KTU LECTURE NOTES

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

DOWNLOAD KTU ASSIST APP FROM PLAYSTORE

SUPERVISED LEARNING NETWORK

LECTURE 3

November 26, 2017

Perceptron Networks

Characteristics

- Perceptron network consists of 3 units: *sensory unit (input unit)*, *associator unit (hidden unit)*, and *response unit (output unit)*.
- Sensory units are connected to associator units with fixed weights having values *1,0 or -1*.
- The binary activation function is used in sensory unit and associator unit.
- The response unit has an activation of *1,0 or -1*.

- The output of the perceptron network is given by,

$$y = f(y_{in})$$

where $f(y_{in})$ is activation function and is defined as,

$$y = f(y_{in}) = \begin{cases} 1 \text{ if } y_{in} \geq \theta \\ 0 \text{ if } -\theta \leq y_{in} \leq \theta \\ -1 \text{ if } y_{in} < \theta \end{cases}$$

- The perceptron learning rule is used in the weight updation between *associator unit and response unit.*
- The error calculation is based on the comparison of the values of targets with those of the calculated outputs.

- The weights will be adjusted on the basis of the learning rule if an error has occurred for a particular training pattern.

$$w_i(new) = w_i(old) + \alpha t x_i$$
$$b(new) = b(old) + \alpha t$$

where,

$$t - \text{ target value} (+1 or -1)$$
$$\alpha - \text{ learning rate}$$

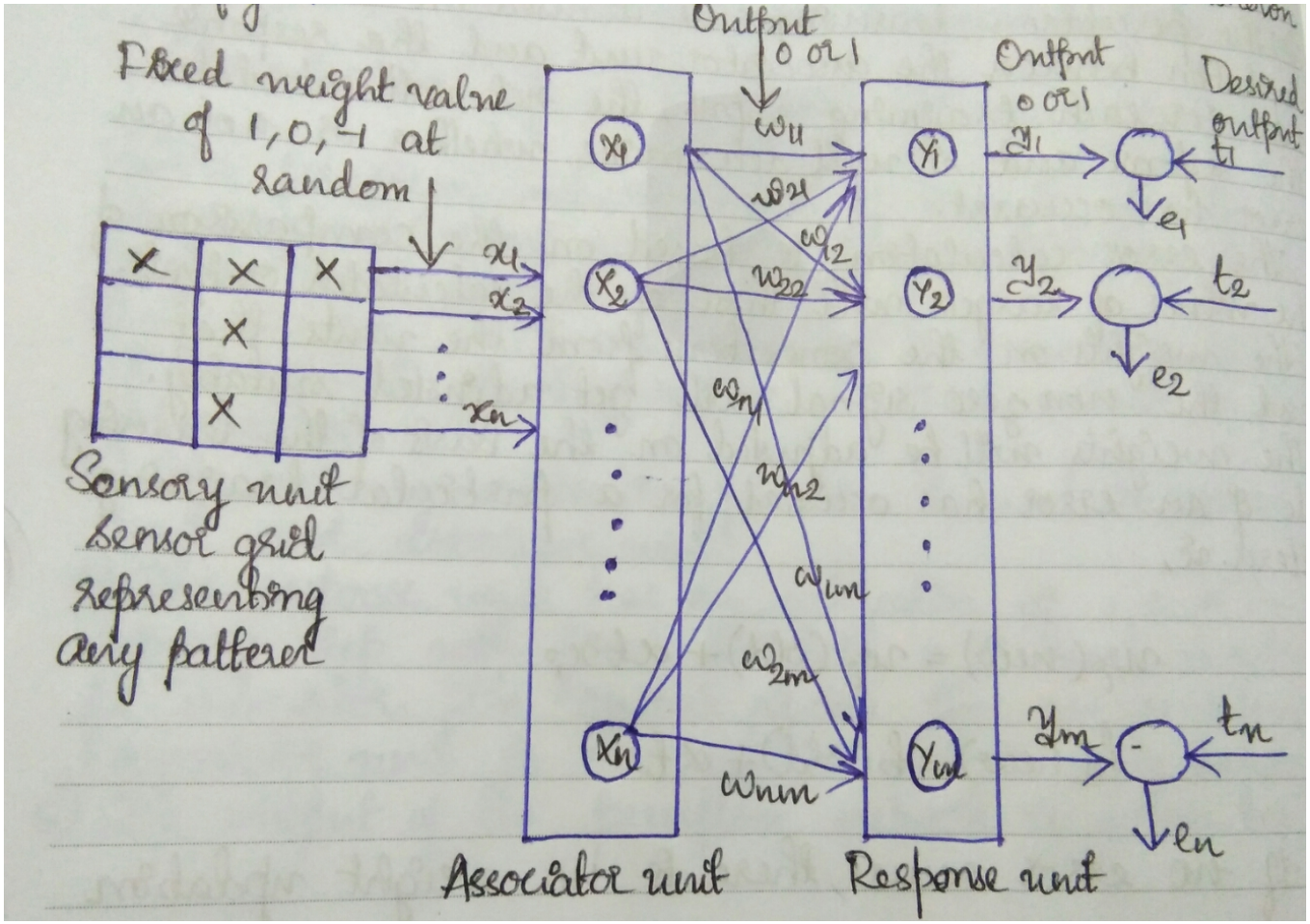


Figure 1.1: A perceptron network with its three units

Learning Rule

- A finite n number of input training vectors with their associated target values, $x(n)$ *and* $t(n)$.
- The output y is obtained on the basis of the net input calculated and activation function being applied over the net input.

