Chapter 2

Background

This chapter discusses in detail the background to the research that was performed for this thesis. Firstly an overview is given of the theory of hidden Markov modelling that was applied. The basic notation is given and algorithms and equations that are required for understanding the proposed ideas are discussed for reference from later chapters. This section also serves to at least partially document the algorithms used in the development of the Hidden Markov Toolkit for Speech Recognition (HMTSR) C++ software by Darryl Purnell and the author during their Ph.D. studies. The software is included on the compact disc inserted inside the back cover of this thesis.

Previous research in multilingual speech recognition is discussed next, focusing on how these systems re-use acoustic information between multiple languages and specifically how cross-language use of acoustic information has benefited the development of speech recognisers in a new target language. Limitations of previous research is pointed out, in particular the partial implementation of speaker adaptation techniques, leading us to consider improved use of these techniques in following chapters.

2.1 Hidden Markov modelling framework

The main components used in the training and testing of the speech recognition system that was developed are:

- **feature extraction** in which speech signals are converted into sequences of mel-scaled cepstral coefficient vectors along with their time derivatives,
- **training** of HMMs, which includes fixed segment initialisation, Viterbi alignment re-estimation and the expectation maximisation or Baum-Welch procedure,
- **continuous speech recognition** in which the feature vectors are matched using dynamic programming to a set of trained HMMs constrained by a finite state grammar.

We now proceed to discuss each of these items in detail, including various choices with respect to parameters of especially the feature extraction process. The selection of parameters of the general system is included in this background section on HMMs because the values of these parameters are fairly standard and are not considered to significantly impact the experiments discussed in a later section.

2.1.1 Feature extraction

The speech signal is blocked into frames of 16 ms spaced 10 ms apart - delivering 6 ms of overlap between successive frames. This choice has been empirically determined to deliver good performance. At a 16 kHz sampling rate, which is used in all experiments, each 16 ms frame consists of 256 samples. Hamming windowing and a fast Fourier transform (FFT) is performed on each frame and the result is multiplied by its complex conjugate to deliver a real valued power spectrum. The next step is applying a mel-spaced filter bank to produce 24 mel-spaced filtered coefficients. The logarithm of each coefficient is taken and a discrete cosine transform (DCT) is performed on the coefficients to deliver what is referred to as mel-scaled cepstral coefficients or MFCCs. In all experiments 13 coefficients are used as we have previously found this to deliver good performance for a connected digit recognition task [35] performed with the HMM software and is also commonly reported in literature. Temporal information about the speech signal is incorporated by estimating first and second time derivatives for each of the 13 coefficients. A second order linear regression is applied to each set of five consecutive coefficients in order to obtain a smoothed estimate of the first and second time derivatives. The observation vector $\mathbf{x}$ thus consists of the 13 mel-scaled cepstral coefficients plus first and second order time derivatives, totalling 39 elements at each frame time. A detailed discussion of issues concerning the feature extraction process can be found in [26].

2.1.2 Continuous density hidden Markov models

A continuous density hidden Markov model (CDHMM), hereafter referred to simply as an HMM, signified by $\boldsymbol{\lambda}$, is described by two sets of parameters:

- a state transition matrix $\mathbf{A} = \{a_{ij}\}$ reflecting the probabilities of making transitions from each state i to each other state j and
- a continuous state observation density function $b_j(\mathbf{x})$ reflecting the likelihood of observing observation vector $\mathbf{x}$ in state j.

To simplify the equations we consider the initial state probabilities to be given by a_{0i} for each state i, without any loss of generality.

The models are first order HMMs since each transition probability to a next state depends only on the current state, and not on which states were previously traversed. Left-to-right HMMs without skipping transitions are commonly used for speech recognition and is the connectionist strategy we use for the purpose of this thesis. The state transition probabilities a_{ij} satisfy the constraints $\sum_{j=1}^{N} a_{ij} = 1$ and for left-to-right models without skipping transitions the additional constraints are that $a_{ij} \neq 0$ only for $j = i$ or $j = i + 1$. The assumption with left-to-right modelling is that observation sequences corresponding to the same HMM traverse the same discrete sequence of statistical properties. This agrees with our phonetic understanding of speech as exhibiting piecewise continuous behaviour to a large degree. This unfortunately does not explicitly allow for the modelling of too much variation in the way that the same word may be pronounced other than for time warping of the speech signal, but at least leads to very efficient implementation.

