Chapter 2

Theory

The work presented here makes use of techniques from various disciplines. The aim of this chapter is to provide the necessary background theory to understand the work presented later. The experienced speech researcher would be familiar with these concepts and therefore could move to Chapter 3 where the problem of speech segmentation is addressed. In Section 2.1 the signal processing techniques, used to extract features from the speech signal are discussed. Hidden Markov models are used for the phoneme recognition experiments and the relevant theory is discussed in Section 2.2. Finally recurrent neural networks are discussed in Section 2.3, as they are used to segment the speech signal into phonemes. This section should be read in conjunction with Appendix A where details on training of the recurrent neural networks are presented.

2.1 Speech signal processing

The human vocal tract mechanism is unable to instantly produce different sounds in speech. Due to the transition-like nature between sounds, there is a significant amount of correlation between segments of speech. By using parametric representations of the speech, rather than the speech signal itself, a more compact, reliable, and robust speech recognition system can be built. The use of parametric representations also reduces the amount of computation needed for both training and decoding. The following front-end is typically used in speech recognition systems.

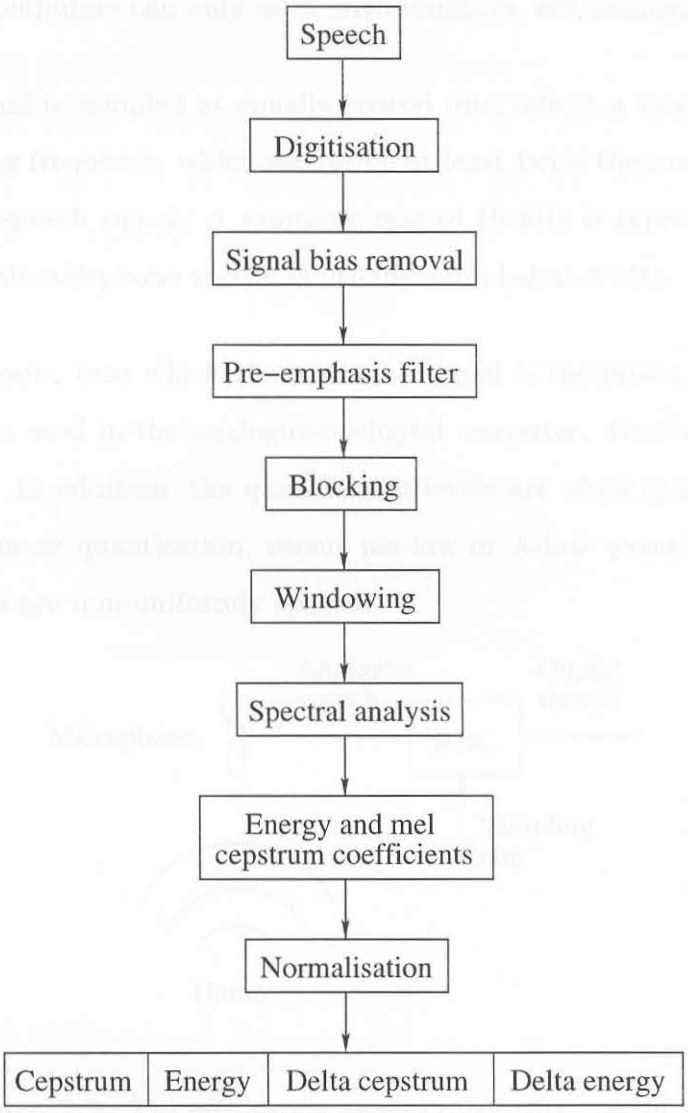

Figure 2.1: Signal processing front-end used.

The following sections describe each of these signal processing steps, in terms of the work presented here, in more detail.

2.1.1 Digitisation

Speech digitisation is the process of converting analogue signals to digital values (numbers), as digital computers can only work with numbers, not analogue values.

The analogue signal is sampled at equally spaced intervals at a rate equal to what is called the sampling frequency, which should be at least twice the maximum frequency of interest in the speech signal. A sampling rate of 16 kHz is typically used for high quality speech, while telephone speech is usually sampled at 8 kHz.

The number of levels, into which the analogue signal is quantised, is dependent on the number of bits used in the analogue-to-digital converter. Resolutions of 12 to 16 bits are common. In addition, the quantisation levels are often spaced equally. This is referred to as linear quantisation, versus mu-law or A-law quantisation, where the quantisation levels are non-uniformly spaced.

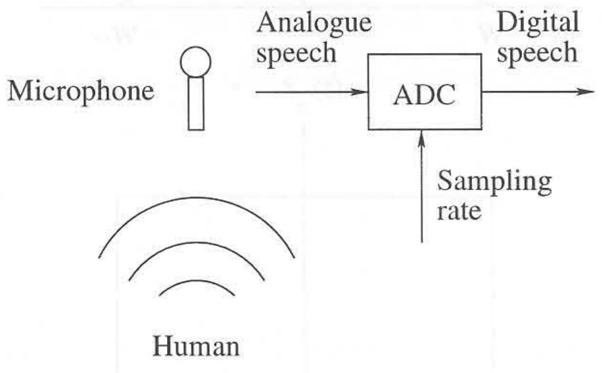

Figure 2.2: The digitisation process commonly used in ASR.

As can be seen from Figure 2.2, a human thus speaks a sentence to be recognised by the system. The speech signals are transmitted as vibrations of air to a microphone, which then converts the air pressure variations into electrical signals. The result is an analogue signal representing the spoken utterance. An analogue anti-aliasing filter (not shown) prevents aliasing effects, but this is not discussed here, and assumed to be part of the analogue-to-digital conversion (ADC) process. An ADC converts the analogue signal to a sequence of numbers (digital), to be processed by a computer.

2.1.2 Pre-emphasis filter

One of the problems that a speech recognition system must face, is the fact that a message (sequence of spoken sounds) will have frequency components that are high at low frequencies, but small amplitude at high frequencies [122]. This is also sometimes referred to as the lip effect. This also occurs because the speech signal is typically represented as a volume velocity [68], but microphones typically measure sound pressure. This difference results in a 6 dB spectral roll-off.

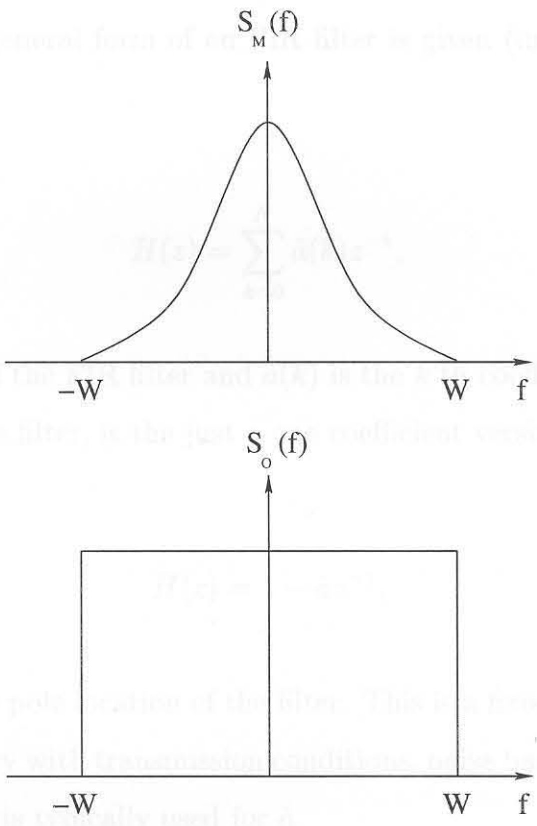

Figure 2.3: Purpose of the pre-emphasis filter.