$$y = f(y_{in}) = \begin{cases} 1 \text{ if } y_{in} \geq \theta \\ 0 \text{ if } -\theta \leq y_{in} \leq \theta \\ -1 \text{ if } y_{in} < \theta \end{cases}$$

- The weight updation is as follows:
  If $y \neq t$ then ,

$$w_i(new) = w_i(old) + \alpha t x_i$$

  else, we have

$$w(new) = w(old)$$

Architecture

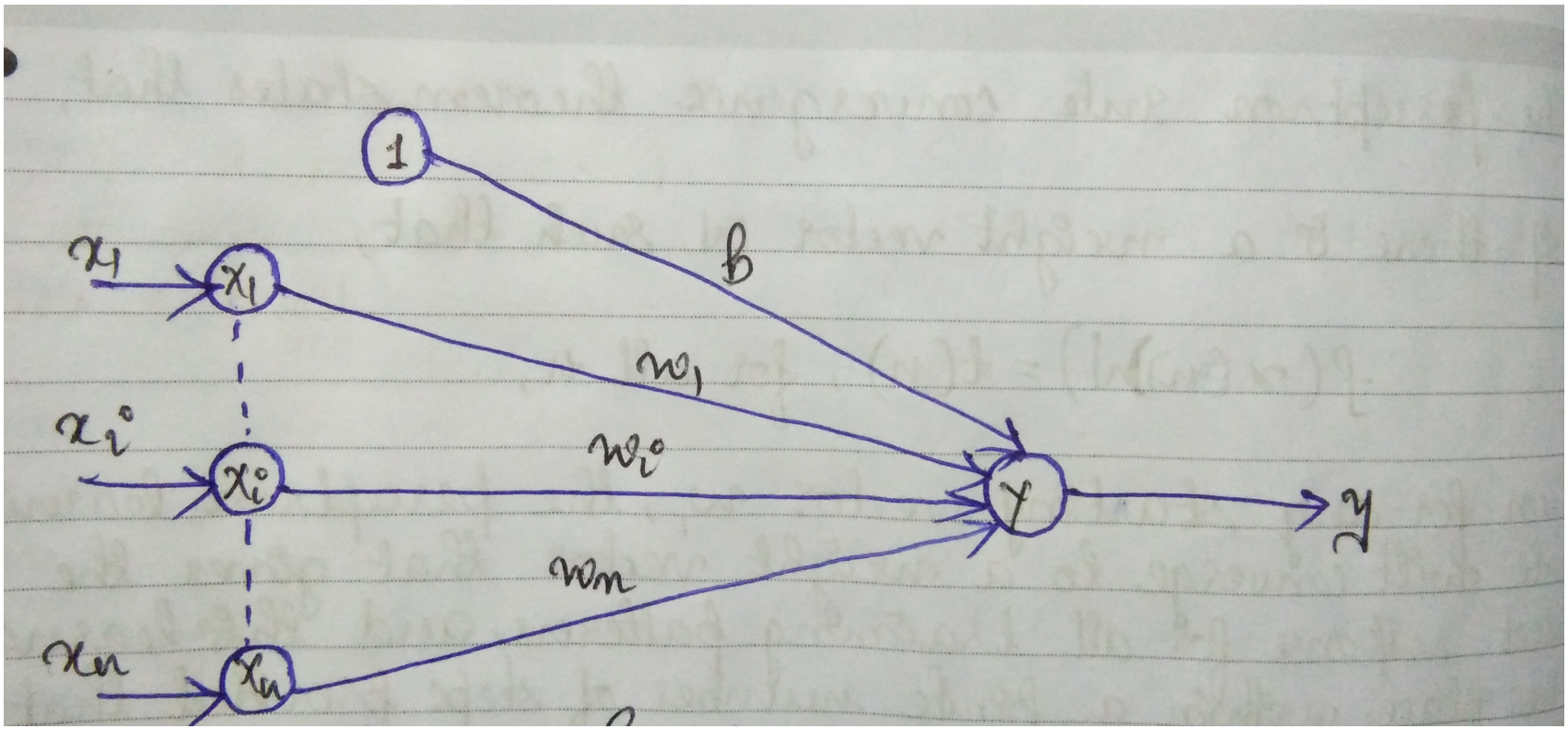


Figure 1.2: Single classification perceptron network

Flowchart

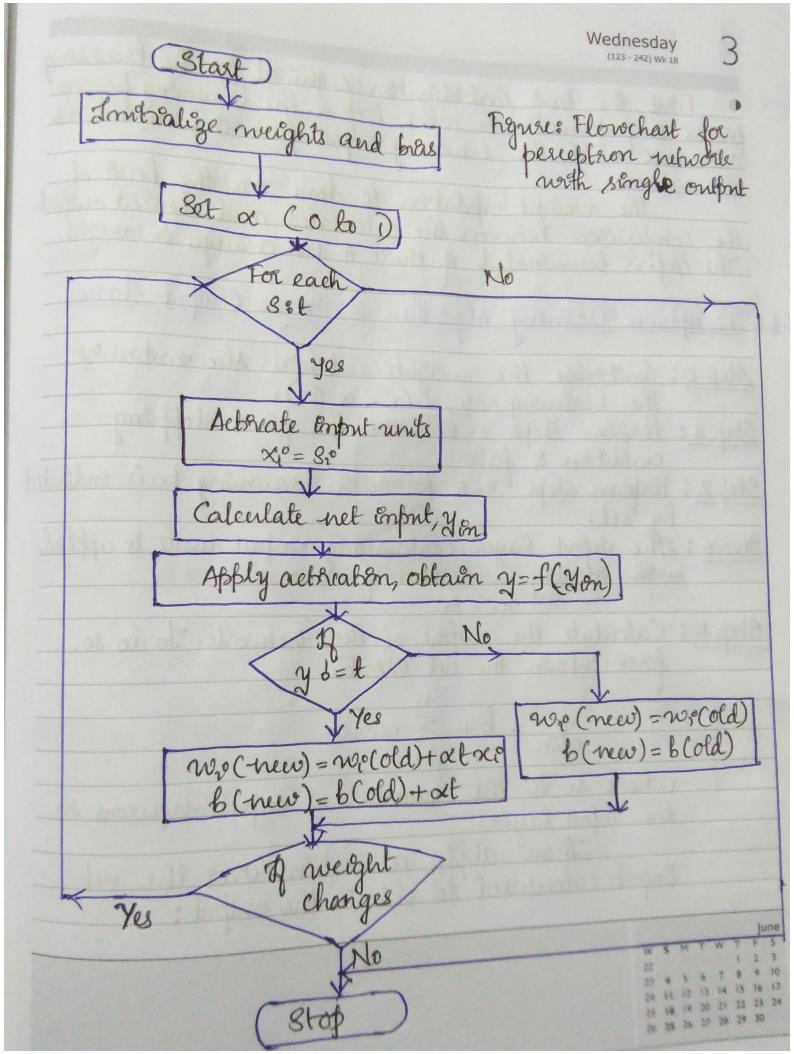


Figure: Flowchart for perceptron network with single output

Perceptron training algorithm for single output classes

- *Step 0*: Initialize the weights and bias. Also, initialize the learning rate,$\alpha(0 < \alpha \leq 1)$.
- *Step 1*: Perform steps 2-6 until the final stopping condition is false.
- *Step 2*: Perform steps 3-5 for each training pair indicated by, $s : t$.
- *Step 3*: The input layer containing input unit is applied with identity activation functions:

$$x_i = s_i$$

- *Step 4*: Calculate the output of the network.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

$$y = f(y_{in}) = \begin{cases} 1 \text{ if } y_{in} \geq \theta \\ 0 \text{ if } -\theta \leq y_{in} \leq \theta \\ -1 \text{ if } y_{in} < \theta \end{cases}$$

- *Step 5*: Weight and bias adjustment:
  If $y \neq t$ then ,

$$w_i(new) = w_i(old) + \alpha t x_i$$
$$b(new) = b(old) + \alpha t$$

  else, we have

$$w(new) = w(old)$$
$$b(new) = b(old)$$

- *Step 6*: Train the network until there is no weight change. Otherwise, start again from Step 2.

Perceptron training algorithm for multiple output classes

- *Step 0*: Initialize the weights and bias. Also, initialize the learning rate,$\alpha(0 < \alpha \leq 1)$.
- *Step 1*: Perform steps 2-6 until the final stopping condition is false.
- *Step 2*: Perform steps 3-5 for each training pair indicated by, $s : t$.
- *Step 3*: The input layer containing input unit is applied with identity activation functions:

$$x_i = s_i$$

- *Step 4*: Calculate the output of the network.

$$y_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}$$

