Advanced Soft Computing Lectuer Notes

Yaowen Mei

May 2021

1 Lecture-1 May, 18, 2021

Soft computing is also known as Artificial Intelligent, AI, which is a new concept comparing to the conventional model-based **hard computing**. There are 3 types of soft computing models:

1. **Neural Networks** & Machine learning
   - Data Driven, Learn from data
   - **Limit**: not good on non-numerical data
2. **Fuzzy Systems**, which mimics human decision
   - Use experience or expert knowledge through Fuzzy rules
   - **Limit**: difficult to handle the uncertainties not in the knowledge domain
3. **Genetic Algorithm** (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO), ACO(???) and Artificial Immune System (AIS).
   - Use evolutionary computing method to provide the optimal solutions
   - **Limit**: normally very slow

These 3 types of mythologies are complementary, based on your specific application or problem, choose the MOST suitable method, or combining some of them, including traditional methods. The proposed solution should be better or least on some aspects better than existing solutions

---

1.1 Neural Networks

There are two types of Neural Networks (NN):

- Biological NN: based on biological structure
- Artificial NN: less biological structure, more like a math unit

There are three types of learning for neural networks:

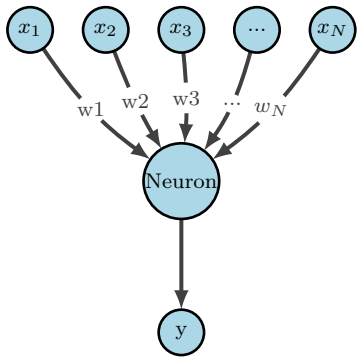
1. Supervised Learning: where the supervisor is always available to tell the error and the learning is to reduce errors
2. Unsupervised Learning: no supervisor, learning is based on the feature of objects. E.g. classifying apples into Good, Normal, Bad categorise. This type of learning is mainly used for classification/recognition. There is no error involved here

3. Reinforcement Learning: No errors/feedback at all the time, however, it has a delayed and overall score/feedback.

1.2 The Model of A Single Neuron

For a neuron, it has many inputs, but only one output. x_1, x_2, ... x_N are the input from other N neurons. w_1,w_2, ... ,w_N, are the connection weights.

$$\text{Total Input} = \sum_{i=1}^{N} w_i x_i \tag{1.1}$$



One fundamental important thing for a neuron is it has a threshold, let θ be the threshold, $f(...)$ is the activation function, also called input-output function, we have the one neuron model as:

$$y = f(\sum_{i=1}^{N} w_i x_i - \theta) \tag{1.2}$$

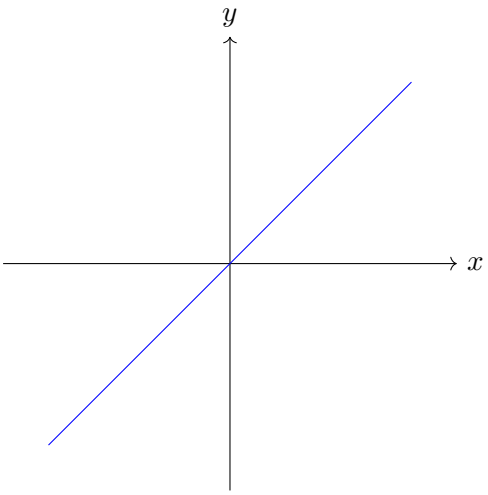
1.3 Commonly used activation functions

There are 6 types of commonly used activation functions:

1.3.1 Linear

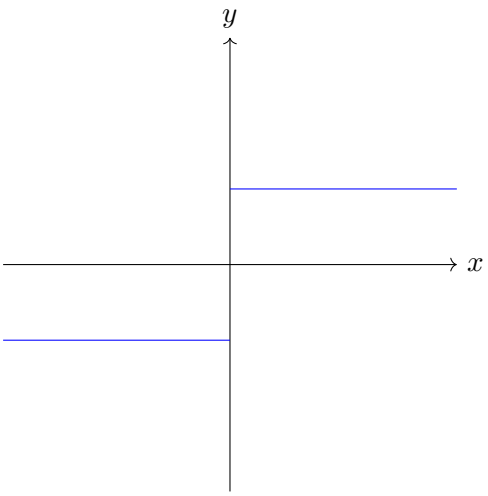
:

$$y = f(x) = x$$



1.3.2 Hard Limiter, HL

$$y = f(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ -1 \text{ if } x < 0 \end{cases} = \text{sign}(x)$$

