PERCEPTRON

- The neuronal model we have just discussed is also known as a perceptron.
- The perceptron is the simplest form of a neural network used for the classification of patterns said to be linearly separable.
- Basically, it consists of a single neuron with adjustable synaptic weights and bias.
- Now we will look at a method of achieving **learning** in our model we have formulated.

- Variables and Parameters

$$\mathbf{x}(n) = (m+1)\times 1 \quad \text{input vector}$$

$$= \left[+1, x_1(n), x_2(n), \ldots, x_m(n)\right]^T$$

$$\mathbf{w}(n) = (m+1)\times 1 \quad \text{weight vector}$$

$$= \left[b(n), w_1(n), w_2(n), \ldots, w_m(n)\right]^T$$

$$b(n) = \text{bias}$$

$$y(n) = \text{actual response}$$

$$d(n) = \text{desired response}$$

$$\eta = \text{learning-rate parameter, a postive constant less than unity}$$

ALGORITHM

1. ***Initialization***. Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time step $n = 1,2, \ldots$
2. ***Activation***. At time step n, activate the perceptron by applying input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. ***Computation of Actual Response***. Compute the actual response of the perceptron:

$$y(n) = \operatorname{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

where $\operatorname{sgn}(.)$ is the signum function.

$$\operatorname{sgn}(x) = \begin{cases} +1, & \text{if} \quad x \geq 0 \\ -1, & \text{if} \quad x < 0 \end{cases}$$

4. ***Adaptation of Weight Vector***. Update the weight vector of the perceptron:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\left[d(n) - y(n)\right]\mathbf{x}(n)$$

where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } C_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } C_2 \end{cases}$$

5. ***Continuation***. Increment time step n by one and go back to step 2.

