Chapter 2

Preliminaries and Methodologies

In this chapter, we will review all the mathematical terminologies related to signal processing and machine learning models that are used in our research work. First, we will see the feature extraction techniques like the mel-frequency cepstral coefficient and the mel-frequency discrete wavelet coefficients, with a brief review of wavelets. Then, we will review a signal processing technique named short-term autocorrelation. Further, various normalisation techniques are also reviewed. After that, various methods used for classification are discussed. These classification techniques are dynamic time warping, artificial neural networks, radial basis function networks, and hidden Markov models. Finally, the chapter ends with a brief on the confusion matrix and performance measures.

To do successful automatic speech recognition, we need to extract vital information from the speech signal, called features. This is essential as the speech signal, in its raw form, is of large dimension. So, using it for recognition directly would require a huge amount of storage and time. Thus, the feature extraction step allows for reducing the dimension of the data without losing important information from the speech. We mention two such techniques that are used in our work.

2.1 Feature Extraction using Mel-Frequency Cepstral Coefficients

Feature extraction is the process of extracting vital features from a digital speech signal. The output of the feature extraction process is a feature vector. They not only contain important features of the speech, but they also have a lesser dimension than the input. These feature vectors represent the information in a small window of the speech signal [6]. It is a spectral and parametric representation of the digital speech signal. There are many ways to represent a speech signal in this way. Some well-known feature extraction techniques for speech are linear predictive coefficients (LPC), reflection coefficients (RC), linear predictive cepstral coefficients (LPCC), perceptual linear predictive coefficients, mel-frequency cepstral coefficients (MFCC), and relative spectra filtering of log domain coefficients (RASTA) [54], [55]. But the most commonly used method for feature extraction from speech is MFCC [5].

The steps for determining MFCC from a speech signal are as follows:

1. First, the speech signal is pre-emphasised. Let $x(t)$ be the recorded digital speech signal, where x is amplitude and t is a sample number representing discrete time. From this, the pre-emphasised speech signal is obtained using equation (2.1). Pre-emphasising of speech signals is done using a first-order high-pass filter [1], [6].

$$s(t) = x(t) - \alpha x(t), \qquad 0.9 \leqslant \alpha \leqslant 1 \tag{2.1}$$

   This step increases the amount of energy for the high frequencies. It makes information from higher formants more available to the recognition process, and hence it improves recognition ability [6]. Moreover, this step makes the signal spectrally flat, which makes it less susceptible to finite precision effects. More commonly, the value of α is considered around 0.95 [1].

2. Speech is a non-stationary signal [6]. As a result, the spectrum of speech changes very quickly. So, the features are not extracted from the entire speech. It is important to extract features from a small part of speech. Hence, after pre-emphasising, in this second step, the pre-emphasised signal $s(t)$ is divided into frames s_i of N samples each, where i is the frame index. There is an overlap of M samples from each frame with its adjacent frame. This gives frames

$$s_i(t), \qquad 0 \leqslant t \leqslant N - 1 \tag{2.2}$$

   The values of N and M are chosen such that the signal is approximately stationary within each frame and its statistical properties are constant within the frame [6]. For a speech with a sampling rate of 8000 samples per second, $N = 300$ makes each frame around 45 milliseconds long, which is called a frame size. And $M = 100$ gives overlapping of 15 milliseconds, which is called a frame shift.

3. The above steps abruptly cut the signal at its boundaries, giving a rectangular window (or a Dirichlet window) in which the sample values are not modified at all. Such a rectangular window can cause discontinuities, which can create a problem while applying the Fourier transform [6]. To overcome this issue, in this third step, a Hamming window given by equation (2.3) is applied to each pre-emphasised frame.

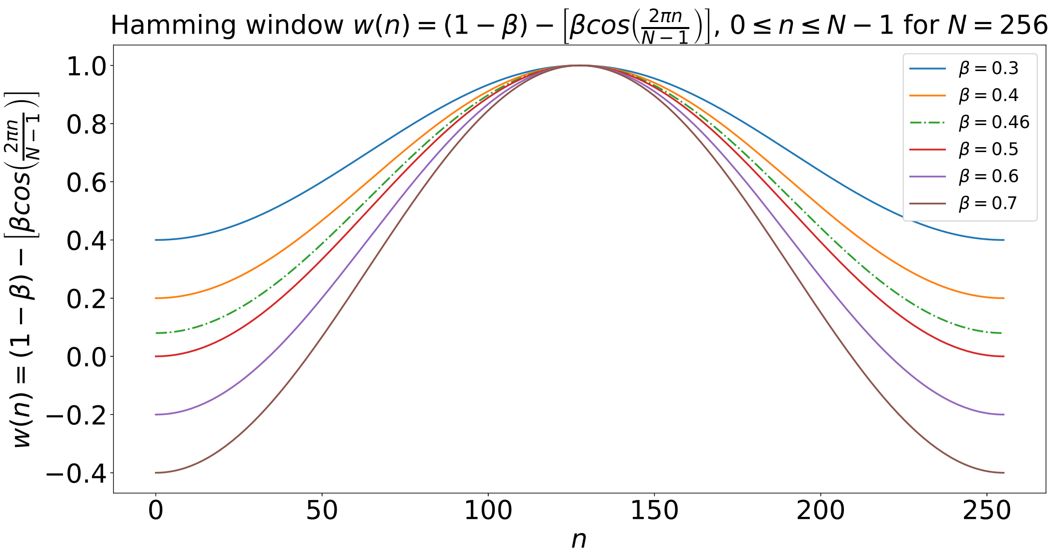


Figure 2.1: Hamming window for different values of β

$$w(t) = (1-\beta) - \beta \cos\left(\frac{2\pi t}{N-1}\right), \qquad 0 \leqslant t \leqslant N-1 \tag{2.3}$$

The typical value of β is taken as 0.46. For different values of β, the hamming windows are shown in the Figure 2.1. This step shrinks the sample values of the signal to zero at both boundaries of the window. To apply the hamming window, a pre-emphasised signal is simply multiplied by the Hamming window, as shown in the equation (2.4).

$$s_i(t)w(t), \qquad \forall i \tag{2.4}$$

This step decreases the discontinuities at both ends of the frames [1], as shown in the Figure 2.2 [4].

4. In the next step, spectral details are obtained from the windowed signal. This is done by applying the discrete Fourier transform (DFT) to a windowed signal. For all frames i, it is defined in the equation (2.5).

$$\mathrm{DFT}_i(f) = \sum_{t=0}^{N-1} s_i(t)w(t)e^{\frac{-2f\pi jt}{N}}, \qquad 0 \leqslant f \leqslant N-1 \tag{2.5}$$

Here, f represents frequencies. This equation gives us information about the amount of energy present at various frequency levels [6]. The output of the discrete Fourier transform is a sequence of complex numbers. This represents the phase and amplitude of the frequency in the windowed signal. To visualise this, the power

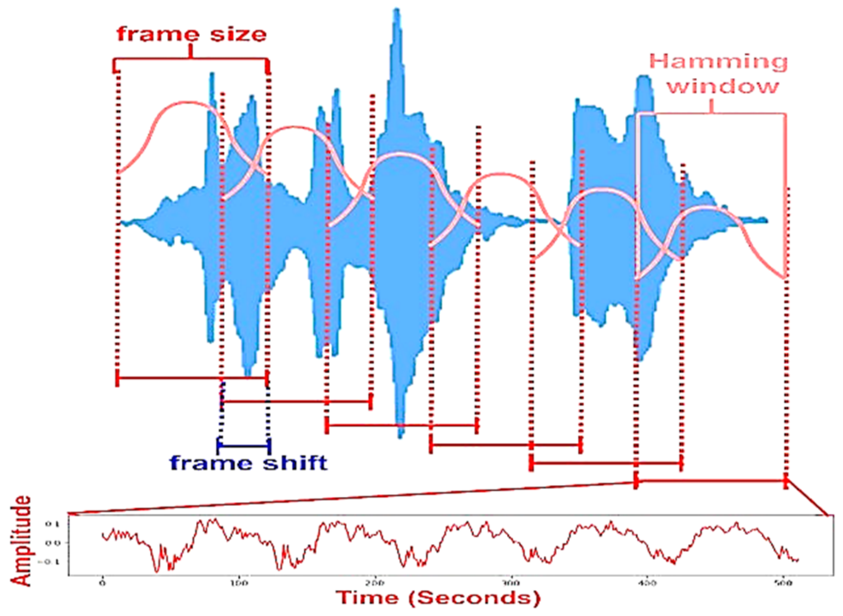


Figure 2.2: Illustration of Framing and Windowing [4]

spectrum defined by equation (2.6) is used for all frames i.

$$\mathrm{PS}_i(f) = \frac{1}{N}|\mathrm{DFT}_i(f)|^2, \qquad 0 \leqslant f \leqslant \mathrm{floor}\left(\frac{N-1}{2}\right) \tag{2.6}$$

5. According to the psychophysical studies, human hearing is not equally sensitive to all frequency components [6]. For sounds with high frequency, the human perception is less sensitive. Human hearing of the frequency component of speech does not follow a linear scale [1]. Thus, the frequency output f by discrete Fourier transform is warped using the "mel" scale $\mathrm{Mel}(f)$ given by the equation (2.7). This is the reason why they are called mel-frequency coefficients.

$$\mathrm{Mel}(f) = 1127 \ln\left(1 + \frac{f}{700}\right) \tag{2.7}$$

According to this function, the relationship between frequency and mel is approximately linear up to 1000 Hz, and then it is logarithmic, as shown in the Figure 2.3. After that, a filter bank of 20-40 triangular filters defined by equation (2.8) is considered, where k is filter index and f is frequency.

$$\mathrm{Filter}_f(k) = \begin{cases} \frac{k-\mathrm{Mel}(f-1)}{\mathrm{Mel}(f)-\mathrm{Mel}(f-1)}, & \mathrm{Mel}(f-1) \leqslant k \leqslant \mathrm{Mel}(f) \\ \frac{\mathrm{Mel}(f+1)-k}{\mathrm{Mel}(f+1)-\mathrm{Mel}(f)}, & \mathrm{Mel}(f) \leqslant k \leqslant \mathrm{Mel}(f+1) \\ 0, & \text{otherwise} \end{cases} \tag{2.8}$$

6. This filter bank is applied to the power spectrum. They collect energy from each

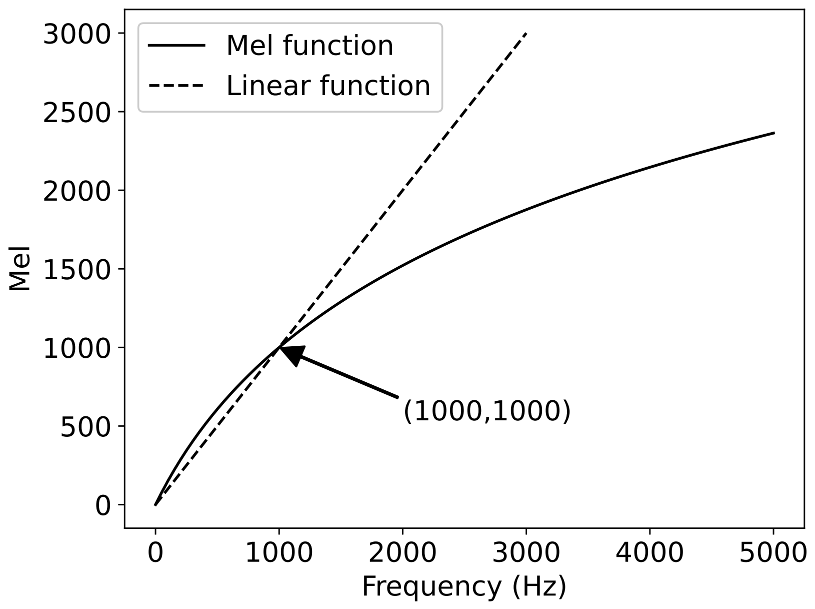


Figure 2.3: Plot of mel-scale converted from frequencies

frequency band [6]. Due to the mel scale, out of 20 filters, the first 10 are linearly spaced below 1000 Hz, and the other 10 are logarithmically spaced above 1000 Hz. Multiplying these mel-scaled filters with the power spectrum given by equation (2.6), we get energy output of k^{th} filter $\hat{S}_k$, defined by equation (2.9) [56].

$$\hat{S}_k = \mathrm{PS}_i(f) \cdot \mathrm{Filter}_f(k) \tag{2.9}$$

7. After that, a logarithm of energy output of k^{th} filter is determined, which gives $\ln(\hat{S}_k)$ [56]. This is important because the response of the human ear is logarithmic when receiving signals. So taking logarithm makes features less sensitive to variations in speech due to the speaker's mouth moving further or closer to the recording microphone [6].

8. When a speaker is speaking, the speech waveform is produced at a glottal source with some fundamental frequency. Then it passes through a vocal tract, which represents a filter. Due to the shape of the vocal tract, various speech sounds are produced. So the characteristics of the filter (vocal tract) are more important as compared to the characteristics of the source (glottal source). Because the exact characteristics of the vocal tract can help us identify the speech sound produced, So, it is important to separate the filter from the source. Cepstrum helps us with this [6], and the discrete cosine transform is a way to convert the spectrum into the cepstrum. This is the reason why they are called cepstral coefficients. So finally, taking a discrete cosine transform of the logarithm of the equation (2.9), we get important features called mel-frequency cepstral coefficients given by equation

(2.10) [56].

$$\hat{c}_n = \sum_{k=1}^{K} \ln\left(\hat{S}_k\right) \cos\left[n\left(k - \frac{1}{2}\right)\frac{\pi}{K}\right] \tag{2.10}$$

Here, K is number of filters and $n = 1, 2, ..., L$ and L is the desired length of the feature vector.

The MFCC features are extracted for each frame. In this method, only the first few cepstral coefficients (9-13) are taken as features because these initial coefficients represent information only about the vocal tract filter, separating the information from the glottal source.

2.2 Feature Extraction using Mel-Frequency Discrete Wavelet Coefficients

The speech signal is a non-stationary signal. The wavelets are better suited for analysing the non-stationary signal. So, discrete wavelet coefficients can give better results when used as a part of the feature extraction process in speech recognition. They are called mel-frequency discrete wavelet coefficients (MFDWC)[57]. Before understanding this feature extraction technique, we will briefly review the concept of wavelets.

2.2.1 Wavelets

The word "wavelet" means a small wave. It is a windowed function of finite length. The nature of this function is oscillatory, so a wavelet is a type of wave. Wavelets have compact support, which means that the oscillations do not last forever. Another interesting property about a wavelet is that the area underneath it is zero, which means that the energy is equally distributed in positive and negative directions. So wavelets are quickly decaying wavelike oscillations that have a zero mean and exist for a finite duration. Let Ψ be a wavelet function, also called a mother wavelet, then it satisfies the following properties:

$$\int_{-\infty}^{\infty} |\Psi(t)|\, dt < \infty \tag{2.11}$$

$$\int_{-\infty}^{\infty} |\Psi(t)|^2\, dt < \infty \tag{2.12}$$

$$\int_{-\infty}^{\infty} \Psi(t) dt = 0 \tag{2.13}$$

Equation (2.11) says that the mother wavelet $\Psi(t)$ is an absolutely integrable function, and equation (2.12) says that the mother wavelet $\Psi(t)$ is a square integrable function. The meaning of these two properties is that the mother wavelet has finite energy and

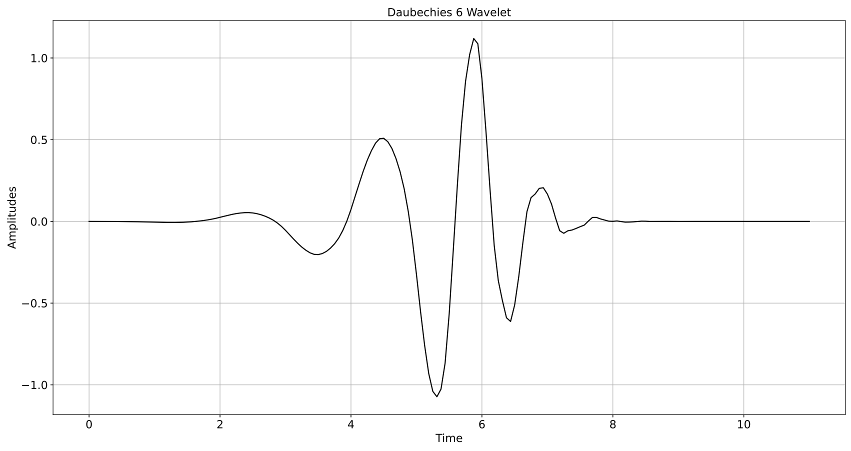


Figure 2.4: The plot of Daubechies-6 wavelet function

$\Psi(t) \to 0$ as $t \to \pm\infty$. Equation (2.13) says that the area under the wavelet function is zero. This means that a mother wavelet has a zero mean. So a wavelet is a function with zero mean, finite energy, compact support, and a fast-decaying nature. One example of such a wavelet function is shown in Figure 2.4. It is called a Daubechies-6 wavelet function, named after Belgian mathematician Ingrid Daubechies.