The top part of Figure 2.3 above illustrates this concept. Here it can be seen that the power spectral density of the message usually falls off appreciably at higher frequencies. These high frequency components (that may carry vital information for accurate speech recognition) will thus be of a much lower magnitude than those at low frequencies. As such, the speech recognition process will not yield the best possible results. It is thus needed to equalise the frequency band over which important signal frequency components in the message exist, to obtain a spectrally flat frequency spectrum, as shown in the bottom part of Figure 2.3. This procedure is called pre-emphasis. Pre-emphasis reduces the effects of glottal pulses and radiation impedance, and enhances the spectral properties of the vocal tract [123, 124]. This makes the signal less susceptible to finite precision effects later in the signal processing front-end [1].

In order to perform equalisation, or pre-emphasis, a high-pass filter is usually used. Since the digitised speech samples are processed sequentially, an FIR filter is a natural implementation. The general form of an FIR filter is given (in the z-transform form) as

$$H(z) = \sum_{k=0}^{N} \tilde{a}(k) z^{-k}, \tag{2.1}$$

where N is the order of the FIR filter and $\tilde{a}(k)$ is the k'th coefficient, and $\tilde{a}(0) = 1$. A high-pass, pre-emphasis filter, is the just a one coefficient version of this, or

$$H(z) = 1 - \tilde{a}z^{-1}, \tag{2.2}$$

where $\tilde{a}$ determines the pole location of the filter. This is a fixed form of pre-emphasis, as $\tilde{a}$ does not slowly vary with transmission conditions, noise backgrounds, etc. A value of between 0.9 and 1.0 is typically used for $\tilde{a}$.

The easiest way to implement the filter of Equation (2.2), is to convert the z-transform equation into a difference equation (in the time domain). It is known that (from Equation (2.2))

$$H(z) = 1 - \tilde{a}z^{-1},$$

but

$$H(z) = \frac{\widetilde{S}(z)}{S(z)},$$

and thus it is possible to write

$$\widetilde{S}(z) = S(z) \cdot (1 - \tilde{a}z^{-1}),$$

with the difference equation which follows as

$$\tilde{s}(n) = s(n) - \tilde{a}s(n-1), \tag{2.3}$$

where $\tilde{s}(n)$ is the output of the pre-emphasis filter, $s(n)$ is the input, and $s(n-1)$ is the previous (one step delayed) input to the pre-emphasis filter.

Although pre-emphasis, as discussed here (using a high-pass filter), is a necessary step and greatly improves speech recognition performance, it has a major disadvantage. While the filter spectrally flattens the range of frequencies of interest, in which useful information lies, it also raises the spectral energy of high frequencies outside the signal bandwidth [6]. These very high frequencies are often associated with noise, and pre-emphasis thus also amplifies noise in the signal. Despite this disadvantage, it remains to be popularly used.

2.1.3 Blocking

After the signal has been pre-emphasised, the next step is to chop the signal into smaller pieces. These smaller pieces are called frames. Usually the frames are chosen as overlapping segments of speech, in order to reduce the noise in spectral estimates (increase correlation between adjacent frames). Figure 2.4 illustrates this process.

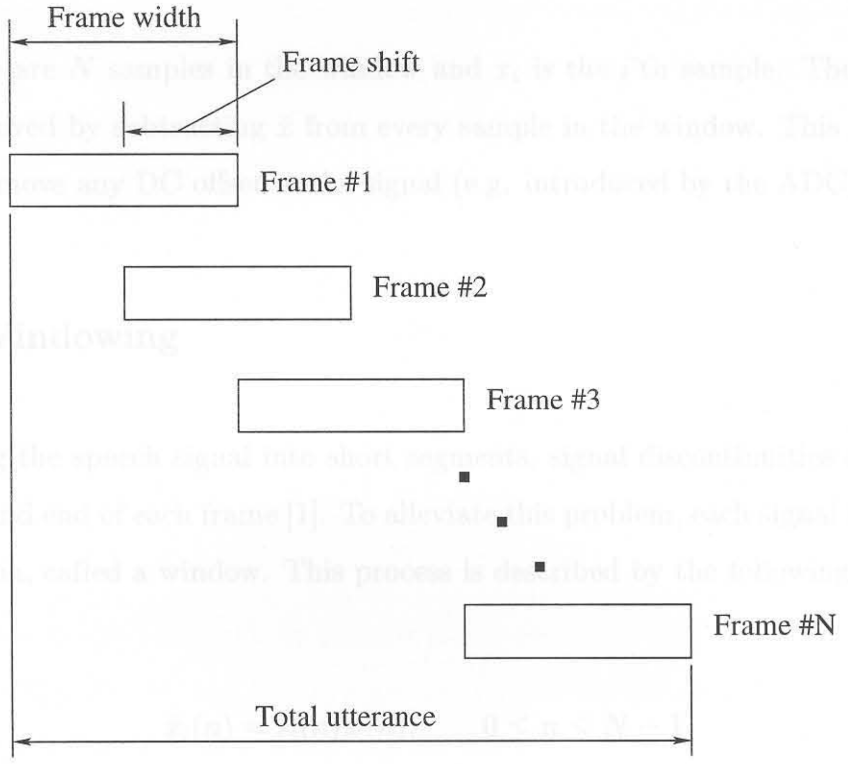

Figure 2.4: Blocking of the speech signal into overlapping frames.

The frame width is usually chosen to be between 20 and 30 ms with a frame shift (or period) of 5 to 10 ms. The bigger the window width, the better the spectral resolution, but the lower the time resolution. A low frame shift results in good temporal resolution, but is computationally expensive.

2.1.4 Signal bias removal

Signal bias removal refers to the process whereby the mean value (here called the signal bias), is removed from the signal. To estimate the mean, the average value of the signal is calculated over the current window through the following equation

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{2.4}$$

where there are N samples in the window and x_i is the i'th sample. The signal bias is then removed by subtracting $\bar{x}$ from every sample in the window. This technique is useful to remove any DC offset in the signal (e.g. introduced by the ADC).

2.1.5 Windowing

By blocking the speech signal into short segments, signal discontinuities occur at the beginning and end of each frame [1]. To alleviate this problem, each signal is multiplied by a function, called a window. This process is described by the following equation:

$$\tilde{x}_l(n) = \tilde{s}_l(n)w(n), \qquad 0 \leq n \leq N-1, \tag{2.5}$$

where $\tilde{x}_l(n)$ is the resultant, windowed signal, $\tilde{s}_l(n)$ is the original signal, $w(n)$ is the window with which the frame is multiplied, and N is the number of samples in the frame. The window with which the frame is multiplied, is typically chosen to be the Hamming window, defined by the following equation [1]:

$$w(n) = 0.54 - 0.46 \cdot \cos\left(\frac{2\pi n}{N-1}\right), \tag{2.6}$$

for $0 \leq n < N$ and $w(n) \equiv 0$ elsewhere. Here $w(n)$ is called the window, and N is the number of samples in the frame with which it should be multiplied.

The purpose of the window is to give more emphasis to samples in the centre of the window. Combined, the overlapping frame analysis and windowing, ensures that smoothly varying parametric estimates can be obtained.

2.1.6 Spectral analysis

For the spectral analysis of the speech signal (the blocked frames of speech multiplied by the window), a Fast Fourier Transform (FFT) of the signal is first computed (called a short-time FFT, because the FFT is taken only over short time segments). This procedure is based on the two equations [86]

$$W \equiv e^{2\pi i/N}, and \tag{2.7}$$

$$Y_n = \sum_{k=0}^{N-1} W^{nk} \tilde{x}_k, \tag{2.8}$$

where W is a complex number, as defined in Equation (2.7), N is the size of the FFT (usually a multiple of 2), and the Y_n's are the FFT components of the signal, with the original signal values denoted by the $\tilde{x}$'s.

After the FFT of the signal is taken, the next step is to calculate the squared magnitude of the spectrum. The squared magnitude spectrum is just the squared, absolute value of the FFT components (complex numbers), and is also called the power spectrum of the signal (or the energy spectrum if the square root is taken). This can be illustrated with the equation

$$Z_k = |Y(k)|^2, \qquad 0 \leq k < K, \tag{2.9}$$

where K can be equal to N, the size of the FFT. Because only half of the (symmetric) spectrum is considered, K is taken to be equal to $N/2$. The result of spectral analysis is thus the squared magnitude spectrum, or power spectrum of the signal, as defined in Equation (2.9).

