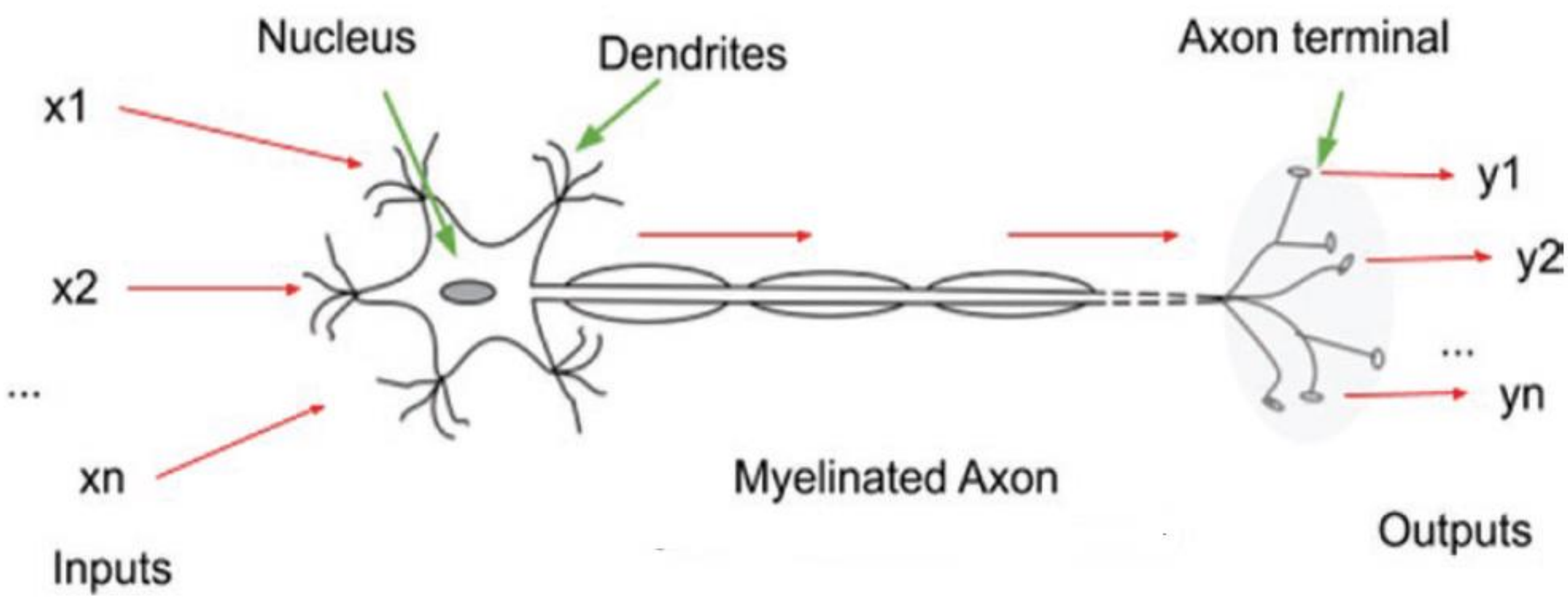
DEEP LEARNING

Dr. Mithun Kumar Kar
Assistant Professor (Sr-Gr)
School of Artificial Intelligence
Amrita Vishwa Vidyapeetham Coimbatore