A mother wavelet can be scaled and translated according to need. This is done by a scale parameter a and a translation parameter b. So, this gives us $\Psi_{a,b}(t)$, which is the same wavelet function but scaled by factor a and translated by factor b. It is given by:

$$\Psi_{a,b}(t) = \frac{1}{\sqrt{a}}\Psi\left(\frac{t-b}{a}\right) \tag{2.14}$$

The continuous wavelet transform of signal $f(t)$ at scale a and translation b is given by

$$W\Big(f(t)\Big) = \int_{-\infty}^{\infty} f(t)\Psi_{a,b}(t)dt \tag{2.15}$$

Now, most of the real-life problems deal with the signals, which are discrete. So, there is a requirement for a discrete wavelet transform of such a signal. Suppose $X[n]$ be a discrete signal defined corresponding to $f(t)$ at M discrete values on $0, 1, 2, \ldots, M-1$. The general expression of a discrete wavelet transform (DWT) for a discrete signal $X[n]$, having M samples, is given by approximation coefficients (2.16) and detail coefficients (2.17).

$$W_\phi[j_0, k] = \frac{1}{\sqrt{M}}\sum_n X[n]\phi_{j_0,k}[n] \tag{2.16}$$

$$W_\psi[j, k] = \frac{1}{\sqrt{M}}\sum_n X[n]\psi_{j,k}[n], j > j_0 \tag{2.17}$$

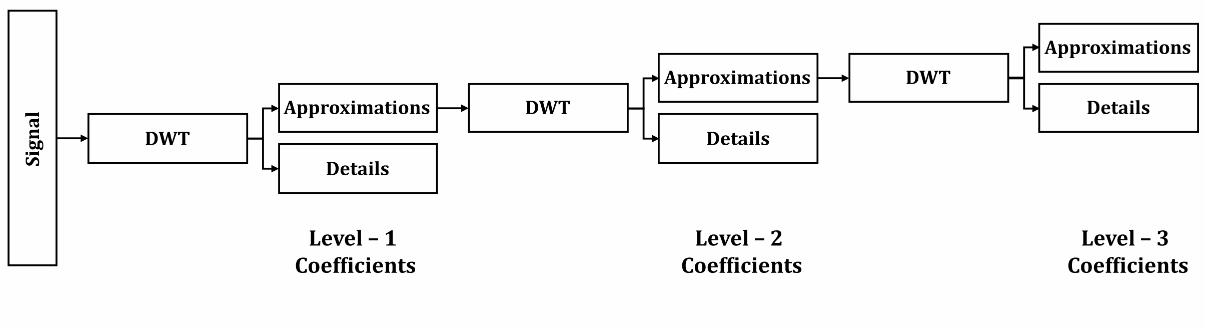


Figure 2.5: Discrete wavelet transform: wavelet coefficients at different levels

Here $\phi_{j_0,k}$ is a discrete scaling function and $\psi_{j,k}[n]$ is a discrete wavelet function having M components each [58]. These approximation and detail coefficients are the discrete wavelet transforms of a given signal. These coefficients are obtained at various levels. Level-1 approximation and detail coefficients are obtained by applying a discrete wavelet transform to the signal. Then, level-2 approximation and detail coefficients are obtained by applying a discrete wavelet transform to level-1 approximation coefficients, and so on. This process is shown up to level 3 in the Figure 2.5.

The advantage of the wavelet transform over the Fourier transform is that the window width is adjustable. It uses short windows for high frequencies of the signal and long windows for low frequencies of the signal. Due to this advantage, the discrete wavelet transform is used in the feature extraction technique for speech recognition, as explained in the following.

Mel-Frequency Discrete Wavelet Coefficients

In the previous section, we described how the discrete cosine transform is used in the derivation of feature vectors by mel-frequency cepstral coefficients. In the discrete cosine transform, the basis vector covers all the frequency levels. But there can be a corruption in frequency levels due to the presence of noise in the speech signal. This affects the values of feature vectors. This is one limitation [55]. Another limitation is the possible presence of two adjacent phonemes, one voiced and another unvoiced, in some frames of a speech signal. In such situations, the voiced phoneme information prevails in the low frequency spectrum, and the unvoiced phoneme information prevails in the high frequency spectrum. The mel-frequency cepstral coefficients are designed in such a way that one speech frame should ideally represent only one phoneme at a time [55].

These above-stated limitations of mel-frequency cepstral coefficients can be overcome by the use of the discrete wavelet transform, due to its property of localisation in time and

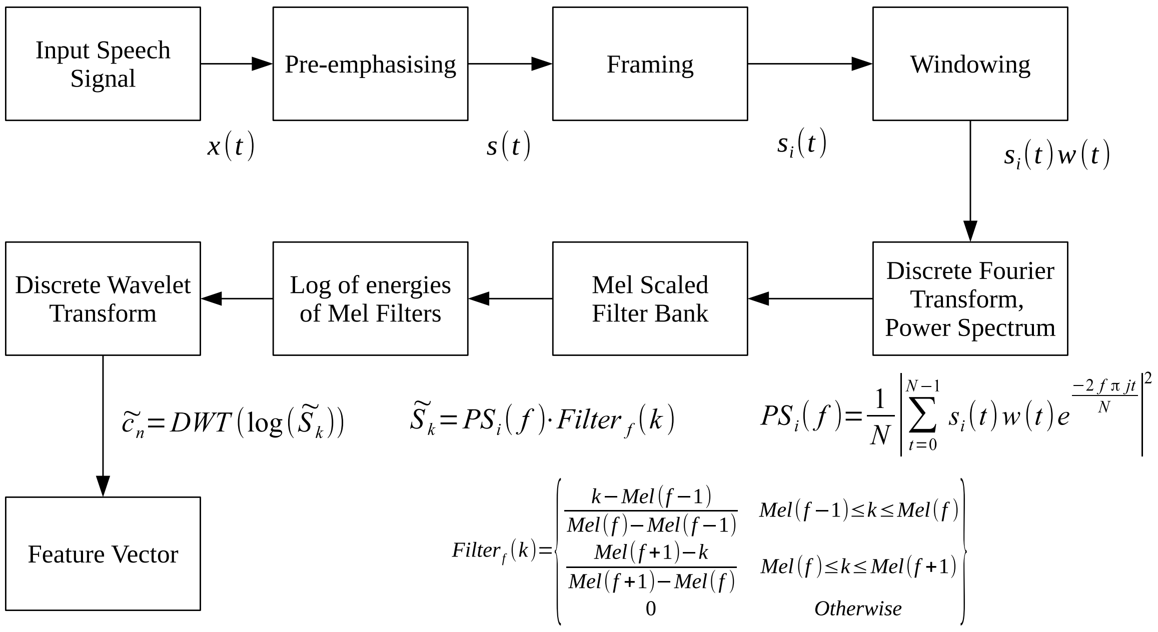

Input Speech Signal
Pre-emphasising
Framing
Windowing
$x(t)$
$s(t)$
$s_i(t)$
$s_i(t)w(t)$
Discrete Wavelet Transform
Log of energies of Mel Filters
Mel Scaled Filter Bank
Discrete Fourier Transform, Power Spectrum
$\tilde{c}_n = DWT(\log(\tilde{S}_k))$
$\tilde{S}_k = PS_i(f) \cdot Filter_f(k)$
$PS_i(f) = \frac{1}{N}\left|\sum_{t=0}^{N-1} s_i(t)w(t)e^{\frac{-2f\pi jt}{N}}\right|^2$
Feature Vector
$Filter_f(k) = \begin{cases} \frac{k-Mel(f-1)}{Mel(f)-Mel(f-1)} & Mel(f-1) \le k \le Mel(f) \\ \frac{Mel(f+1)-k}{Mel(f+1)-Mel(f)} & Mel(f) \le k \le Mel(f+1) \\ 0 & Otherwise \end{cases}$

Figure 2.6: Steps of determining feature vectors.

frequency domains. Also, the discrete wavelet transform is less affected by corruption in the frequency band. This solves the issue of speech frames having noise. Moreover, due to the multi-resolution analysis feature of the discrete wavelet coefficients, the frequency levels are divided into sub-levels, and each sub-level is processed separately. This solves the issue of voiced and unvoiced phonemes in the same frame.

The steps for determining features using MFDWC from the input speech signals are as stated below. They differ from the MFCC in the last step [57].

- Pre-emphasis the digital speech signal.
- Frame the signal with overlapping frames.
- Apply the Hamming window to each frame.
- Obtain a discrete Fourier transform and power spectrum for each frame.
- Find energy of applied Mel-scaled triangular filter-bank.
- Apply suitable wavelets to logarithm of the filter-bank energies, resulting into discrete feature vectors using the discrete wavelet transform.

These steps are summarised in the block diagram: Figure 2.6.

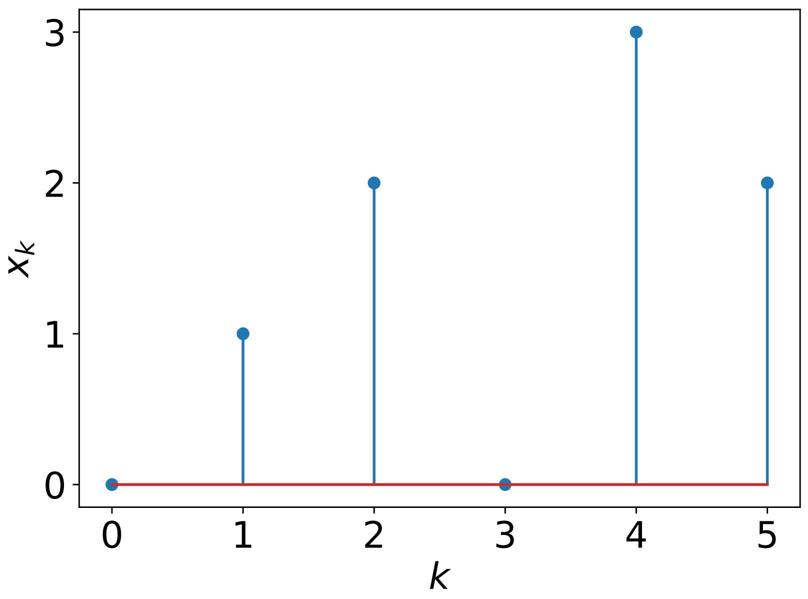


Figure 2.7: Plot of a sample signal x

2.3 Short-Term Autocorrelation

Short-term autocorrelation is a signal processing technique that is useful for isolating words in a sentence. A signal x, in discrete form, can be represented as an array of numbers. The autocorrelation of an array (representing signal) x of length N is given by equation (2.18) [59].

$$a_n = \sum_{k=0}^{N-1} x_k x_{k-n} \tag{2.18}$$

Here, n is called the lag, and it is an integer value, a_n is the n^{th} coefficient of autocorrelation between the original signal x and the same signal shifted by n samples. The sign of n represents whether the signal is shifted to the right or left. If $n = 0$, we get energy from the signal. Moreover, k in equation (2.18) is the index representing the k^{th} element of an array x.

To understand how autocorrelations are calculated, consider the following example: Let $x = (0, 1, 2, 0, 3, 2)$ be a given signal or an array, with length, $N = 6$ as shown in the Figure 2.7.

Then, substituting $n = 0$ in equation (2.18) for this array, we get,

$$a_0 = \sum_{k=0}^{5} x_k x_k = x_0x_0 + x_1x_1 \ldots x_5x_5 = 0^2 + 1^2 + \cdots + 2^2 = 18$$

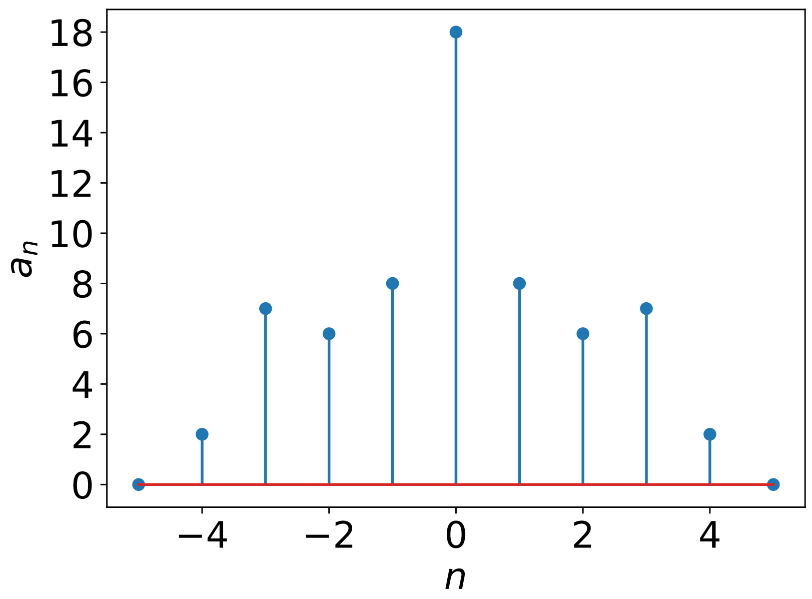


Figure 2.8: Plot of autocorrelations for various values of n

Substituting $n = 1$ in equation (2.18) for this array, we get,

$$a_1 = \sum_{k=0}^{5} x_k x_{k-1} = x_1 x_0 + x_2 x_1 \ldots x_5 x_4 = 1(0) + 2(1) + \cdots + 2(3) = 8$$

Substituting $n = 2$ in equation (2.18) for this array, we get,

$$a_2 = \sum_{k=0}^{5} x_k x_{k-2} = x_2 x_0 + x_3 x_1 \ldots x_5 x_3 = 2(0) + 0(1) + \cdots + 2(0) = 6$$

Similarly, $a_3 = 7$, $a_4 = 2$ and so on. One important property, that equation (2.18) satisfies is $a_n = a_{-n}$. Plotting all the autocorrelations for various values of n, we get a plot as shown in the Figure 2.8.

In the context of speech signals, these autocorrelations are called 'short-term' autocorrelations. Because the signal is divided into several frames, each having N samples, Then, the autocorrelation with lag $n = 1$ is calculated for each frame, which is a non-negative number as seen in the example above.

2.4 Normalisation

Normalisation is a common technique used to rescale the data in a certain range before using it in machine learning models for classification tasks. This improves convergence and reduces the complexity of calculations. In our work, we used a few normalisation techniques that are summarised here.

