The LMS algorithm was introduced by Widrow and Hoff in 1959. It has several names, including the Widrow-Hoff rule and the Recursive Least-Squares (RLS) algorithm. The LMS algorithm is designed to minimize the Mean-Square Error (MSE) of the output of a linear system.

LMS is a fast algorithm that minimizes the MSE. The MSE is the average of the weighted sum of the error for N training sample which defined as:

$$MSE = \frac{1}{2N} \sum_{j=1}^{N} (R_j - C_j)^2$$

where R is the output of the perceptron and Cj is the current test inputs.

In order to train the perceptron by using LMS, we can iterate the test set, taking a set of inputs, computing the output, comparing it to the actual result, and adjusting the weights of the perceptron to minimize the error. The learning rule of LMS is given as:

$$w_{new} = w_{old} + \eta \cdot (R - C)$$

where $\eta$ is the step size or learning rate. This has the effect of adjusting the weights to reduce the output error.

The implementation of LMS is very simple. Initially, the weights vector is initialized with small random weights. Then, the training data is presented to the perceptron, and the output is compared to the expected output. If the error is large, the weights are adjusted. If the error is small, no adjustment is made.

```c
double weights[NUM_WEIGHTS];
#define MAX_TESTS 4
const training_data_t training_set[MAX_TESTS]={{-1.0,-1.0,-1.0},{-1.0,1.0,-1.0},{1.0,-1.0,-1.0},{1.0,1.0,1.0}};
double compute_output(int test)
Learning Algorithms of Neural Network: Least Mean-Square (LMS) Algorithm

```c
{ double result;
    result = ((training_set[test].a*weights[0])+(training_set[test].b*weights[1])+(1.0 * weights[2]));
    return(result);
}
int classify(int test)
{ double result; result = compute_output(test);
    if (result > 0.0) return 1; else return -1; }
double MSE(void)
{ int test; double sum = 0.0;
    for (test = 0; test < MAX_TESTS; test++)
    { sum += sqr(training_set[test].expected-compute_output(test)); } return (sum/(double)MAX_TESTS);
}
int main()
{ int i, test; double result, error; RANDINIT();
    for (i=0; <iNUM_WEIGHTS;i++)
    { weights[i]=RAND_WEIGHT; }
    while (MSE() > 0.26)
    { test = RANDMAX(MAX_TESTS); result = compute_output(test); error = training_set[test].expected - result;
        weights[0] += (p*error*training_set[test].a); weights[1] += (p*error*training_set[test].b);
        weights[2] += (p*error); printf("mse = %g\n",MSE(i)); }
    for(i=0;i <MAX_TESTS;i++)
    { printf("%g AND %g = %d\n", training_set[i].a, training_set[i].b, classify(i)); } return 0; }
```

References