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)
  - Might be useful to define other costs for special purposes

- Eventually:
  - Remember for each state $(i-1, j-1)$ which one was the best predecessor (backpointer)
  - Find the sequence of editing steps by backtracing the predecessor pointers from the final state
The Minimal Editing Distance Problem

Given: Two character sequences (words) $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_m$

Wanted: The minimal number (and sequence) of editing steps that are needed to convert $x$ to $y$

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 (no editing), we can't go back

Example: Convert $x = \text{"BRAKES"}$ to $y = \text{"BAKERY"}$ (one possible solution):

- B = B, move cursor to next character
- Delete character $x_2 = R$
- A = A, move cursor to next character, K = K, move, E = E, move
- Replace character $x_5 = S$ with character $y_5 = R$
- Insert character $y_6 = Y$ (sequence not necessarily unique)

Often referred to as Levinshtein distance, keep in mind, we will revisit this when we talk about how to measure the Performance (Word Accuracy) of a recognizer.
Levenshtein

Y
R
E
K
A
B

B = B, move to next
Levenshtein

Delete character $x_2 = R$
Levenshtein

A=A, move to next
Levenshtein

K=K, move to next
Levenshtein

\[ E=E, \text{ move to next} \]
Levenshtein

replace character $x_6 = S$ with character $y_5 = R$
Levenshtein

insert character $y_6 = Y$
Levenshtein

Sequence is not necessarily unique!

insert character $y_5 = R$
replace character $x_6 = S$ with $y_6 = Y$
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 del/ins, define cost for sub = distance between vectors
- No two vectors are the same? So what.
- Continuous signals → we get continuous distances (no big deal)

The DTW-Algorithm:
- Works like the minimal editing distance algorithm
- Minor modification:
  Allow different kinds of steps (different predecessors of a state)
- Use vector-vector distance measure as cost function
What we could do already

- We can build a first simple isolated-word recognizer using DTW
- We can build a preprocessor such that recorded speech can be processed by the recognizer
- We can recognize speech using DTW and print the score for each of its reference patterns:

Example:
- Build recognizer that can recognize two words $w_1$ and $w_2$
- Collect training examples (in real life: a lot of data)
- 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
Compare Complete Utterances

DTW for Single Word and Sentence Recognizers

Ref Word 1

Ref Word 2

speech

score(w₁)

score(w₂)

MIN

Hypothesis
DTW Summary

- **Optimization of DTW:**
  - Usually, only interested in final score
  - Algorithm requires only values in current and previous frame
  - Keep it simple: Do not allocate new storage but overwrite stuff that is no longer needed
  - For most transition patterns, one frame is enough

- **Drawbacks of DTW:**
  - Does not generalize
  - Speaker dependent
  - Need example(s) for each word from each speaker
  - Gets computationally expensive for large vocabularies
Overview

- Dynamic Programming and Single Word Recognition
  - Isolated Word Recognition with Template Matching
  - From Isolated to Continuous Speech
    - Plan: Cut Continuous Speech Into Single Words
    - Compare Complete Utterances / Words
    - Compare Smaller Units
    - What we can’t do yet OR Problems with Pattern Matching
    - Make a Wish
  - Speech Production as Stochastic Process
    - What’s different?
    - Keep in Mind for HMM Session
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
From Isolated to Continuous Speech

- Sloppier
- Higher speaking rate
- Combinatorial explosion of things that can be said
- Spontaneous effects: restarts, fragments, noise
- Co-articulation: Did you → dija ..
- Segmentation: how to find word boundaries

- Solution: reduce to known problems
Plan 1: Cut Continuous Speech Into Single Words

- Write magic algorithm that segments speech into 1-word chunks
- Run DTW/Viterbi on each chunk
- BUT: Where are the boundaries ????

• No reliable segmentation algorithm for detecting word boundaries other than doing recognition itself, due to:
  – Co-articulation between words
  – Hesitations within words
  – Hard decisions lead to accumulating errors

⇒ Integrated approach works better
Compare Complete Utterances / Words

Hypothesis = recognized sentence?
Compare Smaller Units

Hypothesis = recognized sentence
What we can‘t do yet
OR Problems with Pattern Matching

• Need endpoint detection
• Need collection of reference patterns (inconvenient for user)
• Works only well for speaker-dependent recognition
  (difficult to cover variations)
• High computational effort (esp. for large vocabularies),
  proportional to vocabulary size
• Large vocabulary also means: need huge amount of training data
  since we need training samples for each word
• Difficult to train suitable references (or sets of references)
• Poor performance when the environment changes
• Unsuitable where speaker is unknown and no training is feasible
• Unsuitable for continuous speech, coarticulation
  (combinatorial explosion of possible patterns)
• Impossible to recognize untrained words
• Difficult to train/recognize subword units
Make a Wish

- We would like to work with speech units shorter than words
  → each subword unit occurs often, training is easier, need less data
- We want to recognize speech from any speaker, without prior training
  → store "speaker-independent" references
- We want to recognize continuous speech not only isolated words
  → handle coarticulation effects, handle sequences of words
- We would like to be able to recognize words that have not been trained
  → train subword units and compose any word out of these
    (vocabulary independence)
- We would prefer a sound mathematical foundation
- Solution (particularly successful for ASR): Hidden Markov Models
Speech Production as Stochastic Process

- The same word / phoneme / sound sounds different every time it is uttered.
- We can regard words / phonemes as states of a speech production process.
- In a given state we can observe different acoustic sounds.
- Not all sounds are possible / likely in every state.
- We say: In a given state the speech process "emits" sounds according to some probability distribution/density.
- The production process can make transitions from one state into another.
- Not all transitions are possible, transitions have different probabilities.

When we specify the probabilities for sound-emissions (emission probabilities) and for the state transitions, we call this a model.
What’s different?

Reference in terms of state sequence of statistical models, models consists of prototypical references vectors.

Hypothesis = recognized sentence
Keep in Mind for HMM Session

Hypothesis = recognized sentence
Thanks for your interest!