1. Maximum normalisation
2. Min-max normalisation
3. Z-score

Let $\vec{A} = (a_1, a_2, \ldots, a_n)$ be a given vector or pattern given in the data. In the maximum normalisation, each input vector (pattern) is divided by the maximum value of all the components of that vector. So, the normalised $\vec{A}$ is obtained by dividing it by M where $M = \max\{a_i\}$, where $i = 1, 2, \ldots, n$. So we have:

$$\text{Normalised } a_i = \frac{a_i}{M} \tag{2.19}$$

The min-max normalisation of the same $\vec{A}$ is obtained by using the formula given in the equation (2.20), A_{min} is the minimum of all the components and A_{max} is the maximum of all the components.

$$\text{Normalised } a_i = \frac{a_i - A_{min}}{A_{max} - A_{min}} \tag{2.20}$$

For determining the Z-score, first we need to calculate the mean $\bar{a}$ and standard deviation σ of all the components of vector $\vec{A}$. Then it is determined using the formula given in the equation (2.21).

$$\text{Normalised } a_i = \frac{a_i - \bar{a}}{\sigma} \tag{2.21}$$

Maximum normalisation is used for the data with all the non-negative numbers. In this case, the normalised values are in the range $[0, 1]$. Min-max normalisation is useful for the data having real values, and the normalised values are in the range $[0, 1]$. Z-score is also useful for data with real values, and its usual range is within $[-3, 3]$.

Suppose $\vec{A} = (3, 1, 6, 4, 9, 5, 2)$. Then the normalised vector $\vec{A}$ using various normalisation techniques is as follows:

- Maximum normalisation: $(0.33, 0.11, 0.67, 0.44, 1, 0.56, 0.22)$
- Min-max normalisation: $(0.25, 0, 0.63, 0.38, 1, 0.5, 0.13)$
- Z-score: $(-0.52, -1.32, 0.69, -0.11, 1.89, 0.29, -0.92)$

Once we have generated feature vectors from the speech signal for the recognition phase of the automatic speech recognition process, we would require various techniques that can perform this job. In the next section, we describe the classification techniques that we have used in our work.

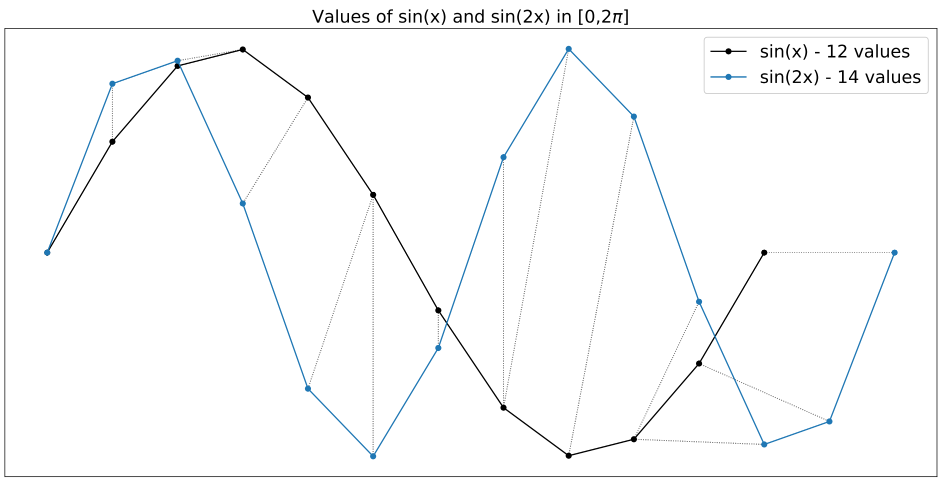


Figure 2.9: Alignment of two sequences of unequal lengths

2.5 Dynamic Time Warping

Dynamic time warping allows us to find the similarity between two vectors by measuring the distance between them. There are various types of distance measures defined to find such similarity. Euclidean distance is the most commonly used distance measure. However, the main limitation of Euclidean distance is that we cannot use it for vectors having different lengths. It is used to determine the similarity between vectors of equal length. In speech recognition problems, we cannot use it for classification purposes because speech feature vectors, produced in the previous phase, may not be equal in length. The length of speech feature vectors will depend on the speed of the speaker's utterances. For speakers who speak fast, the length of the feature vector is short, and for others, the length of the feature vector is long. Even for the same word spoken by the same speaker in two different instances, we may get two feature vectors of unequal length. Hence, to find the distance between two speech feature vectors, we have to explore some other distance measures.

Dynamic time warping was introduced by Sakoe and Chiba in 1978 [60]. Using this, we can align the two vector sequences of unequal lengths to find the distance measure. Figure 2.9 shows the alignment of 12 values of $sin(x)$ and 14 values of $sin(2x)$ in the interval [0, 2π] [61].

Let us try to understand this with one example. Let $X = \{x_1, x_2, \ldots, x_p\}$ and $Y = \{y_1, y_2, \ldots, y_q\}$ be two vectors of unequal lengths p and q respectively. Then the dynamic time warping distance between X and Y is determined by the equation (2.22).

$$D(x_i, y_j) = |x_i - y_j| + K, \qquad 1 \leqslant i \leqslant p, 1 \leqslant j \leqslant q.. \tag{2.22}$$

and K is given as

$$K = \min\{D(x_i, y_{j-1}), D(x_{i-1}, y_{j-1}), D(x_{i-1}, y_j)\} \tag{2.23}$$

Here, $|\cdot|$ is the absolute difference.

Let us try to understand how this works with a small example. Let X be a vector having 5 components defined by $X = (1, 3, 5, 8, 8)$ and Y be a vector having 4 components defined by $Y = (2, 4, 6, 7)$. First of all, the 5×4 matrix D is constructed, where the components of X are arranged row-wise and the components of Y are arranged column-wise, as shown in the Figure 2.10 (a). The elements of this matrix are $D(x_i, y_j)$ for $1 \leqslant i \leqslant p$ and $1 \leqslant j \leqslant q$.
Then, the absolute difference $|x_i - y_j|$ is computed using all the components of vector X with all the components of vector Y, as shown in the Figure 2.10 (b).

The element $D(x_1, y_1)$ is calculated using, $K = 0$ and

$$D(x_1, y_1) = |x_1 - y_1| = |1 - 2| = 1$$

Then all the elements of the first column are determined using $K = D(x_{i-1}, y_1)$ and

$$D(x_i, y_1) = |x_i - y_1| + K$$

$$\therefore D(x_i, y_1) = |x_i - y_1| + D(x_{i-1}, y_1)$$

as shown in the Figure 2.10 (c).

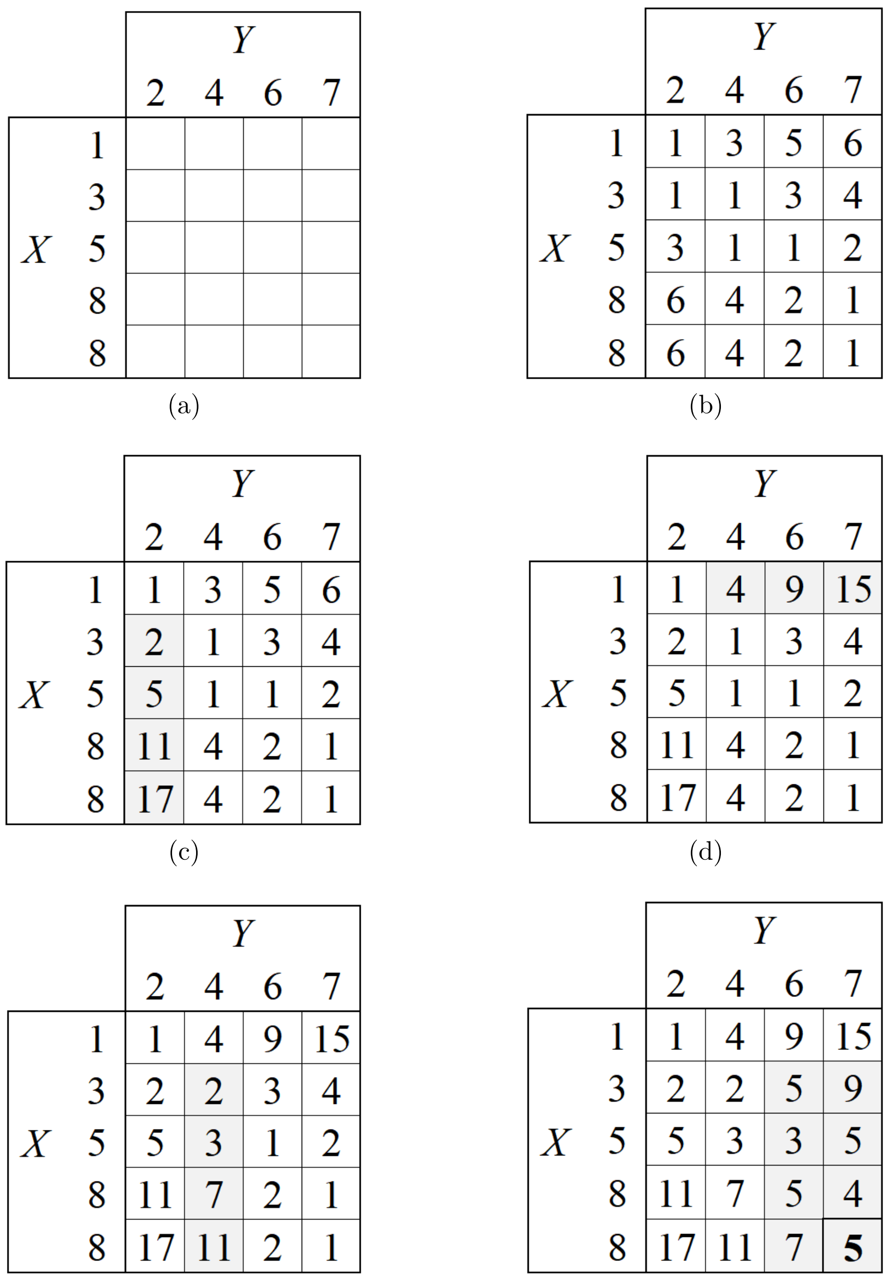
Similarly, the elements of the first row are determined using $K = D(x_1, y_{j-1})$ and

$$D(x_1, y_j) = |x_1 - y_j| + K$$

$$\therefore D(x_1, y_j) = |x_1 - y_j| + D(x_1, y_{j-1})$$

as shown in the Figure 2.10 (d).

Next comes the calculation of all the elements, starting from the second row and second column $D(x_2, y_2)$ to the last row and last column $D(x_5, y_4)$. These values are determined



		Y			
		2	4	6	7
	1				
	3				
X	5				
	8				
	8				

(a)

		Y			
		2	4	6	7
	1	1	3	5	6
	3	1	1	3	4
X	5	3	1	1	2
	8	6	4	2	1
	8	6	4	2	1

(b)

		Y			
		2	4	6	7
	1	1	3	5	6
	3	2	1	3	4
X	5	5	1	1	2
	8	11	4	2	1
	8	17	4	2	1

(c)

		Y			
		2	4	6	7
	1	1	4	9	15
	3	2	1	3	4
X	5	5	1	1	2
	8	11	4	2	1
	8	17	4	2	1

(d)

		Y			
		2	4	6	7
	1	1	4	9	15
	3	2	2	3	4
X	5	5	3	1	2
	8	11	7	2	1
	8	17	11	2	1

(e)

		Y			
		2	4	6	7
	1	1	4	9	15
	3	2	2	5	9
X	5	5	3	3	5
	8	11	7	5	4
	8	17	11	7	**5**

(f)

Figure 2.10: Steps of determining a Dynamic time warping distance between vectors X and Y of unequal lengths: (a) Arrangement of two vectors of unequal lengths as a matrix. (b) Absolute difference between all the elements of X and Y. (c) Determining the elements of first row. (d) Determining the elements of first column. (e) Determining the elements of second row. (f) Determining the elements of the third and fourth column. The element $D(x_5, y_4) = 5$ is the dynamic time warping distance between these two vectors.

		Y			
		2	4	6	7
	1	**1**	4	9	15
	3	2	**2**	5	9
X	5	5	3	**3**	5
	8	11	7	5	**4**
	8	17	11	7	**5**

Figure 2.11: Optimal alignment between X and Y to determine minimum distance

using $K = \min\{D(x_{i-1}, y_j), D(x_{i-1}, y_{j-1}), D(x_i, y_{j-1})\}$ and

$$D(x_i, y_j) = |x_i - y_j| + K$$

$$\therefore D(x_i, y_j) = |x_i - y_j| + \min\{D(x_{i-1}, y_j), D(x_{i-1}, y_{j-1}), D(x_i, y_{j-1})\}$$

as shown in Figures 2.10 (e) and 2.10 (f).

After these calculations, the value of the last row and last column is the dynamic time warping distance between the two vectors X and Y. In this case, it is $D(x_5, y_4) = 5$.

Now, we trace back from $D(x_5, y_4)$ to $D(x_1, y_1)$ to find the optimal alignment of the two vectors, which gives this minimum distance of 5. This optimal alignment is shown in the Figure 2.11.

This way, we can determine the dynamic time warping distance between any two vectors of unequal lengths. This is particularly useful for speech recognition problems because we can determine the similarity between the feature vectors of a template speaker and the feature vectors of an unknown speaker. Based on this distance computed, we can recognise the spoken words by finding the minimum of dynamic time warping distances between the stored template words and the unknown word.

2.6 Multilayered Perceptrons

Scientists were always curious to find out how our brain works, how it remembers, stores, and recognises information. This curiosity of scientists inspired the study of biological aspects of the brain and then the mathematical and logical representation of various capacities of the human brain. This led to the evolution of a new computing area named artificial neural networks. It consists of algorithms that were developed as a mathematical model of the workings of the human brain. These algorithms are useful for real-world applications based on classification, function approximation, and regression problems.

It all started in 1943, when neuroscientist McCulloch and logician Pitts gave a simple mathematical model for the firing rules of biological neurons having dendrites, soma, and axons. The model was named the McCulloch-Pitts neuron model. The calculations of this model were shown as a schematic graph with vertices or nodes. These nodes were named artificial neurons. The objective of this model was to perform a linear classification on selected data with boolean inputs and boolean outputs. But the working of synapses in the brain was missing in this model.

Then, in 1949, Hebb, in his book "The Organisation of Behaviour," gave criteria about how the strength of synapses between two neurons becomes stronger or weaker. Based on this rule, scientists Stent, Changeux, and Danchin gave a model in the 1970s. Unlike the McCulloch-Pitts neuron model, here the synaptic weights were assigned to each connection between two neurons, which could change with training. The model based on Hebb's rule was used for classification tasks with boolean inputs and outputs.

In 1958, American psychologist Frank Rosenblatt proposed the Perceptron model, which was modified by Minsky and Papert in 1969. This model was capable of linear classification of boolean as well as real inputs. The limitation of this model is that it cannot classify a simple non-linear boolean function, XOR, which led to the discovery of multilayered perceptrons in the 1980s. In 1989, Cybenko and Hornik published a universal approximation theorem, which gave the criteria of a multilayered perceptron to approximate any continuous function. Apart from these well-known algorithms, the field of artificial neural networks is full of a variety of algorithms for various applications like classification, function approximation, and regression.

So we can define artificial neural networks as parallel, distributed information processing systems, representing a computational technology built on the analogy of the human information processing system. They are connected networks of processing elements called artificial neurons. They have the ability to learn, store, and make the information available for use. Hence, they play an important role in artificial intelligence and machine learning. In real-world problems, if data cannot be represented by mathematical equations, machine learning models like artificial neural networks can be used to establish the relationship between input-output patterns.

Multilayered perceptrons are one type of architecture of artificial neural networks used as supervised machine learning models. That is, the models for which input-output data pairs are given. They learn generalisation from the patterns presented to them in the form of data by changing their parameters, called weights and biases. There are various algorithms for updating weights and biases. The Error Back-Propagation algorithm, introduced by Rumelhart, Hinton, and Williams [62], is frequently used. It is based on an optimisation algorithm called the gradient descent method.

