

# Appendix-A

## Software for Bayesian neural networks training

- Software for training Bayesian neural networks, called **Model Manager**. It is a collection of scripts working around a **core optimization algorithm due to David MacKay**. This core algorithm is what does the real job, but it is a research tool and, understandably, no efforts were made by the author to make it user-friendly. It is command-line (OK so far), hardly documented and all pre- and post- processing had to be done separately. Also, if you wanted to build a committee of models (as then recommended by the authors), you just had to write your own script for combined errors etc. The software provided is a collection of tcl scripts with a tk interface, and C code provided to carry out all the pre-processing of your data and all the post-processing including graphs etc.
- Software for making predictions with your trained models, aptly called **Predictor**. There again, a lot of scripts and C code around a core algorithm by **David MacKay**. Models are installed as downloadable packages or transferred from the Model Manager.

On the requirements:

- All these softwares are intended for Linux.
- Training a full model (150 models tested) on a database with 3000 entry lines and 20 variables will take a couple of days on a P4-3GHz.

History:

- This software was in great part written by PhD students at Cambridge University. Most of data processing C code is based on methods by **Dr Sree Harsha Lalam**, and other programming by **Dr. Thomas Sourmail**.

Website of **Neuromat Software: Model Manager and Predictor**

<https://thomas-sourmail.net/publications.html>

# **Appendix-B**

## **Provenance of Source Code**

Anne Delorme  
Phase Transformations Group,  
Department of Materials Science and Metallurgy,  
University of Cambridge,  
Cambridge CB2 3QZ, U.K.

## **Purpose**

This program is an implementation of a genetic algorithm which can reach an optimum set of parameters for a target value of yield strength of austenitic stainless steel. This can in theory be applied to any problem, where a neural network exists.

## **Specification**

Language: C  
Product form: Source code

### **How to run the GA to find an optimum set of parameters for a desired yield strength of austenitic stainless steel:**

- First you have to define in the "values" file the inputs you wish to vary or those you wish to fix and the corresponding fixed value
- Then, you must normalised the desired target value of yield strength and enter it in the "nn-input" file, as well as the wanted accuracy.
- Finally, compile the C program "ga\_code" and execute it.

### **How to adapt this GA to optimise other mechanical properties:**

- First, you have to put in the folder "gacode" all the files related to the neural network created for the mechanical properties you want optimise. This files are the following:

generate44.exe  
norm\_test.in  
\_w\*f  
\*.lu  
spec1.tl  
outran.x  
MINMAX

- Then, write the labels of the inputs of the neural network in the "labels.tct" file
- Change as well the file "values" to have the right corresponding parameters in the first column
- Repeat the steps described in the previous paragraph

### **How to change GA parameters:**

The efficiency of the GA depend on the values of parameters such as the number of population, the number of generations ... which are not defined for any problem but must be adapted for each GA. The values are entered directly in the "ga\_code.c" file so you have to edit it and change the desired values in the header of the file.

## Source Code

```
/*
 * File name : annel5_output.c
 *
 * This program is an implementation of a Neural Network (NN) which
attempts to
 * reach a target, by using a optimal search method based on this value
using
 * Genetic Algorithms (GA).
 *
 * It initially chooses several random input datasets (depending on the
number
 * of populations) and various predictions are made on these inputs. The
 * error involved is used to "rank" the predictions.
 * The values with the lowest errors are seen as the "fittest" and are
more
 * likely to reproduce and get closer to the target. The process can stop
after
 * the prescribed number of generations, or if the target has been reached
 * before the max. number of generations requested.
 *
 */

#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <sys/time.h>

/* NN related */
#define LIMIT      150 /* Maximum number of inputs the system can handle
 */
#define SESSIONS  10000 /* Number of generations we'll put the system
through */

/* GA related */
#define POPS       3 /* Number of populations */
#define MAXPOP     20 /* Size of population */
#define BESTPOP    1 /* Number of individuals taken from the best */
#define CROSSPOP   19 /* CROSSPOP-BESTPOP: number of individuals
recombined */
#define LMUTPOP    18 /* LMUTPOP-CROSSPOP: number of individuals slightly
mutated */
#define HMUTPOP    19 /* HMUTPOP-LMUTPOP: number of individuals hardly
mutated */
```

```

#define MIXPOP      200  /* Number of generations between population mixing
*/

int e=0;
int NUM=-1;
int SIZE=0;
char name[20][10];

/* NN related */
float target_0,target_1,target_2;
float *w1;
int total;

/* GA related */
float ***pop;
float ***temp_pop;
double score[POPS][MAXPOP];
double result[POPS][MAXPOP];
double error[POPS][MAXPOP];
float *f,*low,*high;
float *sort_low,*sort_high;
int *v;
int *x;

/*-----
*\
|
|
|   Randomize
|
|
|
|
\*-----
*/

randomize()
{
    struct timeval tp;
    struct timezone tzp;
    /* Use time of day to feed the random number generator seed */
    gettimeofday( &tp, &tzp);
    srandom( tp.tv_sec );
}
/*-----
*\
|
|
|   Get data (reads input file)
|
|
|
|
\*-----
*/

```

```

int get_data()
{
    FILE *fd;

    /* Total number of input values */
    total = 0;

    /* opens the file */
    if ( (fd = fopen("nn-input","r")) == NULL )
    {
        printf ("no-input-file");
        exit(1);
    }
    else
    {
        fscanf(fd,"%f%f",&target_0,&target_2);
        printf("Target is %f\nWanted accuracy is %f\n",target_0,target_2);
    }
    fclose( fd );
    return (0) ;
}

/*-----
*\
|
|
|   Return the number of cases for which the NN returns the correct value
|
|
|
|
\*-----
*/

calc_limit_err()
{
    float err=0;

    err = pow((target_2*target_0),2);
    target_1 = 1.0/sqrt(err) ;
    printf("Target Score is: %f\n",target_1);
    return err;
}

/*-----
*\
|
|
|   Initialize the populations
|
|
|
|
\*-----
*/

```

```

make_initial_population()
{
    int p,i,k,y[20];
    int j,l;
    char label[60];
    FILE *afp;
    float depop_fe, depop_y[20],sum=0;

    /* read low and high values for the SIZE inputs which vary */
    if((afp=fopen("values","r"))==NULL)
    {
        printf("Error:can't read the file Values");
        exit(1);
    }
    else
    {
        l=0;
        fgets (label,sizeof(label),afp);
        for (j = 0; j < NUM; j++)
        {
            fscanf(afp,"%s%d%d%f%f%f",name[j],&x[j],&v[j],&f[j],&low[j],&high[j]);
            if (v[j]==1)
            {
                sort_low[l]=low[j];
                sort_high[l]=high[j];
                ++l;
            }
        }
    }
    fclose(afp);
    for ( p = 0 ; p < POPS ; p++ )
    {
        for ( i = 0 ; i < MAXPOP ; i++ )
        {
            sum =0;
            for ( k = 0 ; k < (SIZE-1) ; k++ )
            {
                y[k] = irand(SIZE-1);
                for ( l = 0; l < k ; l++)
                {
                    while (y[k] == y[l] )
                    {
                        y[k] = irand(SIZE-1);
                        l = 0;
                    }
                }
                depop_y[k] = (random()&1000000) / 10000.0 ;
                sum += depop_y[k];
            }
            depop_fe = (random()&1000000) / 10000.0 ;
            sum += depop_fe;
            for ( k = 0 ; k < (SIZE-1) ; k++ )
            {