DECISION BOUNDARY

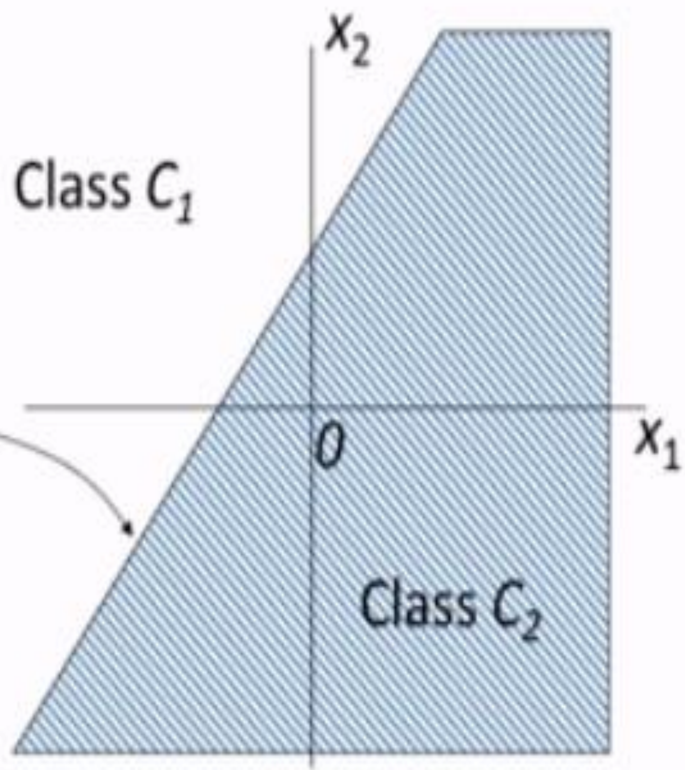
- The hyper-plane

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

or

$$w_1 x_1 + w_2 x_2 + b = 0$$

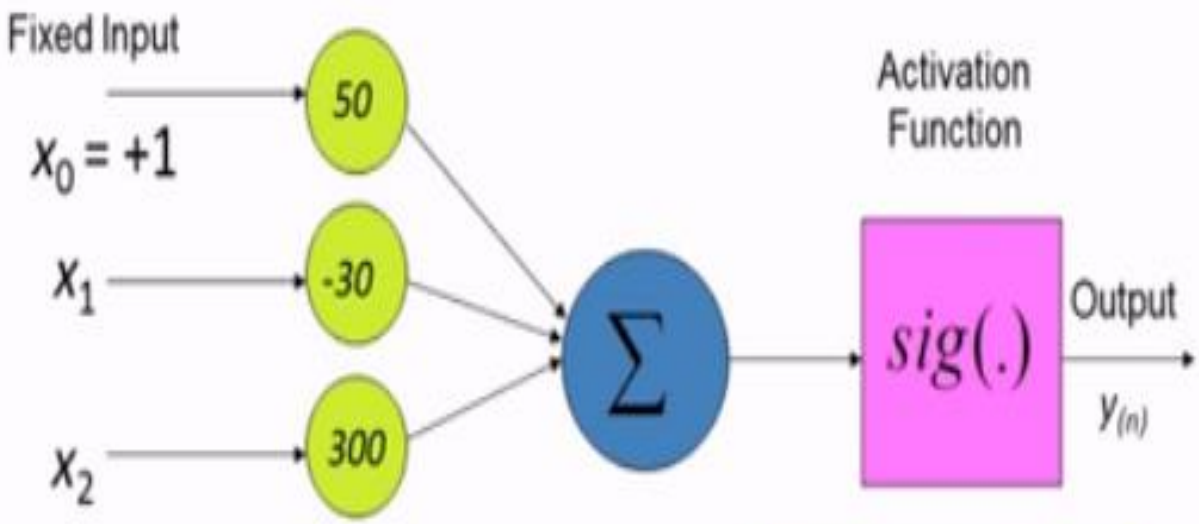
is the decision boundary for a two class classification problem.



EXAMPLE

With correct initial weights and bias

$$\left.\begin{array}{l} w_1(0) = -30,\ w_2(0) = 300, \\ b(0) = 50,\ \eta = 0.01 \end{array}\right\} given$$



$$\mathrm{sgn}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases}$$

$$\left.\begin{array}{l} w_1(0) = -30,\ w_2(0) = 300, \\ b(0) = 50,\ \eta = 0.01 \end{array}\right\}\textit{given}$$

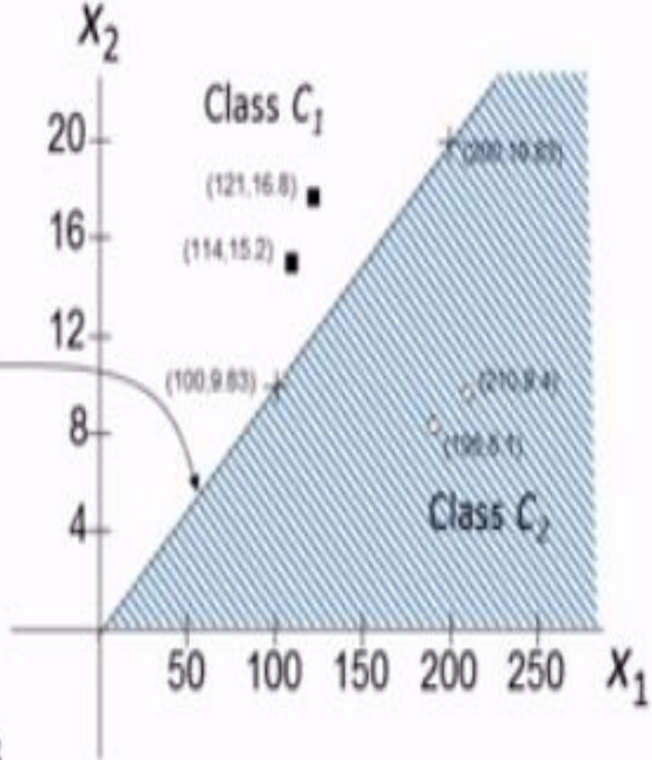
Therefore the Initial Decision Boundary for this example is:

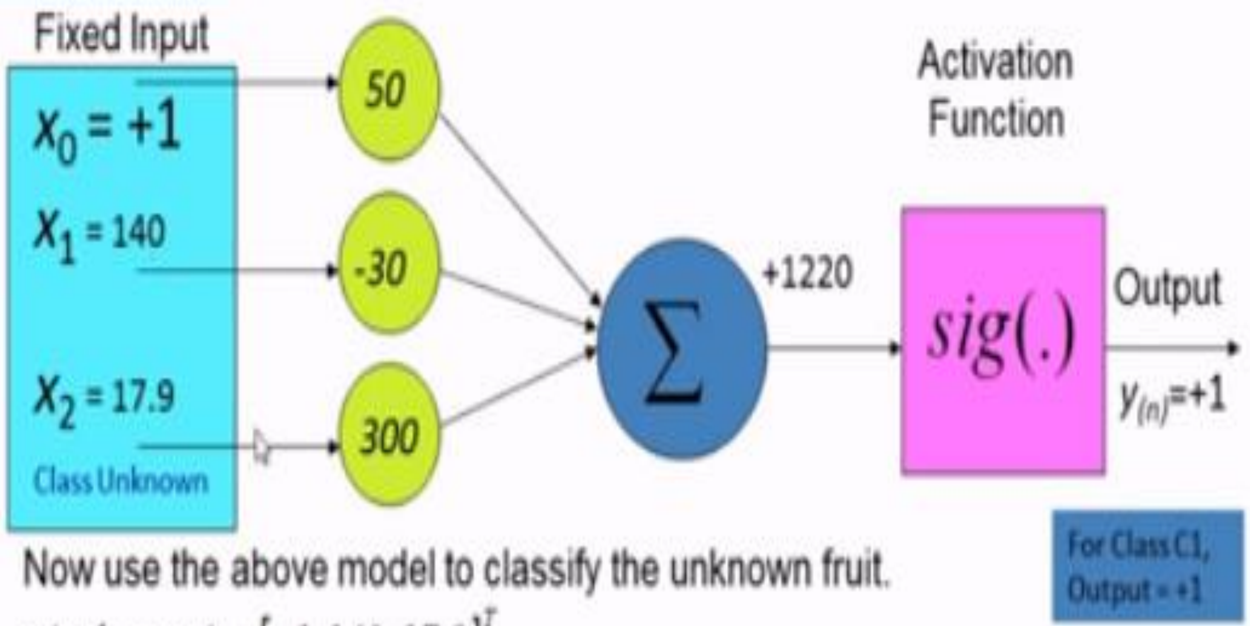
$$w_1x_1 + w_2x_2 + b = 0$$

$$-30x_1 + 300x_2 + 50 = 0$$

$$x_1 = 100,\ x_2 = \frac{30 \times 100 - 50}{300} = 9.83$$

$$x_1 = 200,\ x_2 = \frac{30 \times 200 - 50}{300} = 19.83$$





Now use the above model to classify the unknown fruit.

$$\mathbf{x}(\text{unknown}) = [+1, 140, 17.9]^T$$

$$\mathbf{w}(3) = [50, -30, 300]^T$$

$$y(\text{unknown}) = \text{sgn}\left(\mathbf{w}^T(3)\mathbf{x}(\text{unknown})\right) = \text{sgn}(50\times1 - 30\times140 + 300\times17.9)$$

$$= \text{sgn}(1220) = +1$$

$\therefore$ this unknown fruit belongs to the class C_1.

