# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>PROJECT SCOPE</td>
<td>2</td>
</tr>
<tr>
<td>SYSTEM DESIGN</td>
<td>16</td>
</tr>
<tr>
<td>COMPONENT AND PROCESS DESIGN</td>
<td>3</td>
</tr>
<tr>
<td>DATA DESIGN</td>
<td>9</td>
</tr>
<tr>
<td>USER DISPLAYS AND OUTPUTS</td>
<td>10</td>
</tr>
<tr>
<td>SYSTEM FILES</td>
<td>10</td>
</tr>
<tr>
<td>PROTOTYPE DESCRIPTION</td>
<td>11</td>
</tr>
<tr>
<td>TEST PROCEDURES</td>
<td>12</td>
</tr>
<tr>
<td>SYSTEM DEVELOPMENT SCHEDULES</td>
<td>13</td>
</tr>
<tr>
<td>SPECIAL NOTES</td>
<td>15</td>
</tr>
<tr>
<td>PROJECT EXTENSIONS</td>
<td>15</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>20</td>
</tr>
<tr>
<td>APPENDICES</td>
<td>21</td>
</tr>
</tbody>
</table>
5.1 INTRODUCTION

Texas Coastal Ocean Observation Network (TCOON) presently comprises 50 data collection platforms (DCP) that gather a variety of environmental data along the Texas gulf coast. Data analysis for the various tasks performed by present day Tcoon network requires continuous water-level data over a period of years. Even a gap of data missing for 48 hours will make all the previous readings invaluable. The water-level data is essential in determining the ownership of thousands of acres of disputed land, particularly in the Laguna Madre. If the ownership is decided, the acreage can be leased for mineral production.

At the present time, gaps have been detected in the water-level leadings at some Tcoon stations. Hence, this preliminary attempt is made at designing and implementing the Neural Network to check if the neural network can characterize the water-level data based on two parameters namely wind speed and wind direction and previous days water-level data. TEFL the missing gaps.
5.2 PROJECT SCOPE

OBJECTIVES: The major objectives of the project are outlined as below:

- Selecting an appropriate Artificial Neural Network (architecture)
- Preparing or preprocessing the data for the neural network.
- Training the Neural Network to recognize the relationship between water-level data and other parameters.
- Testing the Neural Network to check the predictions of the missing data.

Constraints: However the system also has limitations like:

- Memory Constraints
  Since large amounts of data is used for training and testing of the Neural Network it takes up a considerable amount of CPU space.

- Time Complexity Factor is very high.
  Time required for training the network is very high on the part of the CPU and the programmer.

Qualifications: In order to use this system, the operator should have knowledge of the programming language ‘c’.
5.4 COMPONENT AND PROCESS DESIGN

The system does not require any special hardware components other than to run our software. However, high-speed computers will greatly enhance the speed of training the neural network. This system is completely implemented by the software. Various software processes involved in the system are

- Pre-processing of the input data
- Neural Network Architecture
- Training the neural network
- Transfer function
- Optimization process
- Output Routines
- Testing of the Neural Network

The flow of data through various above mentioned processes is indicated in the chart on page 14. Each process is explained and their pseudocode is given in the following section.

CONVERSION OR PRE-PROCESSING OF THE INPUT DATA:

The input data for the project can be downloaded from the World Wide Web (URL) this raw data has to be processed to be presented (fed) to the neural network.
The preprocessing of the data is done by a program written in ‘c’.

Pseudocode:

    open    inputfile

    Read    dummy 1, dummy 2, dummy 3, dummy 4 from line one of input file,

        /*
        dummy 1 has station_ID
        dummy 2 has year, day and hr.
        dummy 3 has parameter name.
        dummy 4 has parameter value.
        */

    sscanf    start_yr, start_day, start_hr.

