ADALINES

The Adaline (Adaptive Linear Neuron or "Adaptive Linear Element") is a model similar to the Perceptron. There are several variations:

- One has the **threshold** function similar to a perceptron.
- Another uses a **pure linear** function with no threshold.
- We can devise more.

Adaline Training
With or without the threshold, the Adaline is **trained** based on the output of the linear function rather than the final output.

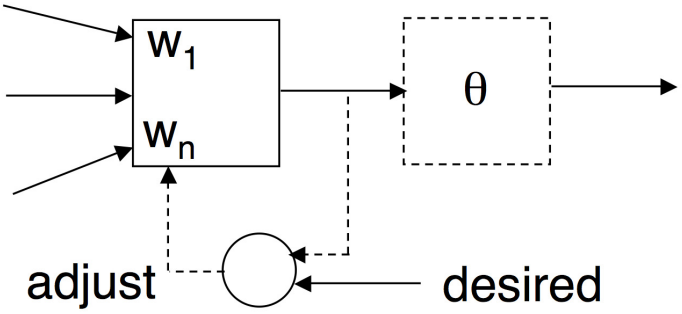


Fig. 24: Adaline training

The catch here is that we have to state the **desired** value in terms of the output of the linear part, rather than the output after the threshold.

What is this for a classifier?

A reasonable approach is to use any nominal target value such as -1 as desired for a "**no**" classification and a +1 for a "**yes**" classification.

The formula for Adaline weight updating is very similar to the Perceptron:

Add to the weights Δw where;

$$\Delta\omega = \varepsilon\ \eta\ [\ -1, x1, x2, \ldots, xn\ ]$$

only now the **error is not limited to 1, -1, 0 as before**; it can have a fractional value, since it is based on the output of the linear part of the device.

One major difference from this vs. the Perceptron is that a learning rate of 1 won't generally be acceptable. It will need to be smaller, say 0.01. There is a theory that tells us how large we can make it.

Adaline Convergence
The Adaline admits a more refined stopping criterion:

> The **Mean-Squared Error (MSE)** is the average of the squares of the error taken over all samples. Squaring makes the measure insensitive to the sign of the error. It also provides certain analytic properties.
> This quantity ideally converges toward a specific minimum (which might never be exactly attained). The algorithm can be set to stop when the MSE reaches a desired value.

Adaline Example

We'll use the same example as before. But now we'll train on the output of the linear portion and target for +1 for a "yes" answer and -1 for a "no" answer.

(4, 5)	+1
(6, 1)	+1
(4, 1)	-1
(1, 2)	-1

Try a learning rate of 0.01.

- Start with initial weights all 0.
- (In general, can use random weights.)
- Progress:
  - trying sample desired: 1, inputs: -1 4 5 output: -1
  - diff is 2
  - error is 1
  - new weights: -0.01 0.04 0.05

  - weights: -0.01 0.04 0.05
  - trying sample desired: 1, inputs: -1 6 1 output: 1
  - diff is 0
  - error is 0.7
  - new weights: -0.017 0.082 0.057

  - trying sample desired: -1, inputs: -1 4 1 output: 1
  - diff is -2
  - error is -1.402
  - new weights: -0.00298 0.02592 0.04298

- trying sample desired: -1, inputs: -1 4 1 output: 1
- diff is -2
- error is -1.402
- new weights: -0.00298 0.02592 0.04298

- trying sample desired: -1, inputs: -1 1 2 output: 1
- diff is -2
- error is -1.11486
- new weights: 0.0081686 0.0147714 0.0206828

- **epoch 1**: wrong = 3, mse = 1.17463

  - epoch 1: wrong = 3, mse = 1.17463
  - epoch 2: wrong = 2, mse = 1.11176
  - epoch 3: wrong = 2, mse = 1.08817
  - epoch 4: wrong = 2, mse = 1.07521
  - epoch 5: wrong = 2, mse = 1.06563

    . . .
  - epoch 30: wrong = 2, mse = 0.888344
  - epoch 31: wrong = 1, mse = 0.881999

    . . .
  - epoch 197: wrong = 1, mse = 0.363726
  - epoch 198: wrong = 0, mse = 0.362493

- Final weights: 1.54436 0.273645 0.252003

- Final weights: 1.54436 0.273645 0.252003

input	desired	weighted sum	actual
(4, 5)	+1	0.810235	1
(6, 1)	+1	0.359513	1
(4, 1)	-1	-0.197777	-1
(1, 2)	-1	-0.766709	-1

Alternate Rule Names
Because the Adaline rule minimizes MSE, it is sometimes called the "**LMS rule**" [LMS = "least mean square"].

The term "**Delta rule**" is also sometimes used, although this will be seen to be a rule for a more general class of networks.

The "**Widrow-Hoff**" rule is also used.

Minimizing Error Iteratively
Consider the task of finding the minimum of a function of one variable. Could try to extend this to functions of multiple variables.

One standard method for doing this, if the derivative of the function is known, is **Newton's method**, which entails constructing a tangent line from a current estimate, then using the intersection of the line with the origin for the next.

What function is minimized?

- Think of the inputs as being **constants**.
- The weights are the variables.
- We want to find the weights that minimize the **MSE** as a function of the weights.
- The fact that the MSE is defined analytically is a big help.

Gradient Descent
Gradient descent is another method for finding the minimum.