$$y = f(y_{inj}) = \begin{cases} 1 \text{ if } y_{inj} \geq \theta \\ 0 \text{ if } -\theta < y_{inj} < \theta \\ -1 \text{ if } y_{inj} \leq \theta \end{cases}$$

- *Step 5*: Make adjustment in weight and bias for $j = 1$ *to* m and $i = 1$ *to* n

  If $t_i \neq y_j$ then ,

$$w_{ij}(new) = w_{ij}(old) + \alpha t_j x_i$$
$$b_j(new) = b_j(old) + \alpha t_j$$

  else, we have

$$w_{ij}(new) = w_{ij}(old)$$
$$b_j(new) = b_j(old)$$

- *Step 6*: Train the network until there is no weight change. Otherwise, start again from Step 2.

Perceptron Network testing algorithm

- *Step 0*: Initial weights is equal to the final weights obtained during training.
- *Step 1*: For each input vector X to be classified, perform Steps 2-3.
- *Step 2*: Set activations of the input unit.
- *Step 3*: Obtain the response of output unit.

$$y_{in} = \sum_{i=1}^{n} x_i w_i$$

$$y = f(y_{in}) = \begin{cases} 1 \text{ if } y_{in} \geq \theta \\ 0 \text{ if } -\theta \leq y_{in} \leq \theta \\ -1 \text{ if } y_{in} < \theta \end{cases}$$

Adaptive Linear Neuron(Adaline)

- The units with linear activation function are called *linear units*.
- A network with single linear unit is called an *Adaline*.
- It uses bipolar activation for its input signals and its target output.
- The weights between the input and the output units are adjustable.
- Adaline is a net which has only one output unit.
- It is trained using *delta rule*.

Delta rule for single output unit

- Also known as *Least Mean Square(LMS) rule* or *Widrow–Hoff rule*.
- It is found to minimize the mean–squared error between the activation and the target value.
- *Widrow–Hoff rule* vs *Perceptron learning rule*:
  1. Perceptron learning rule originates from the Hebbian assumption while the delta rule is derived from the gradient–descent method.
  2. Perceptron learning rule stops after a finite number of learning steps. But the gradient–descent approach continues forever.
- Updates the weights so as to minimize the difference between the net input and the target value.

