5 Automatic Speech Recognition II: Classification and Hidden Markov Models

5.1 Overview

In exercise 4, we derived feature vectors in an abstract feature space (e.g., of dimension 39). Every few milliseconds, a new feature vector is calculated from the incoming speech signal.

Now, in this exercise, we will classify, or *decode* these feature vectors (or whole sequences of feature vectors) in order to obtain the *text* that was originally spoken. Note that in the scope of this exercise, we consider only simple cases which show the basic concepts.

5.2 Classification Problems in ASR

From the earliest automatic speech recognizers (ASRs) in the 1950s for isolated syllables up to the first speaker-independent systems for isolated words in the early 1970s the main focus of research in this area was on *template-matching* techniques. For that (suboptimal) approach, dynamic time warping (DTW) of the patterns in conjunction with some distortion measure is used to compare the incoming patterns with static test patterns in the memory (see lecture notes pp. 234ff.).

Current systems for the automatic recognition of continuously spoken speech are firmly based on the principles of *statistical pattern recognition*.

Question for preparation: What are the advantages and the disadvantages of template-matching techniques (using simple distance measures) vs. statistical pattern recognition (using hidden Markov models)? Consider the following aspects: T

- training of the classification parameters for different speakers and/or for new vocabulary,
- applicability for continuous speech recognition.

▷ _____
▷ _____
▷ _____

For the classification, or decoding, we distinguish between isolated word recognition, and continuous speech recognition.

In any case, we assume that our sequence of feature vectors is given by $\boldsymbol{C} = \langle \boldsymbol{c}(1), \boldsymbol{c}(2), \ldots, \boldsymbol{c}(T) \rangle$. The utterance to be recognized consists of a sequence of words $\boldsymbol{w} = \langle w_1, w_2, ..., w_n \rangle$.

5.2.1 Isolated Word Recognition

In isolated word recognition, we want to find out which single word out of a given list of words was spoken. That word can be labeled by only one number κ.

Template matching: In this traditional method, we just compare the current features $\boldsymbol{c}(m)$ with all templates $\boldsymbol{c}_\lambda$ using a certain distance measure $d(\cdot,\cdot)$:

$$\kappa = \arg\min_{\lambda} d(\boldsymbol{c}_\lambda, \boldsymbol{c}(m)). \tag{18}$$

Optimum (statistical) solution: *maximum a-posteriori solution*

$$\kappa = \arg\max_{\lambda} P(w_\lambda|\boldsymbol{c}(m)) \tag{19}$$

It turns out that the statistical solution is much more flexible than template matching. There is a powerful parameter estimation method for the statistical solution which compresses all the example utterances into very compact parametric representations, the hidden Markov models (HMMs). Moreover, it allows easily the extension to the case of continuous speech recognition.

5.2.2 Continuous Speech Recognition

In this case, the task of the ASR system is to determine the word sequence $\boldsymbol{w}$ with the highest probability, given the observed acoustic sequence $\boldsymbol{C}$, i.e.

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} P(\boldsymbol{w}|\boldsymbol{C}). \tag{20}$$

It can be shown that this *maximum a-posteriori sequence estimation* (MAPSE) is the theoretically optimum solution of the problem at hand.

The required probability in this formula can be decomposed into two components using Bayes' rule[2]:

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{w}} \frac{P(\boldsymbol{w})P(\boldsymbol{C}|\boldsymbol{w})}{P(\boldsymbol{C})} = \arg\max_{\boldsymbol{w}} P(\boldsymbol{w})P(\boldsymbol{C}|\boldsymbol{w}) \tag{21}$$

This equation indicates that to find the most likely word sequence $\boldsymbol{w}$, the word sequence that maximizes the product of $P(\boldsymbol{w})$ and $P(\boldsymbol{C}|\boldsymbol{w})$ must be found.

- The first of these terms represents the *a-priori* probability of observing $\boldsymbol{w}$ independently of the observed signal, and this probability is determined by a *language model*.
- The second term represents the probability of observing the vector sequence $\boldsymbol{C}$ given some specified word sequence $\boldsymbol{w}$, and this probability is determined by an *acoustic model*.

