```
//=========================================================================
==

// 'NN Classifier for Artificial Retina "


// Code for a NN MLP classifier' (C++)

// Code written by Dr. Mircea G Hulea based on general ideas by HN
Teodorescu

// (C) 2013 MG Hulea & HN Teodorescu

// Freely available exclusively for educational purposes.

// Use it "äs such" on your own responsibility.

// No guarantee, direct or implied, is made.

// Authors have tested the code and believe to include no major error.


// NOTA BENE: the code was used by the authors in research reported in at
// least one paper.


// You can contact the authors by e-mail.

// General references explaining the use:

// "H.N.Teodorescu, M. Hulea - Classifiers for decoding patterns in the
// response of an artificial retina", submitted for evaluation to the
// 11-th International Symposium on Signals, Circuits and Systems, 2013,
// organized at "Gheorghe Asachi" Technical University of Iasi, Romania

//=========================================================================
===

#include <stdio.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <math.h>
#include <string.h>

float x[10]; //network input
float x_h[15][15]; //auxiliary variables for hidden layers inputs
int p, q, f, wf; //flags
int area_corners, n_areas;
int sig, div_factor;
int recog_patt;
int m_idx, max_it;
int rand_v;
unsigned r_seed;
int r_idx, vector_idx;
float current_int, input_int;
float digital_err;
float min_err;
float eps_2, eps_2_noise;
float dif; //auxiliary variable for computing epsilon^2
float sq_err;
int mat_conf[16][16];
float dig_d_sig[15];
float y_dig[15];
int generated_patt, current_patt;

float min_v[10], max_v[10], sig_limit_up, sig_limit_down;
```

```
int n_random_values;
int input_set, iteration;
int t_sets; // number of training sets
int idx, idx_out;//read/write file indexes
int nofvalues;//nofvalues read from signal file
int i,j,k; // indexes
int maxint = 32768;
int h; //number of hidden layers
int n_i, n, m, o, m_old; // number of neurons on input, hidden and output
layers
char c[100]; //string used for reading numbers from files
float file_data[5][5];
FILE *err_file, *r_file;
FILE *out_file, *w_file, *o_file, *p_file; //weights file pointer
FILE *s_file; //weighted sums file pointer
FILE *stat;
FILE *conf_file, *sqerr_file;

float a;//learning rate
float s[15][15]; //weighted sums
float w[15][15][15]; // weights matrix
float teta, delta, alfa; //neurons threshold
float y[10]; //network outputs
float d[15000]; //network desired output
float e[15][15]; // output error
float err_factor[15][15];
float v[15000];
float y_aux[15000];
float v_n[2000];
float y_k[2000];
float y_k_n[2000];
float ef_signal;
float p_noise;
float avg_signal;
float err_output;
float v_data[15000];
float l[4][16];
float d_aux[15000];


void write_AND_to_file(int n) {

        FILE *and_file_o, *and_file_i;
        and_file_i = fopen("c:\\C_dir\\inputs.txt","w");
        and_file_o = fopen("c:\\C_dir\\outputs.txt","w");

        fprintf(and_file_i,"AND function inputs:\n");
        fprintf(and_file_o,"AND function outputs:\n");

        for(i=0;i<2;i++) {
                for(j=0;j<2;j++){
                        fprintf(and_file_i, "%d, %d, \n", i, j);
                        fprintf(and_file_o, "%d, \n", i&j);
                }
        }

        fclose(and_file_i);
        fclose(and_file_o);
}

void read_window_limits(int n_psa, int n_w) {

char c_aux;
int psa;
int file_i;
int xy, n_corners;
float w;
float nr;

        p_file = fopen("c:\\c_dir\\clasif.txt", "r");
        n_corners = 4;
```

```
        psa = 0;

        for(psa=0; psa<n_psa; psa++) {

        for(idx=0;idx<10;idx++) {
                c[idx] = 0;
        }

        c_aux = getc(p_file);
        while(c_aux != 'W' && c_aux != EOF) {
                c_aux = getc(p_file);
//
        }

        for(idx=0;idx<10;idx++) {
                c[idx] = 0;
        }

        file_i = 0;
        c_aux = getc(p_file);
        while(c_aux!=':' && file_i < 5) {
                c[file_i] = c_aux;
                c_aux = getc(p_file);
                file_i++;

        }

        w = atof(c)-1;
        printf("w = %2.1f\n", w + 1);

        xy = 0;
        c_aux = getc(p_file);
        while(c_aux==' '||c_aux=='\n') {
                        c_aux = getc(p_file);
                        //printf("%c", c_aux);
                }

        for(i=0;i<n_w;i++) {
        for(j=0;j<n_corners;j++){
                for(idx=0;idx<10;idx++) {