    Input data [0] = value,

    while    not    EOF,

    Read    dummy 1, dummy 2, dummy 3, dummy 4, from next line,

    sscanf    current_yr, current_day, current_hr.,

    index = (current_yr - start_yr) * 365 * 24 +((current_day - start_day)*24) +
    (current_hr/100)

    If dummy 4 is != NA

    Input data [index]= value      //dummy 4 has value

    Else

    Input data [index] = 0

    End while.
The above program first scans the data for 'NA' and 'O's and replaces them with 'O's. If the data is missing for the entire day or days, the index is used to calculate the next data input. All the data values are then put into an array to be used as an input for neural net.

NEURAL NET:

ARCHITECTURE:

The neural net has one input layer, two hidden layers and an output layer. The number of neurons in the hidden layer are arbitrarily chosen. More the number of neurons, more accurate the prediction, but slows down the network. Figure of the network architecture is included in page 16.

TRAINING OF THE NETWORK:

PSEUDOCODE:

Begin

load pwl data
load wsp data
load wdr data

/* Convert range of all parameters between 0 & 1 */

pwl = pwl - min of pwl
     max of pwl - min of pwl

wsp = wsp - min of wsp
     max of wsp - min of wsp

wdr = wdr - min of wdr
     max of wdr - min of wdr

M = Size (pwl)

for i = 1 to M
    x[i] [1-144] = pwl [i - i + M]
            // creates 1st 144 points

    x[i] = x[i] Wsp[144 + i] Wdr [144 + i]
            // total of 146

End for

INITIALIZE WEIGHTS:

W1 = rand [50] [144]          // weight between input layer and 1st hidden layer

W2 = rand [8] [50]            // weight between 1st hidden layer and 2nd hidden layer

W3 = rand [1] [8]             // weight between output layer and 2nd hidden layer
Any optimization function aims to drive the error toward global minimum. Error is defined as the function of the number of parameters.

The optimization algorithm used here is conjugate gradiencc or modified Rosenbrock algorithm.

**PSEUDOCODE:**

```plaintext
optimize (w1, w2, w3, a1, a2, a3, dw1, dw2, dw3, E)

/*
dw1, dw2, dw3 changes in weights proposed by the optimization algorithm based on error E. a1, a2, a3 are the constants used. */
w1 = w1 + dw1
w2 = w2 + dw2
w3 = w3 + dw3
End While.
```

**OUTPUT ROUTINES:**

Plot \( y', y \)  
\( y' = \text{predicted output} \)
\( y = \text{actual output} \)

Save \( w_1, w_2, w_3 \), and \( a_1, a_2, a_3 \) into a file.

**TESTING:**
This is the repetition of the training method, but here the weights that are saved
are used. This process has not been completed at the present time. Will elaborate in the final
design.

5.5 DATA DESIGN:

Data Dictionary

pw1 = primary water level
wsp = wind speed
wdr = wind direction
x  = input data to network, contains 6 prior days of pw1, one element of current wind
     speed, one element of wind direction (146 values total)

w1 = weight matrix between input layer and hidden layer 1
w2 = weight matrix between hidden layer 1 and hidden layer 2
w3 = weight matrix between hidden layer 2 and output layer

y1 = predicted output

x1, x2, x3 = outputs of corresponding layers

y1, y2, y3 = outputs after transfer function

y  = actual output

dw1, dw2, dw3 = corrections to w1, w2, w3, respectively

E   = Prediction error of the network

M   = Total number of output training pairs
5.6 USER DISPLAYS AND OUTPUTS

There are no user displays. The output of the system is a plot comparing the expected output to predict.output. Screen dump will be included in the final design.

5.7 SYSTEM FILES

1. Extract .C:
   Purpose: The purpose of this program is to extract hourly data from a netscape down loaded file which has 6 minutes of data with gaps.

   The processed data contains the hourly data with the missing data filled with zero’s. This hourly data is written to the output file as a one dimensional array.

2. SM2LAYER.M:

   This is a Neural Network Training program. It takes 144 primary water-level data points and two parameters namely wind speed and wind direction to predict the next water-level point. It uses an optimization technique called the Conjugate Gradient Method (modified Rosenbrock method) to change the weights of the network, in order to reduce the prediction error. The optimization is repeated until the stopping criteria is satisfied. The stopping criteria can be one of the following:
(a) The error should be below the indicated error.