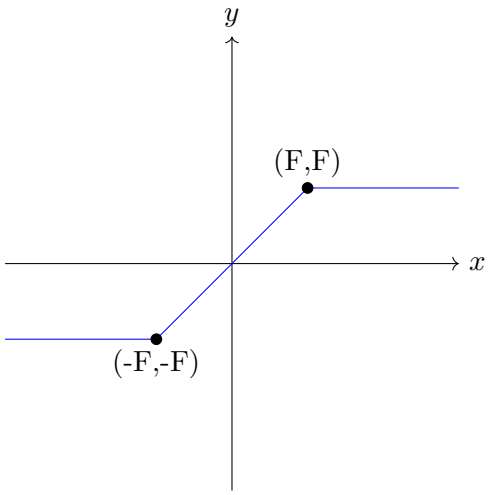


Alternatively, this can be written as:

$$y = f(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$$

1.3.3 Ramplimitter, piecewise limitter

$$y = f(x) = \begin{cases} F & if & x \geq F \\ x & if & -F \leq x \leq F \\ -F & if & x \leq F \end{cases}$$

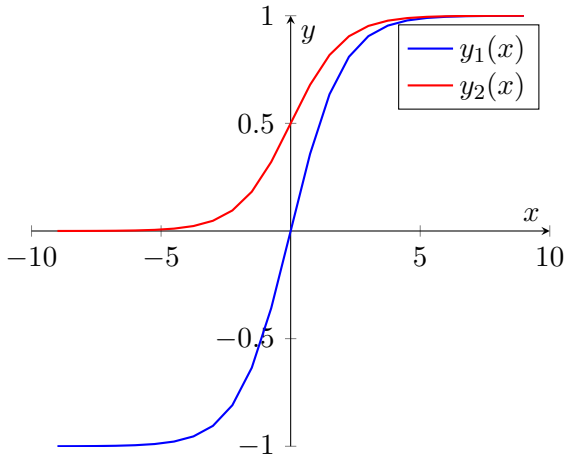


1.3.4 Sigmoid, the most popular and commonly used function

$$y_1 = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \tanh(x)$$

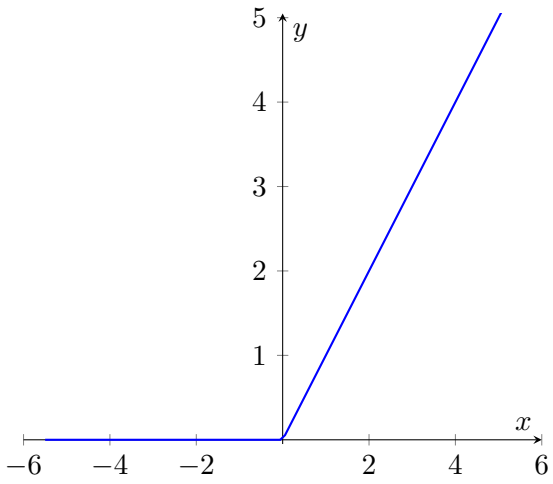
$$y_2 = \frac{1}{1 + e^{-x}}$$

$$y_2 = \tfrac{1}{2}(y_1 + 1)$$
$$y_1 = 2(y_2 - 0.5)$$



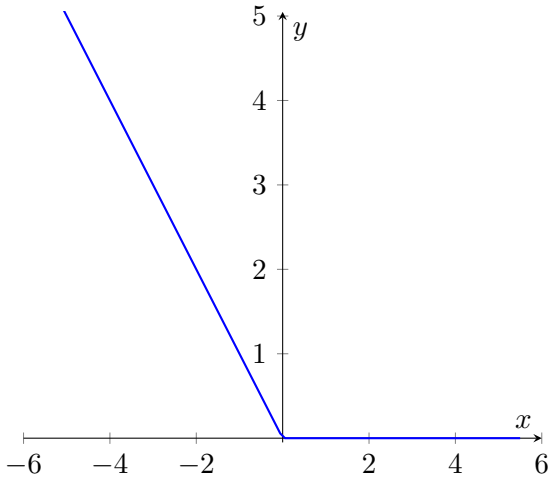
1.3.5 Linear-Above Threshold

$$y = f(x) = [x]^+ = \begin{cases} x & if \quad x \geq 0 \\ 0 & if \quad x < 0 \end{cases} = \max(0, x)$$