                        c[idx] = 0;
                }
                file_i = 0;
                while(c_aux!=' ' && file_i<10) {
                        c[file_i] = c_aux;
                        c_aux = getc(p_file);

                        file_i++;

                }
                nr = atof(c);
                //printf("%2.3f   ", nr);

                if (i==w) {
                        l[xy][psa] = nr;
                        printf("%2.3f   ", l[xy][psa]);
                        xy ++;

                }
                while(c_aux==' '||c_aux=='\n') {
                        c_aux = getc(p_file);

                }

                }
        printf("\n");
        }
        printf("\n");

        }
```

```
        fclose(p_file);
}

void generate_random_values(int n_random, float min_x, float min_y, float
max_x, float max_y) {

int r_i;
int r_v;
float r_v_x, r_v_y;
float delta_x;
int temp;

printf("\n\nArea limits: [%2.1f %2.1f %2.1f %2.1f] values: %d\n", min_x,
min_y, max_x, max_y, n_random);


for (r_i=0; r_i<n_random; r_i ++) {

                r_v = rand();
                r_v_x = min_x + r_v * (max_x - min_x)/maxint;
                r_v = rand();
                r_v_y = min_y + r_v * (max_y - min_y)/maxint;

                printf("(%2.1f, %2.1f)   ", r_v_x, r_v_y);
                fprintf(r_file, "%2.1f, %2.1f, \n", r_v_x, r_v_y);


        }
}

void random_values_in_areas(int n_psa, int t_sets) {
        int i;
        int d_r;
        int nr;
        int b_int[4];
        int b_idx;


        r_file = fopen("c:\\C_dir\\random_xy.txt", "w");
        o_file = fopen("c:\\C_dir\\targets.txt", "w");

        fprintf(r_file, "Random points in areas:\n");
        fprintf(o_file, "Targets for the clasifficator:\n");

        for(i = 0; i<o; i++) {
                b_int[i] = 0;
        }

        for(i = 0; i<n_psa; i++) {
                b_idx = 0;
                nr = i;

                generate_random_values(t_sets,
l[0][i],l[1][i],l[2][i],l[3][i]);
                fprintf(r_file, "\n");

                while(nr > 0) {
                        d_r = nr % 2;
                        b_int[b_idx] = d_r;
                        nr = nr / 2;
                        b_idx++;
                }
                for (j = o-1; j>=0; j--) {
                        fprintf(o_file, "%d, ", b_int[j]);
                }

        fprintf(o_file, "\n");

        }

        fclose(r_file);
```

```
        fclose(o_file);
}

void read_input_outputs(int n, int o, int n_areas, int p) {

char c_aux;
int file_i;
int i_o, c_i;
int i_reads, n_reads;

        c_aux = getc(o_file);

        printf("%c", c_aux);

        while(c_aux != ':' && c_aux != EOF) {
                c_aux = getc(o_file);
//              printf("%c", c_aux);
        }

//      printf("ok\n");

        n_reads = (n + o) * p * n_areas;

        i_o = 1;
        for(i_reads=0; i_reads<n_reads; i_reads++) {

                c_aux = getc(o_file);

                while(c_aux == ' ' || c_aux == '|' || c_aux == '\n') {

                        c_aux = getc(o_file);
                        if (c_aux == '|') {
                                i_o = 0;
                        }
                        if (c_aux == '\n') {
                                i_o = 1;
                        }
                }

                file_i = 0;
                for (c_i = 0;c_i < 10;c_i++) {
                        c[c_i] = 0;
                }
                c[file_i] = c_aux;

//              printf("ok\n");

                while(c_aux!=',' && file_i<10) {
                        c[file_i] = c_aux;
                        c_aux = getc(o_file);
                        file_i++;


                        }
                        if (c_aux == EOF) {
                                printf("Error reading file\n EndOfFile
reached\n");
                        } else {
                                        if (i_o == 1) {
                                                v[idx++] = atof(c);
                                        }
                                        if (i_o == 0) {
                                                d[idx_out++] = atof(c);
                                        }
//                                      printf("%s\n ", c);

                                }

        }

        fclose(o_file);
```

```
}


void read_file(int n, int p) {

char c_aux;
int file_i;
int c_i;

        c_aux = getc(o_file);

        printf("%c", c_aux);