- The delta rule for adjusting the weight of i^{th} pattern $(i = 1 \text{ to } n)$ is,

$$\Delta w_i = \alpha(t - y_{in})x_i$$

where,

Δw_i – weight change

α – learning rate

x_i – activation of input unit

y_{in} – net input to the output unit.ie,

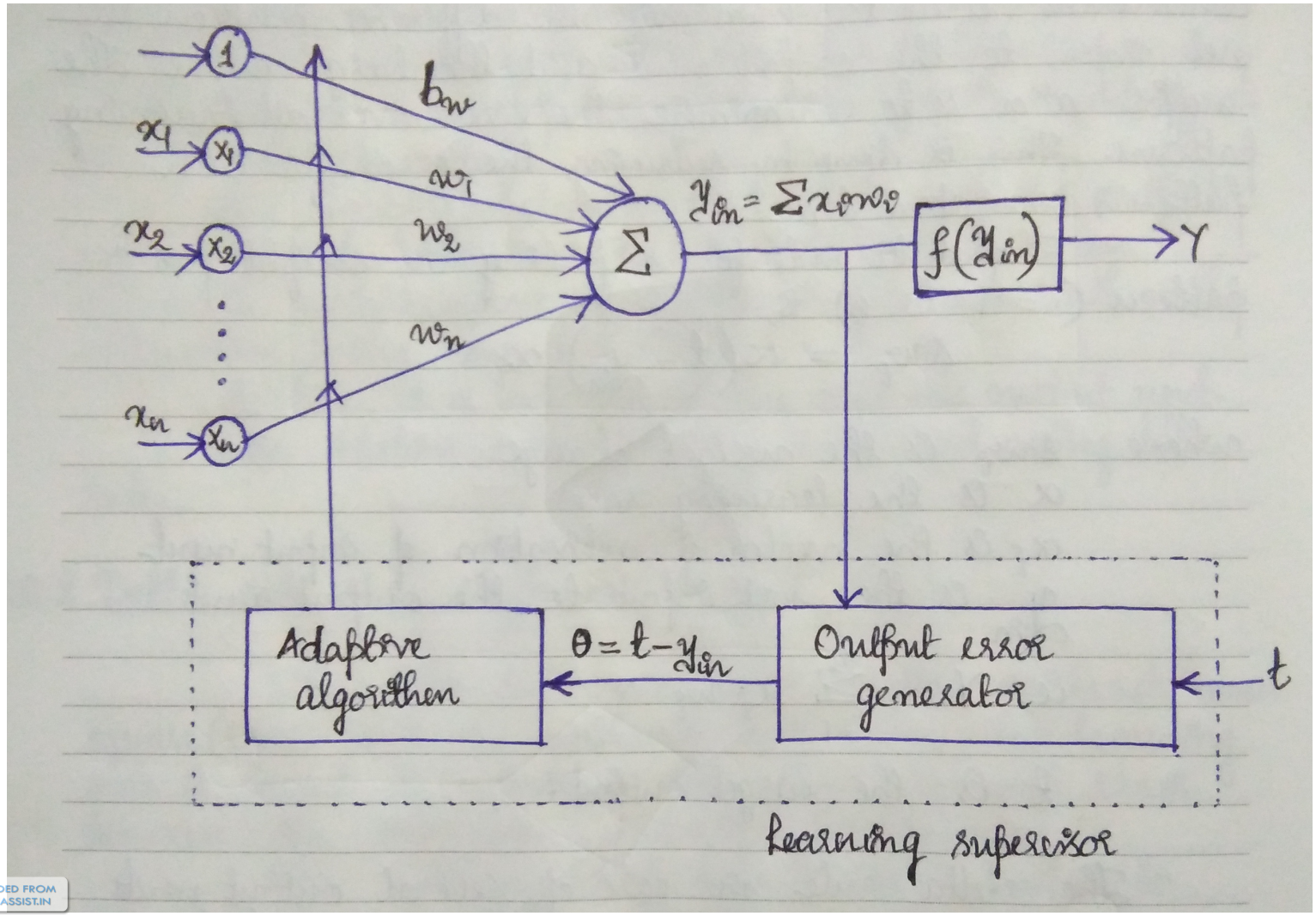