1.3.6 Linear-Below Threshold

$$y = f(x) = [x]^{-} = \begin{cases} 0 & if \quad x \geq 0 \\ -x & if \quad x < 0 \end{cases} = \max(0, -x)$$



1.4 Model of A Single Layer Neural Network

For a fully connected single layer neural network has M neurons (M outputs) and N inputs, for the 1st neuron, we have:

$$y_1 = f(\sum_{i=1}^{N} w_{1i} x_i - \theta_1)$$

for the jth neuron, we have:

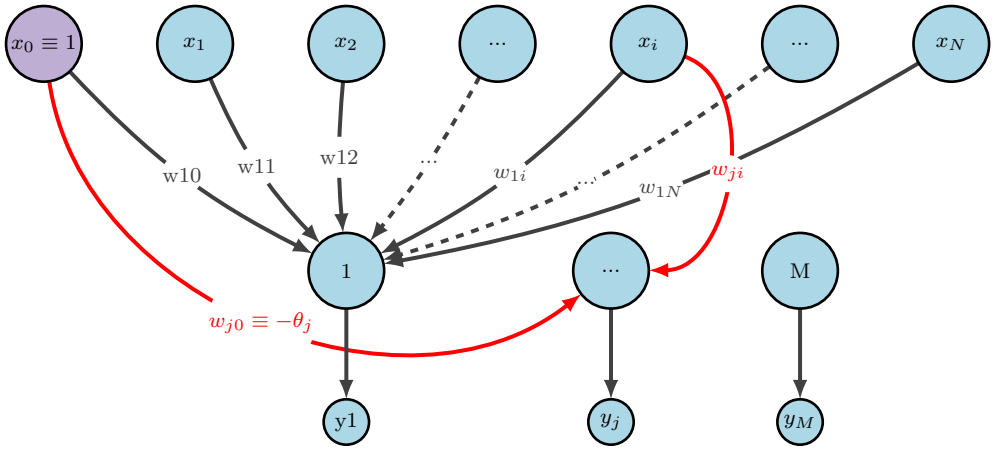
$$y_j = f(\sum_{i=1}^{N} w_{ji}x_i - \theta_j)$$

where j could be any number between 1 to M.

If we choose $-\theta_j = x_0 w_{j0}$: i.e. $x_0 \equiv 1$, and then, $w_{j0} \equiv -\theta_0$ Then, the argumented y becomes:

$$y_j = f(\sum_{i=0}^{N} w_{ji}x_i)$$

where j could be any number between 1 to M, i = 0 is for the threshold.



Write in the compact format: $\mathbb{Y} = f(\mathbb{W}\mathbb{X}^*)$, where $\mathbb{X}^*$ is a N+1 by 1 vector. $\mathbb{W}$ is the weight matrix with size M by N+1

$$\mathbb{W} = \begin{pmatrix} w_{10} & w_{11} & \cdots & w_{1N} \\ w_{20} & w_{21} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M0} & w_{M1} & \cdots & w_{MN} \end{pmatrix}$$

1.5 Multi-Layer Neural Network Structure

3-layer neural network is the most popular choice, actually there are only 2 layer of neurons (input layers which are not neurons, plus one layer of hidden neurons, and one layer of output neurons) For a fully connected 3 layers neural network has M outputs, Q hidden neurons, and N inputs:

$$\mathbb{H} = g(\mathbb{V}\mathbb{X}^*)$$

$$\mathbb{Y} = f(\mathbb{W}\mathbb{H}^*)$$

5-layer neural network has one input layer, 3 hidden layer and 1 output layer:

$$\mathbb{H}_{\mathbb{1}} = g(\mathbb{V}\mathbb{X}^*)$$

$$\mathbb{H}_{\mathbb{2}} = g(\mathbb{V}_{\mathbb{1}}\mathbb{H}_{\mathbb{1}}{}^*)$$

$$\mathbb{H}_{\mathbb{3}} = g(\mathbb{V}_{\mathbb{2}}\mathbb{H}_{\mathbb{2}}{}^*)$$

$$\mathbb{Y} = f(\mathbb{W}\mathbb{H}_{\mathbb{3}}{}^*)$$

1.6 Conclusion

1. the output activation function can be linear or non-linear
2. the activation function for neurons in the hidden layer must be nonlinear
3. the connections are only from layer i to i+1
4. the thresholds are essentially important