        while(c_aux != ':' && c_aux != EOF) {
                c_aux = getc(o_file);
//              printf("%c", c_aux);
        }

        printf("ok\n");

        for(i=0; i<n; i++) {
                for(j=0; j<p; j++){

                c_aux = getc(o_file);

                while(c_aux == ' ' || c_aux=='\n') {
                        c_aux = getc(o_file);

                }

                file_i = 0;
                for (c_i = 0;c_i < 10;c_i++) {
                        c[c_i] = 0;
                }
                c[file_i] = c_aux;

//              printf("ok\n");

                while(c_aux!=',' && file_i<10) {
                        c[file_i] = c_aux;
                        c_aux = getc(o_file);
                        file_i++;


                        }
                        if (c_aux == EOF) {
                                printf("Error reading file\n EndOfFile
reached\n");
                        } else {
                                        v_data[idx++] = atof(c);
//                                      printf("%s\n ", c);

                                }
                }
        }

        fclose(o_file);
        printf("\nNumber of readings: %d\n", idx);
}

void write_training_sets_to_file(float v[], float d[]) {

int in_idx, k, i, j, o_j;

        r_file = fopen("c:\\C_dir\\inputs.txt","w");
        fprintf(r_file, "training sets  -   outputs:\n");

        for(i=0;i<t_sets*n;i+=n) {

        for(j=0;j<n_areas;j++) {
```

```
        for(in_idx=0;in_idx<n;in_idx++) {
                fprintf(r_file, "%2.1f, ", v[j*t_sets*n + i + in_idx]);

        }

                o_j = j*o;
                fprintf(r_file, "| ");

                for(k=0;k<o;k++) {
                        fprintf(r_file, "%2.0f, ", d[o_j + k]);

                }

                fprintf(r_file, "\n");

                }
        fprintf(r_file, "\n");
        }

        fclose(r_file);
        }


void read_weights(int k, int n, int m) {

        int file_i;
        char c_aux;
        char c[10];

        c_aux = getc(w_file);

        printf("%c", c_aux);

        while(c_aux != ':' && c_aux != EOF) {
                c_aux = getc(w_file);
                printf("%c", c_aux);
        }

        for(i=0; i<n; i++) {
                for(j=0; j<m; j++){

                c_aux = getc(w_file);

                while(c_aux == ' ' || c_aux=='\n') {
                        c_aux = getc(w_file);
                }

                file_i = 0;
                c[file_i] = c_aux;

                        while(c_aux!=',' && file_i<10) {
                        c[file_i] = c_aux;
                        c_aux = getc(w_file);
                        file_i++;

                        }
                        w[i][j][k] = atof(c);

                        printf("%s\n ", c);
                }
        }
}

void input_weights(int k, int n, int m) {

float r;

        printf("Layer: %d \n", k);
        if (f==2) {
                read_weights(k, n, m);
        } else {
```

```
        for (i=0;i<n;i++) {
                for(j=0;j<m;j++)
                {
                        if(f==1) {
                        printf("w[%d][%d]:",i, j);
                        cin >> w[i][j][k] ;     //input weights
                        } else {
                                r = rand();
                                w[i][j][k]=r/maxint; //random weights

                                w[i][j][k]=w[i][j][k];
                        }
                }
        }
        }
};

void display_weights(int sv, int k, int n, int m) {

        FILE *w_file;
// write weights to file

                if (sv == 1) {
                        w_file = fopen("c:\\C_dir\\w_final.txt","a");
                        fprintf(w_file, "\nlayer %d: \n", k);
                        for (i=0;i<n;i++) {
                        for(j=0;j<m;j++) {
                                fprintf(w_file, "%2.3f, ", w[i][j][k]);
                        }
                        fprintf(w_file, "\n");
                        }
                        fclose(w_file);

                } else {

                if (f < 2) {
                if (f == 0) {
                        w_file = fopen("c:\\C_dir\\w_random.txt","a");
                }

                if (f == 1) {
                        w_file = fopen("c:\\C_dir\\w_keyb.txt","a");
                }

                fprintf(w_file, "\nlayer %d: \n", k);
                for (i=0;i<n;i++) {