2.1.7 Energy and mel cepstrum coefficients

Energy and mel-scale frequency coefficients are the features most often used in speech recognition systems today. These coefficients provide a parameterised form of the speech that significantly reduces the data rate of the speech input, while still maintaining virtually all of the speech information [6].

To compute the cepstra, a bank of filters is constructed. The number of filters is usually chosen larger than the number of cepstra needed. These filters are equally spaced along the mel-scale frequency axis, but logarithmically spaced on the acoustic frequency axis. This is motivated by the fact that human perception of the frequency content of sounds, either for pure tones or for speech signals, does not follow a linear scale [1]. For each pure tone with an actual frequency, f, measured in Hz, a subjective pitch is perceived by humans, measured on a scale called the "mel" scale. The conversion between acoustic frequency to mel frequency, can be done with the equation [6]

$$melfrequency = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right), \tag{2.10}$$

where f is the acoustic frequency, in Hz, to be converted to a frequency in mels. Conversely, the mel frequency can be converted into an acoustic frequency through the equation (inverse of Equation (2.10))

$$f = 700 \cdot \left(10^{x/2595} - 1\right), \tag{2.11}$$

where x is the mel frequency to be converted into the acoustic frequency, f. The next step in the computation of the cepstra, is to construct a bank of filters. These filters are triangular filters, spaced equally along a mel-scale frequency axis, but logarithmically along the acoustic frequency axis.

Figure 2.5 shows the filter allocation in the frequency domain (acoustic frequency).

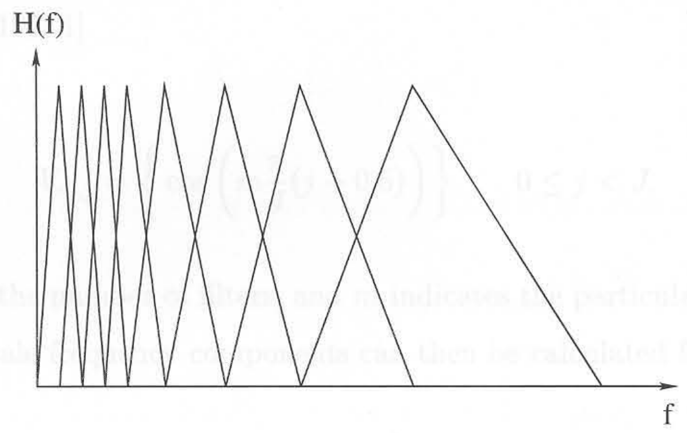

Figure 2.5: Filter allocation in the frequency domain.

The logarithmic nature of the filter spacing, described by Equation (2.10), is clearly seen in the figure. In this figure, each filter has a minimum and maximum frequency that correspond to the centre frequencies of the filters to the left and right, respectively. This results in filters having different bandwidths. In the logarithmic domain, however, these filters have equal bandwidth.

After the mel filters have been constructed, the energy output of each filter must be calculated. The energy output of each of the filters can be calculated as [6]

$$E_j = \sum_{k=0}^{K-1} \phi_j(k) Z_k, \qquad 0 \leq j < J, \tag{2.12}$$

where E_j is the energy from the j'th filter, $\phi_j(k)$ is the value of filter j's transfer function at k, K is equal to half of the size of the FFT (e.g. 256 for a 512-point FFT), and J is the number of filters. The filters must obey the constraint [6]

$$\sum_{k=0}^{K-1} \phi_j(k) = 1 \qquad \forall j. \tag{2.13}$$

To calculate the mel-scale frequency coefficients, a set of weighting factors is first computed. These weighting factors can be computed through an inverse discrete cosine transform, as [6, 102, 1]

$$V_{m,j} = \left\{ \cos\left(m\frac{\pi}{J}(j+0.5) \right) \right\} \qquad 0 \le j < J, \tag{2.14}$$

where J is again the number of filters, and m indicates the particular cepstrum coefficient. The mel-scale frequency components can then be calculated from

$$c_m = \sum_{j=0}^{J-1} \beta_j V_{m,j} \log_{10}(E_j), \tag{2.15}$$

where E_j is the energy output of the j'th filter, as defined in Equation (2.12), $V_{m,j}$ is the weighing factor defined in Equation (2.14), and β is an amplification factor, which accommodates the dynamic range of the coefficients, c_m. The cepstra are normalised by the number of filters as

$$cn_m = c_m\sqrt{\frac{2}{J}}, \tag{2.16}$$

where cn_m is the final m'th mel cepstrum coefficient. The coefficients are sometimes liftered so that the higher and lower order cepstra have similar values. The cepstra are thus re-scaled to have similar magnitudes, according to

$$cn_m^{'} = \left(1 + \frac{L}{2}\sin\frac{\pi m}{L}\right) cn_m, \tag{2.17}$$

where L is the liftering coefficient and cn_m is defined in Equation (2.16).

To obtain the power, component 0 of the cepstrum vector is sometimes taken, which is just the average value of the spectrum, or root mean square (RMS) value of the signal [5]. The logarithm of the true energy, however, normally replaces component 0 of the cepstrum (since it is unreliable), and can be calculated as

$$E = \log_{10}\left(\sum_{n=0}^{N-1} \tilde{x}_n^2\right), \tag{2.18}$$

where $\tilde{x}_n$ is the windowed signal, and N is the number of samples in the window.

The above calculations will thus compute the mel frequency cepstrum coefficients as well as an energy measure. These provide a good parametric representation of the speech signal as well as good discrimination between the different speech sounds.

2.1.8 Normalisation

Normalisation of the speech features is usually done to reduce variability of the features with environment changes, microphone mismatch, etc. Many different algorithms exist, but only cepstral mean normalisation, energy normalisation, and a simple linear re-scaling technique will be discussed here.

Cepstral mean normalisation

After the cepstra have been computed, the next step is their normalisation. Both the power and the mel frequency coefficients can be normalised by the sentence-based mean. The first step is thus to compute the sentence-based mean, as

$$\mu(k) = \frac{1}{T}\sum_t x_t(k), \tag{2.19}$$

where T is the number of frames of the input utterance, k is the index of the cepstral coefficient, and $x_t(k)$ is the k'th cepstral coefficient. Next, the cepstra are normalised by the sentence-based mean, by simply subtracting the mean, as follows:

$$\tilde{x}_t(k) = x_t(k) - \mu(k), \tag{2.20}$$

where $\tilde{x}_t(k)$ is the k'th normalised cepstrum coefficient. This technique is useful to compensate for long-term spectral effects such as those caused by different microphones and audio channels.

Energy normalisation

The log energy is normalised to have values between $-E_{min}$ to 1.0 by subtracting the maximum value of the energy in the utterance from every energy value, and adding 1.0. This technique will cause silence to be indicated by a value of 1.0 and speech samples less than 1.0, and is in a convenient form for other processing tasks, such as silence detection.

