CHAPTER VI
RECOGNITION OF ISOLATED WORD BY USING MULTILAYER PERCEPTRON AND VITERBI SEARCH TECHNIQUE

6.1: INTRODUCTION:

In the present work Multilayer Perceptron (MLP) has been used to estimate the probability of the phonemes. The number of output nodes corresponds to the number of phonemes of Assamese and Bodo languages. The number of hidden nodes has been optimized experimentally. The number of input nodes is the same as the dimension of the input vector. The Multilayer Perceptron has been trained by the error backpropagation algorithm. The output of the MLP provides an observation sequence for each word. Viterbi search algorithm has been used to find the word that is closest to the probability distribution of the observation sequence.

6.2: MLP ARCHITECTURE:

A three-layer MLP with an input layer, a hidden layer and an output layer is shown in Figure 6.1. The neurons in the input, hidden and output layers are denoted by \( x_i \), \( h_j \) and \( y_k \) respectively, where \( i = 1, 2, \ldots, I; j = 1, 2, \ldots, J \) and \( k = 1, 2, \ldots, K \). \( I, J \) and \( K \) are the number of neurons in the layers of inputs, hidden and output respectively. Connection weights from input layer to hidden layer are denoted by \( w_{ij} \). Similarly,
$w_{jk}$ are the connection weights from hidden layer to output layer. The network is fully connected in the sense that every neuron in each layer of the network is connected to every other neuron in the adjacent forward layer by the connection weights.

![Figure 6.1: A three-layer Multilayer Perceptron](image)

The hidden layer and output layer may have biases, which act just like connection weights. These biases are connected to the hidden neurons and output neurons. The biases can be fixed or adjustable according to the network model utilized. The typical value for fixed biases is 1. The biases connected to the hidden layer and output layer are denoted as $b_h$ and $b_y$ respectively.
6.3: ACTIVATION FUNCTION:

Basic operation of a neuron involves summing its weighted input signals and applying an activation function. Typically, a nonlinear activation function is used for all the neurons in the network. According to Fausett (1994), an activation function used for Back-Propagation network should be continuous, differentiable, monotonically non-decreasing and its derivative is easy to compute for computational efficiency. The most typical activation function is the binary sigmoid function, which has a range between zero and one. The binary sigmoid function is defined as:

\[ f(x) = \frac{1}{1 + e^{-x}} \]  

(6.1)

6.4: ERROR BACKPROPAGATION (BP):

Error Backpropagation (BP) or the Generalized Delta Rule (Lippmann, 1989) is the most widely used supervised training algorithm for neural networks especially MLP. Because of its importance, we will discuss it in some detail in this section. We begin with a full derivation of the learning rule.

Suppose we have a multilayered feedforward network of nonlinear (typically sigmoid) units. We want to find values for the weights that will enable the network to compute a desired function from input vectors to output vectors.
Because the units compute nonlinear functions, we cannot solve for the weights analytically; so we will instead use a gradient descent procedure on some global error function $E$.

In the stage of back-propagation, error is first calculated at the output layer. Then, the error is back propagated to the hidden layer and lastly to the input layer. Each output neuron $(y_k)$ compares the calculated or actual output with the corresponding target value to find out the error information term, $\delta_k$. The $\delta_k$ then is used to calculate the weight correction term, $\Delta w_{jk}$ and bias correction term, $\Delta b_{yk}$. The $\Delta w_{jk}$ will be used to update the connection weight, $w_{jk}$ later and similarly the $\Delta b_{yk}$ will be used to update the bias, $b_{yk}$ later. $\eta$ is the learning rate of the network.

\[
\delta_k = (t_k - y_k)f'(y_{\text{input}_k}) \tag{6.2}
\]

\[
\delta_k = (t_k - y_k)y_k(1 - y_k) \tag{6.3}
\]

\[
\Delta w_{jk} = \eta \delta_k h_j \tag{6.4}
\]

\[
\Delta b_{yk} = \eta \delta_k \tag{6.5}
\]
The error information term, $\delta_k$ is sent to the hidden layer as a delta input. Each hidden neuron sums its delta inputs to give

$$\delta_{\text{input}}_j = \sum_{k=1}^{K} \delta_k w_{jk} \quad (6.6)$$

The error information term at the hidden layer, $\delta_j$ is determined by multiplying the derivative of its activation function with $\delta_{\text{input}}_j$.