(b) The error does not change between iterations of the training.

3. SM2tst_M

This program uses the weights derived by the training program and uses the water-level from 6 previous days ad wind speed and wind direction points to predict the next water-level-point. During prediction, it replaces any zeros in the input matrix by their predicted counter parts. The RMS error of prediction is computed and displayed.

The trained weights which are matrices of sizes specified in the earlier section and the constants used in Transfer function are stored in the file for use with testing data.

5.8 Prototype Description

There is no prototype developed in this system. The final product will be delivered as per specification.
5.9 TESTING PROCEDURES

This system is tested with the data for which the output was known. The predicted output of the system was compared to the actual known output to test the validity. The RMS error calculated indicates the accuracy of the prediction.

Pseudocode:

Load input pwl
Load input wsp
Load input wdr

/*
  convert the range of all the parameters to between 0 and 1 */

pwl = \frac{pwl - \text{Min of } pwl}{\text{Max of } pwl - \text{Min of } pwl}

wsp = \frac{wsp - \text{Min of } wsp}{\text{Max of } wsp - \text{Min of } wsp}

wdr = \frac{wdr - \text{Min of } wdr}{\text{Max of } wdr - \text{Min of } wdr}

M = \text{Size} (pwl)

for i = 1 to M
    if pwl[i + 144] = 0 then
        x = [x[2 - 144] y[i - 1] wsp[i] wdr[i]]
Load Weights.dat

\[ x_1 = x \cdot w_1 \quad // x \text{ is the input} \]
\[ y_1 = \sin (a_1 \cdot x_1) \quad // \text{layer x and 1st hidden layer} \]
\[ x_2 = y_1 \cdot w_2 \quad // a_1, a_2, a_3 \text{ are constants} \]
\[ y_2 = \sin (a_2 \cdot x_2) \quad // y_1 \text{ is the output after} \]
\[ x_3 = y_2 \cdot w_3 \quad // \text{the first hidden layer} \]
\[ y^1 = x_3 \quad // y_2 \text{ is output after 1st} \]
\[ E = y^1 - y \quad // \text{hidden layer and before} \]
\[ \text{RMS} = \sqrt{\sum_{i=1}^{N} (E-E)^2} \quad // \text{2nd hidden layer} \]
\[ E \quad // \text{is the error. It is} \]
\[ \sqrt{\sum_{i=1}^{N} (E-E)^2} \quad // \text{the difference between} \]
\[ N \quad // \text{actual} \]

Plot \[ y^1, y \quad // y^1 \text{ is predicted output} \]
\[ y \quad // y \text{ is actual output} \]

5.10 SYSTEM DEVELOPMENT SCHEDULES:

Training the Neural Network \quad Nov. 4th - Nov. 22nd

- Developing the Neural Network Architecture.
- Developing the Neural Network Training Program
• Feeding the Neural Network with training data for station 001 (Naval Air Station)

• Checking the results for validity and making necessary modifications

• Preliminary Design Presentation Nov. 11th

• Training the Neural Network for the remaining stations namely, 006 (Ingleside Station)

  Testing the Neural Network Nov. 23rd - 28th

• Testing the Neural Network

• Modifying the weight matrix, if required.

• Preparation of final design document

• Presentation of the Final Design Nov. 27th

• Preparation of the Final Report Nov. 8th - Dec. 6th
5.11 SPECIAL NOTES AND EXTENSIONS:

SPECIAL NOTES:

Training procedures of the Neural Network requires a lot of CPU time, RAM and time on the part of the operator. The implementation of this project also requires MATHLAB and 'C' installed on the system.

5.12 EXTENSIONS:

The system could be trained with data from more stations to characterize the property of intra-station variability. Inter-station variability can be better characterized by including more parameters.