Simple linear re-scaling

With this technique, each of the speech vector's components is normalised to have zero mean and unit variance. It is a simplified form of the more general whitening transform [69]. In this dissertation, it was used before speech vectors were applied to the input of the neural network that segmented the speech, in order to avoid any saturation problems of the neural network transfer functions. The mean and variance were calculated over all of the speech in the corpus that was used for training purposes, as the mean and variance can vary somewhat from speech utterance to speech utterance, which would degrade the neural network performance. It is, however, not uncommon to calculate the mean and variance only on a sentence by sentence basis. The mean of one speech vector feature is calculated as

$$\mu(k) = \frac{1}{N}\sum_n \frac{1}{T_n}\sum_t x_t^n(k), \tag{2.21}$$

where T_n is the number of frames of the n'th training utterance, k is the index of the feature in the speech vector, $x_t^n(k)$ is the k'th speech feature, and N is the number of training speech vectors. The variance of one of the speech features is calculated as

$$\sigma^2(k) = \frac{1}{N}\sum_n \frac{1}{T_n}\sum_t (x_t^n(k) - \mu(k))^2, \tag{2.22}$$

where $\sigma(k)$ is the standard deviation (square root of the variance). The speech vector components are then normalised by each of the feature means and standard deviations independently, as

$$xn_t^n(k) = \frac{x_t^n(k) - \mu(k)}{\sigma(k)}. \tag{2.23}$$

2.1.9 Delta energy and cepstrum coefficients

By adding time derivatives to the basic static features, the performance of speech recognition systems can be greatly improved. These dynamic features capture the dynamic changes that occur in the speech. To calculate the first derivative of the static features, also called the delta feature, a regression formula [3] such as

$$d_t = \frac{\sum_{\theta=1}^{\Theta} \theta(c_{t+\theta} - c_{t-\theta})}{2\sum_{\theta=1}^{\Theta}\theta^2}, \tag{2.24}$$

is normally used, where d_t is the delta coefficient at time t, computed in terms of the corresponding static coefficients $c_{t-\theta}$ to $c_{t+\theta}$, and θ is the regression window. Equation (2.24) is used to calculate both the delta energy and delta cepstrum coefficients.

2.2 Hidden Markov models

Hidden Markov models (HMMs) are probably the most popular technology of choice for large vocabulary speech recognition systems. This is primarily due to the efficiency with which HMMs model the variation in the statistical properties of speech, both in the time and frequency domains [120]. A number of assumptions are, however, made when using HMMs [120, 1], but these will not be discussed here.

In the following sections, HMMs are discussed only briefly, in the context of how they are used in this study. In particular, training of HMMs is not discussed, as this dissertation only modifies the decoding process. The training of the HMMs is thus fairly standard and details can be found in other studies [1, 66, 67, 120, 3]. The basic concepts of HMMs are discussed in the next section, followed by a section on the Viterbi decoding process (used to recognise the unknown speech). Language models and the use of a word transition penalty are also discussed.

2.2.1 Basic elements and problems of an HMM

Hidden Markov models involve two underlying stochastic processes. One of these is hidden and can only be indirectly observed through the other stochastic process. This is the reason why HMMs are called "hidden" Markov models. The two stochastic processes occur concurrently within an HMM. When HMMs are considered to be used as a parametric model for a process, a number of concepts are of importance. Some of these concepts are discussed briefly in this section.

An important choice to be made when using HMMs is that of the structure of the HMM. An HMM consists of N states, often represented by circles in a pictorial representation. The states are connected by arcs, with arrows indicating the direction of an allowed transition between the states.

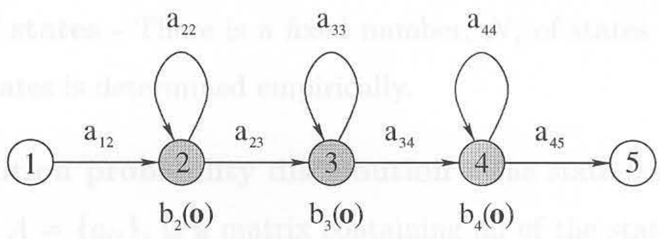

Figure 2.6: Structure of a basic HMM.

Figure 2.6 shows the structure of a basic HMM. In general, the theory allows for arbitrary connections between the states of the HMM. Left-to-right HMMs are, however, more commonly employed, especially in speech recognition, due to the temporal nature of the speech. In HMMs of this kind, as shown in Figure 2.6, there are fewer parameters than in an ergodic HMM. This limits the modeling power of the HMM, but the HMM parameters can be estimated more reliably with limited training data.

At the first time step, an initial state $q_1 = i$ is chosen, according to an initial state distribution π. At the following time instants, a stochastic process decides what the next valid states should be, based only on the current state. This process results in a first order HMM. This stochastic process, described by the transition probabilities, a_{ij}, thus determines what the next state j should be, given that the current state is state i. As each state is entered at time t (either the same or a different state), an observation $\boldsymbol{o}_t$ is generated according to a second stochastic process, described by an observation density $b_j(\boldsymbol{o}_t)$. The sequence of states $\boldsymbol{q}$ can not be observed directly, but only through the observations generated. This is why HMMs are "hidden".

All of the states in an HMM need not be emitting. In Figure 2.6, states 1 and 5 are non-emitting states. These states are used as a convenient way to connect multiple HMMs together. The shaded states (states 2 to 4) are emitting states.

An HMM is thus defined as a parametric model, consisting of the following components [1]:

- **Number of states** - There is a fixed number, N, of states in the model. The number of states is determined empirically.

- **State transition probability distribution** - The state transition probability distribution, $A = \{a_{ij}\}$, is a matrix containing all of the state transition probabilities. Each entry in the state transition matrix is defined as

$$a_{ij} = P[q_{t+1} = j | q_t = i], \qquad 1 \leq i, j \leq N, \tag{2.25}$$

  where q_t is the state at time t, and q_{t+1} is the state at time $t+1$. Also, since a_{ij} are probabilities, they follow the usual stochastic constraints, e.g. $a_{ij} \in [0, 1]$.

- **Observation probability distribution** - The observation probability distribution, $B = \{b_j(\boldsymbol{o}_t)\}$ is the vector containing all of the state observation output probabilities. In this dissertation, these probabilities are modeled by Gaussian mixtures, resulting in continuous observation densities of the form

$$b_j(\boldsymbol{o}_t) = \sum_{m=1}^{M_s} c_{jm} \mathcal{N}(\boldsymbol{o}_t; \boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}), \tag{2.26}$$

  where M_s is the number of mixture components, and c_{jm} is the weight of the m'th component. In (2.26), $\mathcal{N}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a multivariate Gaussian with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, defined as

$$\mathcal{N}(\boldsymbol{o}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^n |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{o}-\boldsymbol{\mu})' \boldsymbol{\Sigma}^{-1} (\boldsymbol{o}-\boldsymbol{\mu})}, \tag{2.27}$$

  where n is the dimensionality of $\boldsymbol{o}$.

- **Initial state distribution** - The initial state distribution, $\pi = \{\pi_i\}$, is defined as the following

$$\pi_i = P[q_1 = i] \qquad 1 \leq i \leq N, \tag{2.28}$$

  with q_1 being the initial state.

The complete parameter set, required to fully specify the HMM, can be compactly given as

$$\lambda = (A, B, \pi), \tag{2.29}$$

with A, B, and π as defined earlier.

To use HMMs, three basic problems must be solved. HMMs are popular, partly due to the efficiency in which they can solve these three problems:

- **Problem 1** - The first includes the need for an efficient way to calculate $P(\boldsymbol{O}|\lambda)$, the probability of the observation sequence ($(\boldsymbol{o}_1\boldsymbol{o}_2\ldots\boldsymbol{o}_T)$, with T the maximum length of the sequence), given the model, and is referred to as an evaluation problem. The forward (or backward) procedure allows this problem to be solved in a computationally efficient means. This is not discussed any further here.

- **Problem 2** - The second involves the problem of finding the most likely state sequence $\boldsymbol{q}$, given the model λ and an observation sequence $\boldsymbol{O} = (\boldsymbol{o}_1\boldsymbol{o}_2\ldots\boldsymbol{o}_T)$. This is referred to as the decoding process, used to uncover the "hidden" part of the HMM. In speech recognition, this is used to recognise the unknown speech (e.g. the most likely phone sequence). The Viterbi algorithm, based on dynamic programming (DP) concepts, is used for this purpose. It is discussed in more detail in the next section since the modification of this algorithm is part of our work.

- **Problem 3** - The third problem is concerned with the process of obtaining the HMM parameter set, $\lambda = (A, B, \pi)$ such that $P(\boldsymbol{O}|\lambda)$ is maximised. This is referred to as the training problem. The Baum-Welch algorithm is used for this purpose, but it is not discussed any further here.