                        for(j=0;j<m;j++) {
                                fprintf(w_file, "%2.1f, ", w[i][j][k]);
                        }
                fprintf(w_file, "\n");
                }

                fclose(w_file);

                }
                }

//display weights
        printf("layer: %d\n", k);

        for (i=0;i<n;i++) {
                for(j=0;j<m;j++)
                {
                        printf("%2.1f ", w[i][j][k]);
                }
        printf("\n");
        }

};

void calculate_w_sums_sig(int k, int n, int m) {
```

```
                for(j=0;j<m;j++) {

                s[j][k]=0;

                        for(i=0;i<n;i++) {
                                //weighted sum computation

                                s[j][k] = s[j][k] + x_h[i][k]*w[i][j][k];

                }

                        y[j] = 1/(1 + exp(-s[j][k])); // sigmoid function
implementation

                }

        for(j=0;j<m;j++)
        {
                x_h[j][k+1] = y[j];
        }

};

void calculate_w_sums(int k, int n, int m) {

        for(j=0;j<m;j++) {

                s[j][k]=0;

                        for(i=0;i<n;i++) {
                                //weighted sum computation
                                s[j][k] = s[j][k] + x_h[i][k]*w[i][j][k];

                }
                        y[j] = 0;
                        if (s[j][k]>=teta) {
                                y[j] = 1; //step function implementation

                        }
                }
        for(j=0;j<m;j++)
        {
                x_h[j][k+1] = y[j];
        }
};


void adjust_hidden_weights_sig(int k, int n, int m) {

float delta_w, x_h_sig;
int j_old;
//claculate weighted errors on hidden layers
        for (j=0;j<m;j++) {

                err_factor[j][k] = 0;

        for (j_old=0;j_old<m_old;j_old++) {

                err_factor[j][k] = err_factor[j][k] + e[j_old][k + 1] *
w[j][j_old][k + 1];

                }
        }

        for (j=0;j<m;j++) {

                x_h_sig = x_h[j][k + 1];
                e[j][k] = x_h_sig * (1 - x_h_sig) * err_factor[j][k];
                printf("error factor: %2.6f\n", err_factor[j][k]);
```

```
    for(i=0;i<n;i++) {

                printf("error delta: %2.6f for input: %2.3f\n", e[j][k],
x_h[i][k]);
                delta_w = e[j][k] * x_h[i][k];
                w[i][j][k] = w[i][j][k] + alfa * delta_w;

        }
        }

                m_old = m; //store the number of neurons for layer k+1 in
m_old
}

void adjust_output_weights_sig(int k, int n, int m) {

float delta_w, d_sig;
float err_out;

        err_out = 0;

        for (j=0;j<m;j++) {

        if(sig == 1) {
                d_sig = 1/(1 + exp(-d[r_idx+j]));
        } else {
                d_sig = d[r_idx+j];
        }

                err_factor[j][k] = d_sig - x_h[j][k + 1];


                err_out = err_out + (err_factor[j][k] *
err_factor[j][k])/2;

                e[j][k] = x_h[j][k + 1]*(1-x_h[j][k + 1]) *
err_factor[j][k];

                printf("generated output: %2.1f desired output: %2.1f \n",
d_sig, x_h[j][k + 1]);

                //err_output = err_output + (e[j][k] * e[j][k])/2;
                err_output = err_output + (err_factor[j][k] *
err_factor[j][k])/2;


    for(i=0;i<n;i++) { //adjust weights for current layer

                printf("error delta: %2.3f for input: %2.3f\n", e[j][k],
x_h[i][k]);
                delta_w = e[j][k] * x_h[i][k];
                w[i][j][k] = w[i][j][k] + alfa * delta_w;
                }
        }

        //fprintf(sqerr_file, "square error: %2.6f \n", err_out);
        m_old = m; //store the number of neurons for layer k+1 in m_old

}

void adjust_hidden_weights(int k, int n, int m) {

float delta_w;
int j_old;

        printf("m old = %d\n", m_old);
        printf("Adjust weights hidden layer: %d\n", k);

        for (j=0;j<m;j++) {

                err_factor[j][k] = 0;
```

```
        for (j_old=0;j_old<m_old;j_old++) {

                err_factor[j][k] = err_factor[j][k] + e[j_old][k + 1] *
w[j][j_old][k + 1];

                }
        }

        for (j=0;j<m;j++) {

                e[j][k] = err_factor[j][k];

                printf("error delta: %2.3f\n", e[j][k]);

    for(i=0;i<n;i++) {
                delta_w = e[j][k] * x_h[i][k];
                printf("x_h: %2.3f\n", x_h[i][k]);
                w[i][j][k] = w[i][j][k] + alfa * delta_w;

        }
        }

                m_old = m; //store the number of neurons for layer k+1 in