$$\delta_j = \delta_{\text{input}}_j f'(h_{\text{input}}_j) \quad (6.7)$$

$$\delta_j = \delta_{\text{input}}_j (h_j)(1-h_j) \quad (6.8)$$

The $\delta_j$ then is used to calculate weight correction term, $\Delta w_j$ and bias correction term, $\Delta b_{h_j}$. The $\Delta w_j$ and the $\Delta b_{h_j}$ will be used to update $w_j$ and $b_{h_j}$ respectively later.

$$\Delta w_j = \eta \delta_j x, \quad (6.9)$$

$$\Delta b_{h_j} = \eta \delta_j \quad (6.10)$$
The weights and biases are updated using the weight and bias correction terms respectively. The adjustment makes use of the current weights and biases. At output layer, each output neuron \( y_k \) updates its weights and biases according to

\[
w_{jk}^{\text{new}} = w_{jk}^{\text{old}} + \Delta w_{jk}
\]

\[
h_{y_k}^{\text{new}} = h_{y_k}^{\text{old}} + \Delta h_{y_k}
\]

The process of adjustment is preferred if a momentum constant, \( \alpha \) can be added to the weight updating formulas. The purpose of the momentum is to accelerate the convergence of the Backpropagation learning algorithm. It can be very useful when some training data are very different from the majority of the data. In Backpropagation with momentum term, the weights and biases are updated not only with the current gradient but also with previous gradient. The modifications of the adjustment are as follows:

1. **At output layer**

\[
w_{jk}(t+1) = w_{jk}(t) + \Delta w_{jk}(t+1),
\]

where \( \Delta w_{jk}(t+1) = \eta \delta_{h} + \alpha \Delta w_{jk}(t) \)
\[ b_{yk}(t+1) = b_{yk}(t) + \Delta b_{yk}(t+1), \]  \hfill (6.15) \\

where \[ \Delta b_{yk}(t+1) = \eta \delta_k + \alpha \Delta b_{yk}(t) \]  \hfill (6.16)

2. **At hidden layer**

\[ w_{\theta}(t+1) = w_{\theta}(t) + \Delta w_{\theta}(t+1), \]  \hfill (6.17) \\

where \[ \Delta w_{\theta}(t+1) = \eta \delta_j h_j + \alpha \Delta w_{\theta}(t) \]  \hfill (6.18)

\[ bh_j(t+1) = bh_j(t) + \Delta bh_j(t+1), \]  \hfill (6.19) \\

where \[ \Delta bh_j(t+1) = \eta \delta_j + \alpha \Delta bh_j(t) \]  \hfill (6.20)

6.5: **IMPROVING ERROR-BACKPROPAGATION:**

There are some methods that can be adopted in order to enhance and improve the Error-Backpropagation. **First**, the weights and biases must be initialized uniformly at small random values. The typical range is between -0.5 and +0.5 \[^{[58]}\]. The choice of initial weights and biases is important because it will affect the training of network toward a global minimum. Improper initialization will lead the network to reach a local minimum. Besides, long training
times and the suboptimal results of MLP may be due to the lack of a proper initialization. Construction of proper initialization of adaptive parameters should enable finding close to optimal solutions for real-world problems.

Second, the target values must be chosen within the range of the sigmoid activation function. If a binary sigmoid activation function is used, the neural network is impossible to reach its extreme value of 0 and 1. If the network is trained to achieve these levels, weights will be driven to such large values that numerical instability will result. Moreover, the derivative of the sigmoid function approaches zero at extreme values, thus results in slow learning.