```





```

/* vary, otherwise the normalised fixed value
*/if ((nmin = fopen("norm_test.in","w")) == NULL )
{
    printf ("Can't write norm_test.in\n");
    exit(1);
}
else
{
    l = 0;
    for ( j = 0; j < NUM ; j++)
    {
        if (v[j]==1)
        {
            fprintf(nmin,"%f\n",w1[l]);
            if (!strcmp(name[j],"Ti"))
                value_Ti = (w1[l]+0.5)*(high[j]-low[j])+low[j];
            else
            {
                if (!strcmp(name[j],"Nb"))
                    value_Nb = (w1[l]+0.5)*(high[j]-low[j])+low[j];
                else
                {
                    if (!strcmp(name[j],"C"))
                        value_C = (w1[l]+0.5)*(high[j]-low[j])+low[j];
                    else
                    {
                        if (!strcmp(name[j],"N"))
                            value_N = (w1[l]+0.5)*(high[j]-low[j])+low[j];
                    }
                }
            }
        }
        l++;
    }
    else
    {
        if (v[j]==0)
        {
            fprintf(nmin,"%f\n", (f[j]-low[j])/(high[j]-low[j])-0.5);
            if (!strcmp(name[j],"Ti"))
                value_Ti = f[j];
            else
            {
                if (!strcmp(name[j],"Nb"))
                    value_Nb = f[j];
                else
                {
                    if (!strcmp(name[j],"C"))
                        value_C = f[j];
                    else
                    {
                        if (!strcmp(name[j],"N"))
                            value_N = f[j];
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    else
        fprintf(nmin,"%f\n",((((value_Ti/4)+(value_Nb/8))/(value_C+value_N)
)-low[j])/(high[j]-low[j]))-0.5);
    }
}
}
fprintf(nmin,"\n");
fclose(nmin);

/* count the number of models */
ofp = popen("ls _w*f","r"); na=0;
while((fgets(d,sizeof(d),ofp))!=NULL)
    ++na;
pclose(ofp);

/* read files _w*f in the current directory and attribute to model[nb]*/
ofp=popen("ls _w*f","r");
for (nb=1 ; nb<na+1 ; nb++)
    fscanf(ofp, "%s",&model[nb]);
pclose (ofp);

/*generate44 for each model[nb]*/
for(nb=1; nb < na +1 ; nb++)
{
    sprintf(s, "generate44 spec1.t1 %d %s %s.lu>
/dev/null",model[nb][2]-96,&model[nb],&model[nb]);
    system(s);
    if ( (nmin2 = fopen("_out","r")) == NULL)
    {
        printf ("Can't read _out stage 1\n");
        exit(1);
    }
    else
    {
        fscanf(nmin2,"%f %f",&res[nb+1],&err[nb+1]);
        printf("res%d err%d %f %f\n",nb+1,nb+1,res[nb+1],err[nb+1]);
        printf("Indices %d %d\n",p,i);

        fclose(nmin2);
    }
    sres += res[nb+1];
    serr += pow(err[nb+1],2);
    serr_ += pow ((res[nb+1]-target_0),2);
}

printf("thisisp %d %d\n", p,i);

result[p][i] = sres / na;
error[p][i] = sqrt((serr/na)+(pow((result[p][i] - target_0),2))) ;

printf("error[p][i] is %f\n\n",error[p][i]);

```

```

e++;

printf("\n-----> sum %d\n",e);
printf("In net() error is %f res is %f\n\n",error[p][i],result[p][i]);
}
/*-----
*\
|
|
|   Return the number of cases for which the NN returns the correct value
|
|
|
|
\*-----
*/

double check_performance (p,i)
    int p,i;
{
    float score_value=0.0;

    net(p,i);

    score_value = error[p][i];
    printf ("Answer is %f and target is %f\n",result[p][i],target_1);

    if (score_value < 0)
        score_value = 0;
    else
        score_value = 1.0/(score_value);

    printf("score_value is: %f\n",score_value);
    return score_value;
}
/*-----
*\
|
|
|   Calculate the scores of all the vectors in all the populations
|
|
|
|
\*-----
*/
*/
calc_score()
{
    int p, i, j;
    double check_performance();
    for ( p = 0 ; p < POPS ; p++ )
    {

```





```

|   Uniform Crossover: pop[p1][i1][j] and pop[p2][i2][j] are swapped
|
|   randomly
|
/*-----
*/

uniform_cross( p1, i1, p2, i2)
    int p1,i1,p2,i2;
{
    int j,k;

    for( j=0; j < SIZE; j++)
    {
        k = irand(2);
        if( k==0 )
            pop[p1][i1][j] = pop[p2][i2][j];
    }
}
/*-----
*\
|
|
|   Show (on the standard output) the best scores of all populations
|
|
|
/*-----
*/
statistics( generation )
    int generation;
{
    int p,i;
    printf("generationi is here: %d\n",generation);
    if ( generation % MIXPOP == 0 )
        printf("-----\n");
    printf(" %4d) First are: ", generation);
    for ( p = 0 ; p < POPS ; p++ ) printf("%f ", score[p][0] );
    printf(" (from %d)\n",total);
}
/*-----
*\
|
|
|   Generate an intermediate generation
|
|
|
/*-----
*/

roulette_wheel()
{

```

```

int p1, p2 = 0, i1, i2, j;
int r, integer_part;
double sum_score, sum_temp = 0;

for (p1 = 0; p1 < POPS; p1++ )
{
    sum_score = 0;
    for( i1 = 0 ; i1 < MAXPOP ; i1++ )
        sum_score += score[p1][i1];
    integer_part = sum_score;

    for( i2 = 0 ; i2 < MAXPOP ; i2++ )
    {
        i1 = 0;
        r = random()&integer_part ;
        sum_temp = score[p1][i1];
        while (sum_temp <= r )
        {
            ++i1;
            sum_temp += score[p1][i1];
        }
        for ( j = 0; j < SIZE ; j++ )
            temp_pop[p2][i2][j] = pop[p1][i1][j];
    }
    ++p2;
}
for (p1 = 0; p1 < POPS; p1++ )
    for( i1 = 0 ; i1 < MAXPOP ; i1++ )
        for ( j = 0; j < SIZE ; j++ )
            pop[p1][i1][j]=temp_pop[p1][i1][j];
}

/*-----
*\
|
|
|   Generate the next generation in all populations
|
|
|
|
\*-----
*/

make_next_generation( generation )
    int generation;
{
    int p, i, j, q, k, l, y[20];
    float dev, sum=0, depop_y[20], depop_fe;
    char c;
    FILE *afp;
    roulette_wheel();
    calc_score();
    sort_population();