The Markov models are termed "hidden" due to the fact that the states are not observed directly in the observation sequence, but rather indirectly through modelling of observation distributions in each state. Gaussian mixtures are used to model the observation probability density functions. The p.d.f. of observation $\mathbf{x}_t$ at time t in state j takes the form

$$\begin{aligned} b_j(\mathbf{x}_t) &= \sum_{k=1}^{K} c_{jk} \mathcal{N}[\mathbf{x}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}] \\ &= \sum_{k=1}^{K} c_{jk} (2\pi)^{-D/2} |\boldsymbol{\Sigma}_{jk}|^{-1/2} e^{-(1/2)(\boldsymbol{\mu}_{jk}-\mathbf{x}_t)^T \boldsymbol{\Sigma}_{jk}^{-1} (\boldsymbol{\mu}_{jk}-\mathbf{x}_t)} \end{aligned} \tag{2.1}$$

where K is the number of mixture components, D is the number of feature vector elements, c_{jk} is the weight associated with the kth mixture in the jth state, $\mathcal{N}$ is the multivariate normal density, $\boldsymbol{\mu}_{jk}$ is the mean vector of the kth mixture in the jth state and $\boldsymbol{\Sigma}_{jk}$ is the covariance matrix of the kth mixture in the jth state. To greatly reduce the number of parameters and since the elements of $\mathbf{x}_t$ are largely uncorrelated, we make the assumption that $\boldsymbol{\Sigma}_{jk}$ is diagonal. The observation density function becomes

$$b_j(\mathbf{x}_t) = \sum_{k=1}^{K} c_{jk} \prod_{l=1}^{D} (2\pi)^{-1/2} \sigma_{jkl}^{-1} \, e^{-(x_{tl}-\mu_{jkl})^2/2\sigma_{jkl}^2} \tag{2.2}$$

where x_{tl} is the lth element of the observation vector at time t, μ_{jkl} is the lth element of the mean vector in mixture k of state j and σ_{jkl}^2 is the lth variance value on the diagonal of $\boldsymbol{\Sigma}_{jk}$.

2.1.3 Duration modelling

It is commonly accepted that the duration modelling aspect of the HMM approach to speech recognition is a major weakness. Conventional HMMs implicitly model state duration by a geometric distribution, i.e.

$$\rho_j(\tau) = a_{jj}^{\tau-1}(1 - a_{jj}), \tag{2.3}$$

where a_{jj} is the auto-transition probability in state j and τ is the duration in number of frames. The geometric distribution is not able to model individual state duration probabilities well since it can only represent an exponentially decreasing probability density function. Explicit duration densities for states may be specified and in such a case the models are called semi-Markov models [26]. State duration density may be modelled with estimated discrete duration probabilities $d_j(\tau), \tau = 1, 2, .., \tau_{max}^j$ for each duration up to a maximum duration τ_{max}^j. This approach has the disadvantage that a large number of parameters have to be estimated. Modelling duration with parametric functions greatly reduces the number of parameters. A popular function for modelling state duration probability is the Gamma distribution

$$\rho(\tau) = \frac{\beta^\alpha}{\Gamma(\alpha)}\tau^{\alpha-1}e^{-\beta\tau}, \tag{2.4}$$

which has only the parameters α and β that have to be estimated for each state duration model. Initial algorithms for duration modelling were very computationally expensive [26], and a post-processing approach [36] was often used. The post-processing method uses duration metrics to re-score a number of the best paths obtained from a search process. This approach fails where the best re-scored path is not amongst the obtained best paths, and is thus not re-scored. Another approach that was more recently investigated was the use of the so-called expanded-state HMM (ESHMM) [37, 38] that provided moderate performance improvement, but at the cost of between 2-times and 4-times speed degradation.

An efficient approach towards incorporating duration modelling into the search process has been proposed by Du Preez [39] and a similar approach was later independently proposed by Burshtein [40]. Both approaches add a duration metric at each time frame to automatically obtain a state duration probability weighted path in a computationally efficient manner, causing only marginal speed degradation. The method proposed by Burshtein was used in the experiments where applicable. Implementation of the method is discussed in the next section along with the algorithms used for the training of HMMs.