The architecture of a multilayered perceptron consists of an input layer, one or more hidden layers, and an output layer. Each layer can have one or more neurons. All the neurons of each layer are connected with all the neurons of the next layer by weights and biases. Each neuron in the hidden layer and the output layer consists of activation functions like logistic (sigmoid), hyperbolic tangent, identity, Rectified Linear Unit (ReLU), normalised exponential function, etc.

Let $\vec{x}$ contain the values of all the neurons from the previous layer, and $\vec{w}$ contains the values of weights. In our work, we have used the following activation functions:

1. Logistic function for hidden layer:

$$f(\vec{w} \cdot \vec{x}) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{x}}} \tag{2.24}$$

2. Rectified linear unit (ReLU) activation function [63] for hidden layer:

$$f(\vec{w} \cdot \vec{x}) = \max(0, \vec{w} \cdot \vec{x}) \tag{2.25}$$

3. Normalised exponential (softmax) activation function for output layer:

$$y_i = \frac{e^{w_i \cdot x_i}}{\sum_{j=1}^{L} e^{w_j \cdot x_j}}, 1 \leqslant i \leqslant L \tag{2.26}$$

   Here y_i is the i^{th} neuron in the output layer.

The output layer of the multilayered perceptron represents the predicted outputs computed by the artificial neural network. Since we also have the desired outputs in the data, these two outputs should be as close as possible. The error or loss between them is determined using loss functions. The following are commonly used loss functions:

1. **Mean squared error**: For regression problems, mean squared error loss function is used, given by:

$$E = \frac{1}{2} \sum_{i=1}^{L} (y_i - \hat{y}_i)^2 \tag{2.27}$$

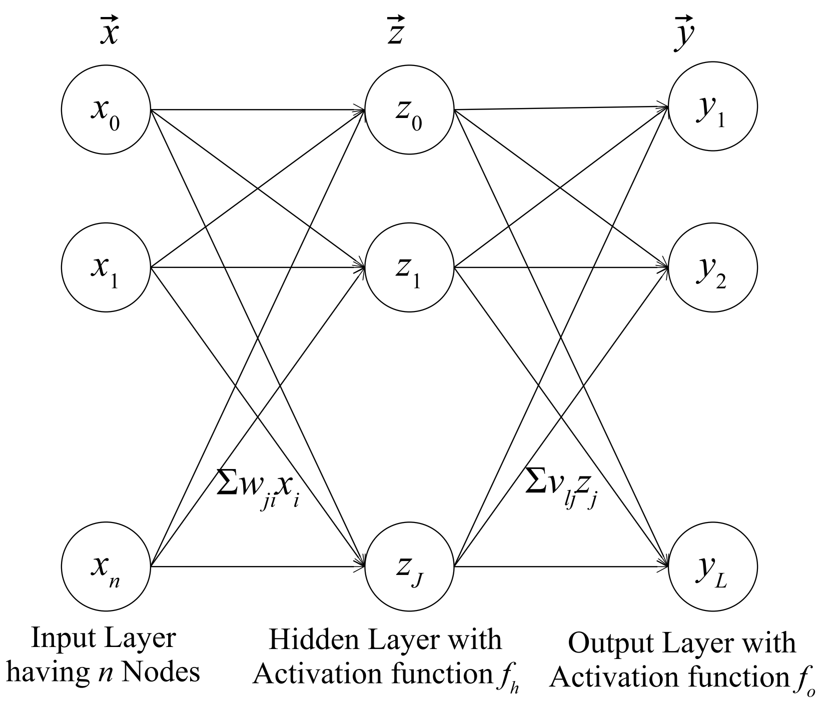

$\vec{x}$
$\vec{z}$
$\vec{y}$
x_0
x_1
x_n
z_0
z_1
z_J
y_1
y_2
y_L
$\Sigma w_{ji} x_i$
$\Sigma v_{lj} z_j$
Input Layer having n Nodes
Hidden Layer with Activation function f_h
Output Layer with Activation function f_o

Figure 2.12: Multilayered perceptron network architecture

Here, N is neurons in the output layer, y_i is the i^{th} desired output and $\hat{y}_i$ is the i^{th} predicted output.

2. **Cross entropy loss (log-loss)**: For classification problems, cross entropy loss (log loss) function is used, given by:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{ij} \cdot \log(p_{ij}) \tag{2.28}$$

Here, N is the neurons in the output layer, M is the number of classes and p_{ij} is determined from the normalised exponential function.

For fitting a model to data, normally it is divided into proportion 80%-20% or 70%-30%, training-testing data. During training, the weights and biases are changed in such a way, that the loss between the desired output and the predicted output obtained from the multilayered perceptron is minimised. The minimisation is achieved by taking gradients of loss with respect to the weights and biases.

Now, we will see how this minimisation is achieved. Consider a two-layered multilayered perceptron network architecture as shown in Figure 2.12 having an architecture $N^2_{n,J,L}$. Let $\vec{x} = (x_0, x_1, \ldots, x_n) \in \mathbb{R}^{n+1}$ be the input vector having bias $x_0 = -1$ and input neurons $x_1, x_2, \ldots x_n$ and let $\vec{z} = (z_0, z_1, \ldots, z_J) \in \mathbb{R}^{J+1}$ be the hidden layer, having bias $z_0 = -1$ and hidden neurons $z_1, z_2, \ldots z_J$. Suppose we denote the weights between neurons x_i and z_j by w_{ji} where $0 \leqslant i \leqslant n$ and $0 \leqslant j \leqslant J$. Let f_h be an activation function for a hidden layer. Then the values of the hidden neuron z_j is given by

$$z_j = f_h\left(\sum_{i=0}^{n} w_{ji}x_i\right) \tag{2.29}$$

Let $\vec{y} = (y_1, y_2, \ldots, y_L) \in \mathbb{R}^L$ be the output layer. Suppose we denote the weight connecting z_j and y_l by v_{lj} where $0 \leqslant j \leqslant J$ and $1 \leqslant l \leqslant L$. Let f_o be an activation function for an output layer. Then the values of the output neuron y_l is given by

$$y_l = f_o\left(\sum_{j=0}^{J} v_{lj}z_j\right) \tag{2.30}$$

Let $\vec{d} = (d_1, d_2, \ldots, d_L) \in \mathbb{R}^L$ be the desired output. Then the sum of squares of all the errors between $\vec{y}$ and $\vec{d}$ is given by

$$E = \frac{1}{2}\sum_{l=1}^{L}(d_l - y_l)^2 \tag{2.31}$$

According to the Delta rule, we can update the weights by considering the partial derivative of error with respect to the weights. So, for the output layer, the change in weights v_{lj} is given by

$$\begin{aligned}
\Delta v_{lj} &= \frac{\partial E}{\partial v_{lj}} \\
&= \frac{\partial}{\partial v_{lj}}\left[\frac{1}{2}\sum_{l=1}^{L}(d_l - y_l)^2\right] \\
&= \frac{1}{2}\sum_{l=1}^{L} 2\,(d_l - y_l)^{2-1}\frac{\partial}{\partial v_{lj}}(d_l - y_l) \\
&= \sum_{l=1}^{L}(d_l - y_l)^1\frac{\partial}{\partial v_{lj}}\left[d_l - f_o\left(\sum_{j=0}^{J} v_{lj}z_j\right)\right] \\
&= \sum_{l=1}^{L}(d_l - y_l)\left[0 - f_o'\left(\sum_{j=0}^{J} v_{lj}z_j\right)z_j\right] \\
&= -\sum_{l=1}^{L}(d_l - y_l)\,f_o'\left(\sum_{j=0}^{J} v_{lj}z_j\right)z_j
\end{aligned} \tag{2.32}$$

Similarly, the change in the weights w_{ji} between an input layer and a hidden layer is given by

$$\begin{aligned}
\Delta w_{ji} &= \frac{\partial E}{\partial w_{ji}} \\
&= \frac{\partial}{\partial w_{ji}}\left[\frac{1}{2}\sum_{l=1}^{L}(d_l - y_l)^2\right] \\
&= \frac{1}{2}\sum_{l=1}^{L} 2\,(d_l - y_l)^{2-1}\,\frac{\partial}{\partial w_{ji}}(d_l - y_l) \\
&= \sum_{l=1}^{L}(d_l - y_l)^1\,\frac{\partial}{\partial w_{ji}}\left[d_l - f_o\left(\sum_{j=0}^{J} v_{lj} z_j\right)\right] \\
&= \sum_{l=1}^{L}(d_l - y_l)\left[0 - f_o'\left(\sum_{j=0}^{J} v_{lj} z_j\right)\frac{\partial}{\partial w_{ji}}\left(\sum_{j=0}^{J} v_{lj} z_j\right)\right] \\
&= -\sum_{l=1}^{L}(d_l - y_l)\,f_o'\left(\sum_{j=0}^{J} v_{lj} z_j\right)\sum_{j=0}^{J} v_{lj}\frac{\partial}{\partial w_{ji}}(z_j) \\
&= -\sum_{l=1}^{L}(d_l - y_l)\,f_o'\left(\sum_{j=0}^{J} v_{lj} z_j\right)\sum_{j=0}^{J} v_{lj}\frac{\partial}{\partial w_{ji}}\left[f_h\left(\sum_{i=0}^{n} w_{ji} x_i\right)\right] \\
&= -\sum_{l=1}^{L}(d_l - y_l)\,f_o'\left(\sum_{j=0}^{J} v_{lj} z_j\right)\sum_{j=0}^{J} v_{lj} f_h'\left(\sum_{i=0}^{n} w_{ji} x_i\right) x_i \qquad (2.33)
\end{aligned}$$

Using equations (2.32) and (2.33), the updated weights at $(k+1)^{st}$ iteration are

$$v_{lj}^{(k+1)} = v_{lj}^{(k)} - \eta_o \Delta v_{lj}^{(k)} \qquad (2.34)$$

$$w_{ji}^{(k+1)} = w_{ji}^{(k)} - \eta_h \Delta w_{ji}^{(k)} \qquad (2.35)$$

In equations (2.34) and (2.35), η_h and η_o are learning rates usually having values between 0.0001 and 2. The neurons calculations are done in a forward manner. But weights and biases updates are done in a backward manner using equations (2.34) and (2.35). Therefore, this algorithm is more commonly known as the error back-propagation algorithm.

Several modifications to this algorithm have occurred over the years. Recently, the Adam algorithm has been widely used to update weights, which is an adaptive momentum algorithm [64]. In this algorithm, the momentum is updated iteratively, and based on that, the weights are updated. In 2015, this algorithm was introduced using equation (2.36) [65].

$$\Delta w = -\frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t \qquad (2.36)$$

Here,

$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$ is updated momentum at step t,

$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla w_t$ is momentum at step t,
$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$ is used to keep history of gradients,
$v_t = \beta_2 v_{t-1} + (1 - \beta_2) (\nabla w_t)^2$,
η is a learning rate,
ϵ is a small constant,
β_1 and β_2 are constants.

To train a multilayered perceptron in our work, we have employed both methods discussed here. Now, we will see another type of artificial neural network named radial basis function networks.

2.7 Radial Basis Function Networks

The radial basis function network is one kind of artificial neural network, proposed by Broomhead and Lowe in 1988 [66], in which specific functions known as radial basis functions are used. It was shown that it has equivalent capabilities as multi-layered perceptrons [67]. It was mainly designed for function approximation and regression problems, but it can be used for classification problems as well. One more advantage of a radial basis function network is that it has fewer complex calculations and fewer function evaluations, and hence it takes less training time as compared to the training done in a multilayered perceptron using the back-propagation algorithm. The main difference between a multilayered perceptron and a radial basis function network is that, unlike multilayered perceptrons, there is only one fixed hidden layer in the radial basis function network. The other difference is that the radial basis functions are used as activation functions in the hidden layer of the radial basis function network.

The radial basis function network, having an architecture of $N^2_{J_1,J_2,J_3}$, is shown in Figure 2.13. It consists of an input layer having J_1 neurons, an output layer having J_3 and one hidden layer having J_2 neurons. In general, the number of hidden neurons is less than or equal to the number of training examples [68]. The hidden layer consists of a radial basis function of the form $\phi(r)$, where r is the distance between an input pattern and a centre. So each hidden neuron is determined using $\phi_i(\vec{x}) = \phi(\vec{x} - \vec{c}_i)$, where $\vec{c}_i$ is a centre corresponding to the i^{th} neuron in a hidden layer [67]. There are many radial basis functions that can be used in the radial basis function network. The most widely used radial basis function used, is the Gaussian radial basis function, defined by equation (2.37).

$$\phi(r) = e^{\frac{-r^2}{2\sigma^2}} \tag{2.37}$$

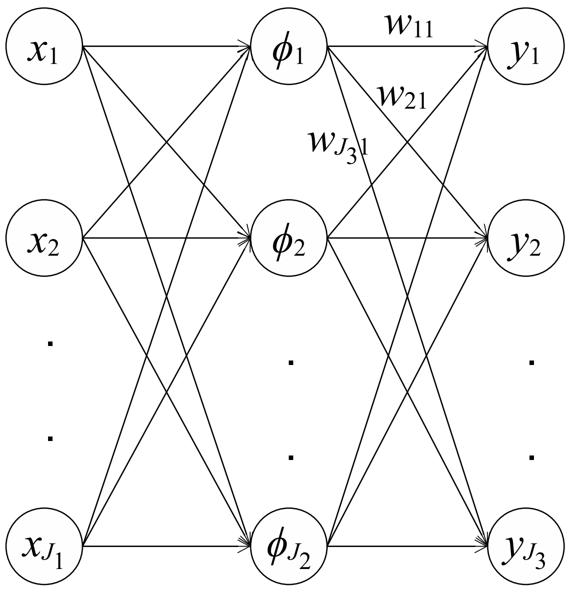


Figure 2.13: Radial basis function network architecture

Just like the multilayered perceptrons, here in the radial basis function network, the connections between the hidden and output layers are assigned weights. These weights and centres are updated to establish relationships between inputs and outputs.

