International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor (2018): 7.426

The Error Correction Learning with Delta Rule

Jogimol Joseph

Assistant Professor, Department of Computer Applications, Mount Zion College of Engineering, Pathanamthitta, Kerala, India

Abstract: ***Learning rule is a method or a mathematical logic includes an iterative process that helps a Neural Network to learn from the existing conditions and improve its performance. In machine learning, the delta rule is a gradient descent learning rule for updating the weights of the inputs to artificial neurons in a single-layer neural network. It is a special case of the more general back propagation algorithm.***

Keywords: Delta Rule, Back Propagation, Perceptron Error learning in ANN, Least Mean Square (LMS) Rule

1. Introduction

By early 1960's, the Delta Rule [also known as the *Widrow & Hoff Learning rule* or the *Least Mean Square* (LMS) rule] was invented by *Widrow and Hoff.* In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (**EW**). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back propagation algorithm is the most widely used method for determining the **EW**. The back-propagation algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each **EW** by first computing the **EA**, the rate at which the error changes as the activity level of a unit is changed. For output units, the **EA** is simply the difference between the actual and the desired output. To compute the **EA** for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the **EA**s of those output units and add the products. This sum equals the **EA** for the chosen hidden unit. After calculating all the **EA**s in the hidden layer just before the output layer, we can compute in like fashion the **EA**s for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the **EA** has been computed for a unit, it is straight forward to compute the **EW** for each incoming connection of the unit. The **EW** is the product of the EA and the activity through the incoming connection.

2. The Learning Rules in Neural Networks

Learning rules improves the Artificial Neural Network's performance and applies this rule over the network and updates the weights and bias levels of a network when a network simulates in a specific data environment. The following are the different learning rules in the Neural Network:

- **Hebbian learning rule –** It identifies how to modify the weights of nodes of a network.
- **Perceptron learning rule –** Network starts its learning by assigning a random value to each weight.
- **Delta learning rule –** Modification in sympatric weight of a node is equal to the multiplication of error and the input.
- **Correlation learning rule –** The correlation rule is the supervised learning.
- **Outstar learning rule –** We can use it when it assumes that nodes or neurons in a network arranged in a layer.

3. Delta Rule

Delta Rule uses the difference between *target activation* (i.e., target output values) and *obtained activation* to drive learning. For reasons discussed below, the use of a *threshold activation function* (as used in both the *McCulloch-Pitts* network and the perceptron) is dropped & instead a linear sum of products is used to calculate the activation of the output neuron (alternative activation functions can also be applied). Thus, the activation function is called a *Linear Activation function*, in which the output node's activation is simply equal to the sum of the network's respective input/weight products. The strength of network connections (i.e., the values of the weights) is adjusted to reduce the difference between *target* and *actual* output activation (i.e., error).

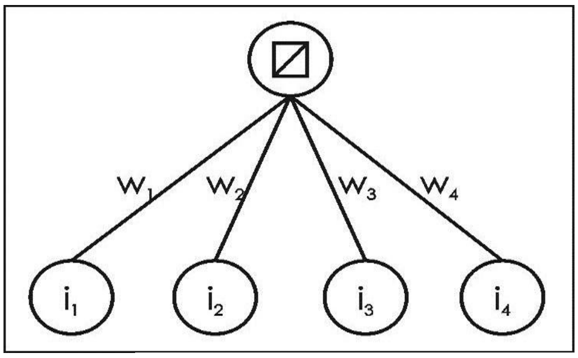

Figure 3.1: A graphical depiction of a simple two-layer network capable of deploying the Delta Rule

Definition 3.1 A linear activation function is a mapping from
f: R→R such that f(t)=t for all t in R
Suppose we are given a single-layer network with n input units and m linear output units, i.e. the output of the i-th neuron can be written as

$O_i = net_i = \langle w_i, x \rangle = w_{i1} x_1 + \cdots + w_{in} x_n$, for $i = 1 \ldots m$.

Assume we have the following training set $\{(x^1, y^1), \ldots, (x^k, y^k)\}$ where $x^k = (x_1^k, \ldots, x_n^k)$, $y^k = (y_1^k, \ldots, y_n^k)$, $k = 1, \ldots, K$. The basic idea of the delta learning rule is to define a measure of the overall performance of the system and then to find a way to optimize that performance.

During **forward propagation** through a network, the output (activation) of a given node is a function of its inputs. The inputs to a node, which are simply the products of the output of preceding nodes with their associated weights, are summed and then passed through an activation function before being sent out from the node. Thus, we have the following:

$$\mathbf{S}_j = \sum_i \boldsymbol{w}_{ij}\mathbf{a}_i \quad \text{and} \quad \mathbf{a}_i = f(\text{Si})$$

Where 'S_j' is the sum of all relevant products of weights and outputs from the previous layer i, 'w_{ij}' represents the relevant weights connecting layer i with layer j, 'a_i' represents the activation of nodes in the previous layer i, 'a_j' is the activation of the node at hand, and 'f' is the activation function.