```

```

for ( p = 0 ; p < POPS ; p++ )
{
    /* keep best - BESTPOP */

    /* crossover */
    for( i = BESTPOP; i < CROSSPOP ; i++)
    {
        if((generation%MIXPOP)==0)
        {
            q=irand(POPS);
            while(q==p)
                q=irand(POPS);
            uniform_cross( p, i, q, irand(CROSSPOP - i) + i);
        }
        else
            uniform_cross( p, i, p, irand(CROSSPOP - i) + i);
    }

    /* slight mutation */
    i = irand(CROSSPOP - BESTPOP) + BESTPOP;
    {
        j = irand(SIZE);
        dev = 1 + ((irand(2000) - 1000)/500000.0);
        pop[p][i][j] *= dev;
        while( pop[p][i][j] < (-(sort_low[j]/(sort_high[j]-
sort_low[j])+0.5)))
        {
            dev = 1 + ((irand(2000) - 1000)/500000.0);
            pop[p][i][j] *= dev;
        }
    }

    /* severe mutation */
    i = irand(CROSSPOP - BESTPOP) + BESTPOP;
    {
        j = irand(SIZE);
        dev = 1 + ((irand(2000) - 1000)/5000.0);
        pop[p][i][j] *= dev;
        while( pop[p][i][j] < (-(sort_low[j]/(sort_high[j]-
sort_low[j])+0.5)))
        {
            dev = 1 + ((irand(2000) - 1000)/5000.0);
            pop[p][i][j] *= dev;
        }
    }

    /* create new individuals */
    for ( i = CROSSPOP ; i < MAXPOP ; i++ )
    {
        sum = 0;
        for ( k = 0 ; k < (SIZE-1) ; k++ )
        {
            y[k] = irand(SIZE-1);
            for ( l = 0; l < k ; l++)
            {

```





```

|   Main
|
|
|
|
\*-----
*/

main()
{
    int generation, i, j, l, p, done = 0, k,x1,x2,w,z;
    float y;
    FILE *fb,*fscore,*afp,*fc;
    char c[10],label[60];
    float value_Ti,value_Nb,value_C,value_N;
    /* read in the file "values" the numbers of inputs (NUM) and the
    */
    /* numbers of which are allowed to vary (SIZE)
    */
    if(( afp = fopen("values","r")) == NULL )
    {
        printf("error: can't read file \"values\"");
        exit(1);
    }
    else
    {
        fgets(label,sizeof(label),afp);
        while(!feof(afp))
        {
            fscanf(afp,"%s%d%d%f%f",c,&w,&z,&y,&y,&y);
            if(z==1)
                ++SIZE;
            ++NUM;
        }
    }
    fclose(afp);
    /* define the size of the array *f,*v,*low,*high,*sort_low,*sort_high
    */
    /* *p,***pop
    */
    f=(float*)malloc(NUM*sizeof(float));
    low=(float*)malloc(NUM*sizeof(float));
    high=(float*)malloc(NUM*sizeof(float));
    sort_low=(float*)malloc(SIZE*sizeof(float));
    sort_high=(float*)malloc(SIZE*sizeof(float));
    v=(int*)malloc(NUM*sizeof(int));
    x=(int*)malloc(NUM*sizeof(int));
    w1=(float*)malloc(SIZE*sizeof(float));
    pop=(float***)malloc(POPS*sizeof(float**));
    for(i = 0; i < POPS; i++)
    {
        pop[i]=(float**)malloc(MAXPOP*sizeof(float*));
        for(j = 0; j < MAXPOP; j++)
            pop[i][j]=(float*)malloc(SIZE*sizeof(float));
    }
}

```

```

    }
    temp_pop=(float***)malloc(POPS*sizeof(float**));
    for(i = 0; i < POPS; i++)
    {
        temp_pop[i]=(float**)malloc(MAXPOP*sizeof(float*));
        for(j = 0; j < MAXPOP; j++)
            temp_pop[i][j]=(float*)malloc(SIZE*sizeof(float));
    }
    randomize();
    get_data(); /* Read input from file */
    calc_limit_err();
    make_initial_population();
    calc_score();
    sort_population();
    for (p=0;p<POPS;p++)
    {
        printf("Score of %d is %f",p,score[p][0]);
    }
    printf("We got here ");
    /* Educate the net */
    generation = 0;
    while ( (done != 1 ) && ( generation++ < SESSIONS ) )
    {
        make_next_generation( generation );
        p = best_pop();
        if ( score[p][0] > target_1)
        {
            printf ("Target error was %f and error is
%f",target_1,score[p][0]);
            done = 1;
        }
        else
            printf("Done all !!\n");
        fb=fopen("../gacode/unnormalise/nn-output_b","w");
        for (i=0;i<POPS;i++)
        {
            for ( j=0;j<BESTPOP;j++)
            {
                apply (i,j);
                l=0;
                for (k=0;k<NUM;k++)
                {
                    if (v[k]==1)
                    {
                        fprintf(fb,"%f\n",w1[l]);
                        if (!strcmp(name[k],"Ti"))
                            value_Ti = (w1[l]+0.5)*(high[k]-low[k])+low[k];
                        else
                        {
                            if (!strcmp(name[k],"Nb"))
                                value_Nb = (w1[l]+0.5)*(high[k]-low[k])+low[k];
                            else

```

```

        {
            if (!strcmp(name[k], "C"))
                value_C = (w1[l]+0.5)*(high[k]-low[k])+low[k];
            else
            {
                if (!strcmp(name[k], "N"))
                    value_N = (w1[l]+0.5)*(high[k]-low[k])+low[k];
            }
        }
        l++;
    }
    else
    {
        if (v[k]==0)
        {
            fprintf(fb, "%f\n", (f[k]-low[k])/(high[k]-low[k])-0.5);
            if (!strcmp(name[j], "Ti"))
                value_Ti = f[k];
            else
            {
                if (!strcmp(name[k], "Nb"))
                    value_Nb = f[k];
                else
                {
                    if (!strcmp(name[k], "C"))
                        value_C = f[k];
                    else
                    {
                        if (!strcmp(name[k], "N"))
                            value_N = f[k];
                    }
                }
            }
        }
        else
            fprintf(fb, "%f\n", (((value_Ti/4)+(value_Nb/8))/(value_C+value_N))-
            low[k])/(high[k]-low[k]))-0.5);
    }
    fprintf(fb, "%f %f\n", result[i][j], (error[i][j]+result[i][j]));
}
}
fclose(fb);
if ((generation%500)==0)
{
    fc=fopen("temp_output", "a");
    for (i=0; i<POPS; i++)
    {
        for (j=0; j<BESTPOP; j++)
        {
            apply(i, j);
            l=0;