Third, learning rate and momentum must be selected properly. The learning rate can be selected as large as possible without causing the learning of network to oscillation. However, a small learning rate will guarantee a true gradient descent. The choice of learning rate depends on the learning problem and also the network architecture. The typical values for learning rate and momentum are 0.1 and 0.9 respectively.

Finally, the number of hidden layer neuron of the network must be chosen appropriately. Fewer neurons increase the possibility of the network to be trapped in local minimum during training. Excessive numbers of hidden neurons not only increases the training but also may cause problem of over-fitting or over-
learning. It is suggested that the approach to find the optimal number of hidden neurons starts with a small number, then the number is increased slightly (Shin Watanabe et al., 1992).

The best number of hidden units depends in a complex way on many factors, including the number of training patterns, the numbers of input and output units, the amount of noise in the training data, the complexity of the function or classification to be learned, the type of hidden unit activation function, the training algorithm. Too few hidden units will generally leave high training and generalization errors due to under-fitting. Too many hidden units will result in low training errors, but will make the training unnecessarily slow, and will result in poor generalization unless some other technique (such as regularization) is used to prevent over-fitting. A sensible strategy is to try a range of numbers of hidden units and see which works best.

One rough guideline for choosing the number of hidden neurons in many problems is the Geometric Pyramid Rule (GPR). It states that, for many practical networks, the number of neurons follows a pyramid shape, with the number decreasing from the input toward output. The number of neurons in each layer follows a geometric progression. The determination of hidden node number using GPR is shown in Figure 6.2. Other investigators (Berke and Hajela, 1991) [127] suggested that the nodes on the hidden layer should be between the average and the sum of the nodes on the input and output layers.
A network as a whole will usually learn most efficiently if all its neurons are learning at roughly the same speed. So, may be different parts of the network should have different learning rates. There are a number of factors that may affect the choices:

**Multilayer Perceptron (MLP)**

\[
\begin{align*}
\text{Output (Y)} & \quad \begin{array}{ccc}
\circ & \circ & \circ \\
\end{array} \\
\text{Hidden (H)} & \quad \begin{array}{cccc}
\circ & \circ & \circ & \circ \\
\end{array} \\
\text{Input (X)} & \quad \begin{array}{ccccccc}
\circ & \circ & \circ & \circ & \circ & \circ & \circ \\
\end{array}
\end{align*}
\]

Hidden Node Number (H)

\[
\begin{align*}
&= \sqrt{X \times Y} \\
&= \sqrt{6} \\
&= 6
\end{align*}
\]

\(X=\text{Number of input node} \quad Y=\text{Number of output node}\)

**Figure 6.2: The determination of hidden node number using Geometric Pyramid Rule (GPR).**
i. The later network layers (nearer the outputs) will tend to have larger local gradients (deltas) than the earlier layers (nearer the inputs).

ii. The activations of units with many connections feeding into or out of them tend to change faster than units with fewer connections.

iii. Activations required for linear units will be different for sigmoid units.

iv. There is empirical evidence that it helps to have different learning rates for the thresholds or biases compared with the real connection weights.

In practice, the learning process is often faster by just using the same rates for all the weights and thresholds, rather than spending time trying to work out appropriate differences. A very powerful approach is to use evolutionary strategies to determine good learning rates (Maniezzo, 1994) [128].
6.6: IMPLEMENTATION OF ERROR-BACKPROPAGATION:

The Error-Backpropagation learning algorithm is summarized in a flow chart shown in Figure 6.3.
The \textit{Viterbi algorithm} is a dynamic programming algorithm for finding the most likely sequence of states (normally hidden), called the \textit{Viterbi path}, that results in a sequence of observed events. The terms "Viterbi path" and "Viterbi algorithm" are also applied to related dynamic programming algorithms that discover the single most likely explanation for an observation. For example, in statistical parsing a dynamic programming algorithm can be used to discover the single most likely context-free derivation (parse) of a string, which is sometimes called the "\textit{Viterbi parse}".