With unknown initial weights and bias

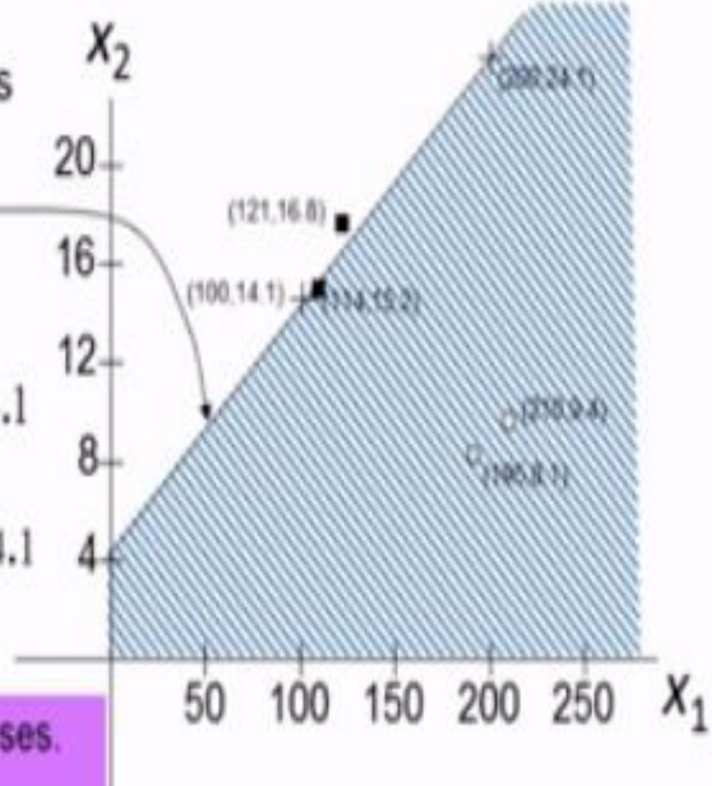
$$\left.\begin{array}{l} w_1(0) = -30,\ w_2(0) = 300, \\ b(0) = -1230,\ \eta = 0.01 \end{array}\right\} given$$

Therefore the Decision Boundary for this case:

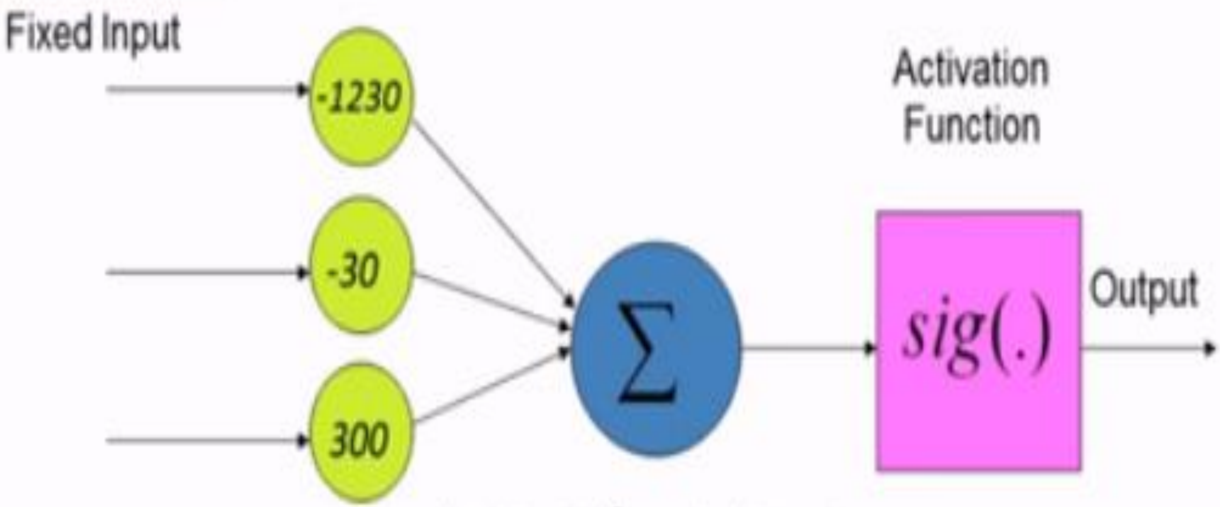
$$-30x_1 + 300x_2 - 1230 = 0$$

$$x_1 = 100, x_2 = \frac{30 \times 100 + 1230}{300} = 14.1$$

$$x_1 = 200, x_2 = \frac{30 \times 200 + 1230}{300} = 24.1$$



itial hyperplane does not separate the two classes.
herefore we need to **Train** the Neural Network