```

```

        for (k=0;k<NUM;k++)
        {
            if (v[k]==1)
            {
                fprintf(fc,"%f\n",w1[l]);
                l++;
            }
            else
            {
                if (v[k]==0)
                    fprintf(fc,"%f\n", (f[k]-low[k])/(high[k]-low[k])-
0.5);
                else

fprintf(fc,"%f\n", (((((value_Ti/4)+(value_Nb/8))/(value_C+value_N))-
low[k])/(high[k]-low[k]))-0.5);
            }
        }
        fprintf(fc,"%f %f
\n",result[i][j],(error[i][j]+result[i][j]));
    }
}
fclose(fc);
}

printf("done! \n");
printf("\n\n**** For unnormalisation, goto /unnormalise and type a.out
(no. of lines in output file) => BEST_POP x POPS = no of lines in output
file (no. of inputs)****\n");
fscore=fopen("score_output","w");
for (i=0;i<POPS;i++) {
    for ( j=0;j<MAXPOP;j++) {
        fprintf(fscore,"Score for score_T_298 pop %d gene %d is %f
\n",i,j,score[i][j]);
        printf("scores for score_T_298 pop %d chromo %d id %f
\n",i,j,score[i][j]);
    }
}
printf("\n Scores printed to 'scores'\n\n");
fclose(fscore);

free(f);
free(low);
free(high);
free(sort_low);
free(sort_high);
free(v);
free(x);
free(w1);
free(pop);
free(temp_pop);
}

```

## References

1. D. E. Goldberg, 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley
2. I. Shah, 2002, [\*Tensile Properties of Austenitic Stainless Steel\*](#), I. Shah, M.Phil. Thesis, University of Cambridge
3. Paper describing this GA: [\*Genetic Algorithm for Optimization of Mechanical Properties\*](#), A. Delrome, 2003

# APPENDIX-C

## Profile string: Statistica Neural Network Software

The profile string in the model summary window is used throughout SNN as a shortcut method to identify the architecture of the network or ensemble. The profile consists of a type code, followed by a code giving the number of input and output variables and number of layers and units (networks) or members (ensembles). For time series networks, the number of steps and the look ahead factor are also given. The individual parts of the profile are the type and architecture.

The type is indicated by one of the following codes:

MLP	multilayer perceptron network
RBF	radial basis function network
SOFM	Kohonen self-organizing feature network
Linear	linear network
GRNN	generalized regression neural network
PCA	principal components network
Cluster	cluster network
Output	output ensemble
Conf	confidence ensemble

A neural network's architecture is of form I:N-N-N:O, where I is the number of input variable, O the number of output variables, N the number of units in each layer.

Example. 2:4-6-3:1 indicates a network with 2 inputs variables, 1 output variable, 4 input neurons, 6 hidden neurons, and 3 output neurons.

For a time series network, the steps factor is prepended to the profile, and signified by an "s".

Example. s10 1:10-2-1:1 indicates a time series network with steps factor (lagged input) 10.

An ensemble's architecture is of form I:[N]:O, where I is the number of input variable, O the number of output variables, and N the number of members of the ensemble.

### ***Training string:***

The training string is used throughout SNN as a shortcut method to identify the methods used to train the respective networks. The training string contains a number of codes, which are followed by the number of epochs for which algorithm ran (if an iterative algorithm) and an optional terminal code indicating how the final network was selected. For example, the code CG213b

indicates that the *Conjugate Gradient Descent algorithm* was used, the best network discovered during that run was selected (for “best” read “lowest selection error”), and this network was found on the 213<sup>th</sup> epoch.

The codes are:

BP Back propagation

CG Conjugate gradient descent

QN Quasi-newton

LM Levenberg-Marquardt

QP Quick propagation

DD Delta-Bar-Delta

SS (sub) sample

KM K-means (center assignment)

EX Explicit (deviation assignment)

IS Isotropic (deviation assignment)

KN K-nearest neighbor (deviation assignment)

PI Pseudo-invert (linear least squares)

KO Kohonen (center assignment)

PN Probabilistic neural network training

GR Generalized regression neural network

PC Principal Components analysis

The terminal codes are:

b Best network (the network with lowest selection error in the run was restored)

s Stopping condition (the training run was stopped before the total number of epochs elapsed as stopping condition was fulfilled)

c Converged (the algorithm stopped early because it had converged; that is, reached and detected a local or global minimum. Note that only some algorithms can detect stoppage in a local minimum, and that this is an advantage not a disadvantage!)



# **Appendix - D**

## **ACADEMIC CONTRIBUTION**

### **(Publications)**

The following is a list of the outcomes that have been published / presented during the period of the doctoral study:

#### **JOURNAL PAPER**

- B. J. Chauhan., S. N. Soman, “Modeling of Yield Strength of Ferritic Steel Welds” *International Journal for Research in Applied Science and Engineering Technology*, Vol. 6, issue 8, August 2018, 524-531. (UGC Approved Journal, ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887).
- B. J. Chauhan., S. N. Soman, “Modeling of Ultimate Tensile Strength of Ferritic Steel Welds” *International Journal for Research in Applied Science and Engineering Technology*, Vol. 6, issue 10, October 2018, 21-28. (UGC Approved Journal, ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 6.887).

#### **INTERNATIONAL CONFERENCE**

- B. J. Chauhan., S. N. Soman, “Modeling of Charpy Toughness of Ferritic Steel Welds”, *International conference on Recent Advances in Metallurgy for Sustainable Development*, Key-note lecture, IC-RAMSD 2018, 1-3 February 2018, Organized by The M S University of Baroda., Under the patronage of Ministry of Steel Government of India, New Delhi.



# Modelling of Yield Strength of Ferritic Steel Welds

B J Chauhan<sup>1</sup>, S N Soman<sup>2</sup>

<sup>1</sup>Associate Professor, Department of Metallurgical and Materials Engineering, Faculty of Technology and Engineering, The M. S. University of Baroda, Vadodara, Gujarat, India.

<sup>2</sup>Professor and Head, Department of Metallurgical and Materials Engineering, Faculty of Technology and Engineering, The M. S. University of Baroda, Vadodara, Gujarat, India.

**Abstract:** The design of ferritic steel welding alloys to match the ever desired properties of newly developed steels is not a simple task. It is traditionally achieved by experimental trial and error, modifying compositions and welding conditions until an adequate result is discovered. Savings in economy and time might be achieved if the trial process could be minimised. The present work outlines the use of an artificial neural network to model the yield strengths of ferritic steel weld deposits from their chemical composition, welding conditions and heat treatments. The development of the General regression neural network (GRNN) models is described, as is the confirmation of their metallurgical fundamentals and accuracy.

**Keywords:** Neural network; Ferritic Steels; Yield Strength; Welding alloys; Variables

## I. INTRODUCTION

The tensile strength test provides the basic design data essential in both the specification and acceptance of welding materials. Although the measurements involved are simple, their values depend in a complicated way on the chemical composition, the welding parameters and any heat treatment.

There is no common fundamental or experimental model capable of estimating the tensile parameters as a function of all these variables [1,2].

The difficulty is the complexity of the nonlinear relationship between input variables and yield strength. The physical models for strengthening mechanisms are not sufficiently sophisticated [3] and the linear regression methods used traditionally are not representing the real behaviour which is far from linear when all the variables are taken into account.

The aim of this work was to use GRNN to empirically model and interpret the dependence of the yield strength of steel weld deposits as a function of many input variables.

General regression neural network is capable of realising a great variety of nonlinear relationships of considerable complexity. Data are presented to the GRNN in the form of input and output parameters. As in regression analysis, the results then consist of the regression coefficients and a specification of the kind of function which in combination with the weights relates the independent or input variables to the dependent or output variables.

The design of a model using the GRNN method requires a large database of experimental measurements was assembled for neural network analysis of ferritic steel welds.