It consists of computing the **gradient** of the function, then taking a small **step** in the **direction of negative gradient**, which hopefully corresponds to decreased function value, then repeating for the new value of the dependent variable.

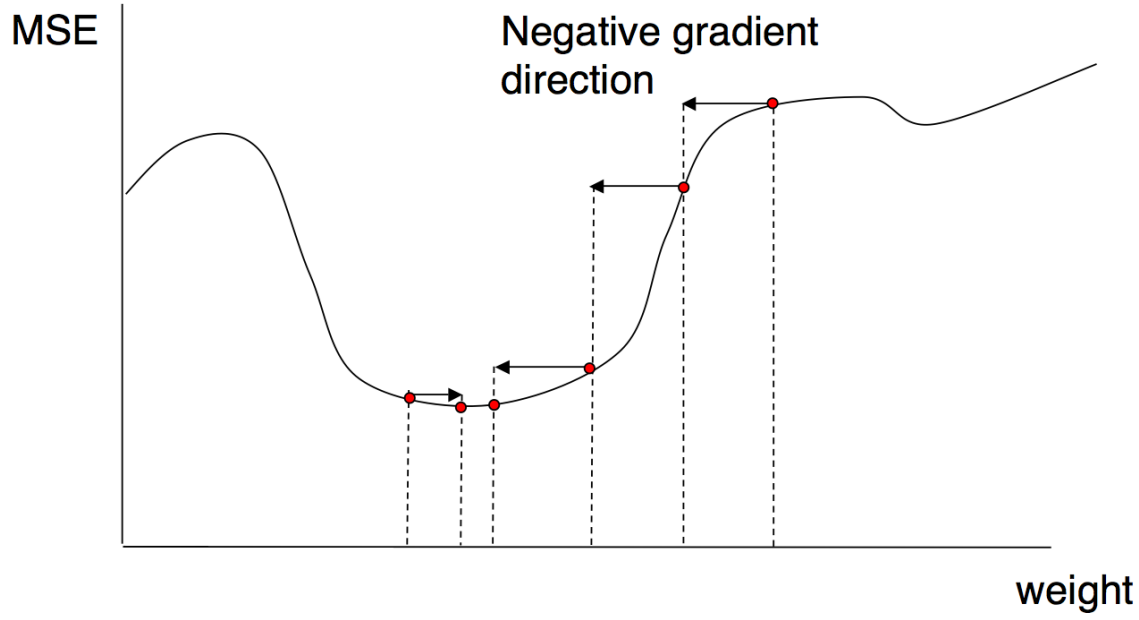


Fig. 26: Gradient Descent

- The previous diagram is mainly to enhance our intuition.
- A single dimension for weights (including bias) is atypical.
- For the general case, the gradient is a **vector** of gradient components, one for each weight (including bias).

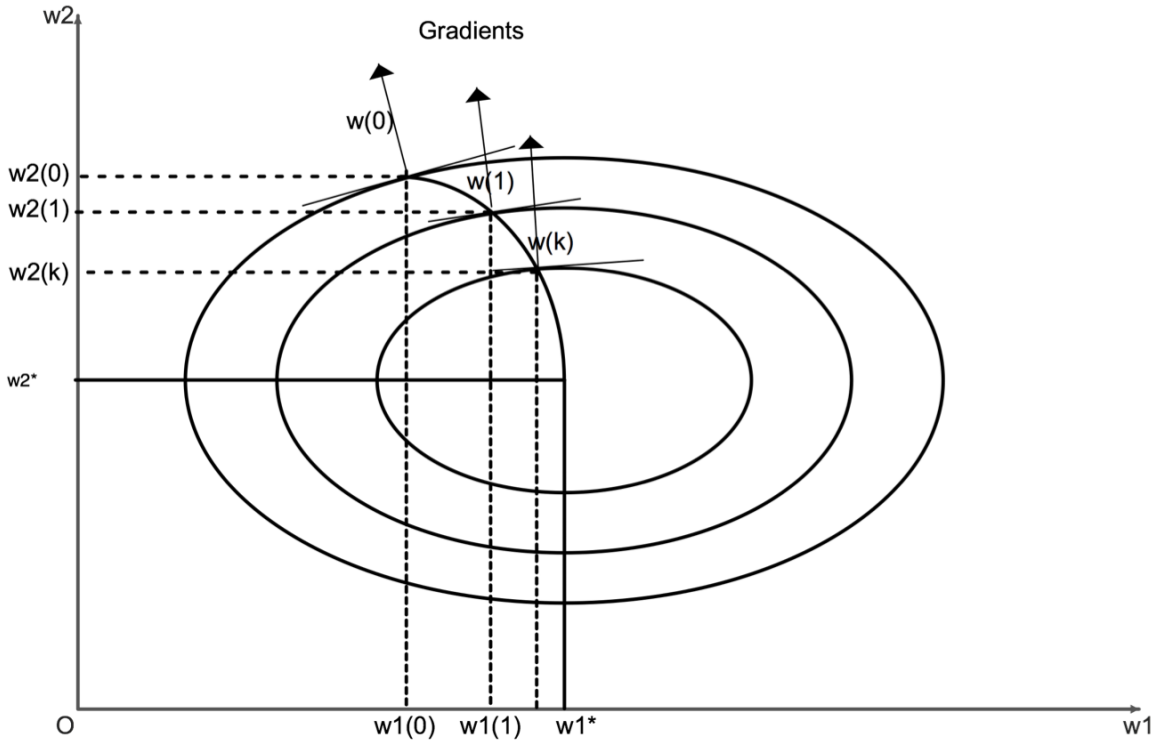


Fig. 27: 2D Gradient descent

Computing Gradients

- MSE = J(w) = Σ(desired-actual)²/n
  where Σ is over n samples.
- desired is a fixed constant for each sample.
- actual = Σwj. xj (Σ over input lines)
- So J(w) = Σ(desired- Σwj.xj)²/n