In speech recognition applications of HMMs, the first stochastic process deals with the temporal sequence in which the HMM states occur and models the temporal structure of the speech signal. This is related to the transition probabilities. The second stochastic process models the locally stationary character of the speech signal and it is described by the conditional output probability density function. The solution to problem 3 is used to train HMM models for each of the phonemes that occur in the language of interest. A network of HMMs is then constructed, called a lattice or network, based on the task grammar, by joining HMMs together. The solution to problem 2, namely the Viterbi algorithm, is then used to determine the optimal sequence of phonemes (in a maximum likelihood sense in this dissertation). The sequence of phonemes is the recognition result from the unknown speech.

2.2.2 The Viterbi decoding process

As mentioned in the previous section, the most likely state sequence, given the model and the observation sequence, is of great importance in continuous speech recognition. The Viterbi algorithm, based on dynamic programming concepts, attempts to uncover the "hidden" part of an HMM. This algorithm attempts to find the single best path (state sequence) with the highest probability. Alternatively, it can be stated that the probability of the observation sequence $\boldsymbol{O} = (\boldsymbol{o}_1\boldsymbol{o}_2\ldots\boldsymbol{o}_T)$ and state sequence $\boldsymbol{q}$, given the model λ, or $P(\boldsymbol{q}, \boldsymbol{O}|\lambda)$, is maximised.

Figure 2.7 illustrates the Viterbi decoding process, as it is commonly used for continuous speech recognition. The vertical dimension represents the HMM states, while the horizontal dimension represents the frames of speech (time). Shown in the figure are two HMMs, namely HMM A and B, each having 5 states, with no skip transitions. In speech recognition systems, many more HMMs are connected together in a network determined by the task grammar. The result can be seen as a single, large HMM, having N states in total. It is also shown that state 5 of HMM A (labeled A5), and state 1 of HMM B (labeled B1) are regarded as being the same state (they are connected through the use of non-emitting states). The thicker lines in the figure represent the

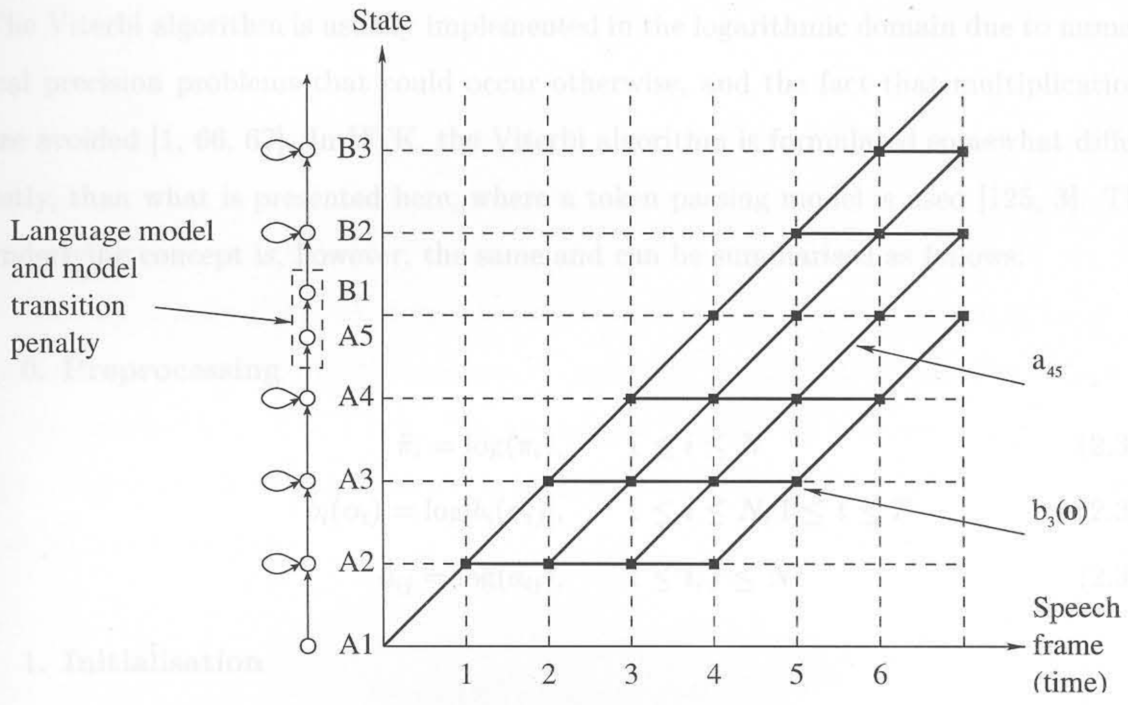

Figure 2.7: The Viterbi decoding process.

allowed paths (sequence of states), the dots represent the log state output probability of observing that frame at the specific time step, and the lines between the dots can be regarded as the log state transition probabilities.

In Figure 2.7, the log probability of a path is calculated by summing the log output probabilities and log transition probabilities along that path. At each time step, the log probability of the previous path that had the highest log probability of all candidate paths, are used to calculate the log probability of a set of new candidate paths. The one with the highest log probability is then chosen for use in the next time step. After each process of selecting the best path, the state sequence followed by that path is recorded. After the best path is calculated at the last time step, the optimal state sequence can be uncovered by backtracking the path followed. From the sequence of states, the times at which the state and model transitions occurred, can be derived. This can be used to align a phonetic transcription against a speech signal, or to recognise the speech and simultaneously provide the times at which model (phoneme) transitions occurred, thereby also segmenting the speech into phonemes.

The Viterbi algorithm is usually implemented in the logarithmic domain due to numerical precision problems that could occur otherwise, and the fact that multiplications are avoided [1, 66, 67]. In HTK, the Viterbi algorithm is formulated somewhat differently, than what is presented here, where a token passing model is used [125, 3]. The underlying concept is, however, the same and can be summarised as follows:

0. **Preprocessing**

$$\tilde{\pi}_i = \log(\pi_i), \qquad 1 \leq i \leq N \tag{2.30}$$

$$\tilde{b}_i(\boldsymbol{o}_t) = \log[b_i(\boldsymbol{o}_t)], \qquad 1 \leq i \leq N, 1 \leq t \leq T \tag{2.31}$$

$$\tilde{a}_{ij} = \log(a_{ij}), \qquad 1 \leq i, j \leq N \tag{2.32}$$

1. **Initialisation**

$$\tilde{\delta}_1(i) = \log(\delta_1(i)) = \tilde{\pi}_i + \tilde{b}_i(\boldsymbol{o}_1), \qquad 1 \leq i \leq N \tag{2.33}$$

$$\psi_1(i) = 0, \qquad 1 \leq i \leq N \tag{2.34}$$

2. **Recursion**

$$\tilde{\delta}_t(j) = \log(\delta_t(j)) = \max_{1 \leq i \leq N} [\tilde{\delta}_{t-1}(i) + \tilde{a}_{ij}] + \tilde{b}_j(\boldsymbol{o}_t) \tag{2.35}$$

$$\psi_t(j) = \arg \max_{1 \leq i \leq N} [\tilde{\delta}_{t-1}(i) + \tilde{a}_{ij}], \qquad 1 \leq j \leq N, 2 \leq t \leq T \tag{2.36}$$

3. **Termination**

$$\tilde{P}^* = \max_{1 \leq i \leq N} [\tilde{\delta}_T(i)] \tag{2.37}$$

$$q_T^* = \arg \max_{1 \leq i \leq N} [\tilde{\delta}_T(i)] \tag{2.38}$$

4. **Backtracking**

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \qquad t = T-1, T-2, \ldots, 1 \tag{2.39}$$

In the above equations $\delta_t(i)$ is the highest probability along a single path, at time t, that accounts for the first t observations and ends in state i, and Equation (2.39) gives the optimal state sequence.