2 Lecture-2 May, 20, 2021

2.1 Why need non-linear activation function for hidden layer?

2.2 Why threshold is essentially important?

2.3 Learning in Artificial Neural Network, ANN

Recall that there are 3 types of learning for neural networks:

- Supervised
- Unsupervised
- Reinforcement

Perceptron is one of the supervised learning method . The activation function is hard limiter (+/- 1) function. The activation function is nonlinear and perceptron is used for classification.

2.3.1 Single Neuron Perceptron

Consider a single neuron with 2 inputs, the model can be written as

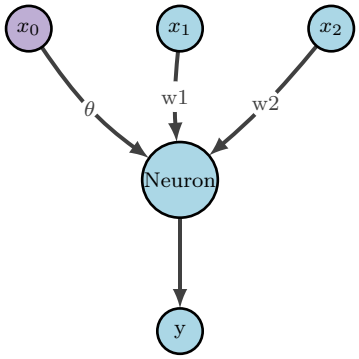
$$y = hardlimit(\mathbb{W}\mathbb{X} + \theta) = hl(w_1x_1 + w_2x_2 + \theta) \tag{2.1}$$

Where we use the second representation of the hard limiter function:

$$y = hl(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases} \tag{2.2}$$

The "decision boundary" is determined by

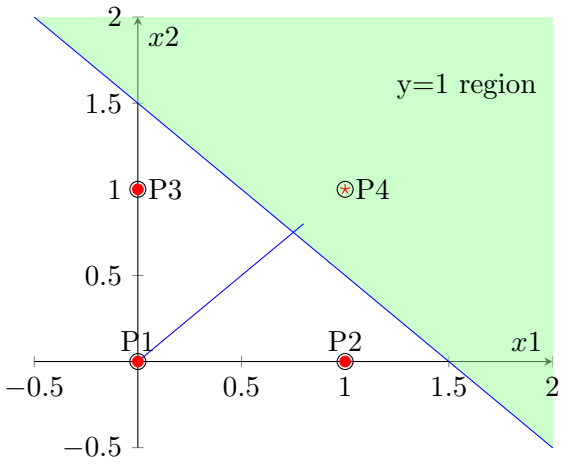
$$w_1x_1 + w_2x_2 + \theta = 0 \tag{2.3}$$



Exmp: Build a perceptron to simulate an AND gate

The question is basically asking to find the decision boundary of a AND gate, there are Inf. many possible solutions for this question, and we are going to find one via supervised learning.

	X1	**X2**	**y**
P1	0	0	0
P2	1	0	0
P3	0	1	0
P4	1	1	1



By observation, one can pick the decision boundary passing through (1.5, 0) and (0, 1.5), so that this decision boundary can separate dot from star.

To find the w_1, and w_2 and θ for this decision boundary: Normally, we choose $\mathbb{W}$ vector perpendicular to the D.B, and toward the y = 1 region. So, what should we do in general case if W is a M by N+1 matrix???

From graph, we can tell that $\mathbb{W}$ is in the 45 degree angle (perpendicular to D.B. and toward to y = 1 region). Let $\mathbb{W} = [1, 1]$, then we have the D.B. as:

$$x_1 + x_2 + \theta = 0 \tag{2.4}$$

We also know that the D.B. passing point (1.5, 0):

$$1.5 + 0 + \theta = 0 \tag{2.5}$$

Finally, we get $\theta = -1.5$, and the D.B can be written as:

$$x_1 + x_2 - 1.5 = 0 \tag{2.6}$$

2.3.2 Multiple Neurons Perceptron

The model for multiple neurons perceptron with N inputs, M outputs, and M neurons, its D.B. of the jth neuron can be written as:

$$\mathbb{W}_j \mathbb{X} + \theta_j = 0 \tag{2.7}$$

In general, we have

$$\mathbb{W}_{(MXN)} \mathbb{X}_{(NX1)} + \theta_{(MX1)} = 0 \tag{2.8}$$

Note that for a single neuron, y = 1 or 0, the perceptron provides two possible classes; while for M neurons, we have $y = 2^M$ possible classes;