[2] $P(\boldsymbol{C})$ does not depend on $\boldsymbol{w}$ and is thus not relevant for the maximization in Eq. (21)

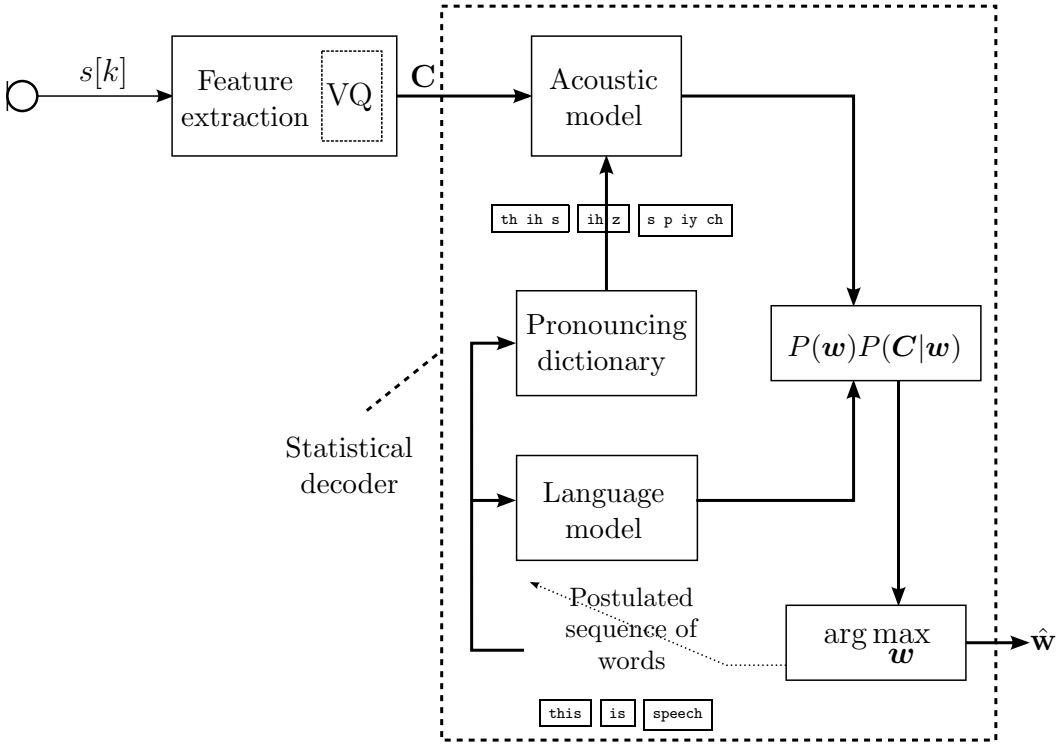

Fig. 9: Overview of the main components of statistical speech recognition

Fig. 9 shows how these relationships might be computed *in theory*. The word sequence $\boldsymbol{w} = \langle$"This","is","speech"$\rangle$ is postulated and the language model computes its probability $P(\boldsymbol{w})$. Each word is also converted into a sequence of basic sounds or *phones* using a pronouncing dictionary. For each phone there is a corresponding HMM. In continuous speech recognition the postulated utterance can be represented by concatenating HMMs to form a single composite model, and the probability of that model generating the observed sequence $\boldsymbol{C}$ is calculated. This is the required probability $P(\boldsymbol{C}|\boldsymbol{w})$. In principle, this process could be repeated for all possible word sequences with the most likely sequence selected as the recognizer output.

The *efficient* realization of this design philosophy into a practical system obviously requires the solution of several challenging fundamental problems which we briefly address in the following.

5.3 Acoustic Modeling by Hidden Markov Models

The purpose of the acoustic models is to provide a method of calculating the likelihood of any vector sequence $\boldsymbol{C}$ given a word w_k. In principle, the required probability distribution could be located by finding many examples of each w_k and collecting the statistics of the corresponding vector sequences. However, this is impractical for systems with large vocabulary and, instead, word sequences are decomposed into basic sounds called *phones*.