2.2.3 Use of a language model

In this dissertation, a word loop grammar is used, which indicates that any word may follow any other word. In order to improve the recognition performance, a stochastic language model can be used. This takes into account that all word sequences are not necessarily equally likely. A simple probabilistic model of speech production relies on the fact that a specified word sequence, W, produces an acoustic observation sequence, Y, with probability $P(W,Y)$ [1]. The recognition or decoding process attempts to find the string of words that maximises the maximum *a posteriori* (MAP) probability. This can be represented as

$$\hat{W} \ni P(\hat{W}|Y) = \max_W P(W|Y). \tag{2.40}$$

Using Bayes' Rule, $P(W|Y)$ can be written as

$$P(W|Y) = \frac{P(Y|W)P(W)}{P(Y)}, \tag{2.41}$$

and consequently, Equation (2.40) can be written as

$$\hat{W} = \arg\max_W P(Y|W)P(W), \tag{2.42}$$

which is called the MAP decoding rule. In Equation (2.42), $P(Y|W)$ is the acoustic model (a probability of a sequence of acoustic observations, given a word string, estimated by the Markov models). $P(W)$ is called the language model. It gives the probability of observing a particular word sequence, W. As shown in Equation (2.42), the language model has a significant effect on the recognition accuracy.

To estimate $P(W)$, a text corpus is used that contains many word sequences. $P(W)$ can then be estimated by

$$P_N(W) = \prod_{i=1}^{Q} P(w_i|w_{i-1}, w_{i-2}, \ldots, w_{i-N+1}), \tag{2.43}$$

for an N-gram language model, where w_i is the i'th word and Q is the total number of words. For a bigram language model ($N = 2$), $P(w_i|w_{i-1})$ can be estimated by building a table of bigram counts, and then output a back-off bigram by using [3]

$$P(w_i|w_{i-1}) = \begin{cases} (F(w_i, w_{i-1}) - D)/F(w_{i-1}) & \text{if } F(w_i, w_{i-1}) > t \\ b(w_i)P(w_{i-1}) & \text{otherwise} \end{cases}, \tag{2.44}$$

where $F(w_i, w_{i-1})$ is the number of times word w_i follows word w_{i-1}, and $F(w_{i-1})$ is the number of times that word w_{i-1} appears. In Equation (2.44), which is based on a process called discounting [3], D is a discount constant, t is a bigram count threshold, and $b(w_i)$ is the back-off weight for word w_i, which ensures that all of the bigram probabilities for a given history sum to one.

The language model probability is added to each word-end transition (in the logarithmic domain), or multiplied with the acoustic model probability, according to Equation (2.42). The language model probability is an *a priori* probability, computed offline. It is often scaled by a language model scale factor s. From Equation (2.42), recognition is thus based on maximising

$$\hat{W} = \arg\max_{W} P(Y|W)P(W)^s, \tag{2.45}$$

which can be conveniently incorporated in the Viterbi algorithm. Equation (2.35) can thus be modified for word-end transitions as

$$\tilde{\delta}_t(j) = \log(\delta_t(j)) = \max_{1\leq i\leq N} [\tilde{\delta}_{t-1}(i) + \tilde{a}_{ij} + s\tilde{P}(W)] + \tilde{b}_j(\boldsymbol{o}_t), \tag{2.46}$$

where $\tilde{P}(W) = \log(P(W))$ is the log language model probability, and s is language model probability scale factor. If the transition from state i to j is not between word-ends, Equation (2.35) is used.

2.2.4 Use of a word transition penalty

In addition to language models, a fixed word transition penalty is often used. The effect of the word transition penalty is to modify the language model log probability according to the equation

$$\tilde{P}(W)' = s\tilde{P}(W) + p, \tag{2.47}$$

where p is the word transition penalty when it is negative, and word confidence when it is positive (in the logarithmic domain). Equation (2.46) of the previous section can thus be further modified (for word-end transitions only), as

$$\tilde{\delta}_t(j) = \log(\delta_t(j)) = \max_{1\leq i\leq N}[\tilde{\delta}_{t-1}(i) + \tilde{a}_{ij} + s\tilde{P}(W) + p] + \tilde{b}_j(\boldsymbol{o}_t). \tag{2.48}$$

In general, when unknown speech is given as input to the recogniser, the number of insertions tend to increase and the number of deletions decrease, as p is increased (more positive and less negative) and s decreased. Alternatively, the number of insertions tend to decrease and the number of deletions increase, as p decreases (more negative and less positive) and s increases. This can be explained due to the fact that the token log-likelihood is decreased when p is a negative number, penalising transitions between words (phonemes), resulting in a decrease in the number of insertions and an increase in the number of deletions. The optimal values for s and p are usually found by maximising the recognition accuracy on a development set, for both of these parameters.

2.3 Recurrent neural networks

For a number of years, neural networks have been a popular choice to solve a variety of problems. Some of these include classification, regression, probability estimation, time series prediction, process modeling and process control.

The most common neural network architecture in use today, is probably the multilayer perceptron [118]. This neural network is regarded as a "static" neural network in that it does a static mapping of inputs to outputs. Recurrent neural networks (RNNs) are regarded as "dynamic" neural networks, since all of the past information is used, in addition to the current input, to calculate the output. Future input information may also be useful in calculating the output of the neural network. Conventional recurrent neural networks use future input information by delaying the output by a number of time steps. Alternatively stated, a "window" of future input information is applied to the neural network's inputs. The window size must be set empirically. Bi-directional recurrent neural networks avoid the choice of window size, in that they use the entire sequence of inputs to calculate the outputs.

This section discusses the use of recurrent neural networks (conventional and bi-directional) as pattern classifiers. In Section 2.3.1 conventional recurrent neural networks are discussed, and bi-directional recurrent neural networks in Section 2.3.2. Appendix A discusses the training of these networks.

2.3.1 Conventional recurrent neural networks

RNNs are generally regarded as being more powerful than multilayer perceptrons. These neural networks contain one or more feedback connections. Although RNNs are not limited to a specific architecture, only recurrent multilayer perceptrons (RMLPs) [70] are discussed here. RNNs map their input space to the output space, by responding temporally to an externally applied input signal. The network acquires state representations and effectively uses all of the past input information to calculate the output. For this reason, they are often considered "dynamically driven recurrent networks".

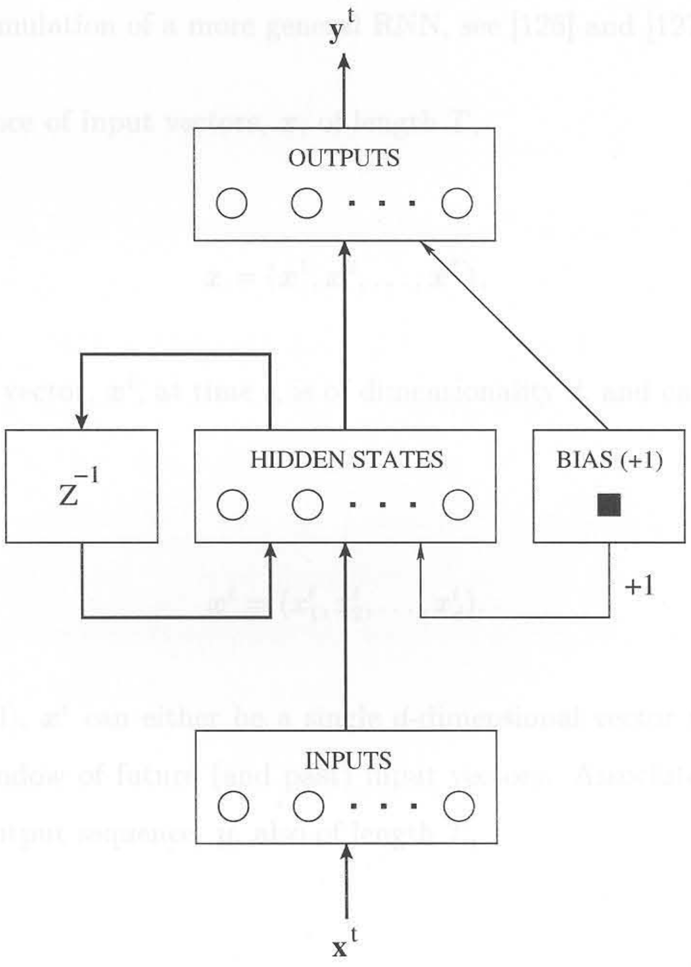

Figure 2.8: A conventional recurrent neural network (RNN).