The change weight can be commonly stated as:
$\Delta \mathbf{w_{ji}} = \boldsymbol{\varepsilon} \times (\mathbf{a_i} - \mathbf{t_i}) \times \mathbf{x_i}$ .Where $\mathbf{w_{ji}}$ is the j's i^th weight, $\boldsymbol{\varepsilon}$ is a constant known as **learning rate**, a_j, is the actual output ,t_j is the targeted output and x_i is the input

The Delta rule employs an error function which is given as the sum of the squares of the differences between all target and actual node activation for the output layer.
Error = $\sum (\mathbf{a_j} - \mathbf{t_j})^2$

The error correction learning procedure is simple enough in conception. The procedure is as follows: During training, an input is put into the network and flows through the network generating a set of values on the output units. Then, the actual output is compared with the desired target, and a match is computed. If the output and target match, no change is made to the net. However, if the output differs from the target a change must be made to some of the connections.

Example 3.1

Imagine the following inputs and outputs:

Table 3.1: Sample inputs and targeted outputs

	Inputs				*Outputs*
Network Node#	0	1	2	3	4
Activations	+1	-1	+1	-1	-1
	+1	+1	+1	+1	+1
	+1	+1	+1	-1	-1
	+1	-1	-1	+1	-1

Let all four weights associated with each input node are initially set to Zero and arbitrary learning rate (ε) of 0.25 is used for the time being.

Each training phase is presented to the network individually and weights are modified according to delta rule.

Training Phase #1:

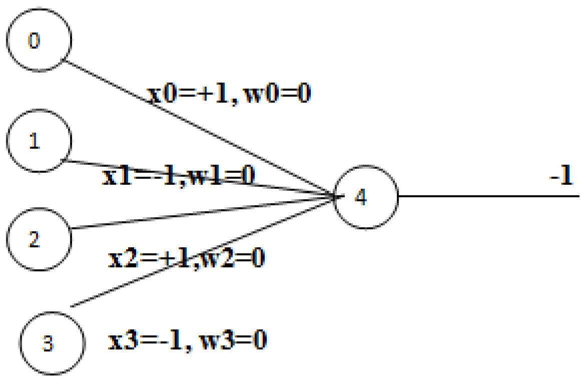

Figure 3.2: Training phase #1

The sum of product of inputs and corresponding weight is given by
$\sum x_i wi = x_0 w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3 = 0$, since all weights are initially set to Zero
Here actual output $a_1=0$ and targeted output $t_1=-1$.

Therefore **Error (1) = $(\mathbf{a_1} - \mathbf{t_1})^2$**
$\qquad\qquad\qquad\quad$ **= $(0-(-1))^2 = 1$** $\qquad$ (1)

$\Delta \mathbf{w_{12}} = \boldsymbol{\varepsilon} \times (\mathbf{a_1} - \mathbf{t_1}) \times \mathbf{x_i}$

$$\begin{cases} 0.25\times1\times(+1) = 0.25 \\ 0.25\times1\times(-1) = -0.25 \\ 0.25\times1\times(+1) = 0.25 \\ 0.25\times1\times(-1) = -0.25 \end{cases}$$

So, the weight to next phase →{0.25,-0.25, 0.25,-0.25}

Training Phase #2:

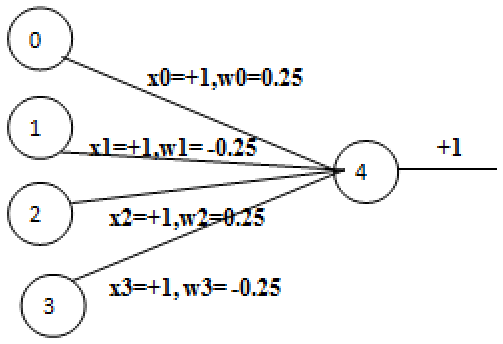

Figure 3.3: Training phase #2

$\sum x_i wi = x_0 w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3$
$= 1\times0.25 + 1\times-0.25 + 1\times0.25 + 1\times-0.25 = 0$

Therefore the presentation of second set of training input causes the network a sum of product to zero again.
Here $t_2=0$, $a_2=1$.
So, **Error (2) = $(\mathbf{a_2} - \mathbf{t_2})$**
$\qquad\qquad\quad$ **= $(1-0)^2 = 1$** $\qquad$ (2)

$\Delta \mathbf{w_{23}} = \boldsymbol{\varepsilon} \times (\mathbf{a_2} - \mathbf{t_2}) \times \mathbf{x_i}$

$$\begin{cases} 0.25\times1\times(+1) = 0.25 \\ 0.25\times1\times(+1) = 0.25 \\ 0.25\times1\times(+1) = 0.25 \\ 0.25\times1\times(+1) = 0.25 \end{cases}$$

Therefore the net weight after second phase=weight at phase1 + weight at phase2.