2.4 Perceptron Learning Rules

$$\mathbb{W}^{new}_{MXN} = \mathbb{W}^{old} + (\mathbb{t} - \mathbb{y})\mathbb{X}^T = \mathbb{W}^{old} + \mathbb{e}_{MX1}\mathbb{X}^T_{1XN}$$

$$\theta^{new}_{MX1} = \theta^{old} + \mathbb{e}_{MX1}$$

If combine the weight and threshold together, we can get:

$$\mathbb{W}^{new}_{MX(N+1)} = \mathbb{W}^{old} + (\mathbb{t} - \mathbb{y})\mathbb{X}^T = \mathbb{W}^{old} + \mathbb{e}_{MX1}\mathbb{X}^{*T}_{1X(N+1)}$$

Where $\mathbb{X}^*$ is the argument-ed input vector, and can be written as below:

$$\begin{bmatrix} x_0 \equiv 1 \\ x_1 \\ \vdots \\ x_N \end{bmatrix}$$

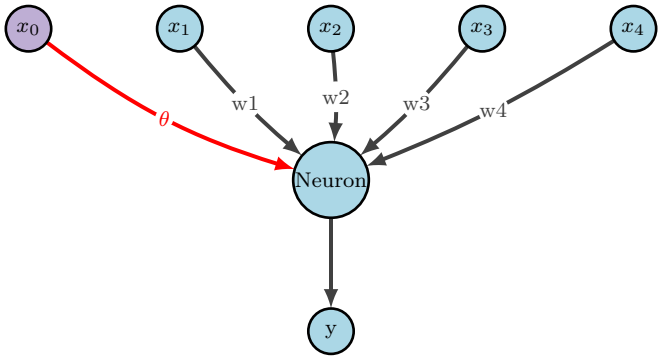
Exmp

Design a single neural perceptron by learning for a 3 input system, where:

$$\mathbb{X}_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} \quad , \quad t_1 = 0$$

$$\mathbb{X}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \quad , \quad t_2 = 1$$

	X_0	X_1	X_2	X_3	t
P1	1	1	-1	-1	0
P2	1	1	1	-1	1



Solution: We will randomly choose:

$$\mathbb{W} = \begin{bmatrix} 0.5 \\ -1 \\ -0.5 \end{bmatrix}^T \quad , \quad \theta = 0.5$$

As a rule of thumb, $\mathbb{W}$ should between -1 and +1. One can re-write $\mathbb{W}$ in argumented format:

$$\mathbb{W}_0 = \begin{bmatrix} 0.5 \\ 0.5 \\ -1 \\ -0.5 \end{bmatrix}^T$$

For $\mathbb{X}_1$:

$$y_1 = hl(\mathbb{W}^0 \mathbb{X}_1^*) = 1$$

$$e = t - y = 0 - 1 = -1$$

$$\mathbb{W}^1 = \mathbb{W}^0 - \mathbb{X}_1^* = \begin{bmatrix} -0.5 \\ -0.5 \\ 0 \\ 0.5 \end{bmatrix}^T$$

For $\mathbb{X}_2$:

$$y_2 = hl(\mathbb{W}^1\mathbb{X}_2^*) = 0 \neq t_2$$

$$e = t - y = 1 - 0 = 1$$

$$\mathbb{W}^2 = \mathbb{W}^1 + \mathbb{X}_2^* = \begin{bmatrix} 0.5 \\ 0.5 \\ 1 \\ -0.5 \end{bmatrix}^T$$

For $\mathbb{X}_1$ (the 3rd iteration):

$$y_1 = hl(\mathbb{W}^2\mathbb{X}_1^*) = 1 \neq t_1$$

$$e = t - y = 0 - 1 = -1$$

$$\mathbb{W}^3 = \mathbb{W}^2 + \mathbb{X}_1^* = \begin{bmatrix} -0.5 \\ -0.5 \\ 2 \\ 0.5 \end{bmatrix}^T$$