$$\mathbf{x}(n) = \mathbf{x}(0) = [+1, 121, 16.8]^T \quad \text{and} \quad d(0) = +1$$

$$\mathbf{w}(n) = \mathbf{w}(0) = [-1230, -30, 300]^T$$

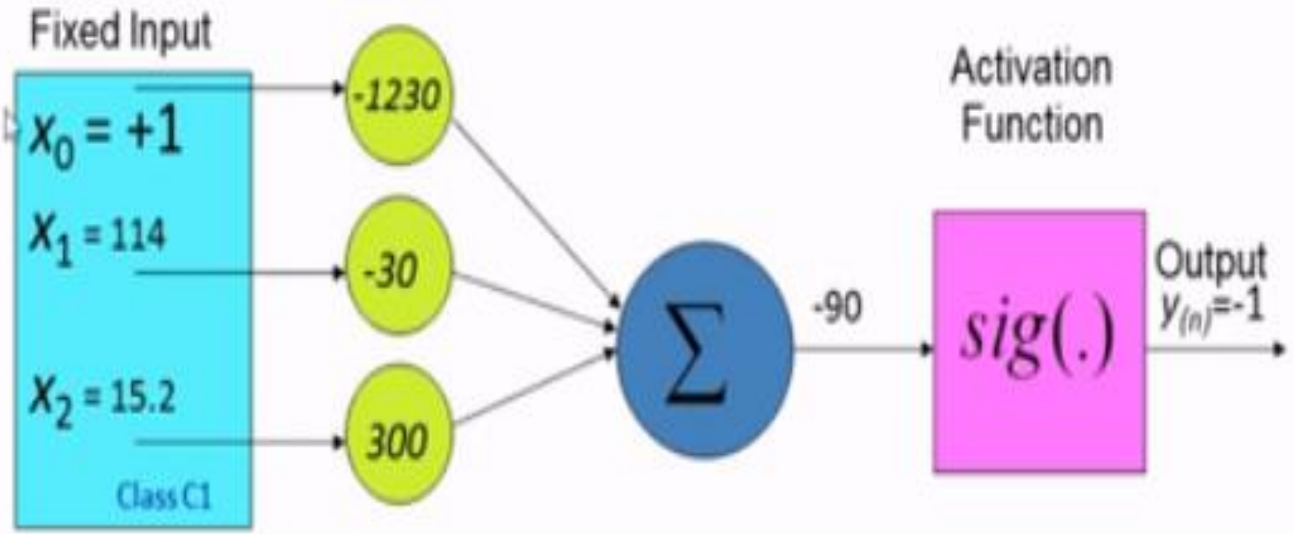
$$y(n) = y(0) = \mathrm{sgn}\left(\mathbf{w}^T(0)\,\mathbf{x}(0)\right)$$

$$= \mathrm{sgn}\left(-1230 \times 1 - 30 \times 121 + 300 \times 16.8\right)$$

$$= \mathrm{sgn}(180) = +1 = d(0)$$

Hence no need to recalculate the weights.

$$\therefore \mathbf{w}(n+1) = \mathbf{w}(1) = [-1230, -30, 300]^T$$



$$\mathbf{x}(1) = [+1, 114, 15.2]^T \quad \text{and} \quad d(1) = +1$$

$$\mathbf{w}(1) = [-1230, -30, 300]^T$$

$$y(1) = \operatorname{sgn}\left(\mathbf{w}^T(1)\mathbf{x}(1)\right) = \operatorname{sgn}(-1230 \times 1 - 30 \times 114 + 300 \times 15.2)$$

$$= \operatorname{sgn}(-90) = -1 \neq d(1)$$

Hence **we have to** recalculate the weights.