The Viterbi algorithm was conceived by \textit{Andrew Viterbi} in 1967 as an error-correction scheme for noisy digital communication links, finding universal application in decoding the convolutional codes used in both CDMA and GSM digital cellular, dial-up modems, satellite, deep-space communications, and 802.11 wireless LANs. It is now also commonly used in speech recognition, keyword spotting, computational linguistics, and bioinformatics. For example, in speech-to-text (speech recognition), the acoustic signal is treated as the observed sequence of events, and a string of text is considered to be the "hidden cause" of the acoustic signal. The Viterbi algorithm finds the most likely string of text given the acoustic signal.

The Viterbi algorithm operates on a \textit{state machine assumption}. That is, at any time the system being modeled is in some state. There are a finite number
of states, however large. Multiple sequences of states (paths) can lead to a given state, but one is the most likely path to that state, called the "survivor path". This is a fundamental assumption of the algorithm because the algorithm will examine all possible paths leading to a state and only keep the one most likely. This way the algorithm does not have to keep track of all possible paths, only one per state.

Another key assumption is that a transition from a previous state to a new state is marked by an incremental metric, usually a number. This transition is computed from the event.

Yet another key assumption is that the events are cumulative over a path in some sense, usually additive. So the crux of the algorithm is to keep a number for each state. When an event occurs, the algorithm examines moving forward to a new set of states by combining the metric of a possible previous state with the incremental metric of the transition due to the event and chooses the best. The incremental metric associated with an event depends on the transition possibility from the old state to the new state. For example in data communications, it may be possible to only transmit half the symbols from an odd numbered state and the other half from an even numbered state. Additionally, in many cases the state transition graph is not fully connected. It must enter the stop state. After computing the combinations of incremental metric and state metric, only the best survives and all other paths are discarded. There are modifications to the basic
algorithm which allow for a forward search in addition to the backwards one described here.

Path history must be stored. In some cases, the search history is complete because the state machine at the encoder starts in a known state and there is sufficient memory to keep all the paths. In other cases, a programmatic solution must be found for limited resources: one example is convolutional encoding, where the decoder must truncate the history at a depth large enough to keep performance to an acceptable level. Although the Viterbi algorithm is very efficient and there are modifications that reduce the computational load, the memory requirements tend to remain constant.

6.8: VITERBI ALGORITHM FOR ISOLATED WORD RECOGNITION:

To find the best matching word by comparing input utterance with speech models in memory is very important and major task in speech recognition system. The Viterbi algorithm is an efficient technique to perform this task. Viterbi scorer computes the probability of generating the test word with each word model, and chooses one word model that gives the highest probability as the recognized word. Given an observation symbol sequence \{O_1, O_2, O_3, \ldots, O_t\}, the following logarithm-integer version of the original Viterbi algorithm is performed for each word model \( \lambda^\nu \) (1 \( \leq \nu \leq V \), V: number of reference words) [147,148].
Initialization:

\[ S_1(1) = b_1(O_1), S_1(i) = -\infty, \quad 2 \leq i \leq N \quad (6.21) \]

Recursion:

\[ S_i(j) = \max \left\{ S_{i-1}(i) + a_y + b_y(O_t) \right\}, \quad 1 \leq i \leq N, \quad 2 \leq t \leq T, \quad 1 \leq j \leq N \quad (6.22) \]

Termination:

\[ P_y = S_{TN} \quad (6.23) \]

Where \( N \) denotes the number of states in the model, and \( \{a_y\} \) and \( \{b_y(O_t)\} \) are obtained from the state transition and the output probabilities, respectively, by taking logarithmic transformation followed by normalization. After computing \( P_y \) for all reference words, one word with the highest \( P_y \) is selected as the recognized word. Since multiplications in the original Viterbi scoring procedure are time consuming operations, they are converted to additions by taking logarithms of the probabilities, and underflow can be avoided efficiently.
\[ a_y = u \log(\alpha_y) \] (6.24)

\[ b_y(O_t) = u \log(\beta_y(O_t)) \] (6.25)

where \( u \log(x) = C \log_{10} x + \Delta \) (6.26)

In the equation (6.26), the exact values of \( C \) and \( \Delta \) must be determined so as to maximize the dynamic range of the transition metrics and the output metrics.