On-Line Approximation to Gradient

"On-line" means based on a **single sample**, vs. "batch", which means using all samples

$$J \approx (d - \Sigma w_j x_j)^2 \quad \text{(note: no outer sum)}$$

$$\text{(d = desired)}$$

$$i^{th} \text{ gradient component} = \partial J / \partial w_i$$

$$= \partial / \partial w_i (d - \Sigma w_j x_j)^2$$

$$= 2 (d - \Sigma w_j x_j) \, \partial / \partial w_i (d - \Sigma w_j x_j) = -2 \varepsilon x_i$$

$$= \text{error, } \varepsilon \qquad -x_i$$

Computing Gradients (contd…)

- i^{th} gradient component = -2 ε xi
- However we want to move in the direction of **negative** gradient, tempered by the **learning rate** η, so:

Amount to add to weight is

$$\Delta w_i = 2\,\varepsilon\,\eta\,x_i \qquad \text{(2 can be folded into } \eta\text{)}$$

which we recognize as the LMS (Adaline) rule.

Vector Version of the Analysis of Gradient Descent for Adaline

- MSE = $J(w) = E[(d - w^T x)^2]$ (E = **expectation** or mean averaged over samples)

  $= E[d^2 - 2dw^T x + w^T x\, x^T w]$
  $= E[d^2] - E[2dw^T x] + E[w^T x\, x^T w]$
  $= E[d^2] - 2w^T E[dx] + wTE[x\, x^T]\, w$
  $= c - 2w^T h + w^T R\, w$, for appropriate const. c, h, R

- $J(w) = c - 2w^T h + w^T R\, w$
  where c = E[d2], h = E[d x], R = E[x xT]

- This is a **quadratic form** in w with coefficients derived from the data vectors x.
- R is called the (auto-)**correlation matrix**

Example

$$\text{Sample1} \left\{ \mathbf{x}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{d}_1 = [-1] \right\} \qquad \text{Sample 2} \left\{ \mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{d}_2 = [1] \right\}$$

$$\mathbf{R} = E[\mathbf{x}\mathbf{x}^T] = \frac{1}{2}\mathbf{x}_1\mathbf{x}_1^T + \frac{1}{2}\mathbf{x}_2\mathbf{x}_2^T$$

$$\mathbf{R} = \frac{1}{2}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} -1 & 1 & -1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

$$\lambda_1 = 1.0, \qquad \lambda_2 = 0.0, \qquad \lambda_3 = 2.0$$

$$\eta < \frac{1}{\lambda_{max}} = \frac{1}{2.0} = 0.5$$

The training – First epoch

Sample 1 $$a(0) = \mathbf{W}(0)\mathbf{p}(0) = \mathbf{W}(0)\mathbf{x}_1 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}\begin{bmatrix}-1\\1\\-1\end{bmatrix} = 0$$

$$\varepsilon(0) = d(0) - a(0) = d_1 - a(0) = -1 - 0 = -1$$

$$\mathbf{W}(1) = \mathbf{W}(0) + 2\eta\varepsilon(0)\mathbf{x}^T(0)$$

$$\mathbf{W}(1) = \begin{bmatrix}0 & 0 & 0\end{bmatrix} + 2(0.2)(-1)\begin{bmatrix}-1\\1\\-1\end{bmatrix}^T = \begin{bmatrix}0.4 & -0.4 & 0.4\end{bmatrix}$$

Second epoch

Sample 2 $$a(1) = \mathbf{W}(1)\mathbf{p}(1) = \mathbf{W}(1)\mathbf{p}_2 = \begin{bmatrix}0.4 & -0.4 & 0.4\end{bmatrix}\begin{bmatrix}1\\1\\-1\end{bmatrix} = -0.4$$

$$\varepsilon(1) = d(1) - a(1) = d_2 - a(1) = 1 - (-0.4) = 1.4$$

$$\mathbf{W}(2) = \begin{bmatrix}0.4 & -0.4 & 0.4\end{bmatrix} + 2(0.2)(1.4)\begin{bmatrix}1\\1\\-1\end{bmatrix}^T = \begin{bmatrix}0.96 & 0.16 & -0.16\end{bmatrix}$$

Third epoch

$$a(2) = \mathbf{W}(2)\mathbf{p}(2) = \mathbf{W}(2)\mathbf{p}_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64$$

$$\varepsilon(2) = d(2) - a(2) = d_1 - a(2) = -1 - (-0.64) = -0.36$$

$$\mathbf{W}(3) = \mathbf{W}(2) + 2\eta\varepsilon(2)\mathbf{x}^T(2) = \begin{bmatrix} 1.1040 & 0.0160 & -0.0160 \end{bmatrix}$$

$$\mathbf{W}(\infty) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

Rattling

If learning rate too high, minimum solution won't be achieved, even if there is no divergence.