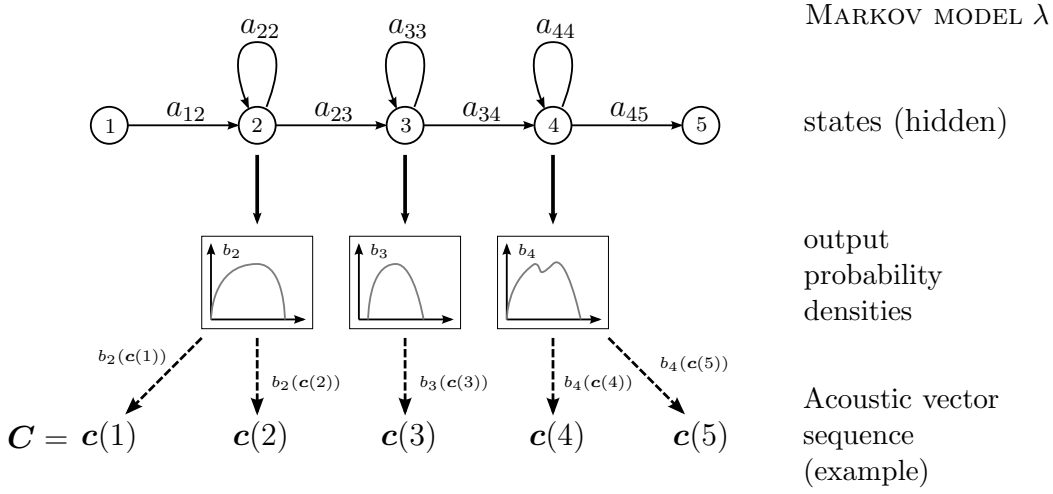

Fig. 10: HMM-based phone model and an example for a state sequence $\boldsymbol{q} = \langle 1, 2, 2, 3, 4, 4, 5 \rangle$ generated by the model

For each individual phone exists an HMM. An HMM has a number of states connected by arcs. HMM phone models typically have three *emitting* states and a simple left-right topology (*Fig.* 10). Entry and exit states are provided to make it easy to join models together. The exit state of one phone model can be merged with the entry state of another to form a composite HMM. This allows phone models to be joined together to form words and words to be joined together to cover complete utterances.

An HMM is most easily understood as a generator of vector sequences (or as a statistically time-varying probability density function describing a vector sequence in the feature space). It is a finite-state machine that changes its state once every time unit, and each time unit, say t, that a state j is entered, an acoustic speech vector $\boldsymbol{c}(t)$ is generated with probability density $b_j(\boldsymbol{c}(t))$. Furthermore, the transition from state i to state j is also probabilistic and is governed by the discrete probability a_{ij}. *Fig.* 10 also shows an example of this process where the model moves through a state sequence $\boldsymbol{q} = \langle 1, 2, 2, 3, 4, 4, 5 \rangle$ in order to generate the sequence $\boldsymbol{c}(1)$ to $\boldsymbol{c}(5)$.

The choice of the output probability density function (pdf) is crucial since it must model all the spectral variability in real speech, both within and across speakers. In current ASR systems the densities $b_j(\boldsymbol{c}(t))$ are chosen from a parametric family of functions. The most common choice is a mixture of multivariate Gaussian pdfs:

$$b_j(\boldsymbol{c}(t)) = \sum_{m=1}^{M} g_{jm} \mathcal{N}(\boldsymbol{c}(t), \boldsymbol{\mu}_{jm}, \boldsymbol{U}_{jm}), \tag{22}$$

where g_{jm} is the weight of mixture component m in state j. $\mathcal{N}$ denotes a multivariate Gaussian pdf with mean vector $\boldsymbol{\mu}_{jm}$ and covariance matrix $\boldsymbol{U}_{jm}$.