$$y = \sum_{i=1}^{n} x_i w_i$$

t – target output

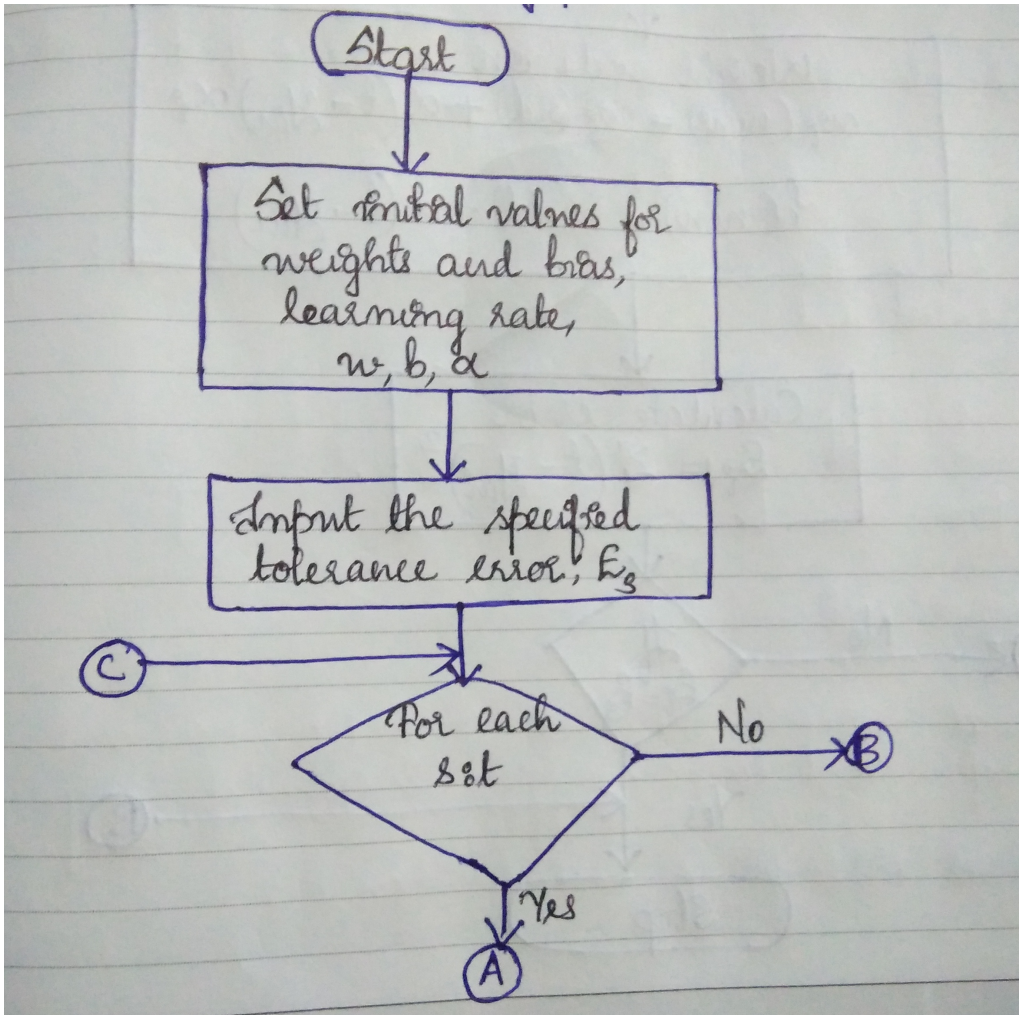
- The delta rule for adjusting the weight from i^{th} input unit to the j^{th} output unit is :

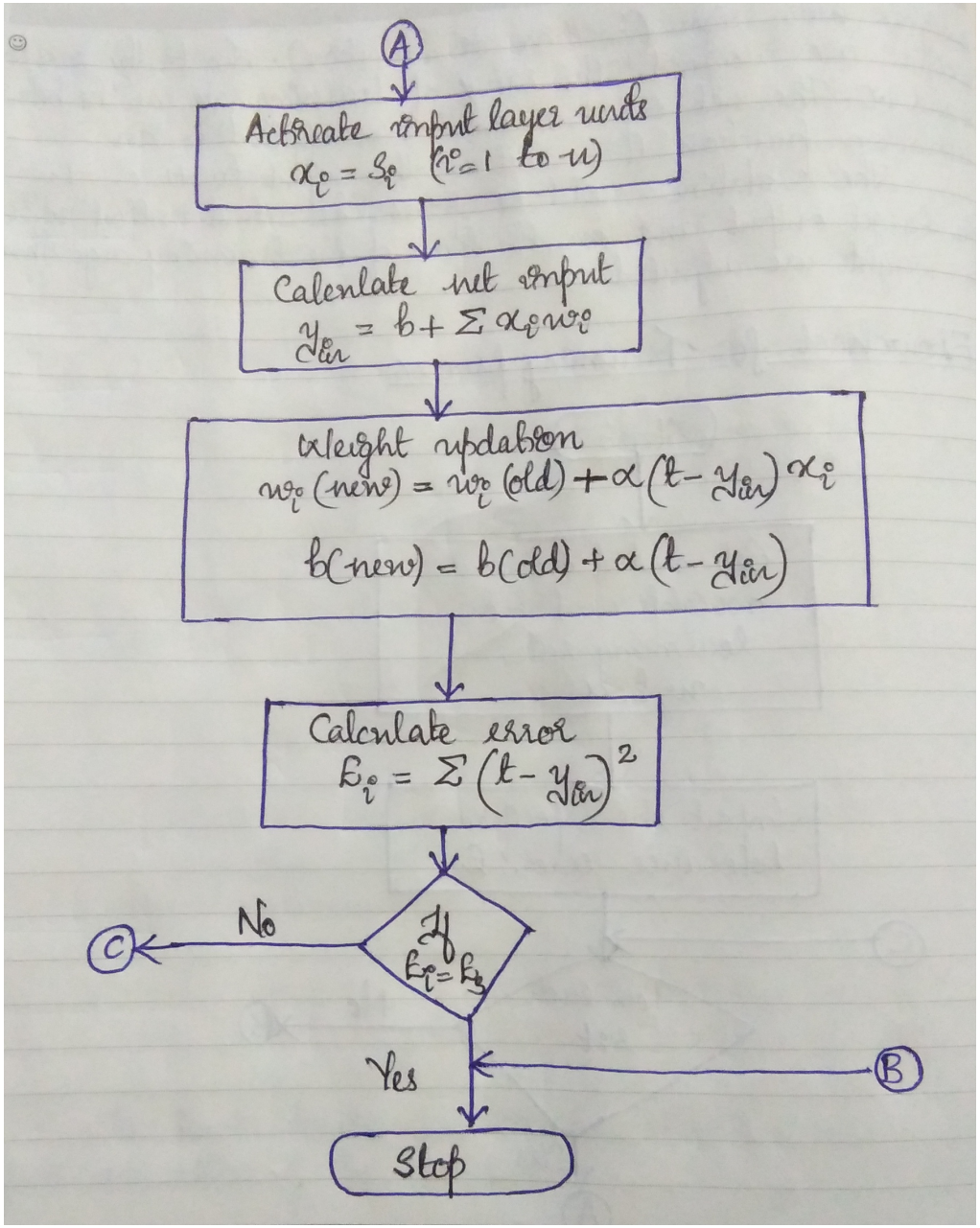
$$\Delta w_{ij} = \alpha(t_j - y_{inj})x_i$$