Figure 2.8 shows the structure of a simple RMLP having two layers of weights. The output of the hidden layer nodes are delayed by one time step and applied to the input of these hidden layer neurons. In this way, past information continues to circulate through the network. The output of the neural network is calculated by using the outputs of the hidden layer neurons. Also shown in the figure is an input whose value is fixed at +1. This bias input is connected to both the hidden neurons and output neurons and allows more freedom to the formation of the decision boundaries by the network. The inputs shown in the figure are not regarded as being neurons. They are simply a convenient way to connect the external inputs to the hidden neurons.

The forward propagation of input vectors to output vectors for the two layer RNN, given in Figure 2.8, is presented below. This is probably the most common form of RNN. For the formulation of a more general RNN, see [126] and [127].

Consider a sequence of input vectors, $\boldsymbol{x}$, of length T,

$$\boldsymbol{x} = (\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^T), \tag{2.49}$$

where each input vector, $\boldsymbol{x}^t$, at time t, is of dimensionality d, and can be conveniently written as

$$\boldsymbol{x}^t = (x_1^t, x_2^t, \ldots, x_d^t). \tag{2.50}$$

In Equation (2.50), $\boldsymbol{x}^t$ can either be a single d-dimensional vector at time t, or may also include a window of future (and past) input vectors. Associated with the input sequence, is an output sequence, $\boldsymbol{y}$, also of length T,

$$\boldsymbol{y} = (\boldsymbol{y}^1, \boldsymbol{y}^2, \ldots, \boldsymbol{y}^T), \tag{2.51}$$

where each output vector, $\boldsymbol{y}^t$, at time t is of dimensionality c, and can be written as

$$\boldsymbol{y}^t = (y_1^t, y_2^t, \ldots, y_c^t). \tag{2.52}$$

A sequence of hidden nodes outputs, $\boldsymbol{o}$, of length T, is also formed,

$$\boldsymbol{o} = (\boldsymbol{o}^1, \boldsymbol{o}^2, \ldots, \boldsymbol{o}^T), \tag{2.53}$$

where each hidden layer vector, $\boldsymbol{o}^t$, at time t is of dimensionality m, and can be written as

$$\boldsymbol{o}^t = (o_1^t, o_2^t, \ldots, o_m^t). \tag{2.54}$$

The input to hidden layer weight matrix, $\boldsymbol{W}^{IH} = \{w_{ji}^{IH}\}$ contains the matrix of input to hidden layer weights. It is of dimension m x d. The bias unit to hidden layer weight matrix is denoted as $\boldsymbol{W}^{H} = \{w_{j}^{H}\}$ (of dimension m x 1), while the bias unit to output layer weight matrix is given by $\boldsymbol{W}^{O} = \{w_{k}^{O}\}$ (of dimension c x 1). The feedback matrix of hidden layer to hidden layer weights, is given by $\boldsymbol{W}^{HH} = \{w_{jk}^{HH}\}$. The hidden to output layer weight matrix, $\boldsymbol{W}^{HO} = \{w_{kj}^{HO}\}$ contains the weights from the hidden layer nodes to the output nodes. It is of dimension c x m. The first index on the weight subscript indicates the neuron to which the weight is going, while the rightmost index indicates the neuron from which the weight originated.

The output of a neuron is calculated as a function of the weighted sum of its inputs. For the hidden neurons this is

$$\begin{aligned} o_j^t = f_j(a_j^t) &= f_j\left(\sum_{i=1}^{d} w_{ji}^{IH} x_i^t + \sum_{k=1}^{m} w_{jk}^{HH} o_k^{t-1} + w_j^H\right), \\ & j = 1, 2, \ldots, m; t = 1, 2, \ldots, T, \end{aligned} \tag{2.55}$$

where o_k^{t-1} is the one step delayed hidden neuron output, taken as 0 at $t = 1$. In Equation (2.55), a_j^t is the weighted sum of inputs to the j'th hidden neuron. Hidden neuron j's transfer function is denoted as f_j, and it is often taken as the hyperbolic tangent function for reasons of faster training convergence,

$$f_j(a_j) \equiv \tanh(a_j) = \frac{e^{a_j} - e^{-a_j}}{e^{a_j} + e^{-a_j}}, \tag{2.56}$$

where e is the exponential function. The outputs of the output layer neurons can be calculated as

$$y_k^t = f_k(a_k^t) = f_k\left(\sum_{j=1}^{m} w_{kj}^{HO} o_j^t + w_k^O\right), \qquad k = 1, 2, \ldots, c; t = 1, 2, \ldots, T, \tag{2.57}$$

where f_k is the transfer function of the output neurons. The softmax activation function [69] is often used for the output neurons,

$$f_k(a_k) \equiv \frac{e^{a_k}}{\sum_{l=1}^{c} e^{a_l}}, \tag{2.58}$$

where the sum is taken over all c outputs. This activation function will ensure that the outputs satisfy the usual stochastic constraints ($y_k \geq 0$, $\sum y_k = 1$) so that they may be interpreted as probabilities.

Forward propagation procedure

The forward propagation of the inputs to the outputs, as used in the simulation of the network once it has been trained, can be summarised as follows:

1. Set activations of all neurons to zero, as well as the vector representing the 1 step delayed version of the hidden layer neuron outputs.

2. Apply an external input vector, $\boldsymbol{x}^t$, at the inputs of the network.

3. Calculate the outputs of hidden layer neurons, using Equations (2.55) and (2.56).

4. Calculate the outputs of output layer neurons, using Equations (2.57) and (2.58).

5. Form the output vector, $\boldsymbol{y}^t$, by taking the outputs of the output neurons.

6. Repeat steps 2 to 5 for all t.

2.3.2 Bi-directional recurrent neural networks

Context information often plays an important role in calculating the output of a neural network. Performance can be significantly increased when either or both past and future vectors are applied to the input of the neural network, in addition to the current input vector. The designer of the neural network is then faced with the problem of choosing the size of the window of future and past input vectors. Bi-directional recurrent neural networks [116, 117, 121] avoid the problem of choosing the window size, by using the entire sequence of input vectors. It thus makes use of all of the available past and future information at any time instant. The primary disadvantage is, however, that the architecture is not suitable for most on-line applications.