Here we use ***Adaptation of Weight Vector*** (Step 4) to update the weight vector of the perceptron.

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\left[d(n) - y(n)\right]\mathbf{x}(n)$$

$$\mathbf{w}(1) = \left[-1230, -30, 300\right]^T$$

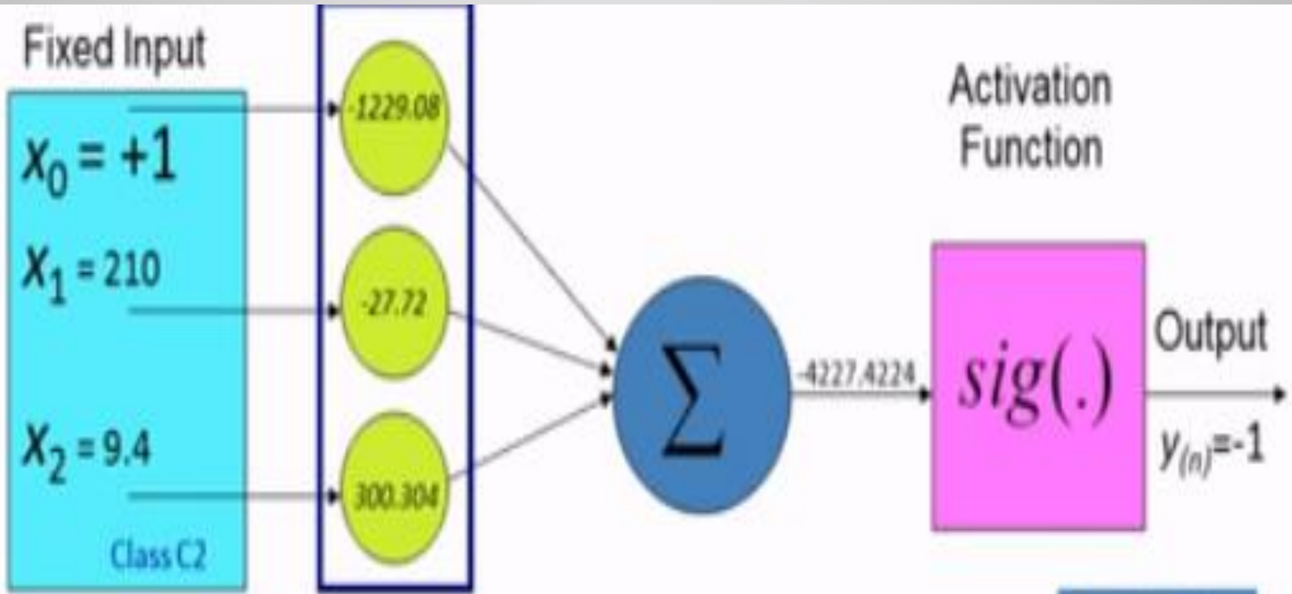
$$\mathbf{x}(1) = \left[+1, 114, 15.2\right]^T$$

$$d(1) = +1, y(1) = -1, \eta = 0.01$$

$$\mathbf{w}(1+1) = \mathbf{w}(2) = \left[-1230, -30, 300\right]^T + 0.01\left[+1-(-1)\right]\left[+1, 114, 15.2\right]^T$$

$$= \left[-1230, -30, 300\right]^T + \left[+0.02, 2.28, 0.304\right]^T$$

$$\therefore \mathbf{w}(2) = \left[-1229.08, -27.72, 300.304\right]^T$$



$$\mathbf{w}(2) = [-1229.08, -27.72, 300.304]^T$$

$$\mathbf{x}(2) = [+1, 210, 9.4]^T \quad \text{and} \quad d(2) = -1$$

$$\mathbf{y}(2) = \text{sgn}\left(\mathbf{w}^T(2)\mathbf{x}(2)\right) = \text{sgn}\left(-1229.08 \times 1 - 27.72 \times 210 + 300.304 \times 9.4\right)$$

$$= \text{sgn}(-4227.4224) = -1 = d(2)$$

Hence no need to recalculate the weights.