2.1.4 Hidden Markov model training

Current large vocabulary continuous speech recognition (LVCSR) systems make use of phone models to efficiently capture the necessary acoustic information for modelling large vocabulary speech through use of pronunciation dictionaries. Separate HMMs are used to model each phone and if a sufficient amount of data is available for training, head-body-tail or explicit trigram-type phone models are used. An HMM is also used to model silence at the beginning and end of utterances and between words. A clustering method, often based on data likelihood e.g. the Bayesian information criterion method, is used to decide on which trigrams to group together to constitute the set of phones. In this study only monophones, which do not take context into account, are used since the experiments mostly do not use enough data to warrant the training of more complex models and also because of the greater computational cost associated with the adaptation of large numbers of context dependent model parameters.

The parameters that have to be estimated in training an N state HMM are:

- $N+1$ independent transition probabilities (since $a_{j,j+1} = 1 - a_{jj}$ for the left-to-right model and other off-diagonal values are zero),
- NK mixture weights,
- NKD mean and covariances values and
- $2N$ duration parameters if duration modelling with the Gamma distribution is used.

i 1578633x
b15221301

Methods used for training HMMs are usually based on the *maximum likelihood* principle. The maximum likelihood estimate $\boldsymbol{\lambda}_{\mathrm{ML}}$ of the parameters of an HMM given an observation sequence $\mathbf{X} = (\mathbf{x}_1, .., \mathbf{x}_T)$ is given by the mode (maximum) of the likelihood function

$$\boldsymbol{\lambda}_{\mathrm{ML}} = \max_{\boldsymbol{\lambda}} f(\mathbf{X}|\boldsymbol{\lambda}). \tag{2.5}$$

The ML estimate is usually found by setting the derivate of either the likelihood function or the log-likelihood function with respect to the parameters of the model to zero and solving for the ML estimate of the parameters.

The likelihood of a sequence of observations given an HMM denoted by $\boldsymbol{\lambda}$ has the form

$$f(\mathbf{X}|\boldsymbol{\lambda}) \propto \sum_{\mathbf{q}} \prod_{t=1}^{T} \left[a_{q_{t-1}q_t} \sum_{k=1}^{K} c_{q_t k} \mathcal{N}(\mathbf{x}_t, \boldsymbol{\mu}_{q_t k}, \boldsymbol{\Sigma}_{q_t k}) \right], \tag{2.6}$$

where summation takes place over all possible observation sequences $\mathbf{q}$. ML estimation of the parameters of an HMM is not trivial because a *sufficient statistic* of fixed dimension does not exist for observations of an HMM. The likelihood function is not expressible in terms of a fixed number of parameters and thus cannot be maximised easily. The lack of a sufficient statistic of fixed dimension is due to the hidden process of an HMM, namely the fact that state and mixture occupancy is not observable. This lack of observability causes HMM estimation to be termed an *incomplete data* estimation problem. The solution is to use an iterative procedure such as expectation maximisation [41] procedure. The EM procedure estimates state and mixture occupancy sufficient statistics in a first part. With the availability of the calculated state and mixture occupancy statistics together with the observation sequence, the problem becomes a *complete data* estimation problem. This enables the computation of the ML parameter estimate in the second part for the complete data problem. The EM procedure consisting of the calculation of occupancy statistics followed by ML parameter estimation is repeated until convergence or a fixed number of iterations have occurred.

In our system we use iterative estimation in each of three different training stages, namely fixed segmentation initialisation in the first stage, then segmental training (encompassing Viterbi-alignment and ML estimation) and finally training with the standard EM or Baum-Welch algorithm that computes statistics with the forward-backward algorithm and uses them for ML estimation. Our three stage training process progresses from simple, computationally inexpensive initialisation to the more complex and slower EM training. The three training stages are discussed next.

Initialisation

The parameters are estimated by examining the distribution of features in training data. The state transition matrix A is initialised according to left-to-right constraints. To bootstrap the parameters, each observation feature vector sequence corresponding to a single HMM is subdivided into as many segments of equal length as there are states in the HMM. The mean and variance of the first Gaussian mixture component in each state is initialised to the sample mean and (diagonal) covariance of the corresponding speech feature segments. After initialisation the training process commences.