m_old
}

void adjust_output_weights(int k, int n, int m) {

float d_sig, delta_w;

                r_idx = vector_idx * m;

        for (j=0;j<m;j++) {

                d_sig = d[r_idx + j];
                err_factor[j][k] = d_sig - x_h[j][k + 1];

                e[j][k] = err_factor[j][k];
                printf("index: %d desired output: %2.1f\ngenerated output:
%2.1f\n", vector_idx, d_sig, x_h[j][k + 1]);
                printf("error factor: %2.3f\n", err_factor[j][k]);
                printf("error delta: %2.3f\n", e[j][k]);
                err_output = err_output + (e[j][k] * e[j][k])/2;

                p = p + 1;

    for(i=0;i<n;i++) {

                delta_w = e[j][k] * x_h[i][k];
                w[i][j][k] = w[i][j][k] + alfa * delta_w;

                }
        }

        m_old = m; //store the number of neurons for layer k+1 in m_old
}

void display_sums(int k, int m) {

        FILE *s_file;

        s_file = fopen("c:\\C_dir\\sum_file.txt", "a");

        printf("layer: %d\n", k);
        fprintf(s_file, "layer: %d\n", k);

        for (j=0;j<m;j++) {

                printf("%2.1f ", s[j][k]);
                fprintf(s_file, "%2.3f ", s[j][k]);
```

```
                }

        printf("\n");
        fprintf(s_file, "\n");
        fclose(s_file);
};

void read_data(int nofvalues) {

        char c_aux;
        int idx;
        int file_i, t;
        FILE *sig_file;
        sig_file = fopen("c:\\C_dir\\record.txt", "r");
        if (sig_file == NULL) {
                printf("error opening file \n");
        }
        cout << "start \n";
        file_i = 0;
        t = -1;
        while(t != 0 && c[i] != EOF) {
                for(i=0;i<2;i++) {

                c[i] = getc(sig_file);
                t = strcmp(c, "mV");

                printf("%d %s\n", t, c);

        }
                file_i++;
        }

        for(idx=0;idx<nofvalues;idx++)
        {

        c_aux = getc(sig_file);
        while(c_aux!=',' && c_aux!=EOF){
                c_aux = getc(sig_file);
        }

                printf("%c", c_aux);

        file_i = 0;
        c[file_i] = getc(sig_file);

        while(c[file_i]!='\n' && c[i]!=EOF){
                file_i++;
                c[file_i] = getc(sig_file);
        }

        v[idx] = atof(c);

        printf("%2.3f", v[idx]);
        }

        fclose(sig_file);
}

void generate_exp(){
        float x_i;
        float y_i;
        x_i = -2.8;
        while (x_i<2.8){
                x_i = x_i + 0.1;
                y_i = 1/(1 + exp(-x_i));
                printf("%2.3f ", y_i);
        }
}

void write_data(int nofvalues, float v[], int fil) {
        FILE *wr_file;
```

```
        int idx;

        if(fil==0) {
                wr_file = fopen("c:\\C_dir\\output_y.txt","w");
        }
        if(fil==1) {
                wr_file = fopen("c:\\C_dir\\y_noise.txt","w");
        }

        if(fil==2) {
                wr_file = fopen("c:\\C_dir\\s_noise.txt","w");
        }

        for(idx=0;idx<nofvalues;idx++) {

                fprintf(wr_file, "%2.3f\n", v[idx]);

        }

        fclose(wr_file);
}

void save_weights(int sv) {

        printf("\nWeight mattrix:\n");
        k = 0;
        display_weights(sv, k, n, m);

//hidden layers
        for(k=1;k<h;k++)
        {
                display_weights(sv, k, m, m);
        }

 //output layer
        k = h;
        display_weights(sv, k, m, o);

}

void gen_MLP_output(int nofvalues, float v[]) {

        int n_random, n_chosen, v_i, random_flag[1000], random_idx[1000];
        FILE *chosen_file;

        idx_out = 0;
        idx = 0;
        a = 1;
        p = 0;
        digital_err = 0;
        err_output = 0;
        vector_idx = -1;

        n_random = n_areas * t_sets;

        for(i=0; i<n_random; i++) {
                random_idx[i] = i;
        }

        chosen_file = fopen("c:\\C_dir\\chosen.txt", "w");

        fprintf(chosen_file, "chosen indexes out of %d:\n", n_random);

        while(idx < nofvalues) {

// read inputs   READ INPUTS
-----------------------------------------------

        rand_v = rand();
        v_i = rand_v * n_random/maxint;
        vector_idx = random_idx[v_i];
```

```
//      vector_idx ++;

        fprintf(chosen_file, "idx: %d \n", vector_idx);

        for(i = v_i; i<n_random;i++) {

                random_idx[i] = random_idx[i + 1];

        }

        n_random --;

        //fprintf(out_file, "index: %d inputs:", vector_idx);