## II. MODELLING WORK

**Database for Modelling:** All of the data collected are from weld deposits in which the joint is designed to minimize dilution from the base metal, to enable specifically the measurement of all weld metal properties. Furthermore, they all represent electric arc welds made using one of the following processes: manual metal arc (MMAW), submerged arc welding (SAW) and tungsten inert gas (TIG). The welding process itself was represented only by the level of heat input. The data were collected from a large number of sources (Table 1).

The aim of the neural network analysis was to predict the Yield Strength as a function of a large number of variables, including the chemical composition, the welding heat input and any heat treatment. As a consequence, the yield strength database consists of 2121 separate experiments with 17 input variables.

In the present work, a neural network method is used as a Generalised Regression Neural Network[4]. All GRNN networks have 17 inputs, 1061 neurons in the first hidden layer, 2 neurons in the second hidden layer and 1 neuron in the output layer. (Figure.1)

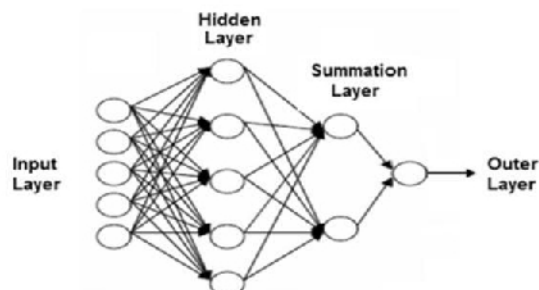


Figure 1. The architecture of Generalized Regression Neural Network

The hundred and thousand of models were trained with this neural network method in statistica software. The training errors, Validation errors (or Selection errors) and testing errors of training dataset(1061), Validation data set(530) (or Selection dataset) and testing dataset(530) of Yield Strength were compared. The lowest errors models were selected because they are best for practical applications.

Table 1 The 17 Input variables used in the analysis of the yield strength

Variables	Min	Max	Average	StDev	Variables	Min	Max	Average	StDev
C wt%	0.01	0.22	0.0708	0.0216	Cu wt%	0	2.18	0.0659	0.2062
Si wt%	0	1.63	0.3467	0.1262	Ti ppm	0	1000	78.6382	122.4481
Mn wt%	0.23	2.31	1.1959	0.4175	B ppm	0	200	9.2504	27.9733
S wt%	0.001	0.14	0.0081	0.0051	Nb ppm	0	1770	53.7704	145.5195
P wt%	0.001	0.25	0.0108	0.0075	HI kJ mm-1	0.55	7.9	1.3573	0.9931
Ni wt%	0	10.66	0.5807	1.4971	IPT C	20	375	205.4668	42.7739
Cr wt%	0	12.1	0.6243	1.5961	PWHTT C	20	780	328.1428	211.1714
Mo wt%	0	2.4	0.2001	0.3591	PWHTt h	0	50	9.4335	6.5893
V wt%	0	0.32	0.0191	0.0507	YS MPa	210	1026	535.7139	119.8611

### III. RESULTS AND DISCUSSION

The normal behaviour of the Predicted Yield Strength and Observed Yield Strength are observed in the Figure. 2 for Training data, Validation data and Testing data. Training of the model is excellent by GRNN method.

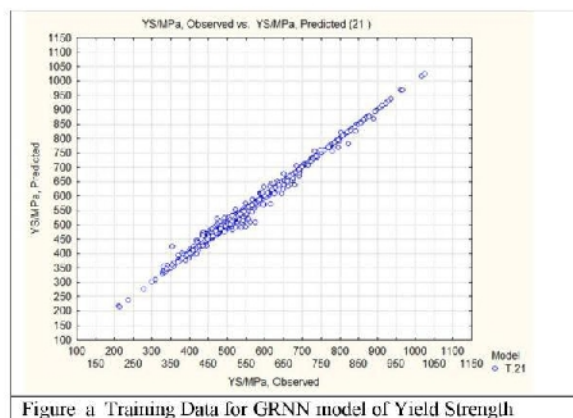


Figure a Training Data for GRNN model of Yield Strength

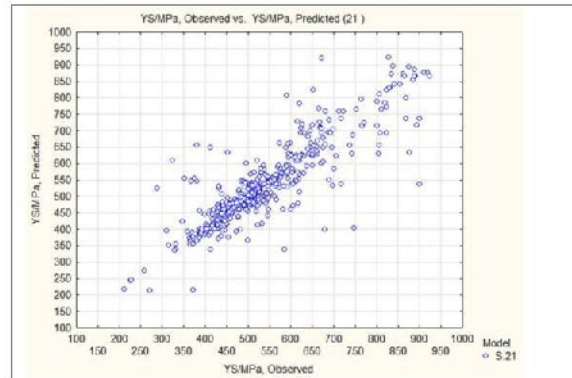


Fig b Validation Data for GRNN model of Yield Strength

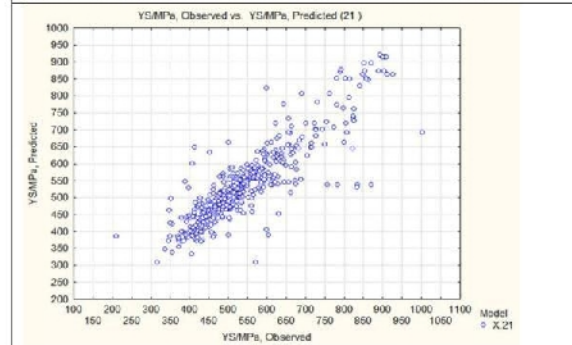
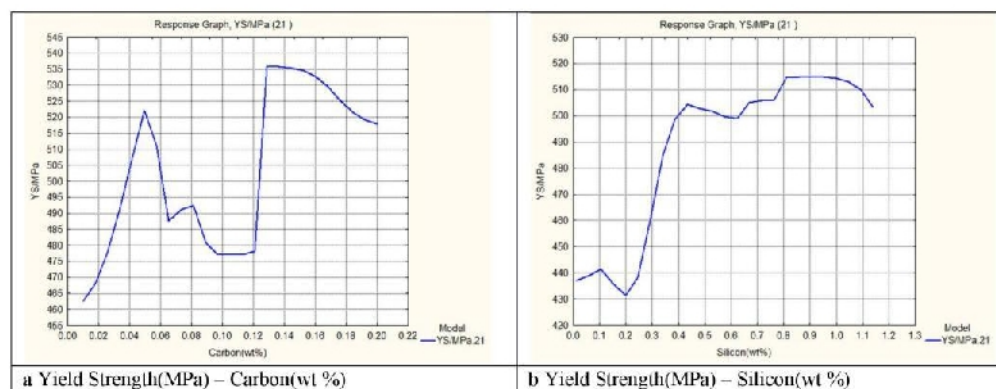