Segmental training

In segmental training the Viterbi [42] algorithm is used to compute the single most likely state alignment of each observation sequence. Given the estimated alignment, the *complete data* modelling problem is solved using maximum likelihood estimates of the mixture weights, means and covariances of the Gaussian mixture models at each state and of the transition probabilities. Alignment and parameter estimation is repeated iteratively. We give details of the Viterbi dynamic programming algorithm since it is used in the implementation of the adaptation algorithms and is also used for both the training and testing of HMMs. We present the Viterbi algorithm mostly following the syntax from [26].

For an HMM, the Viterbi algorithm finds the most likely state sequence $\bar{\mathbf{q}} = (\bar{q}_1, \bar{q}_2, .., \bar{q}_T)$

for a given observation sequence $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_T)$ as well as the likelihood associated with this sequence

$$\begin{aligned} P(\mathbf{X}|\boldsymbol{\lambda}) &= \max_{\mathbf{q}} P(\mathbf{X}, \mathbf{q}|\boldsymbol{\lambda}) \\ &= P(\mathbf{X}, \bar{\mathbf{q}}|\boldsymbol{\lambda}) \\ &= \prod_{t=1}^{T} [a_{\bar{q}_{t-1}\bar{q}_t} b_{\bar{q}_t}(\mathbf{x}_t)]. \end{aligned} \tag{2.7}$$

As part of the definition of the Viterbi algorithm we define

$$\Phi_j(t) = \max_{q_1, q_2, .., q_{t-1}} P[q_1 q_2, .., q_{t-1}, q_t = j, \mathbf{x}_1 \mathbf{x}_2, .., \mathbf{x}_t|\boldsymbol{\lambda}], \tag{2.8}$$

the highest probability along a single path, at time t, that accounts for the first t observations and ends in state j. By induction, the Viterbi recursion is defined as

$$\Phi_j(t+1) = \max_{0 \le i \le N} [\Phi_i(t) a_{ij}]\, b_j(\mathbf{x}_{t+1}), \quad 1 \le j \le N. \tag{2.9}$$

The probability in the final state at the final time frame, $\Phi_N(T)$, indicates the score for the match between model and observation sequence. When the Viterbi algorithm is used to align speech with model states, such as is used in training, it is necessary to keep track of the path followed via

$$\psi_j(t+1) = \arg\max_{0 \le i \le N} [\Phi_i(t) a_{ij}]. \tag{2.10}$$

This path can be backtracked from $\psi_N(T)$ to deliver the highest scoring path $\bar{\mathbf{q}}$.

Note that in Equation 2.9 we have included transitions from state 0 to the current state to incorporate the initial state probabilities. To initialise the Viterbi search for left to right operation we define $\Phi_0(0) = 1$, $\Phi_j(0) = 0, j \neq 0$ and $\Phi_0(t) = 0, t > 0$. When we discuss the implementation of successive Viterbi searches in a later section, the value associated with state zero ($\Phi_0(t)$) for $t > 0$ might not have the value 0, but may represent the final value of a previous level in the search, e.g. $\Phi_N^{(r)}(t)$ for highest scoring model r in the previous level.

The training of parameters takes place after statistics from an entire batch of training utterances are collected. The result of each application of the Viterbi algorithm is a state-aligned set of observation features. After statistics have been collected for the batch of training samples, new mean, variance and transition probability values are computed for the Gaussian mixture models. Training using Viterbi-alignment is also called *segmental* training since the observation sequence is segmented, with each segment being used to update the parameters of a particular state. For the update, we first need to define the posterior state probability variable

$$\gamma_j(t) = P(q_t = j|\mathbf{X}, \boldsymbol{\lambda}), \tag{2.11}$$

which expresses the probability of being in state j at time t, given the observation sequence $\mathbf{X}$ and the model $\boldsymbol{\lambda}$. When *segmental* training is used, $\gamma_j(t)$ is simply equal to 1 when $\bar{q}_t = j$ and zero otherwise, i.e. $\gamma_j(t) = \delta(\bar{q}_t - j)$ where δ denotes the Kronecker delta function. Since we use Gaussian mixture distributions, we proceed to define the posterior mixture observation probability variable

$$\gamma_{jk}(t) = \gamma_j(t)\frac{c_{jk}\mathcal{N}[\mathbf{x}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}]}{b_j(\mathbf{x}_t)} \tag{2.12}$$

which expresses the joint probability of being in state j at time t and observing mixture k, given the observation sequence $\mathbf{X}$ and the model $\boldsymbol{\lambda}$. Finally, we define $\xi_{ij}(t)$, the probability of being in state i at time t and in state j at time $t+1$ by

$$\xi_{ij}(t) = P(q_t = i, q_{t+1} = j|\mathbf{X}, \boldsymbol{\lambda}), \tag{2.13}$$

which simply becomes $\xi_{ij}(t) = \delta(\bar{q}_t - i)\delta(\bar{q}_{t+1} - j)$ for segmental training.