        //fprintf(sqerr_file, "index: %d inputs:", vector_idx);
        r_idx = vector_idx * n;

                for(i=0;i<n;i++){
                x[i] = v[r_idx+i]/div_factor;
                //fprintf(out_file, "%2.3f ", x[i]);
                //fprintf(sqerr_file, "%2.3f ", x[i]);
                printf("Input x[%d]=%2.3f \n", i, x[i]);
                idx++;
        }

        //fprintf(out_file, "\n");
        k = 0;

        for(i=0;i<n;i++)
        {
                x_h[i][k] = x[i]; //x_h the input of the hidden/output
layer
        }


        r_idx = vector_idx * o;

// forward propagation of inputs
        if(sig==1) {
// activation function is sigma
// propagation through input layer
                calculate_w_sums_sig(k, n, m);
// propagation through hidden layers
                for(k=1;k<h;k++) {
                        calculate_w_sums_sig(k, m, m);
                }
// propagation through output layers
        k = h;
        calculate_w_sums_sig(k, m, o);


        } else {
//activation function is step function
                calculate_w_sums(k, n, m);

                for(k=1;k<h;k++) {
                        calculate_w_sums(k, m, m);
                }
        k = h;
        calculate_w_sums(k, m, o);
        }

// getting network outputs
        printf("\nNetwork outputs:\n");
        for (j=0;j<o;j++) {
                printf("%2.1f \n", y[j]);
                y_aux[idx_out] = y[j];
                idx_out ++;
        }
```

```
// writing generated outputs and target output to file "gen_out.txt"


        if (sig == 1) {

                generated_patt = 0;
                current_patt = 0;

                for(i=0;i<o;i++) {
                        if(x_h[i][k+1] >= teta) {
                                y_dig[i] = 1;
                        } else {
                                y_dig[i] = 0;
                        }

                        if( 1/(1 + exp(-d[r_idx+i]))>=teta ) {
                                dig_d_sig[i] = 1;
                        } else {
                                dig_d_sig[i] = 0;
                        }


                        //fprintf(out_file, "idx: %d gen: %2.3f target:
%2.3f\n", r_idx/n, y_dig[i], dig_d_sig[i]);
                        digital_err = digital_err + (y_dig[i] -
dig_d_sig[i]) * (y_dig[i] - dig_d_sig[i])/2;

                }
                for(i=0;i<o;i++) {
                        generated_patt = generated_patt + y_dig[o-i-1] *
pow(2, i);
                        current_patt = current_patt + dig_d_sig[o-i-1] *
pow(2, i);
                }
                mat_conf[generated_patt][current_patt] =
mat_conf[generated_patt][current_patt] + 1;
                fprintf(out_file, "gen patt: %d target patt: %d \n",
generated_patt, current_patt);
        } else {
        /*      for(i=0;i<o;i++) {
                fprintf(out_file, "idx: %d gen: %2.3f target: %2.3f\n",
r_idx/n, x_h[i][k + 1], d[r_idx+i]);
        }*/
        }

// adjusting output weights
        k = h;
        printf("Adjusting output weights...\n");
        printf("layer:%d\n", k);

        if(sig == 1) {
                adjust_output_weights_sig(k, m, o);
        } else {
                adjust_output_weights(k, m, o);
        }
//
//adjustting hidden weights
        printf("Adjusting hidden layer weights...\n");

        for (k=h-1;k>=0;k--) {
                printf("layer:%d\n", k);
                if (k==0) {
                        n_i = n;
                } else {
                        n_i = m;
                }

                if (sig == 1){
                        adjust_hidden_weights_sig(k, n_i, m);
                } else {
                        adjust_hidden_weights(k, n_i, m);
```

```
                }
        }
        //fprintf(out_file, "\n");
        }

        fclose(chosen_file);
}
void save_conf_mattrix() {

float patt_percent;

        recog_patt = 0;
        conf_file = fopen("c:\\C_dir\\conf_matt.txt", "w");
        fprintf(conf_file, "    ");
        for(i = 0; i<n_areas; i++) {
                fprintf(conf_file, "P%d,", i);
        }
                fprintf(conf_file, "\n ");
                for(i=0;i<n_areas;i++) {
                fprintf(conf_file, "P%d,  ", i);
                        for(j=0;j<n_areas;j++) {
                        fprintf(conf_file, "%d,  ", mat_conf[i][j]);
                        }
                fprintf(conf_file, "\n");
                }

        fclose(conf_file);

        conf_file = fopen("c:\\C_dir\\conf_percent.txt", "w");