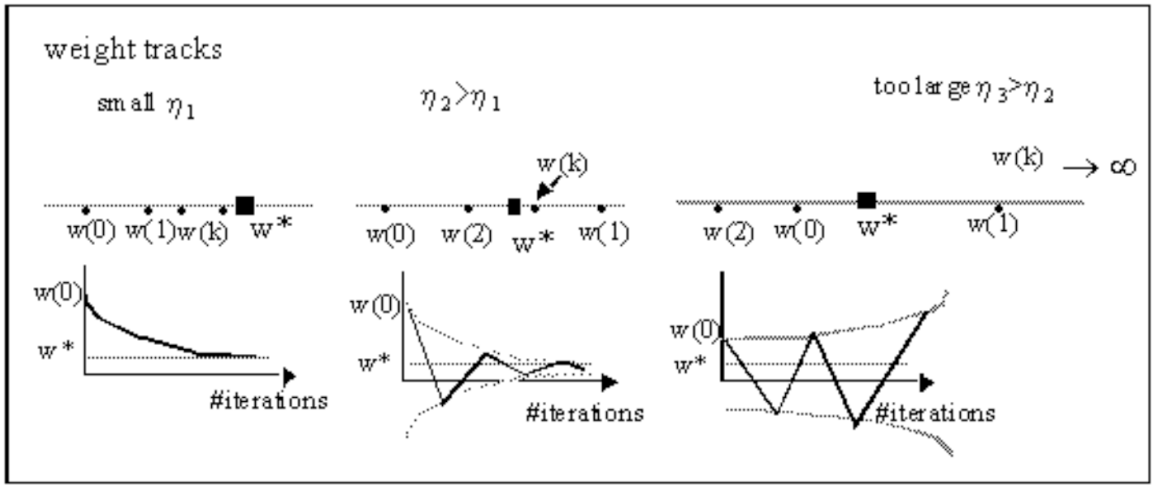


Fig. 30: Rattling

Solutions to Rattling

- Rule of thumb: Use a learning rate 0.1 times the theoretical maximum, **or**
- Gradually decrease the learning rate, e.g. according to a pre-set schedule, or
- Adaptively set the learning rate:
  - If the MSE is taking big steps, make it smaller.
  - If the MSE is taking small steps, make it larger.

Generalizing the Adaline

Suppose that we replace the threshold stage with a general analytic function f and revert to expressing desired in terms of its output:

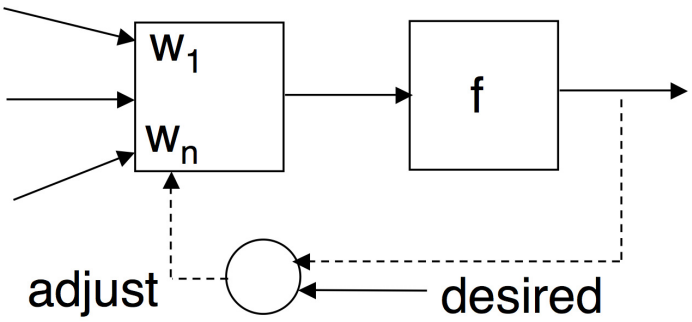


Fig. 31: Generalized adaline

- Consider again the derivation of the LMS rule:
  $J(w) = \Sigma(\text{desired} - f(\Sigma wj.xj))^2/n$
- $J \approx (d - f(\Sigma wj.xj))2$ (d = desired)
- i^{th} gradient component = $\partial J/\partial w_i$
  $= \partial/\partial w_i(d - f(\Sigma wj.xj))^2$
  $= 2\,(d - f(\Sigma wj.xj))\,\partial/\partial\, w_i\,(d - f(\Sigma wj.xj))$
  $= -2\,\varepsilon\,\partial/\partial w_i\, f(\Sigma wj.xj)$
  $= -2\,\varepsilon\, f'(\Sigma wj.xj)\,\partial/\partial w_i\, \Sigma wj.xj = -2\,\varepsilon\, x_i f'(\Sigma wj.xj)$

Generalized LMS Rule (or Delta Rule)

- $\Delta w = 2\,\varepsilon\,\eta\, f'(\Sigma wj.xj)\, xi$
  assuming that f has a derivative f'.
- $\Sigma wj.xj$ is often called the **"net" value** or **"activation" value**, and f the **activation function**.
- For the special case of f being the **identity** function, this reduces to the LMS rule we had before.

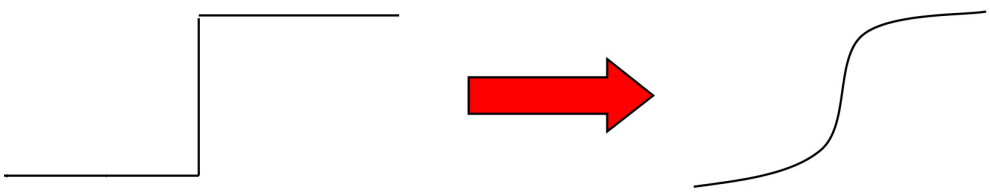
Why worry about this generalization?

$$\Delta w = 2\,\varepsilon\,\eta\, f'(\Sigma wj.xj)\, xi$$

It will have a number important uses.