ALTERNATE APPROACHES:

Functional approximation by using higher order polynomial (5th or 6th order) estimates the previous data and predict the missing values. This process may take less time but it may not be as accurate as a neural network.
DATA PRE-PROCESSING

INPUT FILE
(DATA TAKEN EVERY 6 MINS)

EXTRACT_DATA.C
(ALGORITHM TO PROCESS THE INPUT FILE)

OUTPUT FILE
(CONTAINS INPUT DATA FOR NEURAL NET EVERY 1HR)

* Process Repeated for wind speed and wind direction.
# Network Configuration

<table>
<thead>
<tr>
<th>INPUT</th>
<th>LAYER</th>
<th>hidden layer 1</th>
<th>hidden layer 2</th>
<th>output layer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W&lt;sub&gt;1&lt;/sub&gt;</td>
<td>W&lt;sub&gt;2&lt;/sub&gt;</td>
<td>W&lt;sub&gt;3&lt;/sub&gt;</td>
</tr>
<tr>
<td>144</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>145</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>146</td>
<td></td>
<td>50*8</td>
<td>0</td>
<td>8*1</td>
</tr>
</tbody>
</table>

Y<sub>j</sub>, Y<sub>2</sub>, Y<sup>T</sup>
* Plot a graph using the array of outputs and compare with the actual values.
BIBLIOGRAPHY

1. Knowledge - Based for Engineers and Scientists, Adrian A. Hopgood
2. Software Engineering, Sommerville, 5th Edition
APPENDICES

1. Graphs indicating actual and predicted primary water level readings for the stations 001 (Naval Air Station) and Station 006 (Ingleside Station).

2. Code for the Processing of Data
   Code for the Training of the Neural Network
   Code for the Testing of the Neural Network
#include <stdio.h>
#include <stdlib.h>

main(argc, argv)
int argc;
char **argv;
{
    FILE *fp_in, *fp_out;
    char *str;
    int *pwl_data;
    char *dummy1, *dummy2, *dummy3, *dummy4;
    int point;
    int year, day, hour;
    int st_year, st_day, st_hour;
    int ret;
    int i, index;

    if (argc < 3)
    {
        fprintf(stderr, "\nUsage: extract_data in_file out_file no.of years\n");
        return;
    }

    fp_in = fopen(argv[1], "r");
    fp_out = fopen(argv[2], "w");

    dummy1 = (char *) malloc(3*sizeof(char));
    dummy2 = (char *) malloc(12*sizeof(char));
    dummy3 = (char *) malloc(3*sizeof(char));
    dummy4 = (char *) malloc(4*sizeof(char));
    str = (char *) malloc(200*sizeof(char));

    fscanf(fp_in, "%s %s %s %s\n", dummy1, dummy2, dummy3, dummy4);
    fgets(str, 190, fp_in);
    sscanf(dummy2, "%d%3d+%4d", &year, &st_day, &st_hour);
    pwl_data = (int *) malloc(atoi(argv[3])*365*24*sizeof(int));

    if (strcmp(dummy4, "NA") != 0)
    {
        sscanf(dummy4, "%d", &point);
        pwl_data[0] = point;
    }
    else
    {
        pwl_data[index] = 0;
    }

    while ((ret = fscanf(fp_in, "%s %s %s %s\n", dummy1, dummy2, dummy3, dummy4))
    != EOF)
    {
        /* Process the second string for the array index. */
        fgets(str, 190, fp_in);
        sscanf(dummy2, "%d%3d+%4d", &year, &day, &hour);

        index = (int)((year-st_year)*365*24+(day-st_day)*24+hour/100);

        if (year==st_year)
        else
        index = (int)((year-st_year)*365*24+(day)*24+hour/100);

Page 1
if (strcmp(dummy4, "NA") != 0)
{
    scanf(dummy4, "%d", &point);
    pwl_data[index]= point;
} else
    pwl_data[index]=0;

for (i=0; i<10; i++)
    printf("%d\t", pwl_data[i]);
for (i=0; i<index; i++)
    fprintf(fp_out, "%d\n", pwl_data[i]);