        fprintf(conf_file, "    ");
        for(i = 0; i<n_areas; i++) {
                fprintf(conf_file, "P%d,", i);
        }
                fprintf(conf_file, "\n ");
                for(i=0;i<n_areas;i++) {
                fprintf(conf_file, "P%d,  ", i);
                        for(j=0;j<n_areas;j++) {
                        patt_percent = mat_conf[i][j]*100/t_sets;
                        fprintf(conf_file, "%d,  ",
mat_conf[i][j]*100/t_sets);
                        if(i==j && patt_percent >= 95) {
                                recog_patt ++;
                        }
                        }
                fprintf(conf_file, "\n");
                }

        fprintf(conf_file, "\n recognizable patterns: %d/%d\n", recog_patt,
n_areas);
        fclose(conf_file);
}

main()
 {

//program parameters
        m = 12; //number of neurons on hidden layers

//nominal conditions:
        t_sets = 40; //40       // number of training sets (all
combinations of n inputs)
        n_areas = 16; //16  // number of classification sets

//parameter variation
        printf("\nLearning rate:");
        alfa = 0.1;
        printf("%2.5f\n", alfa);

//constants
        idx_out = 0;
        p_noise = 0.8;
```

```
        area_corners = 4;
        teta = 0.6;
        ef_signal = 0;
        wf = 0;
        r_seed = 2;

        n=8; //number of inputs
        h=1; //number of hidden layers

        o=4; //number of outputs

        sig = 0; // sig = 0 for step function / sig = 1 for sigmoid
function
        max_it = 1; //maximum number of iterations

// random
//      srand(1);

        s_file = fopen("c:\\C_dir\\sum_file.txt", "w");
        fclose(s_file);

//  activate these for new copies

//      read_window_limits(n_areas, area_corners); //read window limits
from file
//      random_values_in_areas(n_areas, t_sets); // generate random values
to file

// preset neural network structure

        for(i=0;i<1000;i++) {
                d[i]=-1;
                v[i]=-1;
        }

        cout << "Reading inputs from file...";
        idx = 0;
        o_file = fopen("c:\\C_dir\\random_xy.txt", "r");
        read_file(n, t_sets * n_areas);

        for(i=0; i<idx; i++) {
                if (i % 2 == 0) {
                        printf("\n");
                }
                if (i % (t_sets*2) == 0) {
                        printf("\n");
                }

                v[i] = v_data[i];
                printf("%2.1f ",v[i]);

        }

        cout << " OK\n";
        cout << "Reading target outputs from file...";
        idx = 0;
        idx_out = 0;
        o_file = fopen("c:\\C_dir\\targets.txt", "r");
        read_file(o, n_areas);

        for(i=0; i<idx; i++) {
                d_aux[i]=v_data[i];
//              printf("%2.1f ",d_aux[i]);

        }

        cout << "OK\n";

        write_training_sets_to_file(v, d_aux); //write training sets to
file "inputs.txt"
```

```
        o_file = fopen("c:\\C_dir\\inputs.txt", "r");

        idx = 0;
        idx_out = 0;

        read_input_outputs(n, o, n_areas, t_sets); //read inputs and
outputs for training sets

        printf("Inputs: %d  Outputs: %d\n", idx, idx_out); //number of read
inputs/outputs

// display read inputs

        for(i=0;i<n;i++) {
                min_v[i] = maxint;
                max_v[i] = -1;
        }

        printf("Inputs: \n");
        for(i=0; i<idx; i++) {

                m_idx = i % n;

                if (i % n == 0) {
                        printf("\n");
                }
                if (i % (n_areas * n) == 0) {
                        printf("\n");
                }

                if(max_v[m_idx] < v[i]) {
                        max_v[m_idx] = v[i];
                }
                if(min_v[m_idx] > v[i]) {
                        min_v[m_idx] = v[i];
                }
                printf("%2.1f ",v[i]);

        }

        printf("\nmin   max  \n");
        for(i=0;i<n;i++) {
                printf("%2.3f %2.3f\n", min_v[i], max_v[i]);
        }

        printf("Press any key...\n");

        getch();

//      srand(r_seed);

//display read outputs

        printf("\nOutputs: \n");
        for(i=0; i<idx_out; i++) {
                if (i % o == 0) {
                        printf("\n");
                }
                if (i % (n_areas*o) == 0) {
                        printf("\n");
                }

                printf("%2.1f ",d[i]);

        }

        nofvalues = t_sets * n_areas * n;