Let $\{(\vec{x}_p, \vec{y}_p)\,|p = 1, 2, \ldots, N\}$, be given data where $\vec{x}_p$ contains inputs and $\vec{y}_p$ contains desired outputs. Let $W = [\vec{w}_1, \vec{w}_2, \ldots, \vec{w}_{J_3}]$ be a $J_2 \times J_3$ weight matrix. Each entry in this weight matrix is a column vector given by $\vec{w}_i = (w_{1i}, w_{2i}, \ldots, w_{J_2 i})^T$. The forward calculations in the radial basis function network are given by equation (2.38).

$$\hat{\vec{y}_i}(\vec{x}) = \sum_{k=1}^{J_2} w_{ki}\phi(||\vec{x} - \vec{c_k}||) \tag{2.38}$$

The calculations of equation (2.38) can be easily implemented using a matrix multiplication as defined in the equation (2.39).

$$Y = W^T \Phi \tag{2.39}$$

In the equation (2.39), W is $J_2 \times J_3$ weight matrix and Φ is $J_2 \times N$ matrix defined by equation (2.40).

$$\Phi = [\vec{\phi}_1, \vec{\phi}_2, \ldots, \vec{\phi}_N] \tag{2.40}$$

In this matrix defined in the equation (2.40), each entry $\vec{\phi}_p$ is column vector defined by $\vec{\phi}_p = (\phi_{p1}, \phi_{p2}, \ldots, \phi_{pj_2})^T$. The result of the matrix multiplication defined in equation (2.39) is the $J_3 \times N$ output matrix Y defined by equation (2.41.)

$$Y = [\vec{y}_1, \vec{y}_1, \ldots, \vec{y}_{J_3}] \tag{2.41}$$

In the output matrix defined in the equation (2.41), each entry $\vec{y}_p$ is column vector defined by $\vec{y}_p = (y_{p1}, y_{p2}, \ldots, y_{pj_2})^T$. The error between desired output and output obtained by the network is given by

$$E = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}^2 \tag{2.42}$$

Here, e_{ni} is the error at i^{th} output neuron for the n^{th} pattern

$$e_{ni} = y_{ni} - \sum_{m=1}^{J_2} w_{mi}\phi\left(\|\vec{x}_n - \vec{c}_m\|\right) \tag{2.43}$$

We can decrease this error by using the gradient descent method. Here we have to minimise the error by optimising weights and centres. Hence, we consider the partial derivatives of error E with respect to weights and centres.
Differentiating E partially with respect to weights, we get

$$\begin{aligned}
\Delta w_{mi} &= \frac{\partial E}{\partial w_{mi}} \\
&= \frac{\partial}{\partial w_{mi}}\left[\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}^2\right] \\
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} 2e_{ni}\frac{\partial}{\partial w_{mi}}(e_{ni}) \\
&= \frac{2}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}\frac{\partial}{\partial w_{mi}}\left(y_{ni} - \sum_{m=1}^{J_2} w_{mi}\phi\left(||\vec{x_n} - \vec{c_m}||\right)\right) \\
&= \frac{2}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}\left(0 - \sum_{m=1}^{J_2}\phi\left(||\vec{x_n} - \vec{c_m}||\right)\right) \\
&= -\frac{2}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}\sum_{m=1}^{J_2}\phi\left(||\vec{x_n} - \vec{c_m}||\right)
\end{aligned} \tag{2.44}$$

Similarly, differentiating E partially with respect to centres, we get

$$\begin{aligned}
\Delta \vec{c_m} &= \frac{\partial E}{\partial \vec{c_m}} \\
&= \frac{\partial}{\partial \vec{c_m}}\left[\frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}^2\right]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} 2e_{ni}\frac{\partial}{\partial \vec{c_m}}\left(e_{ni}\right) \\
&= \frac{2}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}\frac{\partial}{\partial \vec{c_m}}\left(y_{ni} - \sum_{m=1}^{J_2} w_{mi}\phi\left(\|\vec{x}_n - \vec{c}_m\|\right)\right) \\
&= \frac{2}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}\left(0 - \sum_{m=1}^{J_2} w_{mi}\frac{\partial}{\partial \vec{c_m}}\phi\left(\|\vec{x}_n - \vec{c}_m\|\right)\right) \\
&= -\frac{2}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}\sum_{m=1}^{J_2} w_{mi}\phi'\left(\|\vec{x}_n - \vec{c}_m\|\right)\frac{\partial}{\partial \vec{c_m}}\|\vec{x}_n - \vec{c}_m\| \\
&= -\frac{2}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}\sum_{m=1}^{J_2} w_{mi}\phi'\left(\|\vec{x}_n - \vec{c}_m\|\right)\frac{\vec{x}_n - \vec{c}_m}{||\vec{x}_n - \vec{c}_m||}(-1) \\
&= \frac{2}{N}\sum_{n=1}^{N}\sum_{i=1}^{J_3} e_{ni}\sum_{m=1}^{J_2} w_{mi}\phi'\left(\|\vec{x}_n - \vec{c}_m\|\right)\frac{\vec{x}_n - \vec{c}_m}{||\vec{x}_n - \vec{c}_m||} \qquad (2.45)
\end{aligned}
$$

Using equations (2.44) and (2.45), the updated weights and centres at $(k+1)^{st}$ iteration are

$$w_{mi}^{k+1} = w_{mi}^{k} - \eta_1 \Delta w_{mi}^{k} \qquad (2.46)$$

$$\vec{c_m}^{\,k+1} = \vec{c_m}^{\,k} - \eta_2 \Delta \vec{c_m}^{\,k} \qquad (2.47)$$

In equations (2.46) and (2.47), η_1 and η_2 are learning rates, usually having values between 0.0001 and 2. Such a trained radial basis function network has good capabilities for universal approximation [69].

In the next section, we will discuss a method named the hidden Markov model. This was the commonly used technique initially in automatic speech recognition. We have also applied this method to our work.

2.8 Hidden Markov Model

Hidden Markov models are probabilistic models developed by a Russian mathematician named Markov, who was best known for his work on stochastic processes. A stochastic process is a process defined as a family of random variables. A Markov process is a stochastic process satisfying the Markov property. It is the property of a process in which the future is independent of the past, given the present.

Let $\{X(t), t \geqslant 0\}$ be a stochastic process which assumes non-negative integer values. The process is called Markov process if for every $n \geqslant 0$, time steps $0 \leqslant t_0 < t_1 < t_2 < \cdots < t_n < t_{n+1}$ and the states $S_0, S_1, S_2, \ldots, S_{n+1}$, it holds

$$P\big[X(t_{n+1}) = S_{n+1} \mid X(t_n) = S_n, \ldots, X(t_0) = S_0\big] = P\big[X(t_{n+1}) = S_{n+1} \mid X(t_n) = S_n\big]$$

A Markov model is a stochastic model used for modelling a randomly changing system in which the future state depends only on the present state and not on the past state. In Markov models, states are visible, and the parameters of the model are state transition probabilities.

The hidden Markov model is an extension of the Markov model for processes in which states are hidden but the output produced by the hidden state is visible. The parameters of hidden Markov models are state transition probabilities, observation probabilities, and initial state probabilities. Hidden Markov models were introduced by Baum and Petrie in 1966 [70]. Hidden Markov models have many applications in computational biology, speech recognition, handwriting recognition [71], gesture recognition, bioinformatics, etc.

There are three basic problems with the hidden Markov model: [72].

1. Evaluation Problem.
2. Determination of optimal hidden state.
3. Determination of optimal model parameters.

The solutions to these three problems are given by the following three algorithms, respectively: [72].

1. Forward and Backward algorithm
2. Viterbi algorithm
3. Baum - Welch algorithm

To understand the steps of these algorithms, consider the following nomenclature:

N = Number of hidden states

M = Number of observations

$Q = \{q_1, q_2, \ldots, q_N\}$: Hidden state sequence

$V = \{v_1, v_2, \ldots, v_M\}$: Various classes of observations

$a_{ij} = P(q_j \text{ at } t+1 \mid q_i \text{ at } t);\ 1 \leqslant i, j \leqslant N$: State transition probabilities

$A = [a_{ij}]$: State transition matrix

$b_j(k) = P(v_k \text{ at } t \mid q_j \text{ at } t+1); 1 \leqslant j \leqslant N, 1 \leqslant k \leqslant M$: Observation probabilities / emission probabilities

$B = [b_j(k)]$: Observation probabilities (emission probability) distribution

$\pi_i = P(q_i \text{ at } t = 1); 1 \leqslant i \leqslant N$: Initial state probabilities

$\Pi = [\pi_i]$: Initial state probability distribution

A hidden markov model is completely specified by $\lambda = (A, B, \Pi)$. Now we will the description of above mentioned three problems and their respective algorithm.

2.8.1 Evaluation Problem and Forward/Backward Algorithm

Consider a model $\lambda = (A, B, \Pi)$ and an observation sequence O given by $\{O_1, O_2, \ldots, O_T\}$ of length T. Here the problem is to calculate efficiently $P(O)$ [72]. Consider a hidden state sequence $Q = \{q_1, q_2, \ldots, q_T\}$ producing the observation O. Then the conditional probability of observing a sequence O given the hidden sequence Q, $P(O|Q)$ can be derived in the following way [71].

$$\begin{aligned} P(O \mid Q) &= P(O_1, O_2, \ldots, O_T \mid q_1, q_2, \ldots, q_T) \\ &= P(O_1 \mid q_1) P(O_2 \mid q_2) \cdots P(O_T \mid q_T) \\ &= b_{q_1}(O_1) b_{q_2}(O_2) \cdots b_{q_T}(O_T) \end{aligned}$$

$$\therefore P(O \mid Q) = b_{q_1}(O_1) b_{q_2}(O_2) \ldots b_{q_T}(O_T) \tag{2.48}$$

The probability of occurrence of a hidden state sequence $Q = \{q_1, q_2, \ldots, q_T\}$ is derived in the following way [71].

$$\begin{aligned} P(Q) &= P(q_1, q_2, \ldots, q_T) \\ &= P(q_1) P(q_2 \mid q_1) P(q_3 \mid q_2) \ldots P(q_T \mid q_{T-1}) \\ &= \pi_{q_1} P(q_2 \mid q_1) P(q_3 \mid q_2) \ldots P(q_T \mid q_{T-1}) \\ &= \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \ldots a_{q_{T-1} q_T} \end{aligned}$$

$$\therefore P(Q) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \ldots a_{q_{T-1} q_T} \tag{2.49}$$

From this, we can derive the joint probability of occurrence of observation sequence O and hidden state sequence Q in the following way [71].

$$\begin{aligned} P(O,Q) &= P(O \mid Q)P(Q) \\ &= b_{q_1}(O_1)b_{q_2}(O_2)\dots b_{q_T}(O_T)\pi_{q_1}a_{q_1q_2}a_{q_2q_3}\dots a_{q_{T-1}q_T} \\ &= \pi_{q_1}b_{q_1}(O_1)a_{q_1q_2}b_{q_2}(O_2)\dots a_{q_{T-1}q_T}b_{q_T}(O_T) \end{aligned}$$

$$\therefore P(O,Q) = \pi_{q_1}b_{q_1}(O_1)a_{q_1q_2}b_{q_2}(O_2)\dots a_{q_{T-1}q_T}b_{q_T}(O_T) \tag{2.50}$$

Equation (2.50) is derived only for a particular state sequence $Q = \{q_1, q_2, \dots, q_T\}$. Summing equation (2.50) over all the possible state sequences of length T, we can obtain $P(O)$ given a model $\lambda = (A, B, \Pi)$ as equation (2.51).

$$P(O) = \sum_{q_1,q_2,\dots,q_T} \pi_{q_1}b_{q_1}(O_1)a_{q_1q_2}b_{q_2}(O_2)\dots a_{q_{T-1}q_T}b_{q_T(O_T)} \tag{2.51}$$

In practice, the calculation of equation (2.51) is very complex. It can be determined efficiently using the Forward or Backward algorithm which is explained here [72].
Consider a variable $\alpha_t(i)$ defined by equation (2.52).

$$\alpha_t(i) = P(O_1, O_2, \dots O_t \mid q_t = S_i) \tag{2.52}$$

In equation (2.52), $1 \leqslant t \leqslant T$ and $1 \leqslant i \leqslant N$. It represents the probability of partial observation sequence $\{O_1, O_2, \dots, O_t\}$ until time t and state S_i is selected at time t. The calculation of $P(O)$ using the forward variable $\alpha_t(i)$ is done using Forward-Algorithm given by following steps [71].

Step 1: For $1 \leqslant i \leqslant N$

$$\alpha_1(i) = \pi_i b_i(O_1)$$

Step 2: For $1 \leqslant t \leqslant T-1$ and $1 \leqslant j \leqslant N$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(O_{t+1})$$

Step 3:

$$P(O) = \sum_{i=1}^{N} \alpha_T(i)$$

These calculations can be visualised using the Figure 2.14 [71]. Let us try to understand this with an example. Consider a two state problem in which model parameters are given as follows:

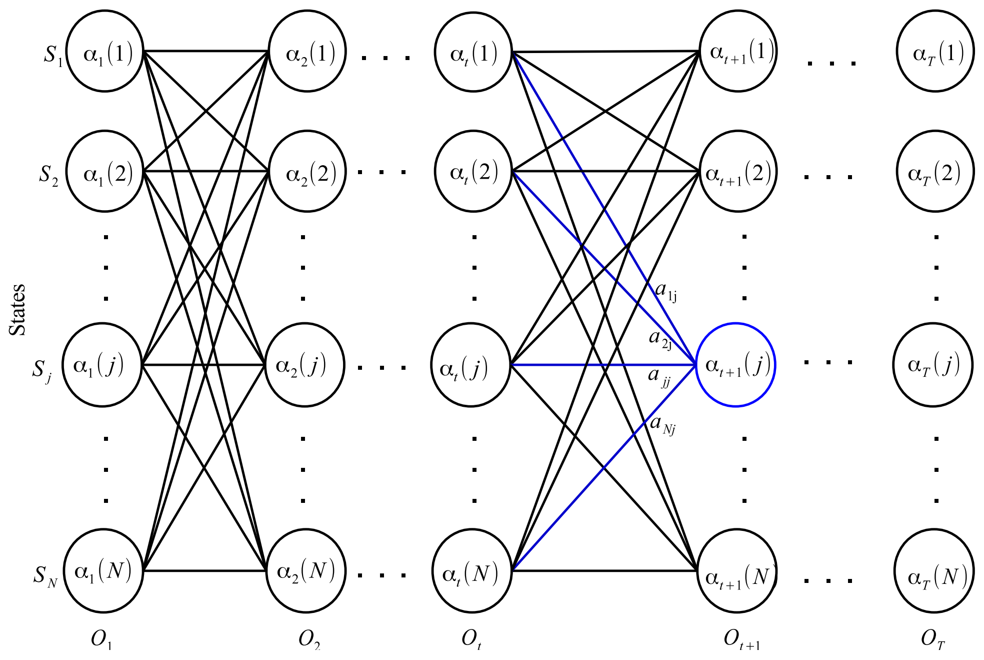


Figure 2.14: Calculations of Forward Algorithm

N = Number of States = 2

T = Number of Observations = 3

$$\Pi = \begin{bmatrix}\pi_1 & \pi_2\end{bmatrix} = \begin{bmatrix}1.0 & 0.0\end{bmatrix}$$

$$A = \begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix} = \begin{bmatrix}0.7 & 0.3\\ 0.5 & 0.5\end{bmatrix}$$

$$B = \begin{bmatrix}b_1(1) & b_1(2) & b_1(3)\\ b_2(1) & b_2(2) & b_2(3)\end{bmatrix} = \begin{bmatrix}0.6 & 0.1 & 0.3\\ 0.1 & 0.7 & 0.2\end{bmatrix}$$

$$O = \begin{bmatrix}O_1 & O_2 & O_3\end{bmatrix} = \begin{bmatrix}3 & 2 & 1\end{bmatrix}$$

Step 1: $\alpha_1(i) = \pi_i b_i(O_1); \qquad 1 \leqslant i \leqslant 2$

$$\alpha_1(1) = \pi_1 b_1(O_1) = \pi_1 b_1(3) = (1)(0.3) = 0.3$$

$$\alpha_1(2) = \pi_2 b_2(O_1) = \pi_2 b_2(3) = (0)(0.2) = 0.0$$

Step 2: For $1 \leqslant i \leqslant 2$ and $1 \leqslant t \leqslant 3$

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{2} \alpha_t(i) a_{ij}\right] b_j(O_{t+1})$$

$$\therefore \alpha_{t+1}(j) = \Big[\alpha_t(1)a_{1j} + \alpha_t(2)a_{2j}\Big] b_j(O_{t+1}) \tag{2.53}$$

Substitute $t = 1$ in equation (2.53)

$$\therefore \alpha_2(j) = \Big[\alpha_1(1)a_{1j} + \alpha_1(2)a_{2j}\Big] b_j(O_2)$$

$$\therefore \alpha_2(j) = \Big[(0.3)a_{1j} + (0.0)a_{2j}\Big] b_j(2)$$

Substitute $j = 1$ in $\alpha_2(j)$

$$\begin{aligned}\therefore \alpha_2(1) &= \Big[(0.3)a_{11} + (0.0)a_{21}\Big] b_1(2)\\ &= \Big[(0.3)(0.7) + (0.0)(0.5)\Big](0.1)\\ \therefore \alpha_2(1) &= 0.021\end{aligned}$$

Substitute $j = 2$ in $\alpha_2(j)$

$$\begin{aligned}\therefore \alpha_2(2) &= \Big[(0.3)a_{12} + (0.0)a_{22}\Big] b_2(2)\\ &= \Big[(0.3)(0.3) + (0.0)(0.5)\Big](0.7)\\ \therefore \alpha_2(2) &= 0.063\end{aligned}$$

Substitute $t = 2$ in equation (2.53)

$$\therefore \alpha_3(j) = \Big[\alpha_2(1)a_{1j} + \alpha_2(2)a_{2j}\Big] b_j(O_3)$$

$$\therefore \alpha_3(j) = \Big[(0.021)a_{1j} + (0.063)a_{2j}\Big] b_j(1)$$

Substitute $j = 1$ in $\alpha_3(j)$

$$\begin{aligned}\therefore \alpha_3(1) &= \Big[(0.021)a_{11} + (0.063)a_{21}\Big] b_1(1)\\ &= \Big[(0.021)(0.7) + (0.063)(0.5)\Big](0.6)\\ \therefore \alpha_3(1) &= 0.02772\end{aligned}$$

Substitute $j = 2$ in $\alpha_3(j)$

$$\therefore \alpha_3(2) = \Big[(0.021)a_{12} + (0.063)a_{22}\Big] b_2(1)$$

$$\begin{aligned} &= \Big[(0.021)(0.3) + (0.063)(0.5)\Big](0.1) \\ \therefore \alpha_3(1) &= 0.00378 \end{aligned}$$

Hence, values of $\alpha_t(j)$ are $\alpha_1(1) = 0.3$, $\alpha_1(2) = 0.0$, $\alpha_2(1) = 0.021$, $\alpha_2(2) = 0.063$, $\alpha_3(1) = 0.02772$, $\alpha_3(2) = 0.00378$.