Use #1

- In the Adaline with **threshold**, we can't very well treat the model analytically, due to the fact that we have a noncontinuous function at the output.
- But we can approximate the noncontinuous function with a continuous one:



Sigmoids

- The "S" shape on the right of the previous figure is called a sigmoid curve.
- This is a generic term and there are several different analytic functions that behave this way.

Logisitic Sigmoid

- Logisitic function ("logsig"–Matlab):

f(x) = 1/(1+exp(-ax))

- f'(x) = f(x)(1-f(x))

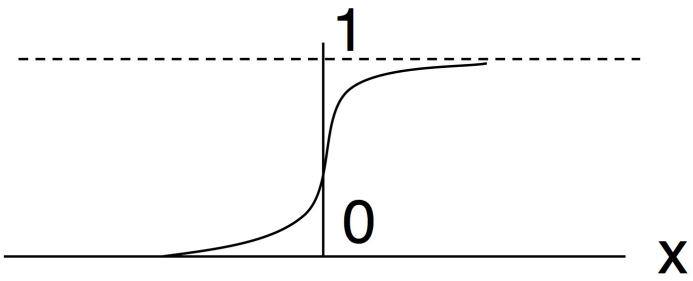


Fig. 33: Logistic Sigmoid

Hyperbolic Sigmoid

- Hyperbolic tangent function ("tansig"):

  f(x) = tanh(x)
  =(exp(x)-exp(-x))/(exp(x) + exp(-x))

- f'(x) = 1-f²(x)

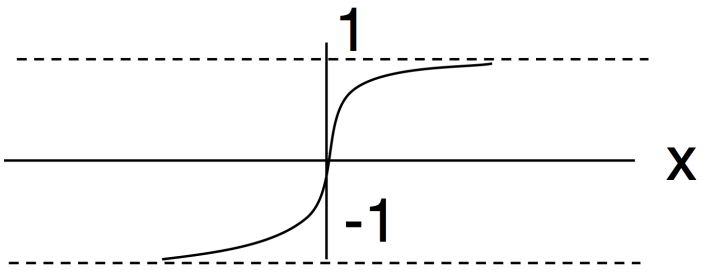


Fig. 34: Hyperbolic sigmoid

Squashing Functions

- Sigmoids, step functions, and other functions that force their results to be in a limited range are called "squashing functions".
- It is generally accepted that biological neural system is based on such functions, since there are physical limits to the response level.

Applications - Medicine

(Refer to :)

References

- Rojas, R., *Neural Networks – A Systematic Introduction*
- Anderson, J.A., *An Introduction to Neural Networks*
- Haykin, S., *Neural Networks – A Comprehensive Foundation*
- Keller, B., *Neural Networks, CS-152*

PRACTICAL # 3

Perceptrons classification
(June 13, 2009)

Objectives
To implement error-corrective learning for classification using threshold activation function.

Description
Implement a perceptron and train it for classification using error-correction and threshold activation function. The perceptron will be trained for 4 different train set.

Platform
Any platform with any language of choice.

Model
- Use a set of **training samples**:
    - Input vectors, each paired with desired output
- Initialize the weights and threshold arbitrarily. (the input X_0 is the threshold)
- Repeat:
    - Choose a sample
    - Test the network against the sample
    - If answer is incorrect, **adjust** weights
- until all samples test correctly. We'

(Refer to page 8 – 14 of Unit 3)