So far there has been an implicit assumption that only one HMM is required per phone. Since the English language, e.g., consists of approximately 45 phones, it

may be thought that only 45 phone HMMs need to be trained. In practice, however, contextual effects cause large variations in the way that different sounds are produced. Hence, to achieve good phonetic discrimination, different HMMs have to be trained for each different context. The simplest and most common approach is to use *triphones*, where every phone has a distinct HMM model for every unique pair of left and right neighbors. There are systems with cross-word triphones (give best accuracy but more complications in the decoder) and simpler systems using just word-internal triphones.

5.3.1 Isolated Word Recognition Using HMMs

The joint probability of a vector sequence $\boldsymbol{C}$ and some state sequence $\boldsymbol{q} = \langle q(1), q(2), ..., q(T)\rangle$, given some model λ can be calculated simply as the product of the transition probabilities and the output probabilities:

$$P(\boldsymbol{C}, \boldsymbol{q}|\lambda) = a_{q(0)q(1)} \prod_{t=1}^{T} b_{q(t)}(\boldsymbol{c}(t)) a_{q(t)q(t+1)}, \tag{23}$$

where we define $q(0)$ as the model entry state and $q(T+1)$ as the model exit state.

Of course, in practice, only the observation sequence $\boldsymbol{C}$ is known and the underlying state sequence $\boldsymbol{q}$ is hidden. (This is why it is called a *hidden Markov model*). Nevertheless the required probability $P(\boldsymbol{C}|\lambda)$ can be found by summing Eq. (23) over all possible state sequences, i.e.,

$$P(\boldsymbol{C}|\lambda) = \sum_{\boldsymbol{q} \in \mathcal{Q}^T} P(\boldsymbol{C}, \boldsymbol{q}|\lambda), \tag{24}$$

where $\mathcal{Q}^T$ denotes the T-fold Cartesian product of the set of possible states. There is a very efficient recursive method of performing this calculation, the so called *Forward-Backward* algorithm.

5.3.2 Training of HMM Parameters

Another important feature of the Forward-Backward algorithm is that it can also be used for the calculation of the probability of being in a specific model state at a specific time. This leads to the *Baum-Welch* algorithm, a very simple and efficient procedure for finding maximum-likelihood estimates of both the a and b parameter sets of the HMMs. We will not go further into parameter estimation here, but it should be emphasized that *the existence of this algorithm and the associated flexibility has been a key factor in making HMMs the dominant technology in acoustic modeling.*

5.3.3 Experiments

In the directory `./SHARED_FILES/spsa/Exercise5`, all necessary files for the experiments are provided. Copy the whole directory into your home directory. In the MATLAB bash, start the code by typing

M

```
>> IsolatedWordRec
```

a) First, we study the training of HMMs using synthetic, randomly generated data. For simplicity, we generate two HMMs (for two words). The principle can very easily be generalized to a larger vocabulary.

   - Create a training set for word 1 by clicking on the button *"Simulate"*. The training signal is randomly generated by a Markov process. Now, you can see the distribution of the feature vectors in the feature space.
   - In the same way, create a training set for word 2. In order to create the training set, a different Markov model than for word 1 is used.
   - The actual training of the parameters for each new HMM can be done by clicking on the button *"train"*. Now, the algorithm creates three states with the according probability densities, transition probabilities (matrix A) and entry state probabilities (vector pi).

b) Now, we have two new HMMs that are ready for use. The next step is the classification of a new, *previously unseen realization* of one of the words in our vocabulary.

   - This new realization is generated by clicking on either the button "Create word 1" or the button "Create word 2".
   - To classify this sequence, click on the button "Classify test sequence". Now, the algorithm calculates the log-likelihood for both HMMs according to Eq. (24).
   - Finally, the algorithm chooses the word with higher log-likelihood.