Now we can define the update equations for the coefficients of the mixture density in iteration n in terms of the sufficient statistics:

$$a_{ij}(n) = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)} \tag{2.14}$$

$$c_{jk}(n) = \frac{\sum_{t=1}^{T} \gamma_{jk}(t)}{\sum_{t=1}^{T} \gamma_j(t)} \tag{2.15}$$

$$\boldsymbol{\mu}_{jk}(n) = \frac{\sum_{t=1}^{T} \gamma_{jk}(t)\mathbf{x}_t}{\sum_{t=1}^{T} \gamma_{jk}(t)} \tag{2.16}$$

$$\Sigma_{jk}(n) = \frac{\sum_{t=1}^{T} \gamma_{jk}(t)(\boldsymbol{\mu}_{jk}(n) - \mathbf{x}_t)(\boldsymbol{\mu}_{jk}(n) - \mathbf{x}_t)^T}{\sum_{t=1}^{T} \gamma_{jk}(t)}. \tag{2.17}$$

Our implementation of the segmental training process is now elaborated in more detail. As a result of the initialisation process described at the start of this section, only the first mixture component has non-zero values after the first iteration. If after an iteration, there are mixtures with zero mixture weights and sufficient data is available to warrant splitting, the component with the largest mixture weight is split to produce two components with means only slightly offset from each other in the direction of maximum variance. This process of alignment, re-estimation and mixture splitting is repeated iteratively until either convergence occurs, or a predetermined number of iterations have been completed.

Expectation maximisation

We use the Baum-Welch method, which is an implementation of the expectation maximisation or EM method for HMMs, to perform final training of the HMMs. Each HMM is trained using the set of speech segments accorded to it in the labelling process. The Baum-Welch method iteratively updates the means, covariances, mixture weights and state transition probabilities at each state in much the same way as done with *segmental* training, but uses the forward-backward algorithm instead of the Viterbi algorithm to obtain statistics from the training utterances. Once the sufficient statistics have been computed, Equations 2.14-2.17 are used for the update. The implementation of the forward-backward algorithm is not discussed here as we do not use it for implementing the adaptation approaches (i.e. we only use it for initial model training). We use a *segmental* implementation for the adaptation algorithms since it is faster and is the method most commonly used in research on adaptation methods. Detail regarding use of the forward-backward algorithm and parameter update using the EM method can be found in [26].

Duration model training

Lastly, training of only the duration parameters is done through the Viterbi state alignment of HMMs to the utterances they represent. For each alignment, the sum of the first and second moments of the number of frames corresponding to each state in each HMM is collected. The empirical expectation values of the mean($\hat{E}\{\tau\}$) and variance($\hat{E}^2\{\tau\}$) of each duration can be calculated and used to obtain the Gamma distribution parameters ($\hat{\alpha}$ and $\hat{\beta}$) through

$$\hat{\alpha} = \frac{\hat{E}^2\{\tau\}}{\widehat{VAR}\{\tau\}}, \quad \hat{\beta} = \frac{\hat{E}\{\tau\}}{\widehat{VAR}\{\tau\}}. \tag{2.18}$$

This concludes the training process. In the next section we discuss the implementation of the recogniser.