$$\therefore \mathbf{w}(n+1) = \mathbf{w}(3) = [-1229.08, -27.72, 300.304]^T$$

Continuous Perceptron Training Algorithm

•Replace the TLU (Threshold Logic Unit) with the sigmoid activation function for two reasons:

- Gain finer control over the training procedure
- Facilitate the differential characteristics to enable computation of the error gradient (TLU is non differentiable as opposed to sigmoid)

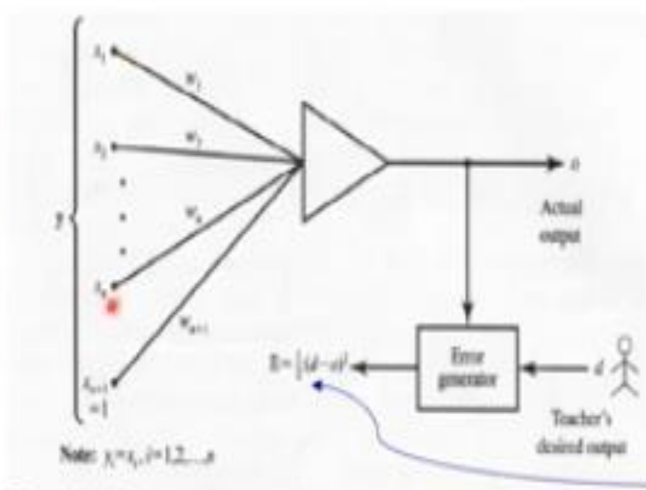


Figure 3.16 Continuous perceptron training using square error minimization.

$$\hat{w} = w - \eta \nabla E(w)$$

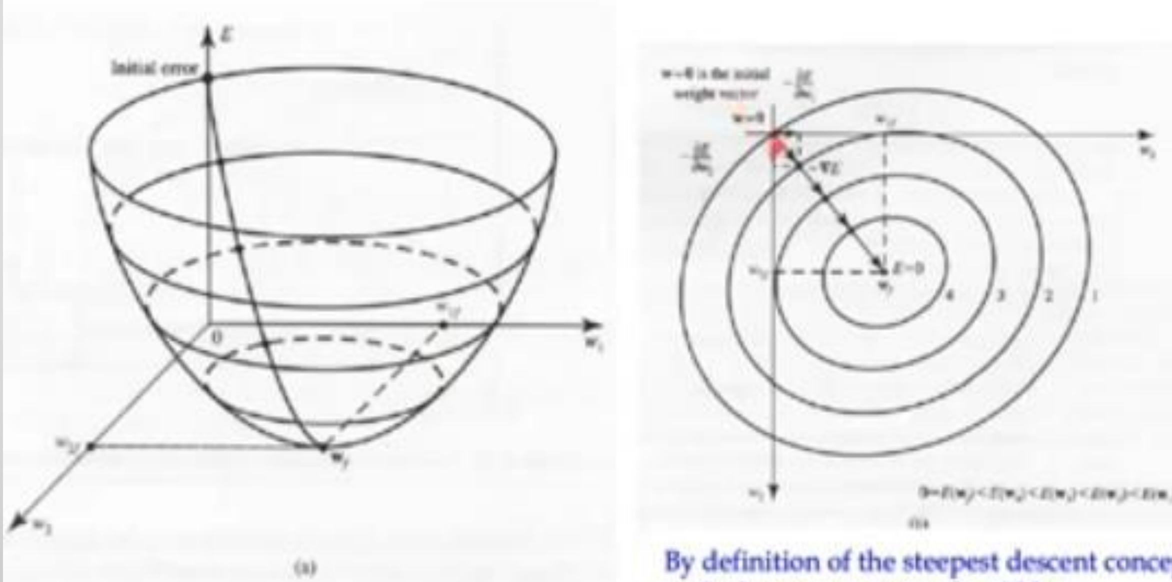
learning constant

error gradient **(of current error function)**

The factor ½ does not affect the location of the error minimum

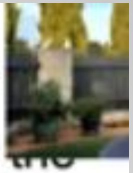
Continuous Perceptron Training Algorithm…

- The new weights are obtained by moving in the direction of the negative gradient along the multidimensional error surface



By definition of the steepest descent concept each elementary move should be perpendicular to the current error contour.