6.9: VITERBI TRAINING:

The MLP output gives the probability estimation for each phoneme. The Viterbi decoding algorithm will be used to find the likelihood of the best path (state sequences) for each model. Further, since the most probable transition was used at each step, backtracking can be made which gives the corresponding state sequence. Conceptually, segmentation of training data with a known model transcription is the same as recognition, except in the former case there are no alternate model sequences to consider.

Since emission probability is obtained for each frame and state category, each of these is used in a process that is often called Viterbi alignment. In
this process, dynamic programming has been done, essentially using the one-pass method, in which the local distances are $-\log(y_j \mid q_i)$, and where there are transition costs $-\log(q_k \mid q_\ell)$ for hypothesizing transitions from states $k$ to $\ell$. Unlike the recognition scenario, the only model sequences that are considered are the ones associated with the known word sequence. All the word models together can be considered as a single model for the entire utterance in this case and there is only one to be evaluated. Backtracking can be done since the best previous state can be preserved for each frame, and so the best state sequence can be found. Additionally, since only one model sequence need to be evaluated, often it is not necessary to use elaborate data structure for this process -- the distance and backtracking information can be held in complete matrices, since the storage is not prohibitive as it would be in the recognition case.

The state sequence that is found through the backtracking procedure is considered to be an alignment of the states with the feature vectors. In the next step, transition and emission probabilities are removed assuming that the state sequence is correct.

Finally, the solution must be accessed. This can be done by looking at the changes in the global likelihood and setting some threshold on the improvement. Another approach is to test for convergence of the segmentation by counting the number of phonemes for which the state label has changed.
6.10: EXPERIMENT:

The clustered output of phoneme segmenter has been considered as input to the MLP based phoneme recognizer. The phonemes are recognized after splitting them from the isolated words uttered by the speaker. The phonemes which are separated from the isolated word are transformed into samples of 300 msec duration by using time-wrapping technique. The phonemes thus obtained are then split into 30 msec frames with one third overlapping. For each frame both pitch related features and cepstral features are extracted and an 18 element feature vector has been obtained as specified in Chapter IV. The feature vector thus obtained is used as input to the MLP based phoneme recognizer.

Initially an MLP based speech recognizer with one hidden layer has been constructed. The number of hidden nodes has been kept at 20 to provide 2160 connections between the input and hidden layer and 1080 connections between hidden and the output layer. The Enhance Back Propagation algorithm has been used to train the recognizer. Gradually, the number of hidden nodes and layers has been increased. With the increasing number of hidden nodes and layers, better performance in terms of recognition accuracy has been obtained. But, with the increasing number of hidden layers and nodes, the time required for convergence is also increased. Considering all parameters, it has been noticed that MLP with 3-layers (two hidden and one output) give the optimal performance for the recognition of phoneme.
The output of the phoneme recognizer is a sequence of phoneme probabilities that are extracted from a word. *Viterbi search algorithm* has been used to find the matching word.

**6.11: RESULT AND DISCUSSION:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Input Node</th>
<th>Output Node</th>
<th>Hidden node in the first layer</th>
<th>Hidden node in the second layer</th>
<th>Hidden node in the third layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>108</td>
<td>53</td>
<td>20</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>2</td>
<td>108</td>
<td>53</td>
<td>40</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>3</td>
<td>108</td>
<td>53</td>
<td>22</td>
<td>11</td>
<td>N/A</td>
</tr>
<tr>
<td>4</td>
<td>108</td>
<td>53</td>
<td>40</td>
<td>20</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>108</td>
<td>53</td>
<td>21</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>108</td>
<td>53</td>
<td>42</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>
### Table (6.2): Convergence time of the recognizer in terms of number of iterations

<table>
<thead>
<tr>
<th>Recognizer Type</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convergence Time (in terms of iteration)</td>
<td>56</td>
<td>103</td>
<td>623</td>
<td>2143</td>
<td>8131</td>
<td>66211</td>
</tr>
</tbody>
</table>

### Table (6.3): Result of the experiment for the recognition of Assamese and Bodo Phonemes with different types of MLP (100 experiment done for recognition of each phoneme). The dataset used for this experiment is the output of the Kohonen SOM algorithm which is the output of phoneme segmenter. All the phonemes are then wrapped into equal length of 300 msec.