2.1.5 Pattern matching

We first discuss the incorporation of duration modelling into the Viterbi algorithm. Duration modelling is implemented according to the synchronous frame by frame method suggested by Burshtein [40]. The method modifies the Viterbi recursion (Equation 2.9) by incorporating a duration penalty C_{ij}^t of making a transition from state i to state j at time $t+1$ within the term that is maximised by the recursion. When written in log format for implementation efficiency, the Viterbi recursion becomes

$$\Phi_j(t+1) = \max_{0 \le i \le N} [\Phi_i(t) + \log(a_{ij}) + \log(C_{i,j}^t)] + \log b_j(\mathrm{x}_{t+1}), \quad 1 \le j \le N. \tag{2.19}$$

To compute the duration penalty, the method keeps track of the number of successive self-transitions in each state. The duration $D_i(t)$ of a state i at time t is equal to one plus the number of successive self-transitions in that state. Let M_i denote the duration at which the Gamma distribution $p(\tau)$ at state i reaches a maximum value. The duration penalty $C_{i,j}^t$ is then given by

$$C_{i,j}^t = \begin{cases} 0 & i = j,\ D_i(t) < M_i \\ \log(D_i(t+1)) - \log(D_i(t)) & i = j,\ D_i(t) \ge M_i \\ \log(D_i(t)) & i \ne j,\ D_i(t) < M_i \\ \log(M_i) & i \ne j,\ D_i(t) \ge M_i. \end{cases} \tag{2.20}$$

The working of the method can be understood in the following way. The duration probability density function is used to modify the probability of a transition occurring, based on the duration spent in the state from which the transition occurs. When a transition to a different state is taken, the exact duration is known and can be used to modify the probability. In considering self-transitions, however, the penalty can not be incorporated on a frame by frame basis since the eventual duration in a state is yet unknown. Incorporating the duration probability at each frame as if it were the last time step in a state would penalise initial self-transitions in a state – causing an incorrect bias towards transitions from the previous state. Therefore the method should not penalise self-transitions until the peak duration probability is reached in a state. After the point of peak duration probability, duration penalty is applied in accordance with the duration probability density.

With duration modelling now incorporated into the Viterbi search, we turn our attention to the implementation of the level building algorithm. When recognition of a sequence of spoken words is attempted, it is desired to find the best match across all possible sequences of word and pause models. An exhaustive search of depth V, containing R possibilities in each level leads to R^V Viterbi alignments. Even for a simple task like connected digit recognition it leads to 10^{10} Viterbi alignments (if pause models are ignored) for a string of at most 10 digits - which is not computationally feasible. The level building algorithm [26] dramatically reduces the computational cost by performing only R searches at each of the V levels, thus effectively $V \times R$ Viterbi alignments.

The level building algorithm works by computing at each successive level l the most likely final state probability (P_t^l) at each frame t over all R models in the search path

$$P_t^l = \max_{1 \le r \le R} [\Phi_N^r(t)]. \tag{2.21}$$

After a level has been completed, the final state probabilities are used as initial state probabilities for all Viterbi searches at the next level, i.e. we now set $\Phi_0^r(t) = P_t^l, 1 \le r \le R$. This process continues until the desired number of levels have been searched. The most likely sequence ends at the level given by

$$\arg \max_{1 \le l \le V} (P_T^l). \tag{2.22}$$

From the most likely final state at the final frame it is easy to backtrack the complete path followed through all levels provided that the backtracking information from each individual Viterbi alignment has been retained. Note that the most likely solution does not necessarily present itself at the last level. The level building technique can thus be used to find unknown length word strings up to the maximum depth for which was searched. In continuous speech recognition experiments we set the maximum depth large enough so as to not influence the results.

The level building algorithm was initially used for experiments, but was later superseded by the use of a frame synchronous trellis search [43] using the Viterbi algorithm, yielding identical results with less computational cost. With the general HMM system background now covered, we turn to previous research in the field of multilingual speech recognition.

2.2 Multilingual speech recognition

To study the similarities between the phones of different languages, one has to examine the relatively new field of multilingual speech recognition. Multilinguality refers to the property of a system to be capable of understanding speech input in more than one language, i.e. it includes both the acoustic and so-called *language* modelling of the relevant languages. As far as acoustic modelling in the multilingual field is concerned, research ranges from systems that have a unified architecture, yet have separate models for each language to systems that share increased numbers of acoustic parameters. Language identification systems are also multilingual systems in a certain sense, but focus mainly on language models to perform discrimination between languages with multiple acoustic models used primarily to extract phone sequences, but also to provide some discriminative information [44, 45].

Speech translation systems from the Verbmobil [10] project, specifically the JANUS [46] system, are amongst the first applications of multilingual speech recognition. The JANUS system is architecturally language independent and each speech recognition module is loaded with models for the specific language it has to recognise. It therefore has a common modelling structure for speech from different languages, but does not share acoustic information between languages.