From this, the probability of observing a sequence O sequence is

$$\begin{aligned} P(O) &= \sum_{i=1}^{N} \alpha_T(i) \\ &= \sum_{i=1}^{2} \alpha_3(i) \\ &= \alpha_3(1) + \alpha_3(2) \\ &= 0.02772 + 0.00378 \\ &= 0.0315 \end{aligned}$$

Hence, the probability of observing a sequence $O = [O_1, O_2, O_3] = [3, 2, 1]$ is

$$P(O) = 0.0315$$

Now, we will discuss the backward algorithm. Unlike the forward algorithm, here the calculations are done in a backward way using a backward variable.

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \dots O_T \mid q_t = S_i) \tag{2.54}$$

Here, $1 \leqslant t \leqslant T$ and $1 \leqslant i \leqslant N$. It is probability of partial observation sequence $\{O_{t+1}, O_{t+2}, \dots O_T\}$ from time step $t + 1$ upto end given that at time t, state is S_i. The steps of Backward algorithm are as follows: [71]

Step 1: For $1 \leqslant i \leqslant N$

$$\beta_T(i) = 1$$

Step 2: For $t = T - 1, T - 2, \dots, 1$ and $1 \leqslant j \leqslant N$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} \beta_{t+1}(j) b_j(O_{t+1})$$

These calculations can be visualised by Figure 2.15 [71]. Let us try to understand this algorithm with the same example as discussed above for the forward algorithm.

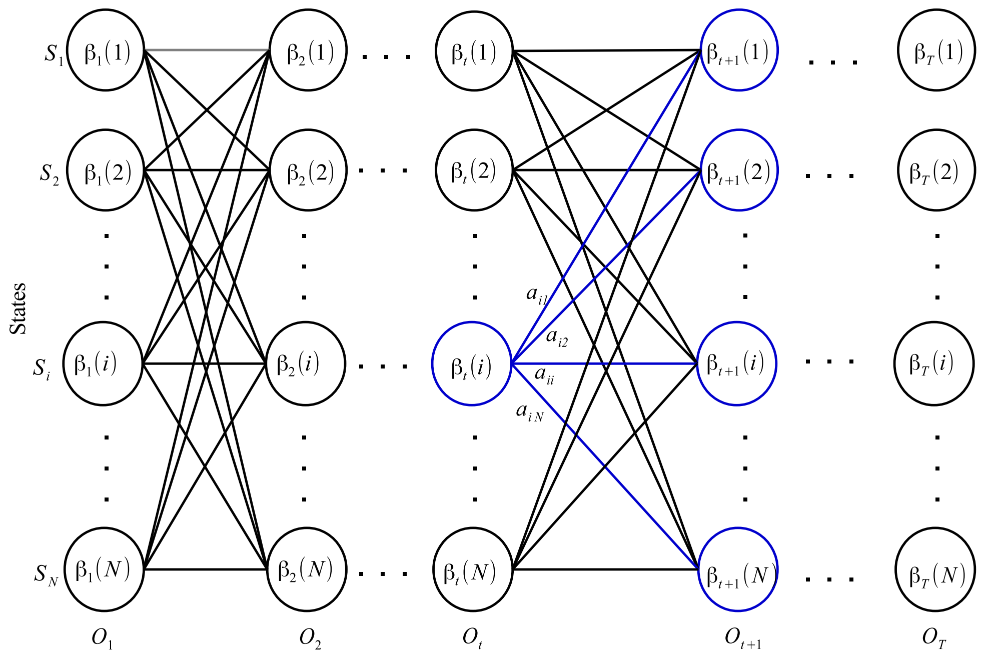


Figure 2.15: Calculations of Backward Algorithm

For $1 \leqslant i \leqslant 2$ and $T = 3$, $\beta_T(i) = 1$

$$\beta_3(1) = 1.0$$

$$\beta_3(2) = 1.0$$

For $t = 2$ and $t = 1$ and $1 \leqslant j \leqslant 2$

$$\beta_t(i) = \sum_{j=1}^{2} a_{ij}\beta_{t+1}(j)b_j(O_{t+1})$$

$$\beta_t(i) = a_{i1}\beta_{t+1}(1)b_1(O_{t+1}) + a_{i2}\beta_{t+1}(2)b_2(O_{t+1}) \tag{2.55}$$

Substitute $t = 2$ in equation (2.55),

$$\beta_2(i) = a_{i1}\beta_3(1)b_1(O_3) + a_{i2}\beta_3(2)b_2(O_3)$$

Substitute $i = 1$ in $\beta_2(i)$,

$$\begin{aligned}\beta_2(1) &= a_{11}\beta_3(1)b_1(1) + a_{12}\beta_3(2)b_2(1)\\ &= (0.7)(1)(0.6) + (0.3)(1)(0.1)\\ &= 0.45\end{aligned}$$

Substitute $i = 2$ in (2.55),

$$\begin{aligned}\beta_2(2) &= a_{21}\beta_3(1)b_1(1) + a_{22}\beta_3(2)b_2(1)\\ &= (0.5)(1)(0.6) + (0.5)(1)(0.1)\\ &= 0.35\end{aligned}$$

Substitute $t = 1$ in equation(2.55),

$$\beta_1(i) = a_{i1}\beta_2(1)b_1(O_2) + a_{i2}\beta_2(2)b_2(O_2)$$

Substitute $i = 1$ in $\beta_1(i)$,

$$\begin{aligned}\beta_1(1) &= a_{11}\beta_2(1)b_1(2) + a_{12}\beta_2(2)b_2(2)\\ &= (0.7)(0.45)(0.1) + (0.3)(0.35)(0.7)\\ &= 0.105\end{aligned}$$

Substitute $i = 2$ in $\beta_1(i)$,

$$\begin{aligned}\beta_1(2) &= a_{21}\beta_2(1)b_1(2) + a_{22}\beta_2(2)b_2(2)\\ &= (0.5)(0.45)(0.1) + (0.5)(0.35)(0.7)\\ &= 0.145\end{aligned}$$

Hence, the values of $\beta_t(i)$ are $\beta_1(1) = 0.105$, $\beta_1(2) = 0.145$, $\beta_2(1) = 0.45$, $\beta_2(2) = 0.35$, $\beta_3(1) = 1.0$, $\beta_3(2) = 1.0$.

The calculations done above for forward variables using the forward algorithm and backward variables using the backward algorithm are summarised in the Figure 2.16. These values are useful in the Viterbi algorithm, which is discussed in the next section.

2.8.2 Determination of Optimal Hidden State using Viterbi Algorithm

The next problem is to determine an optimum hidden state sequence Q that gives the maximum probability, $P(O)$, of producing an observation sequence O [72].
For this, consider a variable

$$\gamma_t(i) = P(q_t = S_i \mid O) \tag{2.56}$$

Here, $1 \leqslant t \leqslant T$ and $1 \leqslant i \leqslant N$. It is the probability of being in state S_i at time t given the observation sequence O. Here, model $\lambda = (A, B, \Pi)$ is known. The equation (2.56)

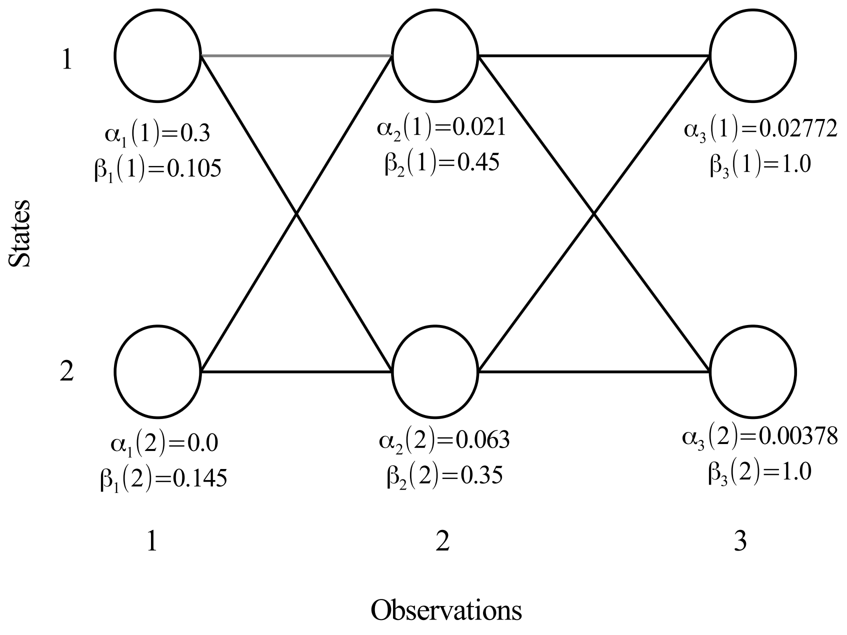


Figure 2.16: Summary of values of forward and backward variables

can be represented in terms of forward and backward variables as equation (2.57) [71].

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)} \tag{2.57}$$

Using the equation (2.57), the individually most likely state q_t at time t is given by equation (2.58).

$$q_t = \underset{1 \leqslant i \leqslant N}{\operatorname{argmax}} \left[\gamma_t(i)\right] \tag{2.58}$$

To understand the calculations of $\gamma_t(i)$, we consider the same example that was discussed for the forward algorithm.
From the equation (2.57),

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{2} \alpha_t(j)\beta_t(j)} = \frac{\alpha_t(i)\beta_t(i)}{\alpha_t(1)\beta_t(1) + \alpha_t(2)\beta_t(2)} \tag{2.59}$$

Substitute $t = 1$ and $i = 1$ in equation (2.59), we get

$$\begin{aligned} \gamma_1(1) &= \frac{\alpha_1(1)\beta_1(1)}{\alpha_1(1)\beta_1(1) + \alpha_1(2)\beta_1(2)} \\ &= \frac{(0.3)(0.105)}{(0.3)(0.105) + (0.0)(0.145)} \\ &= \frac{0.0315}{0.0315} \\ &= 1 \end{aligned}$$

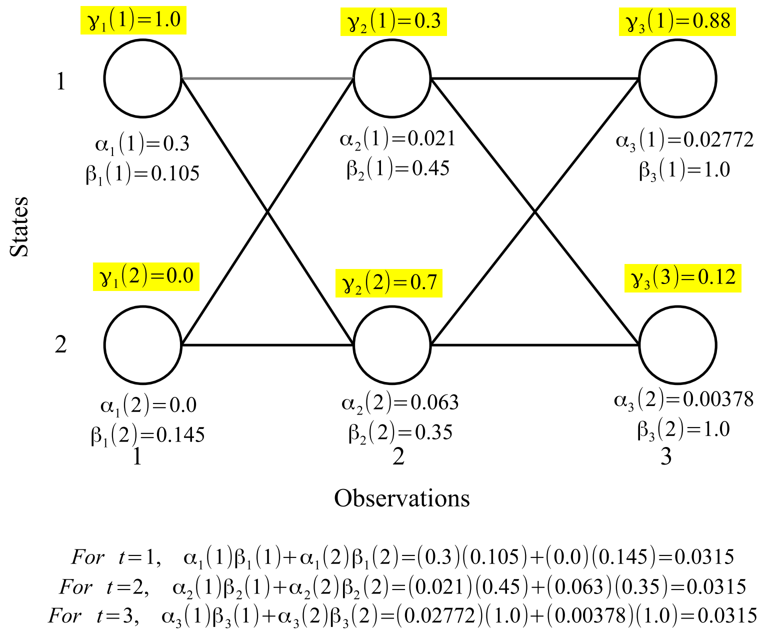


Figure 2.17: Summary of calculated values of $\gamma_t(i)$

Similarly, other values of $\gamma_t(i)$ can be determined.
Substituting $t = 1$ in the equation (2.59),
for $i = 1$, we get $\gamma_1(1) = 1.0$
for $i = 2$, we get $\gamma_1(2) = 0.0$

Substituting $t = 2$ in the equation (2.59),
for $i = 1$, we get $\gamma_2(1) = 0.3$
for $i = 2$, we get $\gamma_2(2) = 0.7$

Substituting $t = 3$ in the equation (2.59),
for $i = 1$, we get $\gamma_3(1) = 0.88$
for $i = 2$, we get $\gamma_3(2) = 0.12$

These values are shown in the Figure 2.17, together with the forward and backward variables.
Now, from the equation (2.58), the individually most likely state is given by

$$q_t = \underset{1 \leqslant i \leqslant 2}{\operatorname{argmax}} \left[\gamma_t(i)\right]$$

For $t = 1$,

$q_1 = \text{argmax}\big[\gamma_1(1), \gamma_1(2)\big] = \text{argmax}\big[1.0, 0.0\big] = 1$

For $t = 2$,
$q_2 = \text{argmax}\big[\gamma_2(1), \gamma_2(2)\big] = \text{argmax}\big[0.3, 0.7\big] = 2$

For $t = 3$,
$q_3 = \text{argmax}\big[\gamma_3(1), \gamma_3(2)\big] = \text{argmax}\big[0.88, 0.12\big] = 1$

Hence, the optimal sequence of hidden states is $Q = \{q_1, q_2, q_3\} = \{1, 2, 1\}$. This way, we can determine the most likely state each time. However, we cannot determine the state transitions accurately from this [72]. For the determination of accurate state transitions, we have followed the Viterbi algorithm.
To understand the steps of the Viterbi algorithm, consider a variable $\delta_t(i)$ for $1 \leqslant t \leqslant T$ and $1 \leqslant i \leqslant N$ as given in the equation (2.60).