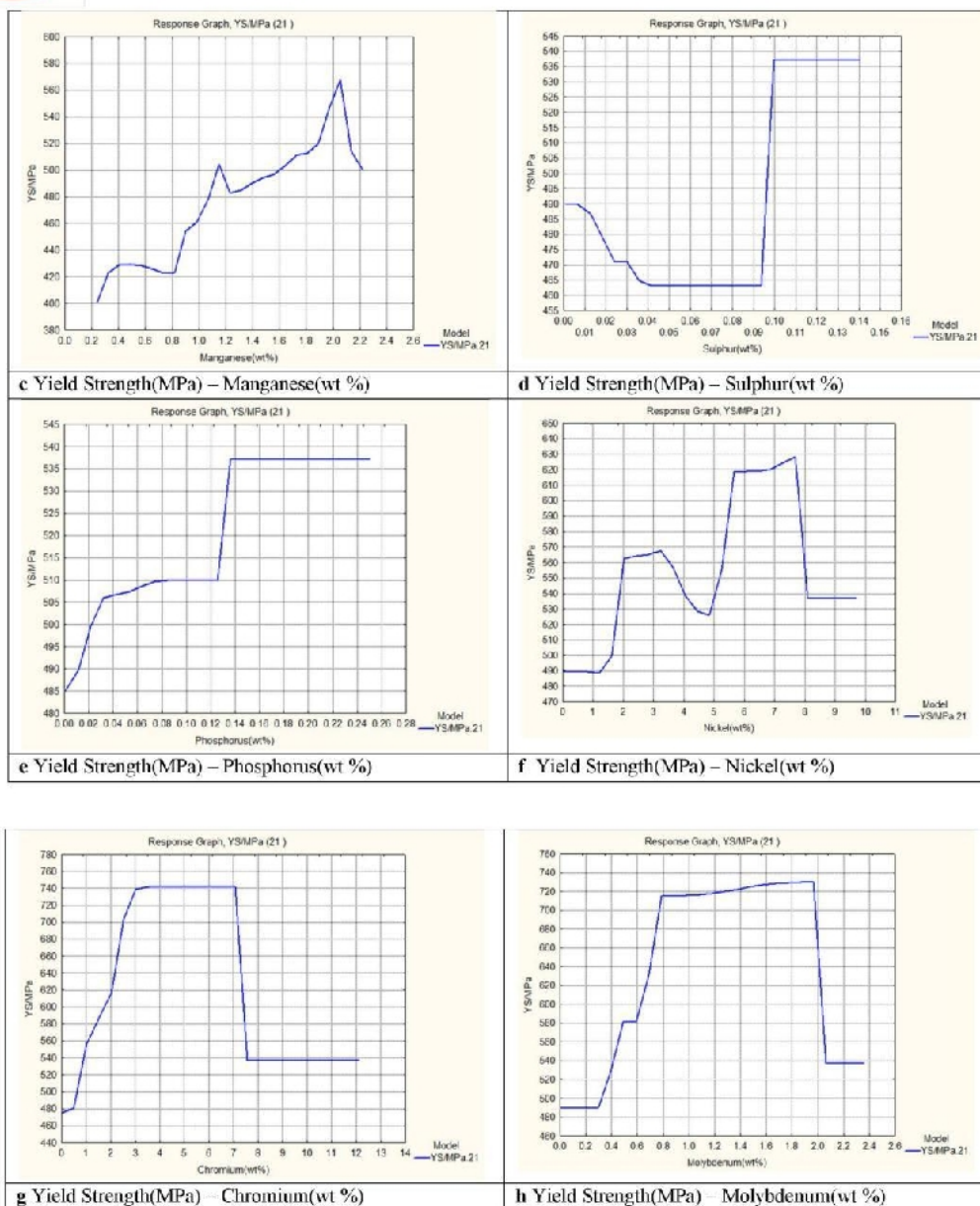
Fig c Test Data for GRNN model of Yield Strength

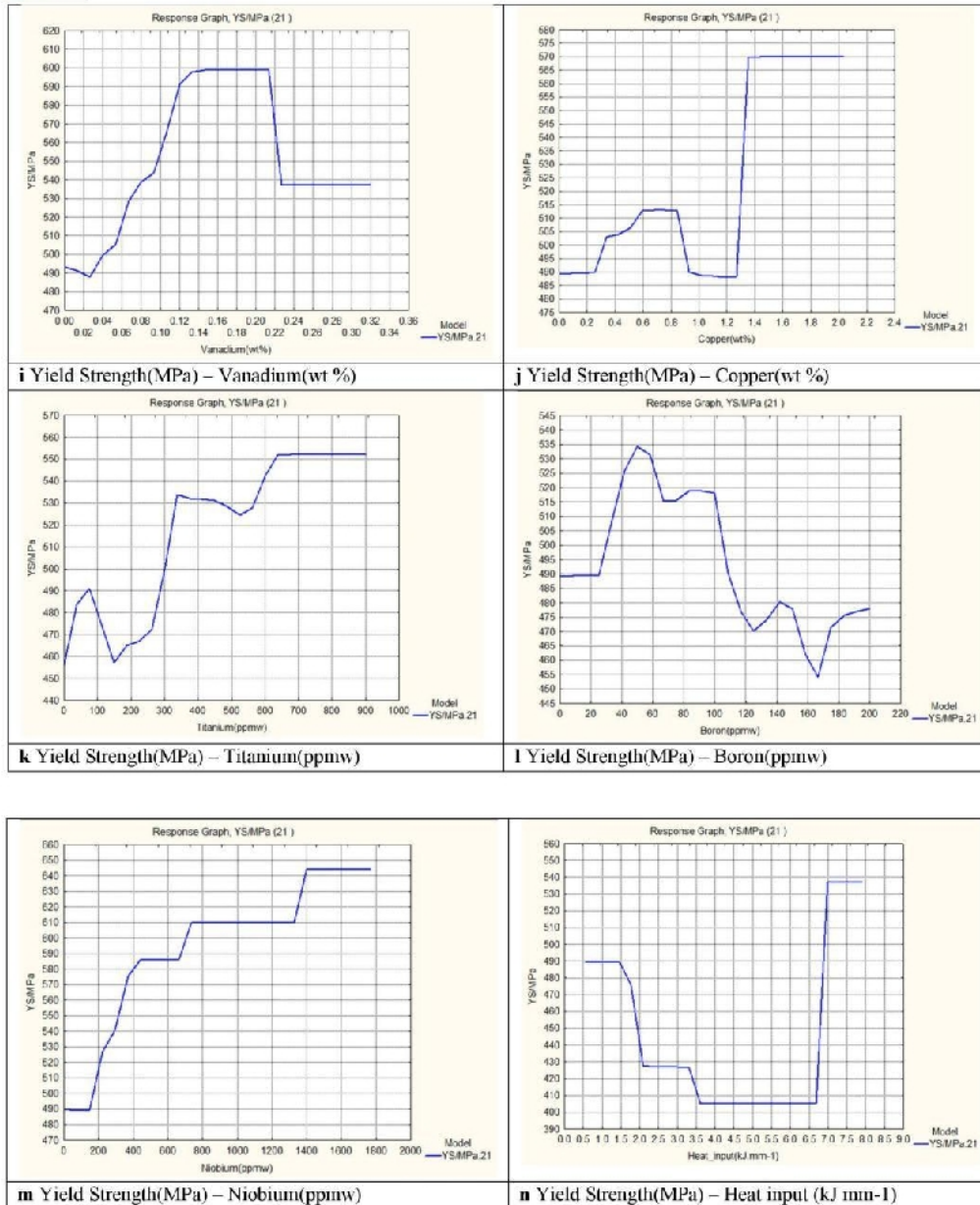
Figure 2 Training data, validation data and test data of the Best GRNN model for Yield Strength.

