SNS COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

Coimbatore – 35.

DEPARTMENT OF BIOMEDICAL ENGINEERING

UNIT 4

BACKPROPAGATION NEURAL NETWORK

To illustrate some problems that often arise when we are attempting to automate complex pattern-recognition applications, let us consider the design of a computer program that must translate a 5 × 7 matrix of binary numbers representing the bit-mapped pixel image of an alphanumeric character to its equivalent eight-bit ASCII code. This basic problem, pictured in figure, appears to be relatively trivial at first glance. Since there is no obvious mathematical function that will perform the desired translation, and because it would undoubtedly take too much time (both human and computer time) to perform a pixel-by-pixel correlation, the best algorithmic solution would be to use a lookup table.

The lookup table needed to solve this problem would be a one-dimensional linear array of ordered pairs, each taking the form.

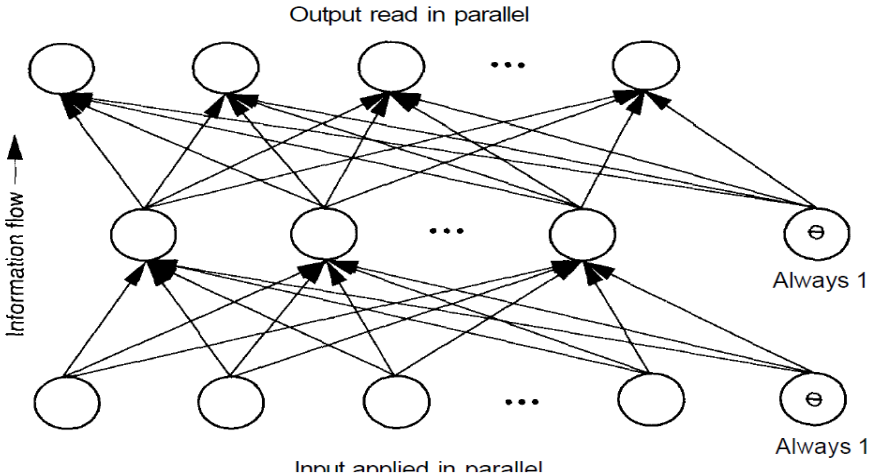
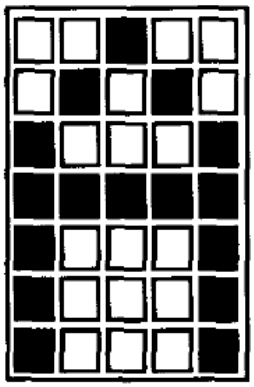


Figure: The general back propagation network architecture is shown.

```
record AELEMENT =
   pattern : long integer;
   ascii : byte;
end record;
```



$= 00100101010001111111000110001100012$

$= 0951FC631_{16}$

$= 65_{10}$ ASCII

The first is the numeric equivalent of the bit-pattern code, which we generate by moving the seven rows of the matrix to a single row and considering the result to be a 35-bit binary number. The second is the ASCII code associated with the character. The array would contain exactly the same number of ordered pairs as there were characters to convert. The algorithm needed to perform the conversion process would take the following form:

```
function TRANSLATE (INPUT : long integer;
                    LUT: AELEMENT [ ]) return ascii;
(performs pixel-matrix to ASCII character conversion)
var TABLE : AELEMENT [ ];
    found : Boolean;
      i : integer;
begin
   TABLE = LUT       {locate translation table}
   found = false;    {translation not found yet}

   for i = 1 to length (TABLE) do {for all items in table}
      if TABLE [i]. pattern = INPUT
      then Found = True; Exit;
                              {translation found, quit loop}
```

```
    End;

  If Found
  Then return TABLE [i]. ascii {return ascii}
  Else return 0
end;
```

Although the look up –table approach is reasonably fast and easy to maintain, there are many situations that occur in real systems that cannot be handled by this method. For example, consider the same pixel-image-to-ASCII conversion process in a more realistic environment. Let's suppose that our character image scanner alters a random pixel in the input image matrix due to noise when the image was read. This single pixel error would cause the look up algorithm to return either a null or the wrong ASCII code, since the match between the input pattern and the target pattern must be exact.

Now consider the amount or additional software (and, hence, CPU time) that must be added to the lookup-table algorithm to improve the ability of the computer to "guess" at which character the noisy image should have been. Single-bit errors are fairly easy to find and correct. Multibit errors become increasingly difficult as the number of bit errors grows. To complicate matters even further, how could our software compensate for noise on the image if that noise happened to make an "O" look like a "Q" or an "E" look like an "F"? If our character-conversion system had to produce an accurate output all the time, an inordinate amount of CPU time would be spent eliminating noise from the input pattern prior to attempting to translate it to ASCII.

One solution to this dilemma is to take advantage of the parallel nature of neural networks to reduce the time required by a sequential processor to perform the mapping. In addition, system-development time can be reduced because the network can learn the proper algorithm without having someone deduce that algorithm in advance.

BPN Operation

In section, we will cover the details of the mechanics of back propagation. A summary description of the network operation is appropriate here, to illustrate how the BPN can be used to solve complex pattern-matching problems. To begin with, the network learns a predefined set of input-output example pairs by using a two-phase propagate-adapt cycle. After an input pattern has been applied as a stimulus to the first layer of network units. It is propagated through each upper layer until an output is generated. This output pattern is then compared to the desired output, and an error signal is computed for each output unit.

The error signals are then transmitted backward from the output layer to each node in the intermediate layer that contributes directly to the output. However, each unit in the intermediate layer receives only a potion of the total error signal, based roughly on the relative contribution the unit made to the original output. This process repeats, layer by layer, until each mode in the network has received an error signal that describes its relative contribution to the total error. Based on the error signal received, connection weights are then updated by each unit to cause the network to converge toward a state that allows all the training patterns to be encoded.

The significance of this process is that, as the network trains, the nodes in the intermediate layers organize themselves such that difference nodes learn to recognize different feature of the total input space. After training, when present with an arbitrary input pattern that is noisy or incomplete, the units in the hidden layers of the network will respond with an active output if the new input contains a pattern that resembles the feature the individual units learned to recognize during training. Conversely, hidden-layer units have a tendency to inhibit their outputs if the input pattern does not contain the feature that they were trained to recognize.

As the signals propagate through the different layers in the network, the activity pattern present at each upper layer can be thought of as a pattern with features that can be recognized by units in the subsequent layer. The output pattern generated can be thought of as a feature map that provides an indication of the presence or absence of many different feature combinations at the input. The total effect of this behaviour is that of BPN provides an effective means of allowing a computer system to examine data patterns that may be incomplete or noisy, and to recognize subtle patterns from the partial input.

Several researchers has shown that during training, BPNs tend to develop internal relationships between nodes so as to organize the training data into classes of patterns .This tendency can be extrapolated to the hypothesis that all hidden-layer units in the BPN are some how associated with specific features of the input pattern as a result of training. Exactly what the association is may or may not be evident to the human observer. What is the important is that the network has found an internal representation that enables is to generate the desired outputs when given the training inputs. This same internal representation can be applied to inputs that were not used during training. The BPN will classify these previously unseen inputs according to the features they share with the training examples.