Here's my program.

```
/*-------------------------------------------------------------------------
 Program name: error corrective learning for classification (program # 3)
 Description : This program demonstrates error corrective learning to establish
                        association  between inputs and logical output and classifies the
                        inputs into two classes.

                        Uses threshold activation function.

 By                      : Pramod Parajuli
 Date            : June 13, 2009
-------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define INPUTS                  3
#define TRAIN_SAMPLES    4

#define THRESHOLD               0

/* let's define a structure to represent the perceptron */
typedef struct perceptron

                long inp[INPUTS];               // inputs to the perceptron
                long weights[INPUTS];           // weights of the inputs
                long threshold;                 // threshold of the neuron
                long function_out;              // functional output of the neuron
                long outp;                      // threshold driven output of the perceptron
                long error;                     // error of the perceptron
                long target;
}PERCEPTRON;

/* let's create a classifier instance of PERCEPTRON*/
PERCEPTRON classifier;

/* this structure defines template for one training set
             - 1 input vector
             - one expected output  */
typedef struct {
             long inputs[INPUTS];
             long expected_out;
} TRAIN_ELEMENT;

/* for logical classifiers with two inputs, let's define 4 training sets */
TRAIN_ELEMENT training_set[TRAIN_SAMPLES] = {
             {{-1, 4, 5}, 1},
             {{-1, 6, 1}, 1},
             {{-1, 4, 1}, 0},
             {{-1, 1, 2}, 0}
};

long counter;                   // general purpose counter

long LEARNING_RATE;             // learning rate of the perceptron
long EPOCH;                     // defines no. of iterations

void menu(void);                // displays a user help menu
void randomize(void);           // assigns random weights to the weight vector
void print(void);               // print the weight vector
void train(void);               // train the perceptron for adaptation
void feedforward(void);         // feedforward function  (inputs * weights)
void backpropaclassifier(void); // backpropagate (weight += error * input * learning rate)
void test(void);                // test the perceptron by using feedforward

int main (int argc, char * argv[]){
             int exit;                  // status variable for flow control
             char user_input[10];       // user input - string

             menu();                    // display menu

             exit = 0;

             classifier.threshold = THRESHOLD;

             do{
```

```
                printf("\ncommand $  ");
                gets(user_input);

                if(strcmp(user_input, "SHUFFLE") == 0){
                        randomize();                            // assign random weights
                        print();                                // print the weight vector
                }
                else if(strcmp(user_input, "TRAIN") == 0){
                        printf("\nEnter the learning rate : ");  // learning rate
                        scanf("%ld", &LEARNING_RATE);
                        train();                                // train
                }
                else if(strcmp(user_input, "TEST") == 0){
                        test();
                }
                else if(strcmp(user_input, "PRINT") == 0){
                        print();
                }
                else if(strcmp(user_input, "EXIT") == 0){
                        exit = 1;
                }
                else{
                        menu();
                }

            }while(exit == 0);

            printf("\npress any key to exit...");
            getc(stdin);

            return 0;
}

/*-------------------------------------------------------------------------
 Function name   : menu
 Description : Displays menu
-------------------------------------------------------------------------*/
void menu(void){
            printf("\n\n\t\t\tPerceptron training for classification  (PROGRAM# 3.0)");
            printf("\n\tUse only integer inputs for data entry");
            printf("\nPlease use following commands");
            printf("\n-------------------------------------------------------------------------");
            printf("\nSHUFFLE : to randomize weights   | TRAIN   : to train the perceptron");
            printf("\nTEST    : to test the perceptron | PRINT   : to print the weights");
            printf("\nHELP    : to display this menu   | EXIT    : to exit the program");
            printf("\n-------------------------------------------------------------------------\n");
}

/*-------------------------------------------------------------------------
 Function name   : randomize weights
 Description : randomizes the weights to new values
 -------------------------------------------------------------------------*/
void randomize(void){
            for(counter = 0; counter < INPUTS; counter++){
                printf("\nEnter a random weight[%ld]: ", counter);
                scanf("%ld", &classifier.weights[counter]);
            }
}

/*-------------------------------------------------------------------------
 Function name   : print
 Description : prints the weights
 -------------------------------------------------------------------------*/
void print(void){
            for(counter = 0; counter < INPUTS; counter++){
                printf("\nWeight [%ld] : %ld", counter, classifier.weights[counter]);
            }
}

/*-------------------------------------------------------------------------
 Function name   : feedforward
 Description : feedforward the inputs by multiplying input vector and weight
                              vector
 -------------------------------------------------------------------------*/
void feedforward(void){
            long sum = 0;
```

```
            for(counter = 0; counter < INPUTS; counter++){
                sum += classifier.inp[counter] * classifier.weights[counter];
            }

            classifier.function_out = sum;

            if(sum > THRESHOLD){
                classifier.outp = 1;
            }
            else{
                classifier.outp = 0;
            }

            printf("\nFunctional output: %ld,  threshold output: %ld", classifier.function_out,
classifier.outp);
}

/*-------------------------------------------------------------------------
 Function name   : backpropaclassifier
 Description : backpropaclassifier by finding the error and update the weight
                                vector
 -------------------------------------------------------------------------*/
void backpropagate(void){

            if(classifier.target - classifier.outp > 0){
                printf("\nTarget > functional output,");
                classifier.error = 1;
            }
            else if(classifier.target - classifier.outp < 0){
                printf("\nTarget < functional output,");
                classifier.error = -1;
            }
            else{
                printf("\nTarget = functional output,");
                classifier.error = 0;
            }

            printf("\terror: %ld", classifier.error);

            for(counter = 0; counter < INPUTS; counter++){
                classifier.weights[counter] += classifier.error * LEARNING_RATE *
classifier.inp[counter];
                printf("\tweights[%ld]: %ld", counter, classifier.weights[counter]);
            }
            printf("\n");
}

/*-------------------------------------------------------------------------
 Function name   : train
 Description : trains the perceptron using following algo.
            1. Select the first train set
            2. Feed-forward the inputs
            3. Back-propagate
            4. Select next train set
            5. Repeat till good level of convergence is achieved
-------------------------------------------------------------------------*/
void train(void){
            int samples;                                // keeps track of training sample
            int iterations=0;
            long current_sample_error = RAND_MAX;

            while(current_sample_error > 0){    // train till errors in all the samples become 0

                current_sample_error = 0;

                for(samples = 0; samples < TRAIN_SAMPLES; samples++){

                        /* set the input of perceptron to be inputs of current train set */
                        for(counter = 0; counter < INPUTS; counter++){
                                classifier.inp[counter] = training_set[samples].inputs[counter];
                        }
                        /* set the target of perceptron to be target of current train set */
                        classifier.target = training_set[samples].expected_out;

                        /* just for debugging */
                        printf("\nCurrent inputs: ");
```

```
                        for(counter = 0; counter < INPUTS; counter++){
                                printf("%ld, ", classifier.inp[counter]);
                        }
                        printf("\ttarget: %ld", classifier.target);

                        feedforward();                  // feed forward
                        backpropagate();        // backpropagate

                        if(abs(classifier.error)){
                           current_sample_error += abs(classifier.error);
                        }
                        //current_sample_error += classifier.error;
                }
                iterations++;
            }

            printf("\nTraining finished in %ld no. of iterations", iterations);
}

/*-------------------------------------------------------------------------
 Function name   : test
 Description : test the perceptron by using feedforward
-------------------------------------------------------------------------*/
void test(void){
            long input;

            /* take user values for inputs */
            for(counter = 0; counter < INPUTS; counter++){
                printf("\nEnter input (%ld) : ", counter);
                scanf("%ld", &input);
                classifier.inp[counter] = input;
            }

            feedforward();

            printf("\nThe functional output of the perceptron : %ld, classifier output : %ld",
                        classifier.function_out, classifier.outp);

}
```