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_T} P\big[q_1, q_2, \ldots, q_T = S_i \mid o_1, o_2, \ldots, o_T\big] \tag{2.60}$$

It is the highest probability along a single path at time t, using the observations up to time t and ends in state S_i [71]. By induction, for $2 \leqslant t \leqslant T$ and $1 \leqslant j \leqslant N$, we get equation (2.61).

$$\delta_{t+1}(j) = \left[\max_{1 \leqslant i \leqslant N} \left\{\delta_t(i) a_{ij}\right\}\right] b_j(O_{t+1}) \tag{2.61}$$

To keep track of the argument which maximises $\delta_t(i)$, consider another variable $\psi_t(i)$ for $2 \leqslant t \leqslant T$ and $1 \leqslant j \leqslant N$ as defined in the equation (2.62).

$$\psi_t(i) = \underset{1 \leqslant i \leqslant N}{\text{argmax}} \big[\delta_t(i) a_{ij}\big] \tag{2.62}$$

The steps of Viterbi algorithm are as follows [71]:
Step 1: For $1 \leqslant i \leqslant N$

$$\delta_1(i) = \pi_i b_i(O_1) \qquad\qquad \psi_1(i) = 1$$

Step 2: For $2 \leqslant t \leqslant T$ and $1 \leqslant j \leqslant N$

$$\delta_t(j) = \left[\max_{1 \leqslant i \leqslant N} \left\{\delta_{t-1}(i) a_{ij}\right\}\right] b_j(O_t)$$

$$\psi_t(j) = \underset{1 \leqslant i \leqslant N}{\text{argmax}} \left[\delta_{t-1}(i) a_{ij}\right]$$

Step 3:

$$p^* = \max_{1 \leqslant i \leqslant N} \left[\delta_T(i)\right] \qquad {q_T}^* = \operatorname*{argmax}_{1 \leqslant i \leqslant N} \left[\delta_T(i)\right]$$

Step 4: For $t = T-1, T-2, \ldots, 1$

$$ {q_t}^* = \psi_{t+1}(q^*_{t+1})$$

Let us try to understand the calculations of these steps using the same example that was considered for the forward and backward algorithms.
From step 1 of Viterbi algorithm, for $1 \leqslant i \leqslant 2$,

$$\delta_1(i) = \pi_i b_i(O_1)$$

$$\psi_1(i) = 1$$

Here, for $i = 1$,

$$\psi_1(1) = 1$$

$$\begin{aligned} \delta_1(1) &= \pi_1 b_1(3) \\ &= (1)(0.3) \\ &= 0.3 \end{aligned}$$

and for $i = 2$,

$$\psi_1(2) = 1$$

$$\begin{aligned} \delta_1(2) &= \pi_2 b_1(3) \\ &= (0)(0.2) \\ &= 0 \end{aligned}$$

Then from the step 2 of the Viterbi algorithm, for $2 \leqslant t \leqslant 3$ and $1 \leqslant j \leqslant 2$

$$\delta_t(j) = \left[\max_{1 \leqslant i \leqslant 2} \left\{\delta_{t-1}(i) a_{ij}\right\}\right] b_j(O_t)$$

$$\psi_t(j) = \operatorname*{argmax}_{1 \leqslant i \leqslant 2} \left[\delta_{t-1}(i) a_{ij}\right]$$

Here, for $t = 2$

$$\delta_2(j) = \left[\max\left\{\delta_1(1) a_{1j}, \delta_1(2) a_{2j}\right\}\right] b_j(O_2)$$

$$\psi_2(j) = \text{argmax}\left[\delta_1(1)a_{1j}, \delta_1(2)a_{2j}\right]$$

Substituting $j = 1$ in above equations, we get

$$\begin{aligned}\delta_2(1) &= \big[\max\big\{\delta_1(1)a_{11}, \delta_1(2)a_{21}\big\}\big]b_1(2)\\ &= \big[\max\big\{(0.3)(0.7), (0)(0.5)\big\}\big](0.1)\\ &= \big[\max\big\{0.21, 0\big\}\big](0.1)\\ &= 0.021\end{aligned}$$

$$\begin{aligned}\psi_2(1) &= \text{argmax}\big[\delta_1(1)a_{11}, \delta_1(2)a_{21}\big]\\ &= \text{argmax}\big[(0.3)(0.7), (0)(0.5)\big]\\ &= \text{argmax}\big[0.21, 0\big]\\ &= 1\end{aligned}$$

Similarly, we can find other values of $\delta_t(i)$ and $\psi_t(i)$. All the values are summarised below and in the Figure 2.18.

For $t = 1$,

$i = 1$,	$\delta_1(1) = 0.3$	$\psi_1(1) = 1$
$i = 2$,	$\delta_1(2) = 0.0$	$\psi_1(2) = 1$

For $t = 2$,

$i = 1$,	$\delta_2(1) = 0.021$	$\psi_2(1) = 1$
$i = 2$,	$\delta_2(2) = 0.063$	$\psi_2(2) = 1$

For $t = 3$,

$i = 1$,	$\delta_3(1) = 0.0189$	$\psi_3(1) = 2$
$i = 2$,	$\delta_3(2) = 0.0032$	$\psi_3(2) = 2$

Next, we apply step 3 of the Viterbi algorithm. From this step,

$$p^* = \max_{1 \leqslant i \leqslant 2}\big[\delta_3(i)\big] \qquad {q_3}^* = \underset{1 \leqslant i \leqslant 2}{\text{argmax}}\big[\delta_3(i)\big]$$

So expanding the brackets for various values of i, we get

$$p^* = \max[\delta_3(1), \delta_3(2)]$$

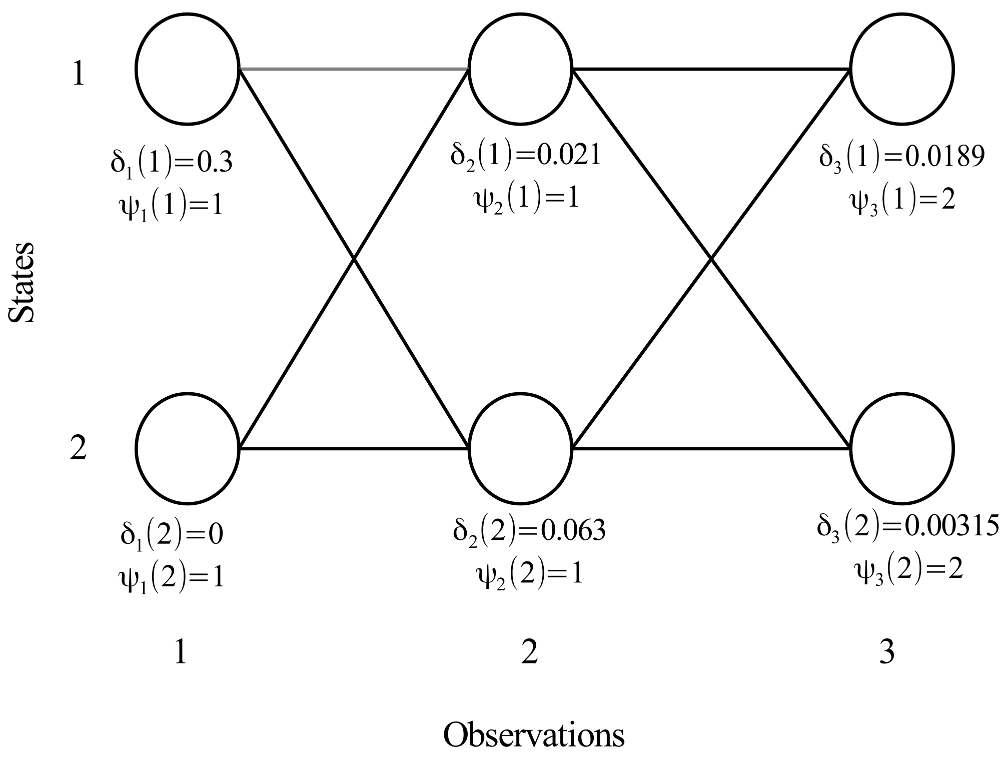


Figure 2.18: Summary of calculated values of variables $\delta_t(i)$ and $\psi_t(i)$

$$
\begin{aligned}
&= \max[0.0189, 0.00315] \\
&= 0.0189
\end{aligned}
$$

$$
\begin{aligned}
{q_3}^* &= \operatorname{argmax}[\delta_3(1), \delta_3(2)] \\
&= \operatorname{argmax}[0.0189, 0.00315] \\
&= 1
\end{aligned}
$$

Finally, we apply step 4 of Viterbi algorithm. By this step, $t = 2, 1$, we have

$$ {q_t}^* = \psi_{t+1}(q^*_{t+1}) $$

For $t = 2$,

$$ {q_2}^* = \psi_3(q^*_3) = \psi_3(1) = 2 $$

For $t = 1$,

$$ {q_1}^* = \psi_2(q^*_2) = \psi_2(2) = 1 $$

Hence, the optimal path of hidden state sequence is

$$Q = \{q_1^*, q_2^*, q_3^*\} = \{1, 2, 1\}$$

2.8.3 Determination of Optimal Model Parameters using Baum-Welch Algorithm

This problem is about determining the optimal model parameters $\lambda = (A, B, \Pi)$ such that it maximises the probability of the observation sequence $P(O)$ [72]. There is no known analytical method to solve this problem. However, model parameters can be chosen in such a way that $P(O)$ is locally maximised [71]. This is done by an iterative method named the Baum-Welch algorithm. The aim is to re-estimate the parameters (A, B, Π). It is done iteratively, and parameters improve such that $P(O)$ is maximum.
For this, consider a variable defined in the equation (2.63).

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j \mid O) \tag{2.63}$$

where $1 \leqslant t \leqslant T$ and $1 \leqslant i, j \leqslant N$. It is the joint probability of being in state S_i at time t and in state S_j at time $t+1$. We can write $\xi_t(i,j)$ in terms of forward variable $\alpha_t(i)$ and $\beta_t(i)$ using the equation (2.64) [71].

$$\xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{P(O)}$$

$$\therefore \xi_t(i,j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)} \tag{2.64}$$

These calculations can be visualised by the Figure 2.19 [71].
By the equation (2.54), the relation between $\xi_t(i,j)$ and $\gamma_t(i)$ is given by equation (2.65) [71].

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{2.65}$$

Summing $\gamma_t(i)$ over t gives expected number of transitions made from state S_i excluding the time step $t = T$, we get equation (2.66).

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{Expected number of transitions from } S_i \tag{2.66}$$

Summing $\xi_t(i,j)$ over t gives the expected number of transitions made from state S_i to

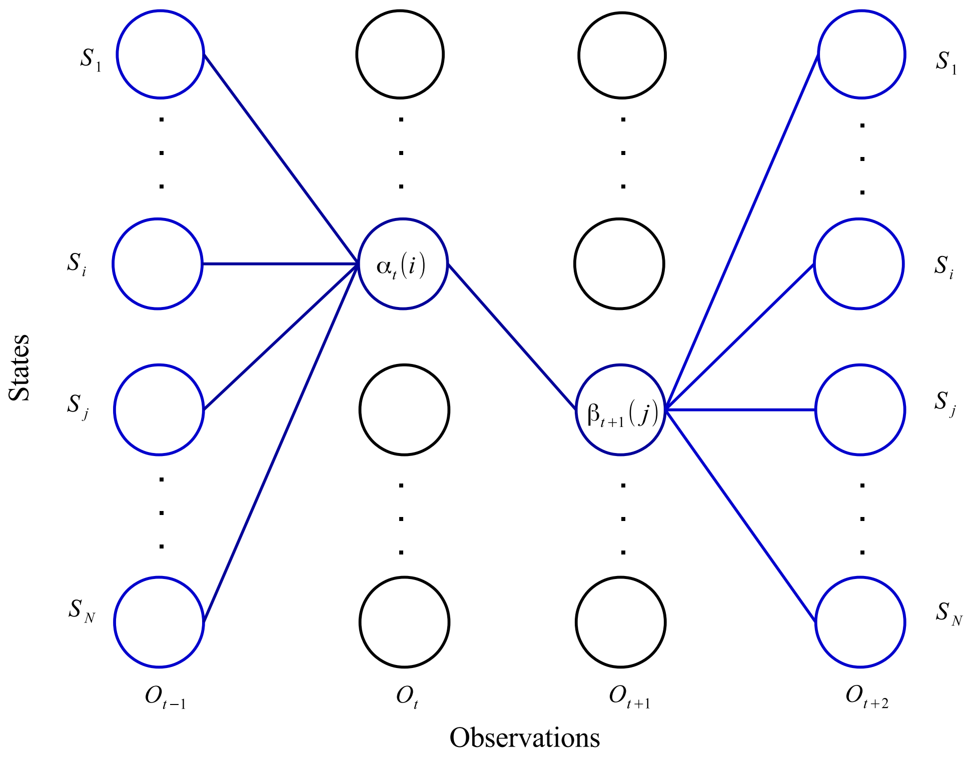


Figure 2.19: Determination of $\xi_t(i,j)$ for Baum-Welch algorithm

state S_j excluding the time step $t = T$, we get equation (2.67).

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{Expected number of transitions from } S_i \text{ to } S_j \tag{2.67}$$

Using equations (2.66) and (2.67), the re-estimation formulas of model parameters A, B and Π are as follows [71].
$\overline{\pi}_i$ = Expected number of times state S_i is reached at first time step $t = 1$

$$\therefore \overline{\pi}_i = \gamma_1(i) \tag{2.68}$$

$$\overline{a}_{ij} = \frac{\text{Expected number of transitions from } s_i \text{ to } s_j}{\text{Expected number of transitions from } s_i}$$

$$\therefore \overline{a}_{ij} = \sum_{t=1}^{T-1} \xi_t(i,j) \bigg/ \sum_{t=1}^{T-1} \gamma_t(i) \tag{2.69}$$

$$\overline{b}_j(k) = \frac{\text{Expected number of times in state } s_j \text{ and } v_k \text{ is observed}}{\text{Expected number of times in state } s_j}$$

$$\therefore \overline{b}_j(k) = \sum_{\substack{t=1 \\ O_t = v_k}}^{T} \gamma_t(j) \bigg/ \sum_{t=1}^{T} \gamma_t(j) \tag{2.70}$$

Hence, the steps of the Baum-Welch algorithm are as follows [71]:
Step 1: Define current model $\lambda = (A, B, \Pi)$

Step 2: Use (2.68), (2.69) and (2.70) to determine a re-estimated model $\overline{\lambda} = (\overline{A}, \overline{B}, \overline{\Pi})$.

Step 3: One of the following two cases may arrive:

1. The initial model λ defines critical points of likelihood. In this case $\lambda = \overline{\lambda}$ and process is over.

2. $\overline{\lambda}$ is more likely than λ. In this case, $P(O \mid \overline{\lambda}) \geqslant P(O \mid \lambda)$. Take $\lambda = \overline{\lambda}$ and go to Step 1.

Now let us try to understand the calculations involved in the Baum-Welch algorithm using the example that was considered for the forward, backward, and Viterbi algorithms.
First, we determine various values of the variable $\xi_t(i, j)$. We have,

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{2} \sum_{j=1}^{2} \alpha_t(i) a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)}$$

Substituting $t = 1$ in $\xi_t(i, j)$,
For $i = 1$,
If $j = 1$, then $\xi_1(1, 1) = 0.3$
If $j = 2$, then $\xi_1(1, 2) = 0.7$

For $i = 2$,
If $j = 1$, then $\xi_1(2, 1) = 0$
If $j = 2$, then $\xi_1(2, 2) = 0$

Substituting $t = 2$ in $\xi_t(i, j)$,
For $i = 1$,
If $j = 1$, then $\xi_2(1, 1) = 0.28$
If $j = 2$, then $\xi_2(1, 2) = 0.02$
For $i = 2$,
If $j = 1$, then $\xi_2(2, 1) = 0.6$
If $j = 2$, then $\xi_2(2, 2) = 0.1$

Substituting $t = 3$ in $\xi_t(i, j)$,
For $i = 1$,
If $j = 1$, then $\xi_3(1, 1) = 0.616$
If $j = 2$, then $\xi_3(1, 2) = 0.264$
For $i = 2$,

If $j = 1$, then $\xi_3(2,1) = 0.06$
If $j = 2$, then $\xi_3(2,2) = 0.06$

By equation (2.68), for $1 \leqslant i \leqslant 2$, $\overline{\pi}_i = \gamma_1(i)$. From this, the re-estimate initial state probabilities are

$$\overline{\pi}_1 = \gamma_1(1) = 1$$

$$\overline{\pi}_2 = \gamma_1(2) = 0$$

By equation (2.69), for $1 \leqslant i,j \leqslant 2$, we can determine the re-estimated state transition probabilities in the following way.

$$\overline{a}_{ij} = \sum_{t=1}^{2} \xi_t(i,j) \Bigg/ \sum_{t=1}^{2} \gamma_t(i)$$

$$\therefore \overline{a}_{ij} = \frac{\xi_1(i,j) + \xi_2(i,j)}{\gamma_1(i) + \gamma_2(i)}$$

Substituting $i = 1$ and $j = 1$ in above equation, we get

$$\begin{aligned} \overline{a}_{11} &= \frac{\xi_1(1,1) + \xi_2(1,1)}{\gamma_1(1) + \gamma_2(1)} \\ &= \frac{0.3 + 0.28}{1 + 0.3} \\ &= 0.446154 \end{aligned}$$

Similarly, the other state transition probabilities are

$$\overline{a}_{12} = 0.553846$$

$$\overline{a}_{21} = 0.857143$$

$$\overline{a}_{22} = 0.142857$$

.