        printf("\n\nPlease select activation function:\n 1 - sigmoid\n 0 -
unit step function (threshold = 1)\nYour option:");
```

```
        cin >> sig;

        printf("\n\nPlease choose:\n 2 - read weights from file\n 1 - input
weights from keyboard\n 0 - random weights\nYour option:");
        cin >> f;

        printf("\nNumber of itterations:");
        cin >> max_it;

        //cin >> alfa;

        if (f == 0) {
                w_file = fopen("c:\\C_dir\\w_random.txt","w");
                fclose(w_file);
        }

        if (f == 1) {
                w_file = fopen("c:\\C_dir\\w_keyb.txt","w");
                fclose(w_file);
        }

        if (f == 2){
                cout << "Please choose: \n 1 - load the last input weights
from keyboard \n 0 - load the last randomly generated weights ";
                cin >> wf;
                if(wf == 0) {
                        w_file = fopen("c:\\C_dir\\w_random.txt","r");
                }
                if(wf == 1) {
                        w_file = fopen("c:\\C_dir\\w_keyb.txt","r");
                }
        }

// scale input vector to input interval for sigma

        if(sig == 1) {

                div_factor = 1;
                sig_limit_down = -2;
                sig_limit_up = 2;
                input_int = sig_limit_up - sig_limit_down;
                for(i=0;i<idx;i++) {
                        m_idx = i % n;
                        current_int = max_v[m_idx] - min_v[m_idx];
                        printf("%2.3f %2.3f %2.3f\n", v[i], min_v[m_idx],
max_v[m_idx]);
                        v[i]=(v[i] - min_v[m_idx]) * input_int/current_int
- (input_int/2);
                        printf("%2.3f \n", v[i]);
                }
        } else {
                div_factor = 1000;
        }

        k = 0;
        input_weights(k, n, m);

// randomize

// hidden layers
        for (k=1;k<h;k++){

                input_weights(k, m, m);

        }

//output layer
        k = h;
        input_weights(k, m, o);

        if (f==2){
```

```
                fclose(w_file);
        }

        save_weights(0);


//train network
        //out_file = fopen("c:\\C_dir\\gen_out.txt", "w");
        err_file = fopen("c:\\C_dir\\o_error.txt", "w");

        iteration = 0;
        //fprintf(out_file, "Outputs all layers: \n");
        fprintf(err_file, "Output errors: \n");
        input_set = 0;

// train network using backpropagation
        min_err = 1000;
        stat = fopen("c:\\c_dir\\statiscs.txt", "a");
        //sqerr_file = fopen("c:\\c_dir\\sq_err.txt","w");

        do {

                for(i=0;i<n_areas;i++) {
                        for(j=0;j<n_areas;j++) {
                                mat_conf[i][j] = 0;
                        }
                }

                gen_MLP_output(nofvalues, v);

//              save_weights(1);

                iteration = iteration + 1;

                digital_err = digital_err/(n_areas * t_sets);
                sq_err = err_output/(n_areas * t_sets);

                printf("Output error: %2.6f\n", sq_err);

                fprintf(err_file, "Iteration: %d Output error: %2.6f",
iteration, sq_err);

                if(sig == 1) {
                        fprintf(err_file, " 1/0 error: %2.3f",
digital_err);
                }

                fprintf(err_file, "\n");

                //fprintf(out_file, "\nIteration:%d\n", iteration);

                if (min_err > sq_err) {
                        min_err = sq_err;
                        w_file = fopen("c:\\C_dir\\w_final.txt","w");
                        fclose(w_file);
                        // save the new set of weights
                        save_weights(1);
                        save_conf_mattrix();
                }

        } while(digital_err>0 && iteration<max_it);

        printf("sig = %d\n", sig);
        if(sig == 0) {
                fprintf(stat, "step ");
        }
        if(sig == 1) {
                fprintf(stat, "sigma ");
        }

        fprintf(stat, "%d, %d, %d, %d, %d, %2.5f,   %d,   %2.3f", n, h, m,
```

```
o, t_sets, alfa, max_it, min_err);

        fprintf(stat, "\n");
        //fclose(out_file);
        fclose(stat);
        fclose(err_file);
        //fclose(sqerr_file);

        printf("Number of iterations:%d\n", iteration);
        printf("Minimum error: %2.6f\n", min_err);
        printf("Recognizable patterns: %d/%d\n", recog_patt, n_areas);
        printf("\n Press any key...\n");
        getch();

  }
```