The field of multilingual information systems has also been actively researched. In the development of the MIT VOYAGER [14] multilingual system, separate context-independent acoustic models were trained for English, Japanese and Italian. The English version used 58 models based on the labels used in the TIMIT [31] database and was trained with data from the same database. For the Japanese and Italian versions, the models were initialised by seeding them from their most phonetically similar English counterparts. This was reported to enable the further training of the Japanese and Italian models on language specific speech data that was only transcribed but not aligned, thus saving the great amount of work needed to manually align the speech. The Mandarin Chinese version of the GALAXY system, called YINHE [18], followed on the VOYAGER system and also used English models to seed near-neighbour Mandarin acoustic models. Another large multilingual speech recognition system is the BYBLOS Callhome system [15]. The system performs task specific speech recognition in multiple languages using the same architecture, but using separate acoustic models for each language in question.

2.2.1 Bootstrapping of new target language recognisers

Studies have been performed to quantify the effect on system performance of the cross-language bootstrapping of acoustic models. Wheatley *et al.* [16] compared the performance when Japanese acoustic models are bootstrapped with English acoustic models trained on TIMIT to flat-start training of the Japanese models, as well as initialisation of the Japanese models with a limited number of hand-picked representative examples. The application was a connected digit recognition system with some control words and was modelled with whole word models. In the case of bootstrapping with English models the Japanese word models were initialised with sequences of English phone models. Compared to the flat-start approach, the cross-language bootstrapped models and the hand-picked representative example approaches achieved better performance after 2 training iterations. After 10 iterations, the bootstrapped models exhibited a small improvement in overall performance over the other two approaches. The authors also performed a cross-language smoothing experiment. When only a small amount of Japanese data was used, smoothing of the final model by interpolation between English and Japanese models achieved slightly better performance than using the Japanese models directly.

Schultz *et al.* performed bootstrapping of a Japanese recogniser with models from a German recogniser [47] and showed bootstrapping to be an efficient method of initialising the target language models. In a subsequent study [17] a multilingual phoneme set comprising of the collection of the language dependent phonemes of German, English, Japanese and Spanish was created and used to bootstrap recognisers in Chinese, Croatian and Turkish. Bootstrapping was done through a five step process namely

1. the determining of a mapping of language specific phones to the multilingual set by phonetic experts,
2. initialisation of the acoustic models according to the mapping,
3. maximum likelihood linear regression (MLLR) transformation of the models using language specific data along with language specific linear discriminant analysis (LDA) calculation and K-means codebook clustering,
4. four training iterations and
5. repetition of steps 3 and 4.

Bootstrapping was shown to result in better performance than is achievable with flat start training on target language data when only a few iterations of training is done. Schultz & Waibel later also investigated a simpler form of bootstrapping by performing cross-language training [30] of acoustic models. Various monolingual model sets as well as a multilingual set of HMMs were used as starting models for 2 iterations of Viterbi training on German data. It was shown that using the multilingual phone set as initial model was slightly superior to using 3 of the 5 languages (Turkish, Croatian and Spanish) and was far superior to using Japanese and Korean initial models.

The bootstrapping results discussed in this subsection indicate that cross-lingual models provide good initial models for training in a new language. None of the bootstrapping results, unfortunately, indicate a real advantage in terms of recognition rate of using cross-lingual information. The methods do, however, show the advantage of requiring fewer training iterations for convergence when cross-language seed models are used.

2.2.2 Explicitly multilingual systems

When acoustic information in a source language is used to bootstrap models for a new language, the source language data is only used to construct seed models for initial alignment of unlabelled data and is not used in subsequent re-alignment and re-estimation of models for the new language. In this case separate recognisers are realised for each target language. Other studies have explicitly used multilingual phoneme sets, in which case the eventual models exhibit characteristics of multiple languages. Köhler [48] studied isolated phoneme recognition on continuous American English, German and Spanish telephone speech. He found that the sharing of acoustic information across languages leads to some performance degradation, but that a representation with fewer mixtures than that of the combined models from the three languages still delivered reasonable performance. Weng *et al.* [19] used shared Gaussian codebooks across Swedish and English phones and reported that allowing the sharing of data across phones from the two languages also did not improve performance, but lead to a system capable of performing language identification as part of the decoding process. Bonaventura *et al.* [20] performed experiments to quantify the performance of a system with a language-independent phonetic inventory on Italian, Spanish, English and German words. Dissimilarity measures were proposed to enable automatic determination of which phones from the different language to merge into multilingual phones. Significant reduction of the total number of phones needed was achieved at the cost of some degradation of performance with respect to language dependent phones.