PRACTICAL # 4

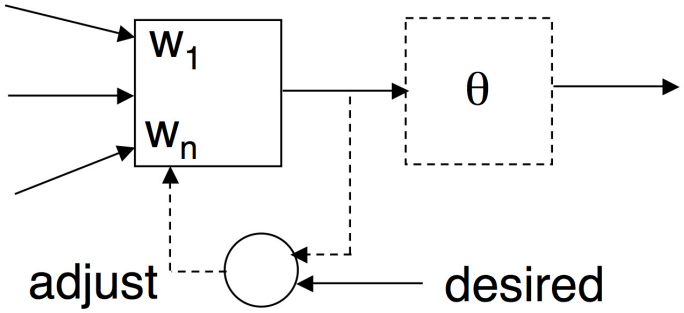
Adaline classification
(June 14, 2009)

Objectives
To implement error-corrective learning for classification using threshold activation function for adaline.

Description
Implement an adaline and train it for classification using error-correction and threshold activation function. The adaline will be trained for 4 different train set.

Platform
Any platform with any language of choice.

Model



$$\Delta\omega = \varepsilon \eta [-1, x1, x2, \ldots, xn]$$

(Refer to page 23 – 25 of Unit 3)

Here's my program.

```
/*-------------------------------------------------------------------------
 Program name: error corrective learning for classification using adaline
                        (program # 4)
 Description : This program demonstrates implementation of adaline for
                        error corrective learning to establish association  between inputs
                        and logical output and classifies the inputs into two classes.

                        Uses threshold activation function.

 By                      : Pramod Parajuli
 Date            : June 13, 2009
-------------------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define INPUTS          3
#define TRAIN_SAMPLES   4

#define THRESHOLD       0

/* let's define a structure to represent the adaline */
typedef struct
{
                double inp[INPUTS];                     // inputs to the adaline
                double weights[INPUTS];         // weights of the inputs
                double threshold;                       // threshold of the neuron
                double function_out;            // functional output of the neuron
                double outp;                            // threshold driven output of the
adaline
                double error;                           // error of the adaline
                double target;                          // target of adaline
}ADALINE;

/* let's create a classifier instance of ADALINE*/
ADALINE classifier;

/* this structure defines template for one training set
             - 1 input vector
             - one expected output  */
typedef struct {
             double inputs[INPUTS];
             double expected_out;
} TRAIN_ELEMENT;

/* for logical classifiers with two inputs, let's define 4 training sets */
TRAIN_ELEMENT training_set[TRAIN_SAMPLES] = {
             {{-1, 4, 5}, 1},
             {{-1, 6, 1}, 1},
             {{-1, 4, 1}, -1},
             {{-1, 1, 2}, -1}
};

long counter;                   // general purpose counter

double LEARNING_RATE;                   // learning rate of the perceptron
long EPOCH;                             // defines no. of iterations

void menu(void);                // displays a user help menu
void randomize(void);           // assigns random weights to the weight vector
void print(void);                       // print the weight vector
void train(void);                       // train the perceptron for adaptation
void feedforward(void);         // feedforward function  (inputs * weights)
void backpropaclassifier(void); // backpropagate (weight += error * input * learning rate)
void test(void);                // test the perceptron by using feedforward

int main (int argc, char * argv[]){
             int exit;                          // status variable for flow control
             char user_input[10];       // user input - string

             menu();                                    // display menu

             exit = 0;

             classifier.threshold = THRESHOLD;
```

```
            do{
                printf("\ncommand $  ");
                gets(user_input);

                if(strcmp(user_input, "SHUFFLE") == 0){
                        randomize();                            // assign random weights
                        print();                                // print the weight vector
                }
                else if(strcmp(user_input, "TRAIN") == 0){
                        printf("\nEnter the learning rate : ");  // learning rate
                        scanf("%lf", &LEARNING_RATE);
                        train();                                                    // train
                }
                else if(strcmp(user_input, "TEST") == 0){
                        test();
                }
                else if(strcmp(user_input, "PRINT") == 0){
                        print();
                }
                else if(strcmp(user_input, "EXIT") == 0){
                        exit = 1;
                }
                else{
                        menu();
                }

            }while(exit == 0);

            printf("\npress any key to exit...");
            getc(stdin);

            return 0;
}

/*-------------------------------------------------------------------------
 Function name   : menu
 Description : Displays menu
-------------------------------------------------------------------------*/
void menu(void){
            printf("\n\n\t\t\tPerceptron training for classification  (PROGRAM# 3.0)");
            printf("\n\tUse float value inputs for data entry");
            printf("\nPlease use following commands");
            printf("\n-------------------------------------------------------------------------");
            printf("\nSHUFFLE : to randomize weights   | TRAIN   : to train the perceptron");
            printf("\nTEST    : to test the perceptron | PRINT   : to print the weights");
            printf("\nHELP    : to display this menu   | EXIT    : to exit the program");
            printf("\n-------------------------------------------------------------------------\n");
}

/*-------------------------------------------------------------------------
 Function name   : randomize weights
 Description : randomizes the weights to new values
 -------------------------------------------------------------------------*/
void randomize(void){