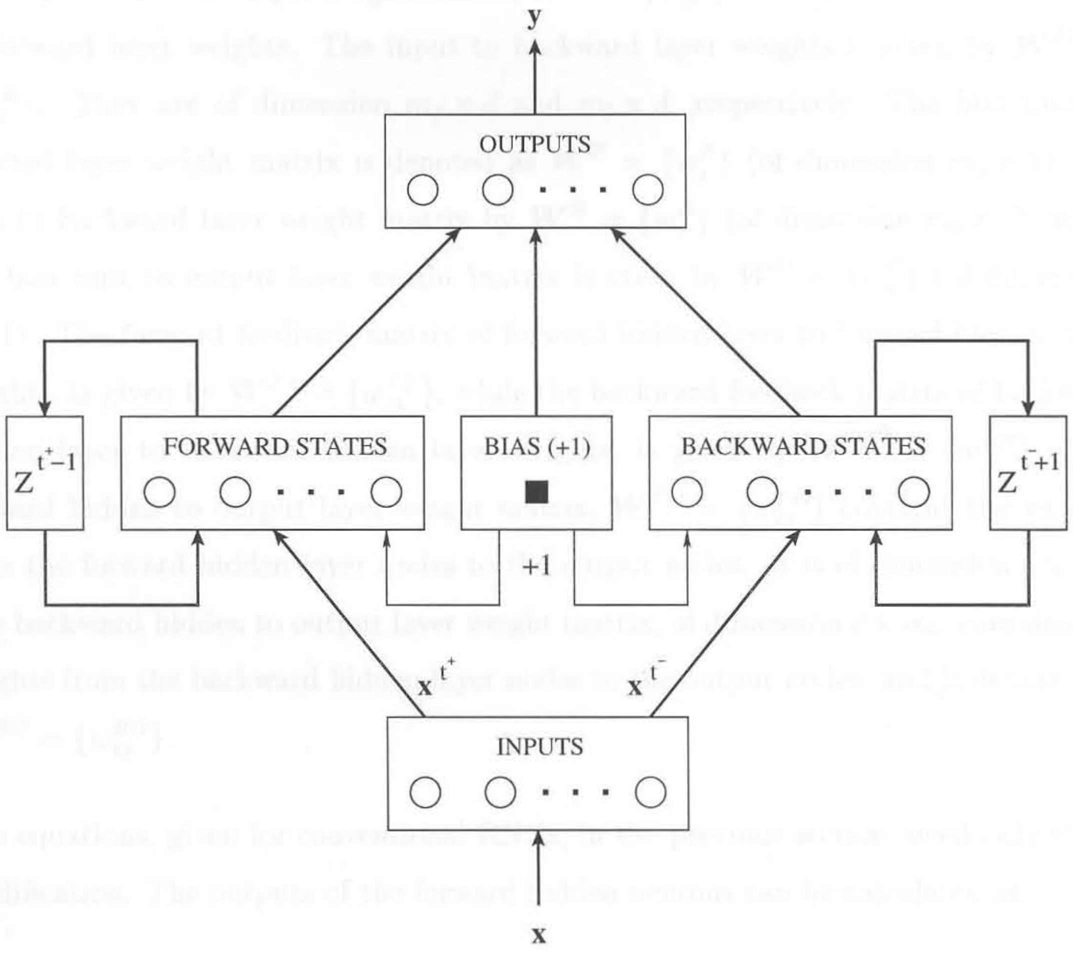

Figure 2.9: A bi-directional recurrent neural network (BRNN).

Figure 2.9 gives the structure of a bi-directional recurrent neural network. It consists of two underlying conventional recurrent neural networks, one operating in the forward time direction ($t^+ = 1$ to T) and one in the negative time direction ($t^- = T$ to 1). The hidden states of both these forward and backward RNNs are used to calculate the outputs of the neural network. The delay operators, z^{t^+-1} and z^{t^-+1}, are also shown for the forward and backward states, respectively. The former delays the forward state (neurons) outputs by 1 time step, while the latter advances the backward state (neurons) outputs by 1 time step. In the figure, $\boldsymbol{x}^{t^+}$ denotes the positive time input sequence, starting from $t = 1$, proceeding to $t = T$, while $\boldsymbol{x}^{t^-}$ denotes the negative time input sequence, starting from $t = T$, proceeding to $t = 1$.

The input to forward layer weight matrix, $\boldsymbol{W}^{IF} = \{w_{ji}^{IF}\}$ contains the matrix of input to forward layer weights. The input to backward layer weights is given by $\boldsymbol{W}^{IB} = \{w_{ji}^{IB}\}$. They are of dimension m_f x d and m_b x d, respectively. The bias unit to forward layer weight matrix is denoted as $\boldsymbol{W}^{F} = \{w_{j}^{F}\}$ (of dimension m_f x 1), the bias to backward layer weight matrix by $\boldsymbol{W}^{B} = \{w_{j}^{B}\}$ (of dimension m_b x 1), while the bias unit to output layer weight matrix is given by $\boldsymbol{W}^{O} = \{w_{k}^{O}\}$ (of dimension c x 1). The forward feedback matrix of forward hidden layer to forward hidden layer weights, is given by $\boldsymbol{W}^{FF} = \{w_{jk}^{FF}\}$, while the backward feedback matrix of backward hidden layer to backward hidden layer weights, is given by $\boldsymbol{W}^{BB} = \{w_{jk}^{BB}\}$. The forward hidden to output layer weight matrix, $\boldsymbol{W}^{FO} = \{w_{kj}^{FO}\}$ contains the weights from the forward hidden layer nodes to the output nodes. It is of dimension c x m_f. The backward hidden to output layer weight matrix, of dimension c x m_b, contains the weights from the backward hidden layer nodes to the output nodes, and is denoted by $\boldsymbol{W}^{BO} = \{w_{kj}^{BO}\}$.

The equations, given for conventional RNNs, in the previous section, need only slight modification. The outputs of the forward hidden neurons can be calculated as

$$o_j^{f,t^+} = f_j^f(a_j^{f,t^+}) \;=\; f_j^f\left(\sum_{i=1}^{d} w_{ji}^{IF} x_i^{t^+} + \sum_{k=1}^{m_f} w_{jk}^{FF} o_k^{f,t^+-1} + w_j^F\right),$$
$$j = 1,2,\ldots,m_f; t^+ = 1,2,\ldots,T. \tag{2.59}$$

The outputs of the backward neurons are calculated as

$$o_j^{b,t^-} = f_j^b(a_j^{b,t^-}) \;=\; f_j^b\left(\sum_{i=1}^{d} w_{ji}^{IB} x_i^{t^-} + \sum_{k=1}^{m_b} w_{jk}^{BB} o_k^{b,t^-+1} + w_j^B\right),$$
$$j = 1,2,\ldots,m_b; t^- = T, T-1,\ldots,1. \tag{2.60}$$

The outputs of the neural network are obtained only after calculating the forward and backward neurons' outputs for the entire input sequence, $\boldsymbol{x}$, as

$$y_k^t = f_k(a_k^t) \;=\; f_k\left(\sum_{j=1}^{m_f} w_{kj}^{FO} o_j^{f,t} + \sum_{i=1}^{m_b} w_{kj}^{BO} o_j^{b,t} + w_k^O\right),$$
$$k = 1,2,\ldots,c; t = 1,2,\ldots,T, \tag{2.61}$$

where $o_i^{f,t}$ is the output of the i'th forward neuron at time t, $o_i^{b,t}$ is the output of the i'th backward neuron at time t, m_f is the number of forward neurons, m_b is the number of backward neurons, and f_j^f, f_j^b and f_k are the neuron transfer functions for the forward hidden, backward hidden, and output layers, respectively. In the equations above, o_j^{f,t^+-1} denotes the 1 step delayed forward hidden node output, and o_j^{b,t^-+1} is the 1 step advanced backward hidden node output. The j'th forward hidden node output at time t and t^+ is given by $o_j^{f,t}$ and o_j^{f,t^+}, respectively. The j'th backward hidden node output at time t and t^- is given by $o_j^{b,t}$ and o_j^{b,t^-}, respectively.

Forward propagation procedure

The forward propagation of the inputs to the outputs, as used in the simulation of the network once it has been trained, can be summarised as follows:

1. Set activations of all neurons to zero, as well as the vectors representing the 1 step delayed, and 1 step advanced version of the forward and backward neurons' outputs.

2. Apply an external input sequence, $\boldsymbol{x}$, at the inputs of the network.

3. Calculate the outputs of the forward hidden layer neurons, using Equations (2.59) and (2.56) for $t^{+} = 1$ to T (the entire sequence).

4. Calculate the outputs of the backward hidden layer neurons, using Equations (2.60) and (2.56) for $t^{-} = T$ to 1 (the entire sequence).

5. Calculate the outputs of the output layer neurons, using Equations (2.61) and (2.58) (the entire sequence of outputs).

6. Form the output sequence, $\boldsymbol{y}$, by taking the sequence of network outputs.