<table>
<thead>
<tr>
<th>Recognizer Type → Language</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assamese</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vowel</td>
<td>39.81</td>
<td>56.34</td>
<td>63.27</td>
<td>72.8</td>
<td>88.43</td>
<td>90.34</td>
</tr>
<tr>
<td>Consonant</td>
<td>32.66</td>
<td>43.18</td>
<td>56.76</td>
<td>68.22</td>
<td>79.88</td>
<td>81.67</td>
</tr>
<tr>
<td>Average</td>
<td>36.235</td>
<td>49.76</td>
<td>60.02</td>
<td>70.51</td>
<td>84.16</td>
<td>86.01</td>
</tr>
<tr>
<td>Bodo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vowel</td>
<td>40.55</td>
<td>57.34</td>
<td>65.64</td>
<td>71.46</td>
<td>87.32</td>
<td>90.32</td>
</tr>
<tr>
<td>Consonant</td>
<td>35.88</td>
<td>37.71</td>
<td>39.08</td>
<td>65.22</td>
<td>69.23</td>
<td>71.93</td>
</tr>
<tr>
<td>Average</td>
<td>38.22</td>
<td>47.53</td>
<td>52.36</td>
<td>68.34</td>
<td>78.28</td>
<td>81.13</td>
</tr>
</tbody>
</table>
Table (6.4): Recognition of isolated word of different syllable structure (100 experiments done for recognition of each phoneme)

<table>
<thead>
<tr>
<th>SYLLABLE TYPE</th>
<th>ASSAMESE</th>
<th>BODO</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>71.81</td>
<td>70.67</td>
</tr>
<tr>
<td>VC</td>
<td>69.32</td>
<td>69.83</td>
</tr>
<tr>
<td>CV</td>
<td>69.12</td>
<td>69.92</td>
</tr>
<tr>
<td>CVC</td>
<td>67.88</td>
<td>68.04</td>
</tr>
<tr>
<td>CCV</td>
<td>66.89</td>
<td>66.39</td>
</tr>
<tr>
<td>VCC</td>
<td>66.12</td>
<td>66.76</td>
</tr>
<tr>
<td>CVCC</td>
<td>61.21</td>
<td>63.56</td>
</tr>
<tr>
<td>CCVC</td>
<td>61.78</td>
<td>61.57</td>
</tr>
</tbody>
</table>

Table (6.3) shows the recognition accuracy of the phoneme recognizer. It has been observed that with increasing hidden node and layers the recognition accuracy increases. But with increasing nodes and layers the convergence time, which is represented in the form of number of iteration is also increases. Thus, considering all parameters, the configuration 4 that is neural network with input layer, three hidden layers and a output layer with number of
nodes in the hidden layers 21, 12 and 12 respectively is considered as optimal and has been used throughout the experiment. The output of the recognizer is an ordered set of isolated phonemes which work as input to the Viterbi search unit. The Viterbi search unit recognized the word associated with the sequence of phonemes. It has been observed that with the recognition accuracy depends on the syllabic structure of the word. Table (6.4) shows the recognition accuracy of words for different syllabic structures. It has been observed that with increasing number of syllables the recognition accuracy degrades. The main reason of this variation is due to the fact that errors in the phoneme recognition block have an accumulating affect on the error in recognition of the word.

Thus, it has been observed that Viterbi search performed on the ordered sequence of phoneme is an effective measure for the recognition of isolated word. The recognition accuracy of the system depends on the recognition accuracy of the individual phoneme.