The best model of GRNN has training error 0.011404, validation error (selection error) 0.018101, and testing error 0.018669. This model is used for getting the results in form of various response graphs to understand the trend between the input variables and output variable(Yield Strength).(Figure 3)









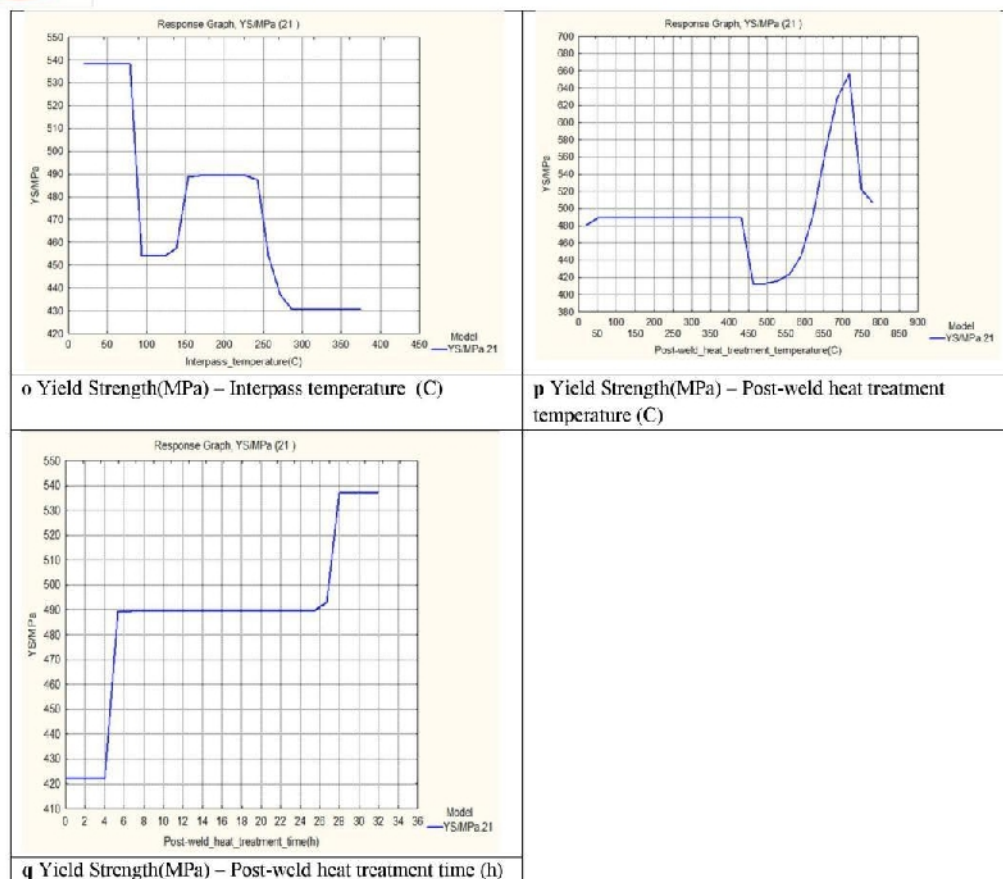


Figure 3 Response graphs(a to q) of Input variables and Yield Strength of Ferritic Steel Welds

The influence of each of the variables on the yield strength of welding alloys which is discussed here. The carbon increases the yield strength up to 522 MPa with 0.05% then drop to 477 MPa at 0.1%. After 0.15% C, yield strength increases to 536 MPa then decrease to 519 MPa at 0.2% C. In the case of silicon between 0.1% to 0.2%, there is a drop of the 440 MPa to 431 MPa in yield strength and then increases to 505 MPa at 0.45%. At 0.8%, yield strength is 515 MPa and decreases between 1.0% to 1.2% from 515 MPa to 504 MPa. The trend of manganese shows the increase in the Mn% the value of the yield strength is also increased from 400 MPa to 563 MPa. At various points, 0.8%, 1.1%, 2.1% the decrease in yield strength is observed. The sulphur shows the first decrease in the yield strength from 490 MPa to 464 MPa. At slightly more than 0.09%, it is increased from 464 MPa to 537 MPa. The Phosphorus gives the increase in yield strength from 485 MPa to 537 MPa. The nickel has the maximum yield strength of 629 MPa at 7.8% and minimum 490 MPa at 1%. In figure. It shows at 4.9% the yield strength value drop to 528 MPa. More than 7.8 % Ni gives a further drop in yield strength 539 MPa. The Chromium has a maximum yield strength of 740 MPa between 3% to 7%. More than 7% Cr reduces the yield strength to 539 MPa. Increase in the yield strength from 479 MPa to 740 MPa only by the gradual addition of chromium up to 3%. Molybdenum increases the yield strength from 490 MPa to 730 MPa at 1.98%. At 0.8% Mo gives yield strength 719 MPa. More than 1.98% Mo decreases yield strength from 730 MPa to 539 MPa. Vanadium increases the yield strength from 492 MPa to 600 MPa at 0.15%. At 0.22% V, yield strength decreases to 538 MPa. Copper increases the yield strength from 490 MPa to 513 MPa at 0.6%. At 1.2% Cu, yield strength decreases to 488 MPa. Cu gives maximum yield strength of





570 MPa when it is more than 1.27%. Titanium gives a minimum yield strength of 457 MPa to maximum 553 MPa. At 700 ppm yield strength is the highest. In between some range of Titanium from 90 ppm to 630 ppm, up and down in yield strength. Boron shows maximum yield strength of 535 MPa at 50 ppm. More than 50 ppm decreases the yield strength to 454 MPa. Niobium has a trend of increase in yield strength from 490 to 644 MPa with an increase from 180 to 1400 ppm.

Heat Input has stated of the yield strength of 490 MPa, then drops in between 1.5 to 6.6 kJ mm<sup>-1</sup> to 406 MPa. The highest value of yield strength 537 MPa is obtained at and more than 6.7 kJ mm<sup>-1</sup>. When the Interpass temperature is less than 70 °C, the yield strength is 538 MPa. More than 70 °C decrease in yield strength is observed to 470 MPa and further increase to 490 MPa at 150 °C. Minimum yield strength is 430 MPa at 270 °C. Post weld heat treatment temperature increases up to 425 °C shows yield strength 480 MPa and 490 MPa. More than 455 °C, the yield strength increases to maximum 655 MPa at 710 °C then drop to 510 MPa. Post weld heat treatment time has a trend of increase in yield strength from 420 to 490 MPa between 4 to 5 hours. More than 25 hours, it increases maximum yield strength to 538 MPa.

The relationship between the input variables and yield strength is a nonlinear as seen above in response graphs (Figure 3).

The GRNN model has good accuracy in prediction of yield strength of ferritic steel welds on unseen data which is excellent for the design of welds (Table.2) The predicted yield strength for the unseen data of three weld alloys are compared with measured values of yield strength shows the prediction capacity of the GRNN model. This GRNN model can be used for practical applications, research and development of ferritic steel alloys.

