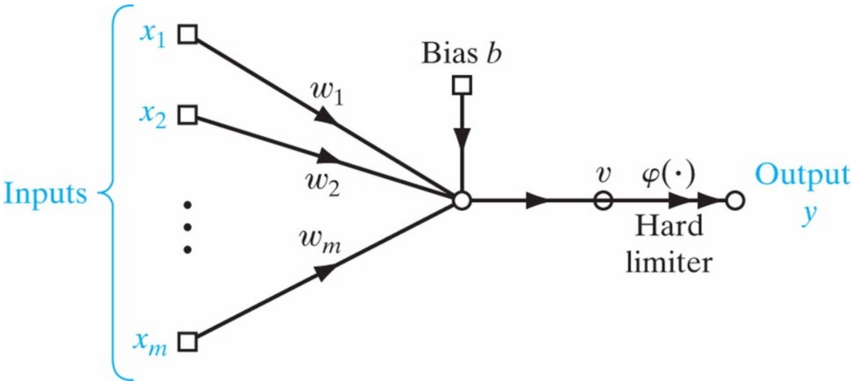
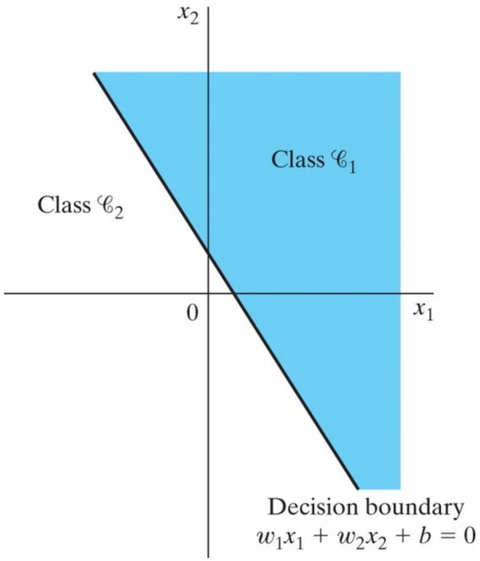
Perceptron

Deep Learning & Applications

(Rosenblatt's) Perceptron

- LMS algorithm was described with a linear neuron but the perceptron is built around a non-linear neuron
- Output is +1 for positive and -1 for negative (can be thought as 2 classes)
- The synaptic weights can be adapted by using *perceptron convergence algorithm*





Perceptron Convergence Theorem

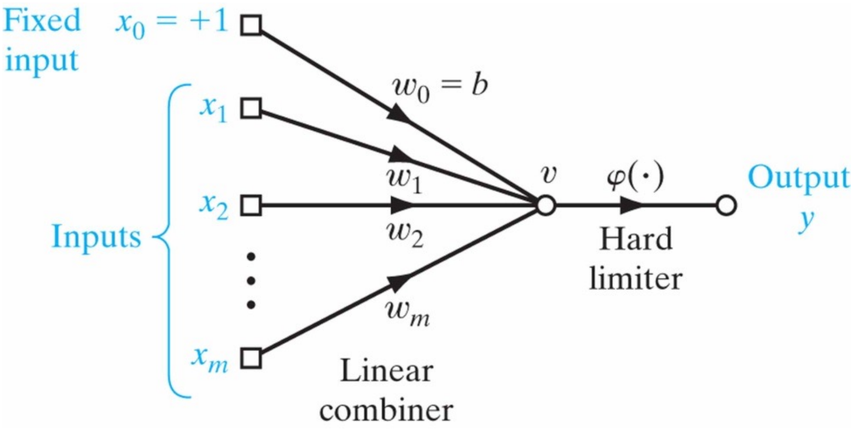
- Input vector
  $\mathbf{x}(n) = [+1, x_1(n), x_2(n), \dots, x_m(n)]^T$

where n is the iteration step

- Weight vector
  $\mathbf{w}(n) = [w_0(n), w_1(n), , \dots, w_m(n)]^T$

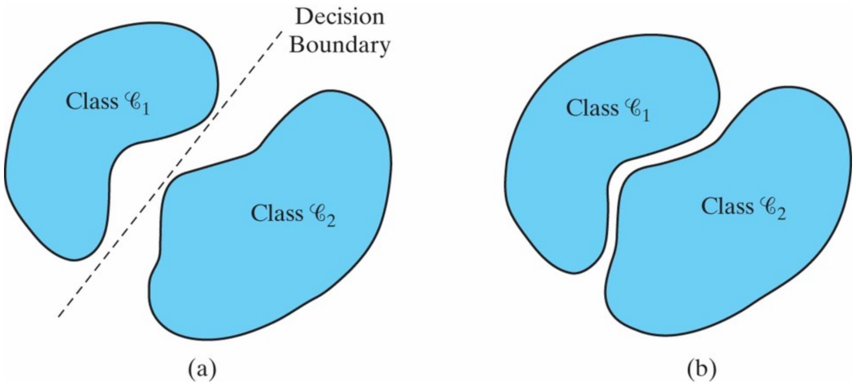
where $w_0(n) = b(n)$

- Linear combiner output can be written
  $$v(n) = \mathbf{w}^T(n)\mathbf{x}(n)$$
- For fixed n, equation $w^T x = 0$ defines a hyperplane in m-dimensional space as the decision surface between 2 classes



Perceptron Convergence Theorem

- For fixed n, equation $w^T x = 0$ defines a hyperplane in m-dimensional space as the decision surface between 2 classes
- For the perceptron to function properly, the 2 classes must be *linearly separable*



Perceptron Convergence Theorem

- The algorithm for adapting the weights
  1. If the nth member of the training set $\mathbf{x}(n)$ is correctly classified by $\mathbf{w}(n)$ at the nth iteration, no correction is made
  2. Otherwise, weight vector is updated as follows

$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta(n)\mathbf{x}(n)$ if $\mathbf{w}^T(n)\mathbf{x}(n) > 0$ and $\mathbf{x}(n)$ belongs to $\mathcal{C}_2$

$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{x}(n)$ if $\mathbf{w}^T(n)\mathbf{x}(n) \leq 0$ and $\mathbf{x}(n)$ belongs to $\mathcal{C}_1$

TABLE 1.1 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

$$\mathbf{x}(n) = (m+1)\text{-by-1 input vector}$$
$$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$$
$$\mathbf{w}(n) = (m+1)\text{-by-1 weight vector}$$
$$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$$
$$b = \text{bias}$$
$$y(n) = \text{actual response (quantized)}$$
$$d(n) = \text{desired response}$$
$$\eta = \text{learning-rate parameter, a positive constant less than unity}$$

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 continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as
$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$
where $\text{sgn}(\cdot)$ is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$
where
$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$
5. *Continuation.* Increment time step n by one and go back to step 2.