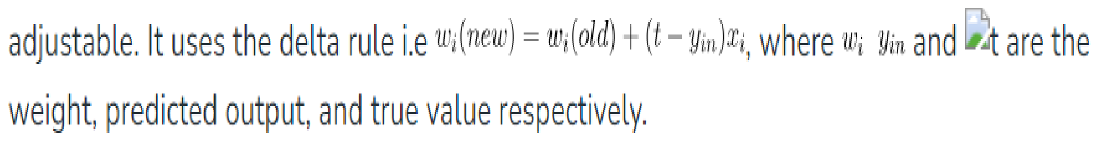
An artificial neural network inspired by the human neural system is a network used to process the data which consist of three types of layer i.e input layer, the hidden layer, and the output layer. The basic neural network contains only two layers which are the input and output layers. The layers are connected with the weighted path which is used to find net input data. In this section, we will discuss two basic types of neural networks Adaline which doesn’t have any hidden layer, and Madaline which has one hidden layer.

1. Adaline (Adaptive Linear Neural) :

A network with a single linear unit is called Adaline (Adaptive Linear Neural). A unit with a linear activation function is called a linear unit.

In Adaline, there is only one output unit and output values are bipolar (+1,-1).

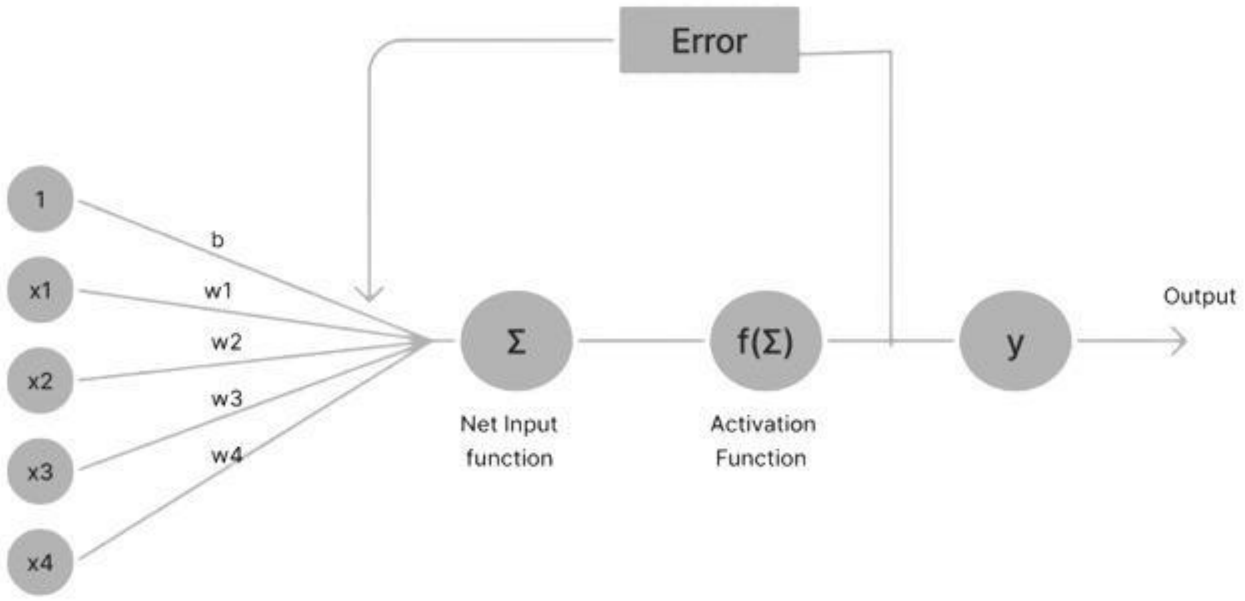
Weights between the input unit and output unit are adjustable.



adjustable. It uses the delta rule i.e $w_i(new) = w_i(old) + (t - y_{in})x_i$, where w_i y_{in} and t are the weight, predicted output, and true value respectively.

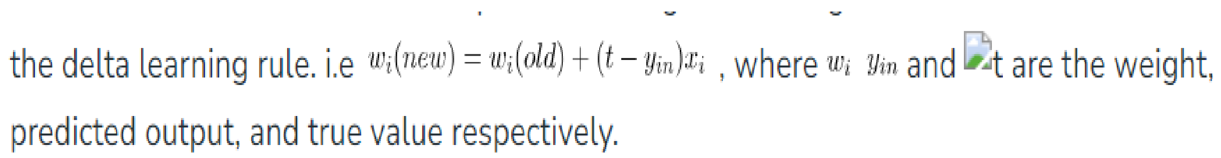
- The learning rule is found to minimize the mean square error between activation and target values. Adaline consists of trainable weights, it compares actual output with calculated output, and based on error training algorithm is applied.