In a detailed study covering five languages (Croatian, Japanese, Korean, Spanish and Turkish), Schultz & Waibel [30] found that monolingual systems outperformed a system with shared multilingual acoustic models and the same number of parameters as the five monolingual systems combined, by approximately 1% (27% versus 28%) in terms of word error rate (WER). The reason given for the decrease in performance is that language independent modelling decreases the precision of the acoustic models. In a study also covering five languages (French, German, Italian, Portuguese and Spanish) Köhler [21] found that a multilingual approach to acoustic modelling yielded a 3.2% (14.2% versus 11.0%) increase in average WER for an isolated word recognition task and a 4.9% (43.7% versus 48.6%) decrease in correct phone recognition rate compared to a monolingual approach.

A study performed by Uebler *et al.* [49] targeted performance improvement in a bilingual environment where L1 German and Italian speakers spoke both languages, producing L1 and L2 German and Italian speech. The study found that a bilingual German/Italian system outperformed two separate monolingual systems on the test database of L1 and L2 German and Italian speech. The improvement of 1.2% (11.3% versus 12.5%) in WER of the bilingual system is attributed to the large variation in accents and dialects of the L2 speakers in both languages being better captured by the bilingual system than by the monolingual systems.

The research discussed so far in this section has focussed either on bootstrapping to avoid the manual labelling effort in a new language, or on creating shared multilingual phone sets to facilitate integrated multilingual recognition. The latter approach has mostly lead to a degradation in performance over monolingual systems, except where L1 and L2 speech were mixed in an application [49]. Little research has been performed with the goal of improving performance in a specific target language through explicit use of cross-language acoustic information. Research conducted with this specific goal in mind is discussed next.

2.2.3 Cross-language use of acoustic data for new target languages

Bonaventura *et al.* [20] performed experiments where it was assumed that little data was available for training Spanish models. The application was an isolated word recognition system with a vocabulary of 70 words. It was found that the use of phone models trained on both Italian and Spanish data, i.e. on the pooled multilingual data, lead to between 0.6% (12.9% versus 13.5%) and 3% (20% versus 23%) reduction in WER over a system trained only on the Spanish data, depending on the amount of adaptation data used.

The use of on-line Bayesian learning for cross-language adaptation was investigated by Bub *et al.* [22] and applied to Slovene isolated digit recognition. The method used the on-line maximum *a posteriori* (MAP) algorithm, updating only the Gaussian means via linear interpolation between the original and sample means - i.e. the Gaussian variance, mixture weight and transitions probabilities from the original models were not changed. The adaptation of monolingual and a multilingual (German, American English and Spanish) HMM systems to Slovene was considered. Results show that MAP adaptation on 646 utterances of 12 isolated Slovene digits improved the performance of the baseline multilingual HMM system from 76.5% to 85.0%. Unfortunately no comparable results for direct training on the Slovene digit data are given. The WER of 15% is also high for an isolated digit recognition system.

Köhler [21] investigated the cross-language use of multilingual acoustic models (trained on American English, Italian, French, Portuguese and Spanish) in developing a German speech recognition system. A bootstrapping method from Schultz & Waibel [17] was compared to cross-language mean-only MAP adaptation on German adaptation data and it was found that for little adaptation data, the cross-language adaptation approach achieved better performance than a bootstrapping or a flat-start approach. When most of the adaptation data was used, a flat-start German system was found to achieve the best performance. The relatively poor performance of the adapted system when more data is available is probably due to the fact that only the mean parameters were adapted- since it is known that the performance of the MAP algorithm is asymptotic with the task dependent performance as the amount of data increases (Lee *et al.* [24]).

In this section we discussed the various approaches that were followed in previous research on multilingual recognition. Some adaptation algorithms, notably MAP and MLLR were used, albeit in a limited fashion. Proper use of adaptation algorithms presents the logical extension to the research covered in this chapter. In the next chapter we therefore proceed to discuss adaptation algorithms in depth to apply these methods for cross-language adaptation.