Adaline model



Flowchart of training process





Training algorithm

- *Step 0*: Weights and bias are set to some random values but not zero. Set the learning rate parameter, α.
- *Step 1*: Perform Steps 2-6 when stopping condition is false.
- *Step 2*: Perform Steps 3-5 for each bipolar training pair, $s : t$.
- *Step 3*: Set activations for input units $i = 1$ *to* n:

$$x_i = s_i$$

- *Step 4*: Calculate the net input to the output unit:

$$y_{in} = b_w + \sum_{i=1}^{n} x_i w_i$$

- *Step 5*: Update the weights and bias for $i = 1$ *to* n:

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

- *Step 6*: If the highest weight change that occurred during training is smaller than a specified tolerance then stop the training process, else continue.

Testing algorithm

- *Step 0*: Initial weights is equal to the final weights obtained during training.
- *Step 1*: Perform Steps 2−4 for each bipolar input vector, x.
- *Step 2*: Set activations of the input units to x.
- *Step 3*: Calculate the net input to the output unit:

$$y_{in} = b_w + \sum_{i=1}^{n} x_i w_i$$

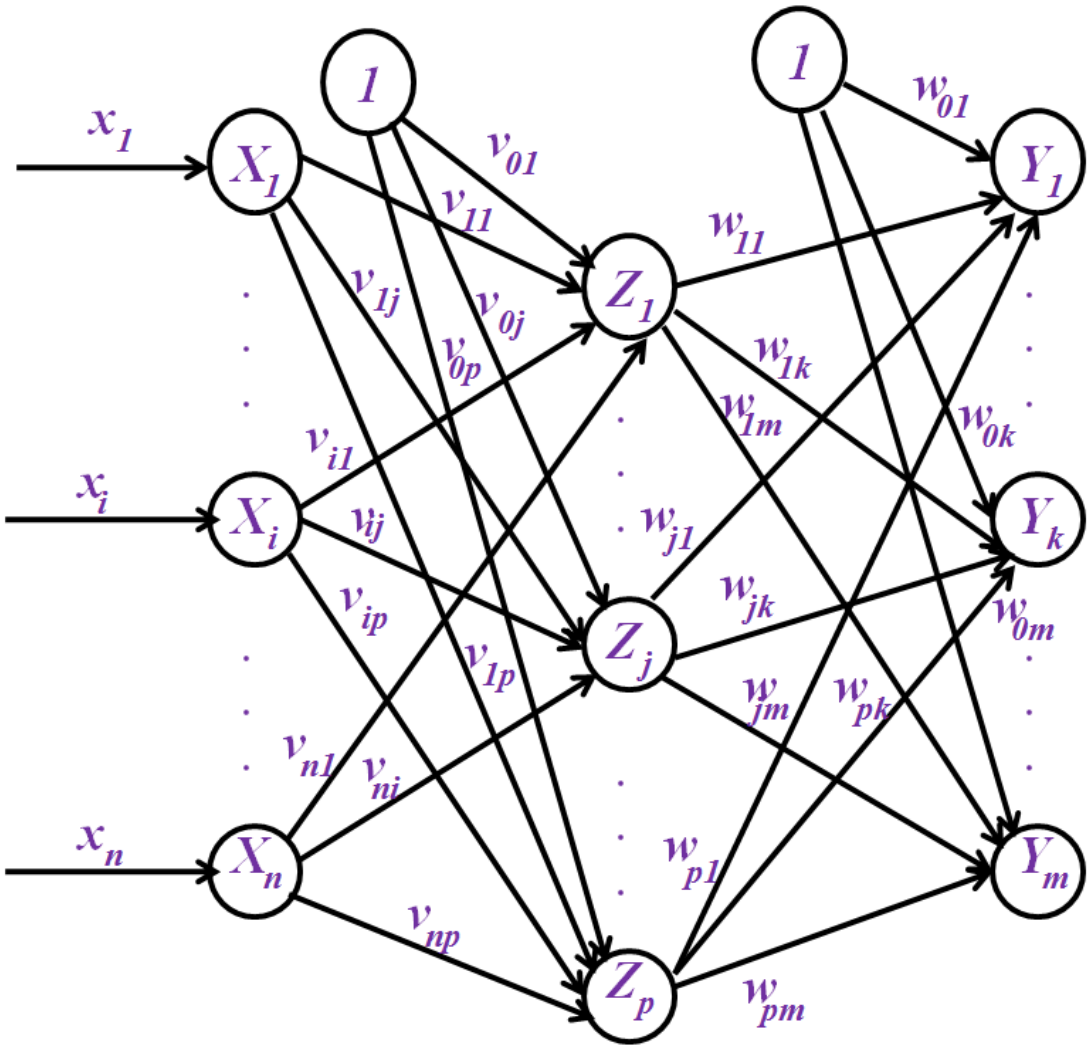
- *Step 4*: Apply the activation function over the net input calculated:

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

Back−Propagation Network

- This learning algorithm is applied to multilayer feed−forward networks consisting of processing elements with continuous differentiable activation functions.
- The networks associated with back−propagation learning algorithm are called back−propagation networks(*BPNs*).
- Algorithm provides a procedure for changing the weights to classify the given input patterns correctly.
- It uses *gradient−descent method*.
- This is a method where the error is propagated back to the hidden unit.