Table 2 Predicted yield strength by GRNN model for unseen data of three ferritic weld deposits

Variable	Weld alloy 1	Weld alloy 2	Weld alloy 3
Carbon(wt%)	0.041	0.049	0.081
Silicon(wt%)	0.30	0.35	0.24
Manganese(wt%)	0.62	1.37	0.59
Sulphur(wt%)	0.007	0.007	0.009
Phosphorus(wt%)	0.010	0.013	0.012
Nickel(wt%)	2.38	1.06	10.8
Chromium(wt%)	0.03	0.03	1.17
Molybdenum(wt%)	0.005	0.005	0.300
Vanadium(wt%)	0.012	0.012	0.006
Copper(wt%)	0.03	0.03	0.30
Titanium(ppm)	55	55	00
Boron(ppm)	2	2	1
Niobium(ppm)	20	20	10
Heat_input(kJ.mm-1)	1.0	1.0	1.2
Interpass_temperature(°C)	200	200	150
Postweld_heat_treatment_temperature(°C)	250	250	250
Post-weld_heat_treatment_time(h)	14	14	14
Measured YS/MPa	466	498	896
Predicted YS/MPa	466	497	913

#### IV. CONCLUSIONS

General Regression Neural Network is the best for capturing trends of input variables and output variables in weld alloys which is nonlinear. A neural network method based within a General regression neural network has been used to rationalize an enormous quantity of published experimental data on the yield strength. It is now possible, therefore, to estimate the yield strength as a function of the chemical composition, welding conditions and a variety of heat treatment parameters.

The model formulated has been applied towards the understanding of ferritic steels alloys used in welding for various equipment construction in industries (eg. Power plants, Submarines, Liquid Gas Storage Tanks..etc.) It has been used successfully on unseen data on ferritic steel welds for various applications.

The design of the ferritic weld alloys become easier, accurate, economical and time-saving with the help of the GRNN modelling. The control of the effective input variables gives the desired yield strength of weld alloys for real applications in industries.





#### REFERENCES

- [1] H.K.D.H. Bhadeshia and L.-E. Svensson, in H. Cerjak and K.E. Easterling (eds.), *Mathematical Modelling of Weld Phenomena*, Institute of Materials, London, 1993, pp. 109-192.
- [2] H. K. D. H. BHADSHIA: in 'Mathematical modelling of weld phenomena 3', (ed. H. Cerjak), 229-284; 1997, London, The Institute of Materials
- [3] L. E. Svensson, *Control of Microstructure and Properties in Steel Arc Welds*, CRC Press, London, 1994, pp. 188
- [4] Speck, D.F. (1991). A Generalized Regression Neural Network. *IEEE Transactions on Neural Networks* 2 (6), 568-576.



# Modelling of Ultimate Tensile Strength of Ferritic Steel Welds

B J Chauhan<sup>1</sup>, S N Soman<sup>2</sup>

<sup>1</sup>Associate Professor, Department of Metallurgical and Materials Engineering, Faculty of Technology and Engineering, The M. S. University of Baroda, Vadodara, Gujarat, India.

<sup>2</sup>Professor and Head, Department of Metallurgical and Materials Engineering, Faculty of Technology and Engineering, The M. S. University of Baroda, Vadodara, Gujarat, India.

**Abstract:** The design of ferritic steel welding alloys to fit the ever expected properties of newly evolved steels is not a very easy task. It is traditionally attained by experimental trial and error, changing compositions and welding conditions until a sufficient result is established. Savings in the economy and time might be achieved if the trial process could be minimised. The present work outlines the use of an artificial neural network to model the ultimate tensile strength of ferritic steel weld deposits from their chemical compositions, welding conditions and heat treatments. The development of the General regression neural network (GRNN) models is explained, as is the confirmation of their metallurgical principles and precision.

**Keywords:** Neural network; Ferritic Steels; Ultimate Tensile Strength; Welding alloys; Variables

## I. INTRODUCTION

The tensile strength test provides the basic design data essential in both the specification and acceptance of welding materials. Although the measurements involved are simple, their values depend in a complicated way on the chemical compositions, the welding parameters and the heat treatments. There is no common fundamental or experimental model capable of estimating the tensile parameters as a function of all these variables [1,2].

The difficulty is the complexity of the nonlinear relationship between input variables and ultimate tensile strength. The physical models for strengthening mechanisms are not sufficiently sophisticated [3] and the linear regression methods used traditionally are not representing the real behaviour which is far from linear when all the variables are taken into account.

The aim of this work was to use GRNN to empirically model and interpret the dependence of the ultimate tensile strength of steel weld deposits as a function of many input variables.

The General regression neural network is capable of realising a great variety of nonlinear relationships of considerable complexity. Data are presented to the GRNN in the form of input and output parameters. As in regression analysis, the results then consist of the regression coefficients and a specification of the kind of function which in combination with the weights relates the independent or input variables to the dependent or output variables.

The design of a model using the GRNN method requires a large database of experimental measurements was assembled for neural network analysis of ferritic steel welds.

## II. MODELLING WORK

**Database for Modelling:** All of the data collected are from weld deposits in which the joint is designed to minimize dilution from the base metal, to enable specifically the measurement of all weld metal properties. Furthermore, they all represent electric arc welds made using one of the following processes: manual metal arc (MMAW), submerged arc welding (SAW) and tungsten inert gas (TIG). The welding process itself was represented only by the level of heat input. The data were collected from a large number of sources (Table I).

The aim of the neural network analysis was to predict the Ultimate Tensile Strength as a function of a large number of variables, including the chemical compositions, the welding parameters and heat treatments. As a consequence, the Ultimate Tensile strength database consists of 2091 separate experiments with 18 input variables.

In the present work, a neural network method is used as a Generalised Regression Neural Network[4]. All GRNN networks have 18 inputs, 1047 neurons in the first hidden layer, 2 neurons in the second hidden layer and 1 neuron in the output layer. (Figure.1)



INTERNATIONAL CONFERENCE  
ON



RECENT ADVANCES IN METALLURGY FOR SUSTAINABLE DEVELOPMENT  
(IC-RAMSD 2018)  
FEBRUARY 1-3, 2018

This is to certify that Dr./Mr./Ms.


B. J. CHAUHAN


from METALLURGICAL AND MATERIAL ENGINEERING DEPARTMENT has  
presented Plenary/Key Note/Special/Technical paper titled MODELLING OF TOUGHNESS OF THE  
FERRITIC STEEL WELDS

in the International Conference on RECENT ADVANCES IN METALLURGY FOR SUSTAINABLE  
DEVELOPMENT (IC-RAMSD 2018) organized by the Department of Metallurgical and Materials  
Engineering, Faculty of Technology and Engineering, The M.S. University of Baroda under the patronage  
of Ministry of Steel, Government of India, New Delhi on February 1-3, 2018 held at Faculty of Technology  
& Engineering, The Maharaja Sayajirao University of Baroda, Vadodara.



  
(Prof. S.N. Soman)  
Chairman and Head

  
(Dr. Ashok Kr. Vaish)  
Organizing Secretary  
Ministry of Steel Chair Professor

  
(Dr. Vandana J. Rao)  
Joint Organizing Secretary