Finally, by equation (2.70), for $1 \leqslant j \leqslant 2$ and $1 \leqslant k \leqslant 3$, we can determine the re-estimated observation probabilities using

$$\overline{b}_j(k) = \sum_{\substack{t=1 \\ O_t = v_k}}^{3} \gamma_t(j) \Bigg/ \sum_{t=1}^{3} \gamma_t(j)$$

$$\therefore \overline{b}_j(k) = \frac{\gamma_1(j) + \gamma_2(j) + \gamma_3(j) \text{ Such that } O_t = v_k}{\gamma_1(j) + \gamma_2(j) + \gamma_3(j)}$$

For $k = 1$ and $j = 1$ in above equation, we get,

$$\begin{aligned}\bar{b}_1(1) &= \frac{\gamma_1(1)+\gamma_2(1)+\gamma_3(1) \text{ Such that } O_t = v_1 = 3}{\gamma_1(1)+\gamma_2(1)+\gamma_3(1)} \\ &= \gamma_3(1)/(\gamma_1(1)+\gamma_2(1)+\gamma_3(1)) \\ &= 0.88/(1+0.3+0.88) \\ &= 0.403669\end{aligned}$$

In the same way, we can find the other values of observation probabilities,
Similarly other value of $\bar{b}_j(k)$ are

$$\bar{b}_1(1) = 0.40366972$$

$$\bar{b}_2(1) = 0.14634146$$

$$\bar{b}_1(2) = 0.13761468$$

$$\bar{b}_2(2) = 0.85365854$$

$$\bar{b}_1(3) = 0.4587156$$

$$\bar{b}_2(3) = 0$$

Summary of the calculations of the Baum-Welch algorithm is as follows:
Given model was $\lambda = (\Pi, a, b)$

$$\Pi = \begin{bmatrix}\pi_1 & \pi_2\end{bmatrix} = \begin{bmatrix}1 & 0\end{bmatrix}$$

$$A = \begin{bmatrix}a_{11} & a_{12} \\ a_{21} & a_{22}\end{bmatrix} = \begin{bmatrix}0.7 & 0.3 \\ 0.5 & 0.5\end{bmatrix}$$

$$B = \begin{bmatrix}b_1(1) & b_1(2) & b_1(3) \\ b_2(1) & b_2(2) & b_2(3)\end{bmatrix} = \begin{bmatrix}0.6 & 0.1 & 0.3 \\ 0.1 & 0.7 & 0.2\end{bmatrix}$$

Then the re-estimated model $\overline{\lambda} = (\overline{\Pi}, \overline{a}, \overline{b})$ using one iteration of the Baum-Welch algorithm is

$$\overline{\Pi} = \begin{bmatrix}\overline{\pi}_1 & \overline{\pi}_2\end{bmatrix} = \begin{bmatrix}1 & 0\end{bmatrix}$$

$$\overline{A} = \begin{bmatrix}\overline{a}_{11} & \overline{a}_{12} \\ \overline{a}_{21} & \overline{a}_{22}\end{bmatrix} = \begin{bmatrix}0.446154 & 0.553846 \\ 0.857143 & 0.0.142857\end{bmatrix}$$

$$\overline{B} = \begin{bmatrix}\bar{b}_1(1) & \bar{b}_1(2) & \bar{b}_1(3) \\ \bar{b}_2(1) & \bar{b}_2(2) & \bar{b}_2(3)\end{bmatrix} = \begin{bmatrix}0.40367 & 0.13762 & 0.45872 \\ 0.14634 & 0.85366 & 0.0\end{bmatrix}$$

This process can be continued iteratively until the desired accuracy is obtained [71]. So the Baum-Welch algorithm is very useful to determine the hidden Markov model parameters, which maximises the probability of observing a particular sequence given some initial values of the model parameters. However, in practice, there is a need to model the observation (emission) probabilities. For that, a Gaussian mixture model is used.

In the context of speech recognition problems, the observation sequence O represents the values of feature vectors produced from speech recording, and the hidden state sequence W represents the correct sequence of words uttered. The aim is to determine the most likely sentence $\widehat{W}$ out of all the possible sentences in language $\mathcal{L}$ given some observation sequence O of acoustic input. This can be represented with the help of conditional probabilities, as mentioned in the equation (2.71) [6].

$$\widehat{W} = \underset{W \in \mathcal{L}}{\operatorname{argmax}} P(W|O) \tag{2.71}$$

Using Bayes's rule of conditional probabilities,

$$\widehat{W} = \underset{W \in \mathcal{L}}{\operatorname{argmax}} \frac{P(O|W)P(W)}{P(O)} \tag{2.72}$$

Here, $P(W)$ = Prior probability, $P(O|W)$ = Observation likelihood and $P(O)$ = Probability of observation sequence. Here $P(O)$ can be ignored because it doesn't change for each sentence. Thus, most probable sentence $\hat{W}$ given some observation sequence O is the one for which product on left hand side of equation (2.73) is greatest [6].

$$\widehat{W} = \underset{W \in \mathcal{L}}{\operatorname{argmax}} P(O|W)P(W) \tag{2.73}$$

In the equation (2.73), $P(W)$ is called a language model, and it is determined using the technique called N-gram model [6]. An N-Gram model uses the previous $N-1$ words to predict the next one. It is determined using probability of a word given previous $N-1$ words. Consider a sentence W made up of n-words $w_1, w_2, \ldots, w_n$. Then probability of this sentence is

$$P(W) = P(w_1, w_2, \ldots, w_n) \tag{2.74}$$

Using the chain rule of probabilities,

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \cdots P(w_n|w_1, w_2, \ldots w_{n-1}) \tag{2.75}$$

Using Markov assumption that a current word depends only on previous word, the equation (2.75) becomes equation (2.76).

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1) \cdots P(w_n|w_{n-1}) \tag{2.76}$$

In the equation (2.73), $P(O|W)$ is called the acoustic model, and it is determined using hidden Markov models [6]. In this case, the Gaussian mixture models are used.

Gaussian mixture models are used to model the observation (emission) probabilities when the observations are continuous. In speech recognition problems, the observations are continuous values obtained from the speech signal using the process of feature extraction. It is a probabilistic model, and it assumes all the values are from a mixture of a finite number of Gaussian distributions with unknown parameters: the mean and the covariance. It is the linear combination of multivariate Gaussian distributions. Using the Gaussian mixture model, we can learn these parameters. These models are based on computing the probability density function. Observation probabilities are computed using the equation (2.77) [6].

$$b_j(o_k) = \sum_{m=1}^{M} c_{jm} \frac{1}{\sqrt{2\pi|\Sigma_{jm}|}} e^{-\frac{1}{2}(o_k-\mu_{jm})^T \Sigma_{jm}^{-1}(o_k-\mu_{jm})} \tag{2.77}$$

Here, Σ_{jm} is the covariance matrix, and μ_{jm} is the mean of the m^{th} Gaussian probability density function and the j^{th} feature vector. c_{jm} are the coefficients of Gaussian mixture models. The probabilities determined here are useful in the hidden Markov model for the classification of words.

2.9 Performance Measures

Whenever machine learning models are used in any application, one is always striving to have the best model. This can be checked using various performance measures.

2.9.1 Confusion Matrix

In classification problems, a confusion matrix is a contingency table to determine how a trained machine learning model is performing over unseen data that was not trained. From this matrix, we can determine the overall accuracy of the classification. Moreover, we can also determine various performance measures like precision, recall, and F1-score based on this matrix. If there are n classes for the classification, then the confusion matrix is $n \times n$ matrix in which each cell represents the number of patterns that are classified or misclassified.

Consider an example of a 4-class classification problem with classes A, B, C and D. Suppose there are 15 test patterns having desired outputs given as a row matrix: $d = [A\ D\ B\ C\ B\ D\ A\ B\ C\ C\ B\ B\ C\ D\ A]$. Suppose the trained artificial neural network is giving the predicted output $y = [A\ D\ D\ B\ B\ D\ A\ B\ A\ C\ A\ B\ C\ C\ B]$. Then the

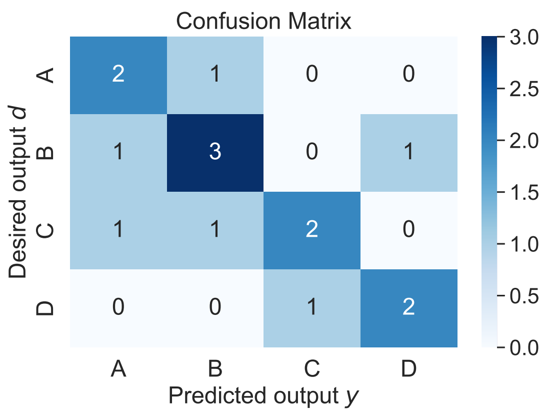


Figure 2.20: Confusion matrix of 4-class classification problem

confusion matrix is given by equation (2.78).

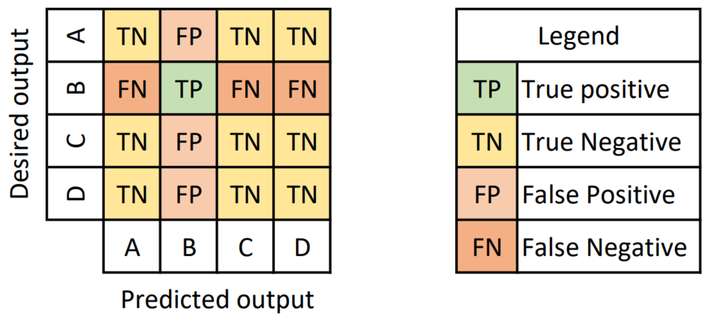
$$M = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 1 & 3 & 0 & 1 \\ 1 & 1 & 2 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix} \tag{2.78}$$

The 4 rows of the confusion matrix M correspond to the 4 classes in the desired output: class A, class B, class C and class D. Similarly, 4 columns in the confusion matrix M correspond to the 4 classes in the predicted output: class A, class B, class C and class D. The entry m_{ij} in the confusion matrix M represents the number of times the desired output having i^{th} class is equal to the predicted output having j^{th} class. So, all the values on the diagonal represent the correct classification. This matrix can also be visualised by a graph, as shown in the Figure 2.20. From the confusion matrix, the accuracy of testing a model and other performance measures can be easily determined.

All the entries in the confusion matrix are divided into the following four categories [73]:

1. True Positive (TP): The current class was predicted while it was actually desired.
2. True Negative (TN): The current class was not predicted and it was not desired.
3. False Positive (FP): The current class was predicted, but it was not desired.
4. False Negative (FN): The current class was not predicted but it was desired.

We can determine these 4 categories for each class using the confusion matrix by identifying cell positions. For example, suppose we are determining these four categories for class B. Then these 4 categories from the confusion matrix are determined using the Figure 2.21 [74].



Desired output \ Predicted output	A	B	C	D
A	TN	FP	TN	TN
B	FN	TP	FN	FN
C	TN	FP	TN	TN
D	TN	FP	TN	TN

Legend	
TP	True positive
TN	True Negative
FP	False Positive
FN	False Negative

Figure 2.21: Identifying True positive, True Negative, False Positive and False Negative in the confusion matrix of 4-class classification problem

After determining true positive, true negative, false positive, and false negative, we can use these values to determine the following performance measures [73]:

1. Recall or True Positive rate or Sensitivity.
2. False Positive rate (False alarm rate).
3. Specificity.
4. Precision.
5. F-measure.
6. Accuracy.

2.9.2 Recall

The recall, also called the true positive rate or sensitivity, for each class is defined by an equation (2.79) [73].

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.79}$$

For the example explained by the confusion matrix in the Figure 2.20, for class B, according to the Figure 2.21, TP is 3 and the total outputs having value B is 5. Hence, using equation (2.79), we get Recall $= 3/5 = 0.6$.

2.9.3 False Positive Rate

The false positive rate for each class is defined by equation (2.80) [73].

$$\text{False Positive rate} = \frac{FP}{TN + FP} \tag{2.80}$$

For the example explained by the confusion matrix in the Figure 2.20, for class B, according to the Figure 2.21, the sum of all FP values is 2, and the total outputs not having value B are 10. Hence, using equation (2.80), we get false positive rate = $2/10 = 0.2$.

2.9.4 Specificity

The specificity for each class is defined by equation (2.81) [73].

$$\text{Specificity} = \frac{TN}{FP + TN} \tag{2.81}$$

For the example explained by the confusion matrix in the Figure 2.20, for class B, according to the Figure 2.21, the sum of all TN values is 2, the sum of all FP values is 2, and the sum of all TN is 8. Hence, using equation (2.81), we get Specificity = $\frac{2}{2+8} = 2/10 = 0.2$.

2.9.5 Precision

The precision for each class is defined by equation (2.82) [73].

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.82}$$

For the example explained by the confusion matrix in the Figure 2.20, for class B, according to the Figure 2.21, TP is 3 and the sum of all FP values is 2. Hence, using equation (2.82), we get Specificity = $\frac{3}{3+2} = 3/5 = 0.6$.

2.9.6 F-Measure

The F-measure or F1-score for each class is defined by equation (2.83) [73].

$$\text{F-measure} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} \tag{2.83}$$

For the example explained by the confusion matrix in the Figure 2.20, for class B, according to equations (2.82) and (2.79) respectively, precision is 0.6 and recall is 0.6. Hence, using equation (2.83), we get F-measure = $\frac{2}{\frac{1}{0.6} \times \frac{1}{0.6}} = \frac{2}{\frac{10}{3}} = \frac{6}{10} = 0.6$.

2.9.7 Accuracy

The accuracy of classification is determined using equation (2.84) [74].

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.84}$$

In other words, it is defined as the trace of a confusion matrix divided by the sum of all elements of the confusion matrix. For the example explained by the confusion matrix in the Figure 2.20, the trace of the confusion matrix M is 9 and the sum of all elements is 15. Hence, accuracy is $9/15 = 0.6$. In other words, we can say that 60% of the predicted outputs are correctly classified. The above-mentioned accuracy formula and performance measures are used in our work to determine how each machine learning model is performing. This gives us an idea of how accurate our model is for speech recognition.

2.10 Conclusion

In this chapter, we discussed preliminaries and methodologies for speech recognition. The chapter begins with sections 2.1 and 2.2 which describe feature extraction techniques. Then, section 2.3 discusses the signal processing method to extract words from the speech. Various normalisation techniques are explained in section 2.4. The methods to classify speech are discussed in sections 2.5 to 2.8. Finally, the confusion matrix and post-classification analysis measures are described in section 2.9.