$$\begin{cases} 0.25+0.25=0.50 \\ 0.25-0.25=0 \\ 0.25+0.25=0.50 \\ 0.25-0.25=0 \end{cases}$$

So, the weight to next phase →{0.50, 0, 0.50, 0}

Training Phase #3:

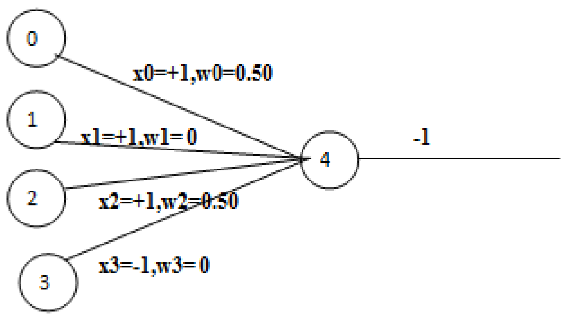

Figure 3.4: Training phase #3

$\sum x_i w_i = x_0 w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3$
$= 1\times0.50+1\times0+1\times0.50+1\times0=1$

Hence $t_3=1$, $a_3=-1$.

So, **Error (3) $= (a_3-t_3)$**
$= (-1-1)^2 = 4$ (3)

$\Delta w_{34}=\varepsilon \times (a_3 - t_3) \times x_i$

$$\begin{cases} 0.25\times(-2)\times(+1)=-0.50 \\ 0.25\times(-2)\times(+1)=-0.50 \\ 0.25\times(-2)\times(+1)=-0.50 \\ 0.25\times(-2)\times(+1)=0.50 \end{cases}$$

Therefore the net weight after Third phase

$$\begin{cases} -0.50+0.50=0 \\ -0.50+0=-0.50 \\ -0.50+0.50=0 \\ 0.50+0=0.50 \end{cases}$$

So, the weight to next phase →{0,-0.50, 0, 0.50}

Training Phase #4:

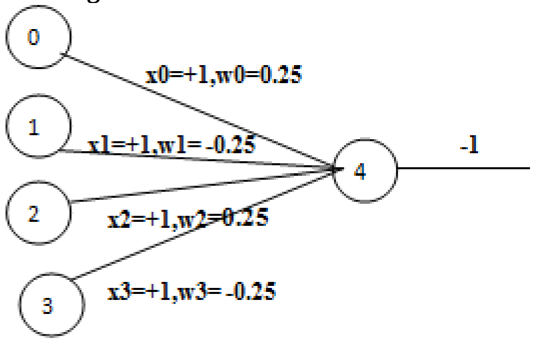

Figure 3.5: Training phase #4

$\sum x_i w_i = x_0 w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3$
$= 1\times0+-1\times-0.50+-1\times0+1\times0.50=1$

Hence $t_4=1$, $a_3=-1$.

So, **Error (4) $= (a_4-t_4)$**
$= (-1-1)^2 = 4$ (4)

$\Delta w_{4out}=\varepsilon \times (a_4 - t_4) \times x_i$

$$\begin{cases} 0.25\times(-2)\times(+1)=-0.50 \\ 0.25\times(-2)\times(-1)=0.50 \\ 0.25\times(-2)\times(-1)=0.50 \\ 0.25\times(-2)\times(+1)=-0.50 \end{cases}$$

Therefore the net weight after Fourth phase

$$\begin{cases} -0.50+0=-0.50 \\ 0.50+-0.50=0 \\ 0.50+0=0.50 \\ -0.50+0.50=0 \end{cases}$$

So, the change in weight after phase 4→{-0.50, 0, 0.50, 0}

Hence the Total Sum of Errors = 1+1+4+4=10

4. Conclusion

We have intensively studied the learning characteristics of Delta Rule on feed forward networks and how to employ back propagation for learning error. Our learning procedure requires only change in weight to be proportional. True gradient descent requires that infinitesimal steps be taken. The constant of proportionality is the learning rate here. Larger this constant, larger the changes in the weight. For practical purpose we choose a learning rate that is as large as possible without leading to oscillation.

References

[1] An introduction to neural computing. Aleksander, I. and Morton, H. 2nd edition
[2] Learning internal representations by error propagation by Rumelhart, Hinton and Williams (1986).
[3] A Tutorial on The delta learning rule, Robert Fuller, Institute for Advanced Management Systems Research Department of Information Technologies Abo Akademi University
[4] A Journal on “A simple procedure for pruning back-propagation trained neural networks”, E.D Karnin, IBM Science. And Tech., Technioncity, Haifa, Israel
[5] Industrial Applications of Neural Networks (research reports Esprit, I.F.Croall, J.P.Mason)
[6] DARPA Neural Network Study (October, 1987-February, 1989). MIT Lincoln Lab. Neural Networks, Eric Davalo and Patrick Naim
[7] Neural Networks, Eric Davalo and Patrick Naim.