Architecture



Learning factors of BPN

- *Initial Weights*
  - Initialized at small random values.
  - The choice of initial weight determines how fast the network converges.
  - One method of choosing the weight w_{ij} is choosing it in the range,
    $$[\frac{-3}{\sqrt{o_i}}, \frac{3}{\sqrt{o_i}}]$$
    where o_i is the number of processing elements j that feed−forward to processing element i.
  - *Nyugen− Widrow initialization:* Based on the geometric analysis of the response of hidden neurons to a single input.
  - Random initialization of weights connecting the input neurons to the hidden units is obtained by,
    $$v_{ij}(new) = \gamma \frac{v_{ij}(old)}{\|\overline{v_j(old)}\|}$$
    $$\gamma = 0.7(P)^{1/n}$$

- *Learning rate, α*
  - The range of α from 10^{-3} to 1.0 has been used successfully.
- *Momentum factor, η*
  - $\eta \in [0, 1]$ and the value of 0.9 is often used for the momentum factor.
  - Weight updation formulas used here are,

$$w_{jk}(t+1) = w_{jk}(t) + \alpha\delta_k z_j + \eta[w_{jk}(t) - w_{jk}(t-1)] \text{ and}$$
$$v_{ij}(t+1) = v_{ij}(t) + \alpha\delta_j x_i + \eta[v_{ij}(t) - v_{ij(t-1)}]$$

- *Generalization*
- *Number of training data*
- *Number of hidden layer nodes*

Training alorithm

- *Step 0*: Initialize weights and learning rate.
- *Step 1*: Perform *Steps*2−9 when stopping condition is false.
- *Step 2*: Perform *Steps*3 − 8 for each training pair.

Feed–forward phase(Phase I)

- *Step 3*: Each input unit receives input signal x_i and sends it to the hidden unit*(i=1 to n)*.
- *Step 4*: Each hidden unit z_j *(j=1 to p)* sums its weighted input signals to calculate net input:

$$z_{inj} = v_{oj} + \sum_{i=1}^{n} x_i v_{ij}$$

Calculate output of the hidden unit by applying activation function,

$$z_j = f(z_{inj})$$

- *Step 5*: For each output unit y_k *(k=1 to m)*, calculate the net input:

$$y_{ink} = w_{ok} + \sum_{j=1}^{p} z_j w_{jk}$$

and apply the activation function to compute output signal:

$$y_k = f(y_{ink})$$

Back−propagation of error(Phase II)

- *Step 6*: Each output unit y_k *(k=1 to m)* receives a target pattern corresponding to the input training pattern and computes the error correction term:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

Then update the weights and bias:

$$\Delta w_{jk} = \alpha \delta_k z_j$$
$$\Delta w_{ok} = \alpha \delta_k$$

Send δ_k to the hidden layer backwards.

- *Step 7*: Each hidden unit z_j $(j=1 \text{ to } p)$ sums its delta inputs from the output units:

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

The term δ_{inj} gets multiplied with the derivative of $f(z_{inj})$ to calculate the error term:

$$\delta_j = \delta_{inj} f'(z_{inj})$$

Then update the weights and bias:

$$\Delta v_{ij} = \alpha \delta_j x_i$$
$$\Delta v_{oj} = \alpha \delta_j$$

Weight and bias updation(Phase III)

- *Step 8*: Each output unit y_k *(k=1 to m)* updates the bias and weights:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$
$$w_{0k}(new) = w_{0k}(old) + \Delta w_{0k}$$

Each output unit z_j *(j=1 to p)* updates the bias and weights:

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$$
$$v_{0j}(new) = v_{0j}(old) + \Delta v_{0j}$$

- *Step 9*: Check for the stopping condition *may be certain number of epochs reached or when the actual output equals to target output.*

Testing alorithm of BPN

- *Step 0*: Initialize the weights. The weights are taken from the training algorithm.
- *Step 1*: Perform $Steps 2-4$ for each input vector.
- *Step 2*: Set the activation of input unit for x_i $(i=1 \text{ to } n)$.
- *Step 3*: Calculate the net input to hidden unit x and its output. For $j=1 \text{ to } p$,

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$
$$z_j = f(z_{inj})$$

- *Step 4*: Now compute the output of the output layer unit. For $k=1 \text{ to } m$,

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$
$$y_k = f(y_{ink})$$

Use sigmoidal activation functions for calculating the output.