For $\mathbb{X}_2$ (the 4th iteration), $e = 0$, requires no learning:

$$y_2 = hl(\mathbb{W}^3\mathbb{X}_2^*) = 1 = t_2$$

$$e = t - y = 1 - 1 = 0$$

For $\mathbb{X}_1$ (the 5th iteration), $e = 0$, requires no learning:

$$y_1 = hl(\mathbb{W}^3\mathbb{X}_1^*) = 0 = t_1$$

$$e = t - y = 0 - 0 = 0$$

We found one possible solution for this preceptron is:

$$\mathbb{W}^3 = \begin{bmatrix} -0.5 \\ -0.5 \\ 2 \\ 0.5 \end{bmatrix}^T$$

□

3 Lecture-3 May, 25, 2021

3.1 Review from the Last Lecture

We have learned Perceptron and it's learning strategy:

- Activation function **MUST** be Hard-Limiter [0,1]; WHY???
- Learningis: $\mathbb{W}^{new} = \mathbb{W}^{old} + (\mathbb{t} - \mathbb{y})\mathbb{X}^{*T} = \mathbb{W}^{old} + \mathbb{e}\mathbb{X}^{*T}$, for both weights and threshold;

3.2 Adaline and Madaline

Today, we are going to look at Adaline and Madaline:

- **Adaline** stands for **Ada**ptive **Line**ar Neuron Network;What the word linear means here as the activation function can be non-linear???
- **Madaline** stands for **M**any **Adaline** in Parallel; Mainly used for communication, DSP, Control, and Robotics.

These concepts were introduced by Barnard Widraw and Marcian Hoff in the 60s, together with LMS (least mean square) training algorithms.

3.3 Single Neuron Adaline Model

For a fully connected single neuron Adaline neural network, which has one neurons (one outputs) and N inputs, for the 1st neuron, we have:

$$y = f(\sum_{i=1}^{N} w_i x_i - \theta)$$

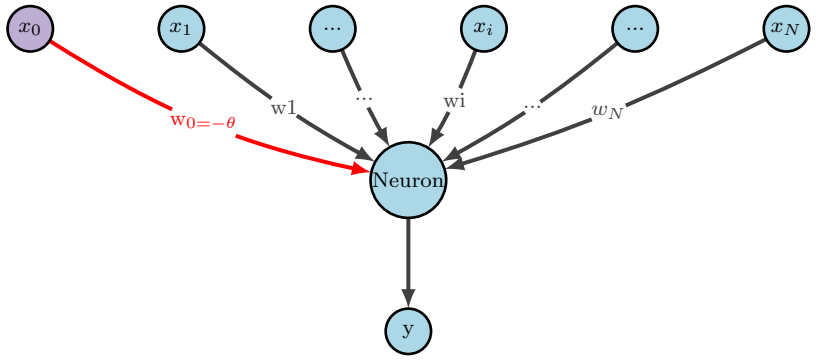
If we choose $-\theta_j = x_0 w_{j0}$: i.e. $x_0 \equiv 1$, and then, $w_{j0} \equiv -\theta_0$ Then, the argument-ed y becomes:

$$y = f(\sum_{i=0}^{N} w_i x_i)$$

Where i could be any number between 0 to N:

- w_i is the weight from the ith input to the neuron,
- $i = 0$ is the term for the threshold, where $w_0 = -\theta$

All these w_i are randomly initialized to $[-1, +1]$.



Activation function $f(a)$ can be either **linear** or **non-linear**. The popular choices are:

- $f(a) = a$ for linear activation function
- $f(a) = Sigmoid(a)$ for non-linear activation function

3.4 Least Mean Square (LMS) Learning rules

3.4.1 LMS for Adaline

Definition 3.4.1 (LMSE, LMS error)**.** For a single neuron Adaline model with one actual output, y, and target output, t (also called reference input or desired output), the Least Mean Square Error, e, is:

$$error = e = t - y$$

$$E = \frac{1}{2}e^2 = \frac{1}{2}(t-y)^2 \text{ (this is the squared error)}$$

The learning rule is simply that:

$$\begin{array}{rcccl} \Delta w_i & = & w_i^{new} - w_i^{old} & = & w_i^{k+1} - w_i^k \\ \Delta w_i & = & -\eta \frac{\partial E}{\partial w_i} & where & i \in [0, N] \end{array}$$

Where the partial derivative is the gradient of the squared error, the negative sign before η means the learning is in the direction of the reduction of squared error in the steepest path.

Bibliography