            for(counter = 0; counter < INPUTS; counter++){
                printf("\nEnter a random weight[%ld]: ", counter);
                scanf("%lf", &classifier.weights[counter]);
            }
}

/*-------------------------------------------------------------------------
 Function name   : print
 Description : prints the weights
 -------------------------------------------------------------------------*/
void print(void){
            for(counter = 0; counter < INPUTS; counter++){
                printf("\nWeight [%ld] : %lf", counter, classifier.weights[counter]);
            }
}

/*-------------------------------------------------------------------------
 Function name   : feedforward
 Description : feedforward the inputs by multiplying input vector and weight
                            vector
 -------------------------------------------------------------------------*/
```

```
void feedforward(void){
            double sum = 0.0;
            for(counter = 0; counter < INPUTS; counter++){
               sum += classifier.inp[counter] * classifier.weights[counter];
            }

            classifier.function_out = sum;

            if(sum > THRESHOLD){
               classifier.outp = 1;
            }
            else{
               classifier.outp = -1;
            }

            printf("\nFunctional output: %lf,  threshold output: %lf, difference: %lf",
                        classifier.function_out, classifier.outp, classifier.target -
classifier.outp);
}

/*-------------------------------------------------------------------------
 Function name   : backpropaclassifier
 Description : backpropaclassifier by finding the error and update the weight
                              vector
 -------------------------------------------------------------------------*/
void backpropagate(void){

            classifier.error = classifier.target - classifier.function_out;

            printf("\terror: %lf", classifier.error);

            for(counter = 0; counter < INPUTS; counter++){
               classifier.weights[counter] += classifier.error * LEARNING_RATE *
classifier.inp[counter];
               printf("\tweights[%ld]: %lf", counter, classifier.weights[counter]);
            }

}

/*-------------------------------------------------------------------------
 Function name   : train
 Description : trains the perceptron using following algo.
            1. Select the first train set
            2. Feed-forward the inputs
            3. Back-propagate
            4. Select next train set
            5. Repeat till good level of convergence is achieved
-------------------------------------------------------------------------*/
void train(void){
            int samples;                                  // keeps track of training sample
            int iterations=0;
            long wrong = TRAIN_SAMPLES;
            double MSE;

            while(wrong > 0){   // train till errors in all the samples become 0

               wrong = 0;
               MSE = 0.0;

               for(samples = 0; samples < TRAIN_SAMPLES; samples++){

                        /* set the input of perceptron to be inputs of current train set */
                        for(counter = 0; counter < INPUTS; counter++){
                                 classifier.inp[counter] = training_set[samples].inputs[counter];
                        }
                        /* set the target of perceptron to be target of current train set */
                        classifier.target = training_set[samples].expected_out;

                        /* just for debugging */
                        printf("\nCurrent inputs: ");
                        for(counter = 0; counter < INPUTS; counter++){
                                 printf("%lf, ", classifier.inp[counter]);
                        }
                        printf("\ttarget: %lf", classifier.target);

                        feedforward();                    // feed forward
```

```
                        backpropagate();        // backpropagate

                        if(classifier.target != classifier.outp){
                                wrong++;
                        }
                        MSE += classifier.error * classifier.error;

                        //current_sample_error += classifier.error;
                }
                printf("\tMSE: %lf\n\n", MSE / TRAIN_SAMPLES);
                iterations++;
            }

            printf("\nTraining finished in %ld no. of iterations", iterations);
}

/*-------------------------------------------------------------------------
 Function name   : test
 Description : test the perceptron by using feedforward
-------------------------------------------------------------------------*/
void test(void){
            double input;

            /* take user values for inputs */
            for(counter = 0; counter < INPUTS; counter++){
                printf("\nEnter input (%ld) : ", counter);
                scanf("%lf", &input);
                classifier.inp[counter] = input;
            }

            /* feed forward */

            feedforward();

            printf("\nThe functional output of the perceptron : %lf, classifier output : %lf",
                        classifier.function_out, classifier.outp);

}
```

PRACTICAL # 5

Adaline classification
(June 14, 2009)

Objectives

To implement perceptron program in Matlab using Neural Net ToolBox

Steps

1. Create input vectors and target vectors

```
p = [0 0 1 1; 0 1 0 1]
t = [0 0 0 1]

net = newp(p, t)
```

2. Train the network

```
net = train(net, p, t)
```

3. Simulate the network

```
output = sim(net, p)
```

4. Check weights.

```
net.IW{1, 1}
```

To modify the property of training, use `net.trainParam.<parameter name>`.

Use '`nntool`' GUI tool to model Neural Networks.

Test following activation functions

```
n = -5:0.1:5;
plot(n, hardlim(n), 'c+:');
plot(n, hardlims(n), 'c+:');
plot(n, purelin(n), 'c+:');
plot(n, logsig(n), '*');
```