Continuous Perceptron Training Algorithm…



•Define the error as the squared difference between the desired output and the actual output

$$E = \frac{1}{2}(d-o)^2$$

or $$E = \frac{1}{2}\left[d - f\left(\mathbf{w}^t\mathbf{y}\right)\right]^2 = \frac{1}{2}\left[d - f(net)\right]^2$$

$$\nabla E(\mathbf{w}) = \frac{1}{2}\nabla\left(\left[d - f(net)\right]^2\right)$$

$$\nabla E(\mathbf{w}) \triangleq \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \cdot \\ \cdot \\ \frac{\partial E}{\partial w_{n+1}} \end{bmatrix} = -(d-o)f'(net)\begin{bmatrix} \frac{\partial(net)}{\partial w_1} \\ \frac{\partial(net)}{\partial w_2} \\ \cdot \\ \cdot \\ \frac{\partial(net)}{\partial w_{n+1}} \end{bmatrix} = -(d-o)f'(net)\mathbf{y}$$

Training rule of continous perceptron (equivalent to delta training rule)

Since $net = \mathbf{w}^t\mathbf{y}$, we have $\frac{\partial(net)}{\partial w_i} = y_i \quad i = 1,2,\ldots,n+1$

Continuous Perceptron Training Algorithm…

- Bipolar Continuous Activation Function

$$f(net)=\frac{2}{1+\exp(-\lambda\cdot net)}-1 \qquad f'(net)=\lambda\cdot\frac{2\exp(-\lambda\cdot net)}{[1+\exp(-\lambda\cdot net)]^2}=\lambda\cdot\left\{1-[f(net)]^2\right\}=\lambda(1-o^2)$$

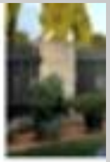
$$\boxed{\hat{\mathbf{w}}=\mathbf{w}+\frac{1}{2}\eta\cdot\lambda(d-o)(1-o^2)\mathbf{y}}$$

- Unipolar Continuous Activation Function

$$f(net)=\frac{1}{1+\exp(-\lambda\cdot net)} \qquad f'(net)=\frac{\lambda\cdot\exp(-\lambda\cdot net)}{[1+\exp(-\lambda\cdot net)]^2}=\lambda\cdot f(net)[1-f(net)]=\lambda\cdot o(1-o)$$

$$\boxed{\hat{\mathbf{w}}=\mathbf{w}+\eta\cdot\lambda\cdot(d-o)o(1-o)\mathbf{y}}$$

Continuous Perceptron Training Algorithm…



$$E_k = \frac{1}{2}\left\{d^k - \left[\frac{2}{1+\exp(-\lambda\, net^k)} - 1\right]\right\}^2$$

$$E_1(\mathbf{w}) = \frac{1}{2}\left\{1 - \left[\frac{2}{1+\exp\left[-\lambda\,(w_1 + w_2)\right]} - 1\right]\right\}^2$$

$\lambda = 1$ and reducing the terms simplifies this expression to the following form

$$E_1(\mathbf{w}) = \frac{2}{\left[1+\exp(w_1 + w_2)\right]^2}$$

similarly

$$E_2(\mathbf{w}) = \frac{2}{\left[1+\exp(0.5w_1 - w_2)\right]^2}$$

$$E_3(\mathbf{w}) = \frac{2}{\left[1+\exp(3w_1 + w_2)\right]^2}$$

$$E_4(\mathbf{w}) = \frac{2}{\left[1+\exp(2w_1 - w_2)\right]^2}$$