5.4 Continuous Speech Recognition

5.4.1 Language Modeling

Note: *This section provides just additional information which is not essential for the exercise.*

As suggested in *Fig.* 9, the language model in a statistical ASR system is intended to restrict the space of possible solutions for the optimization problem Eq. (21) by some weighting of that space. In statistical language modeling we have to provide a mechanism for the estimation of the probability of some word w_i in an utterance given the preceding words. The required probability for Eq. (21) could then be calculated after, say word k by

$$P(\boldsymbol{w}) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdot \ldots \cdot \ldots \cdot P(w_k|w_1, \ldots, w_{k-1}) \qquad (25)$$

In theory, each word depends on all its predecessors, however in practice Eq. (25) is approximated. A simple but effective way of doing this is to use so called *n-grams*,

in which it is assumed that a certain word w_i depends only on the preceding $n-1$ words, e. g. $n = 3$ (trigrams):

$$P(\boldsymbol{w}) \approx P(w_1) \cdot P(w_2|w_1) \cdot \prod_{i=3}^{k} P(w_i|w_{i-2}, w_{i-1}) \tag{26}$$

n-grams simultaneously encode syntax, semantics and pragmatics and they concentrate on local dependencies. This makes them very effective for most languages where word order is important and the strongest contextual effects tend to come from near neighbors. Furthermore, n-gram probability distributions can be computed directly from text data and hence there is no requirement to have explicit linguistic rules such as a formal language.

In principle, n-grams can be estimated from simple frequency counts and stored in a look-up table (histogram). The problem is of course – similarly to the one of the acoustic models – that of acute data sparsity: For a vocabulary of V words, there are V^3 potential trigrams. Even for a relatively limited vocabulary of 2000 words[3], this is already a very large number. Therefore, most of the required words will not even occur in a short training period or it will be at least very difficult to obtain robust statistics. Although robust estimation of trigram probabilities requires considerable care, there are a number of tricky approaches today that give relatively good performance. However n-grams do have obvious deficiencies resulting from their inability to exploit long-range constraints such as subject-verb agreement. Various alternatives have been studied. However, in general, all of these attempts have yielded only small improvements at considerable computational cost.

5.4.2 Decoding of Continuous Speech

In order to perform recognition using the components described above, the sequence of words $\hat{\boldsymbol{w}}$ that maximizes Eq. (21) must be found. This is a search problem and its solution is the domain of the decoder.

There are generally two main approaches for search problems:

- In **depth-first** designs, the most promising hypothesis is pursued until the end of the speech is reached. Examples of depth-first decoders are *stack-decoders* and A^**-decoders*.
- In **breadth-first** designs, all hypotheses are pursued in parallel. Breadth-first decoding exploits the optimality principle of Bellman and is often referred to as *Viterbi decoding*. Since ASR systems for large vocabulary are complex and pruning of the search space is essential, a process called *beam search* is typically used.

[3]Information systems for relatively simple dialogs such as Information on the schedule of trains typically have a vocabulary of some 2000 words. One example is the system "EVAR" of the chair of pattern recognition, univ. of Erlangen-Nuremberg. Future voice-operated auto navigation systems might need to have a similar vocabulary.

The Viterbi Algorithm also provides an alternative way to the direct calculation of Eq. (24) as it can be approximated by *finding the state sequence* that maximizes Eq. (23).

In most ASR decoders, the Viterbi algorithm is used. Here, the determination of the most likely state sequence is the key to recognizing an unknown word sequence.

5.4.3 Experiment

For this experiment, we also provide all necessary files in the directory `./SHARED_FILES/spsa/Exercise5`.

- Start the GUI by typing
  `>> ContinuousSpeechRec`
- In this experiment, a trained HMM is already available. The parameters (densities, transition probabilities, ...) are shown in the top left of the GUI.
- When you click on the button "Create sequence", an arbitrary state sequence is created (with the transition probabilities of the HMM), and plotted. Moreover, for each state, the values of the two features are simulated (according to the densities of the HMM).
- Now, the most likely state sequence (depending on the observed feature vectors) is determined by means of the Viterbi algorithm:
  - For each time index and state, the most likely state sequence resulting in the respective state is determined and sketched in gray.
  - When the final time index is reached, the sequence with the highest likelihood is selected and sketched in blue.
  - The red stars illustrate the actual state sequence.