Workflow:



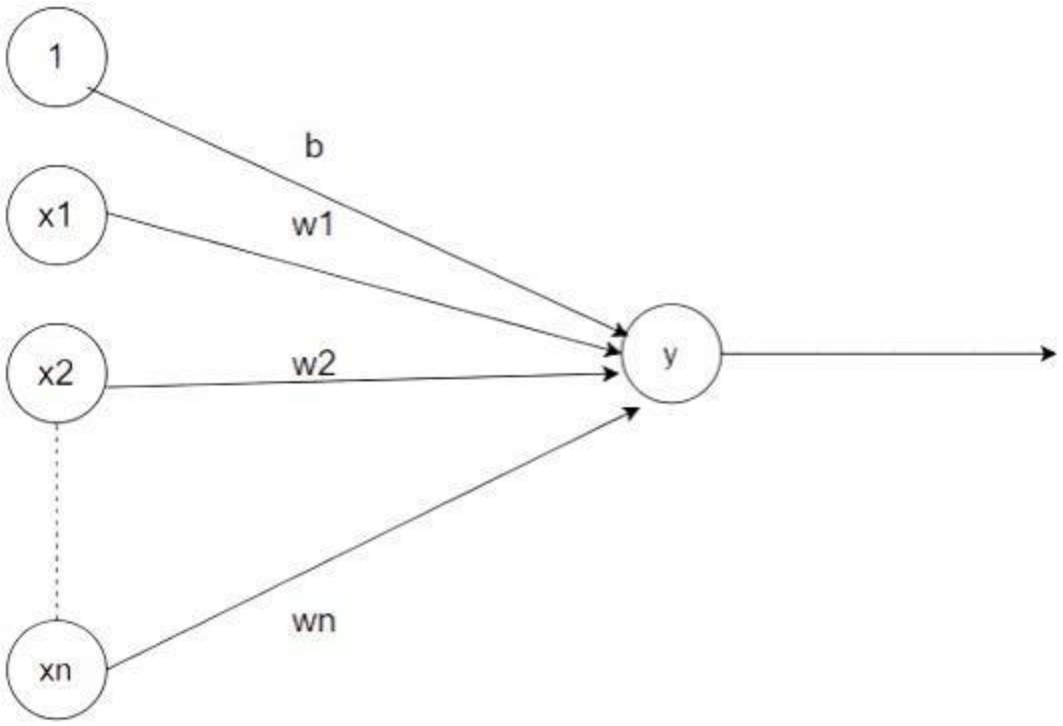
Adaline

First, calculate the net input to your Adaline network then apply the activation function to its output then compare it with the original output if both the equal, then give the output else send an error back to the network and update the weight according to the error which is calculated by the delta learning rule.



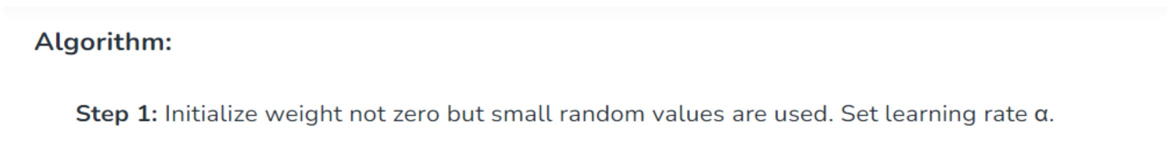
the delta learning rule. i.e $w_i(new) = w_i(old) + (t - y_{in})x_i$, where w_i y_{in} and t are the weight, predicted output, and true value respectively.