Sigmoidal activation functions

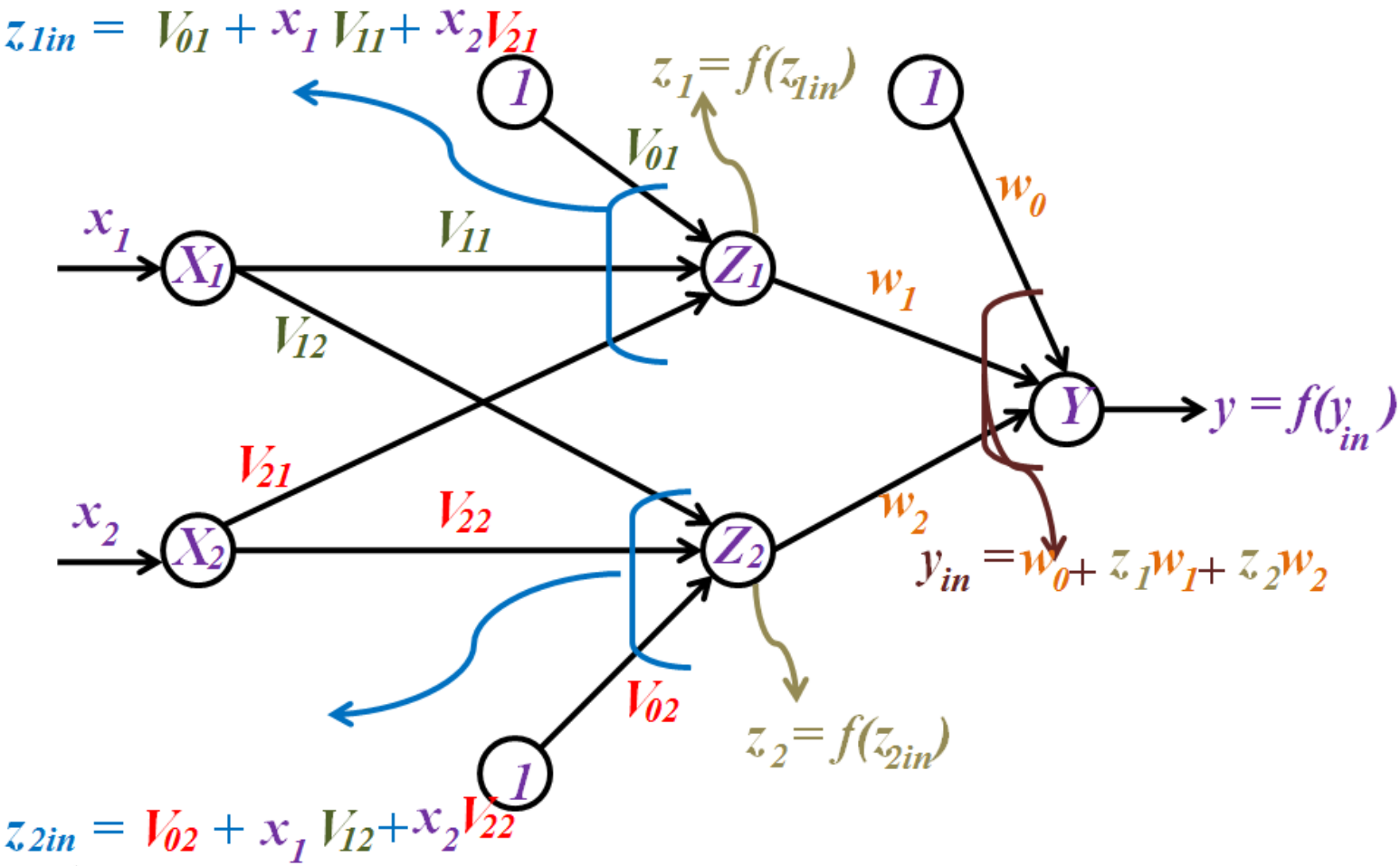
1. *Binary sigmoid function*

$$f(x) = \frac{1}{1 + e^{-x}}$$
$$f'(x) = f(x)[1 - f(x)]$$

2. *Bipolar sigmoid function*

$$f(x) = \frac{2}{1 + e^{-x}} - 1$$
$$f'(x) = 0.5[1 + f(x)][1 - f(x)]$$

Summarization of algorithm



- Compute the error, δ_k, here, $k = 1$, only one output neuron.

$$\delta_k = (t_k - y_k) f'(y_{ink})$$
$$\delta_1 = (t - y) f'(y_{in})$$

- changes in weight between hidden and output layer:

$$\Delta w_0 = \alpha \delta_1$$
$$\Delta w_1 = \alpha \delta_1 z_1$$
$$\Delta w_2 = \alpha \delta_1 z_2$$

- Compute the error portion δ_j between input and hidden layer, j=1,2(ie,z_{in1}, z_{in2}):

$$\delta_j = \delta_{inj} f'(z_{inj})$$
$$\delta_{inj} = \sum_{k=1}^{w_{jk}}$$

Here, k=1(only one output neuron)

$$\delta_{inj} = \delta_1 w_{j1}$$
$$\delta_{in1} = \delta_1 w_{11}, \delta_{in2} = \delta_1 w_{21}$$
$$w_{11} = w_1, w_{21} = w_2$$

- Error,

$$\delta_1 = \delta_{in1} f'(z_{in1})$$
$$\delta_2 = \delta_{in2} f'(z_{in2})$$

- Change in weights between input and hidden layer:

$$\Delta v_{11} = \alpha \delta_1 x_1$$
$$\Delta v_{21} = \alpha \delta_1 x_2$$
$$\Delta v_{01} = \alpha \delta_1$$
$$\Delta v_{12} = \alpha \delta_2 x_1$$
$$\Delta v_{22} = \alpha \delta_2 x_2$$
$$\Delta v_{02} = \alpha \delta_2$$

- Compute the final weights of the network:

$$
\begin{aligned}
v_{11}(new) &= v_{11}(old) + \Delta v_{11} \\
v_{12}(new) &= v_{12}(old) + \Delta v_{12} \\
v_{21}(new) &= v_{21}(old) + \Delta v_{21} \\
v_{22}(new) &= v_{22}(old) + \Delta v_{22} \\
w_1(new) &= w_1(old) + \Delta w_1 \\
w_2(new) &= w_2(old) + \Delta w_2 \\
w_0(new) &= w_0(old) + \Delta w_0 \\
v_{01}(new) &= v_{01}(old) + \Delta v_{01} \\
v_{02}(new) &= v_{02}(old) + \Delta v_{02}
\end{aligned}
$$

END