Architecture:



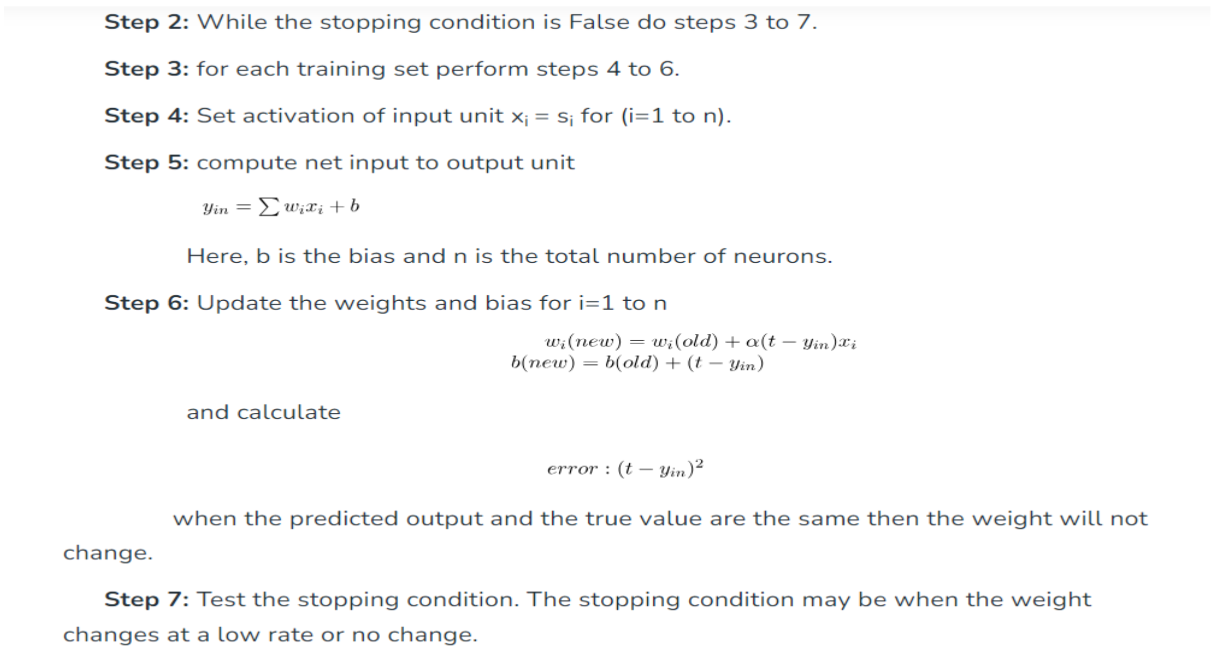
Adaline

In Adaline, all the input neuron is directly connected to the output neuron with the weighted connected path. There is a bias b of activation function 1 is present.



Algorithm:

Step 1: Initialize weight not zero but small random values are used. Set learning rate α.



Step 2: While the stopping condition is False do steps 3 to 7.

Step 3: for each training set perform steps 4 to 6.

Step 4: Set activation of input unit $x_i = s_i$ for (i=1 to n).

Step 5: compute net input to output unit

$$y_{in} = \sum w_i x_i + b$$

Here, b is the bias and n is the total number of neurons.

Step 6: Update the weights and bias for i=1 to n

$$w_i(new) = w_i(old) + \alpha(t - y_{in})x_i$$
$$b(new) = b(old) + (t - y_{in})$$

and calculate

$$error : (t - y_{in})^2$$

when the predicted output and the true value are the same then the weight will not change.

Step 7: Test the stopping condition. The stopping condition may be when the weight changes at a low rate or no change.

Implementations

Problem: Design OR gate using Adaline Network?
Solution :

- Initially**,** all weights are assumed to be small random values, say 0.1, and set learning rule to 0.1.
- Also, set the least squared error to 2.
- The weights will be updated until the total error is greater than the least squared error.

x_1	x_2	t
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

- Calculate the net input

(when $x_1=x_2=1$)

- Now compute, $(t-y_{in})=(1-0.3)=0.7$
- Now, update the weights and bias

- calculate the error

Similarly, repeat the same steps for other input vectors and you will get.

x_1	x_2	t	y_{in}	$(t-y_{in})$	Δw_1	Δw_2	Δb	w_1 (0.1)	w_2 (0.1)	b (0.1)	$(t-y_{in})$^2
1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	0.17	0.83	0.083	-0.083	0.083	0.253	0.087	0.253	0.69
-1	1	1	0.087	0.913	-0.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
-1	-1	-1	0.0043	-1.0043	0.1004	0.1004	-0.1004	0.2621	0.2787	0.2439	1.01

This is epoch 1 where the total error is 0.49 + 0.69 + 0.83 + 1.01 = 3.02 so more epochs will run until the total error becomes less than equal to the least squared error i.e 2.