Artificial Neural Networks

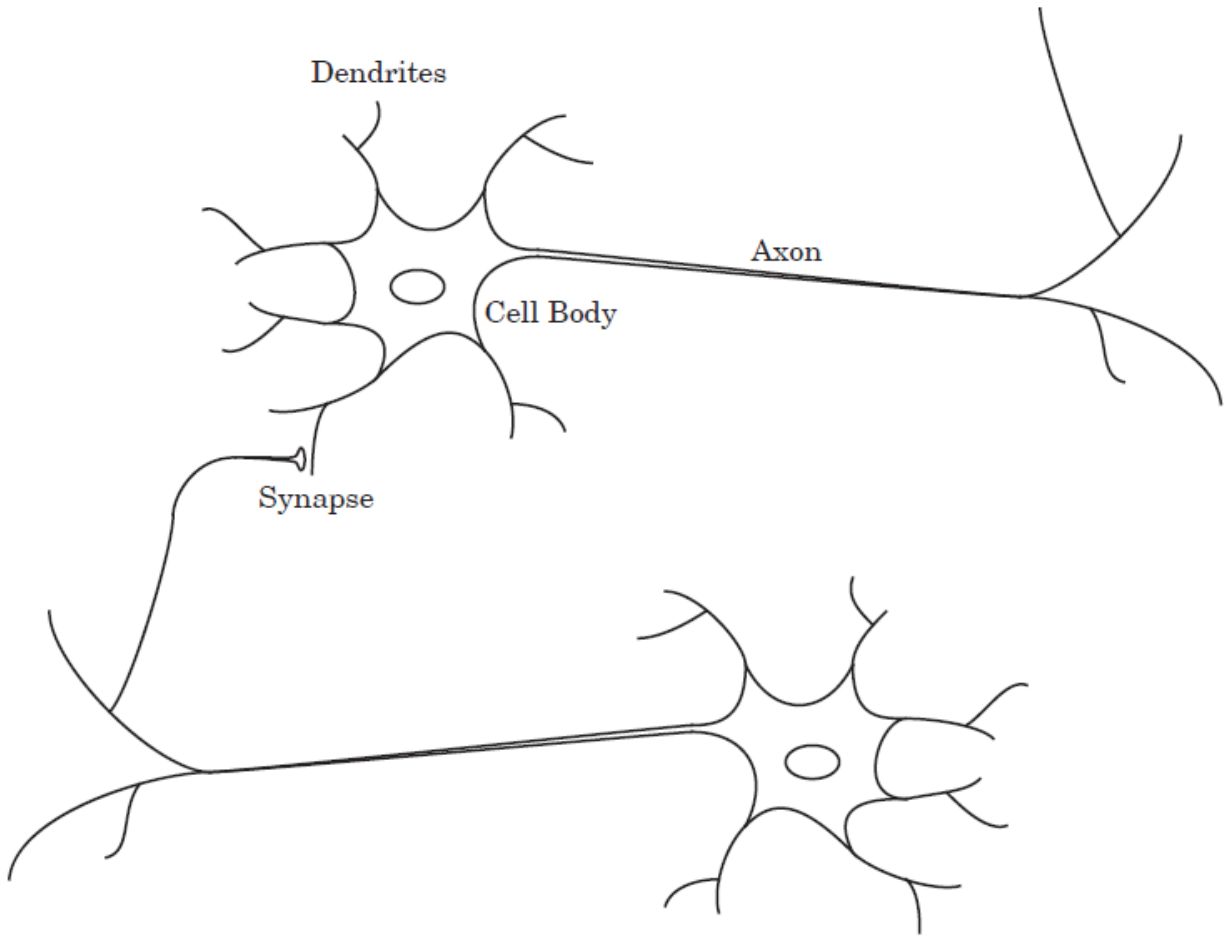
- An artificial neural network (ANN) is a computing system that is designed to work the way the human brain works.



- **When we go in to a new environment, we adapt to the new environment that is we learn**.

Artificial Neural Networks

- Neurons have three principal components: the **dendrites**, the **cell body** and the **axon**.

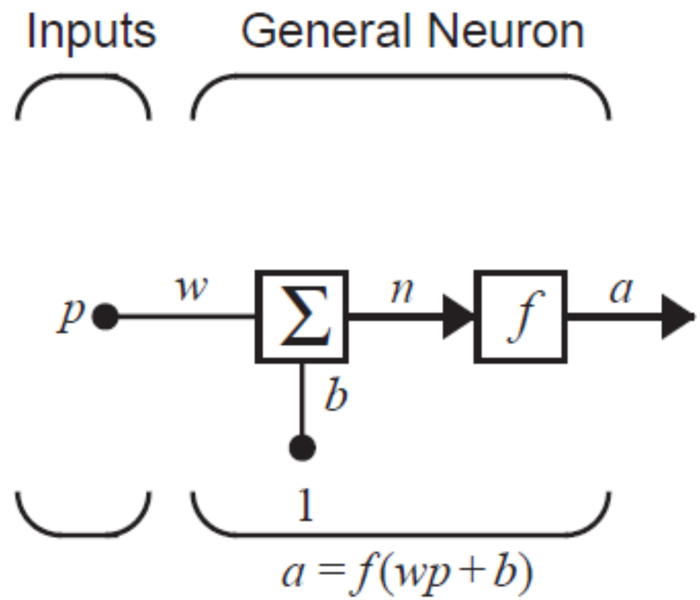


Artificial Neural Networks

- The **dendrites** are tree-like **receptive networks** of nerve fibers that carry electrical signals into the cell body.
- The **cell body** effectively **sums and thresholds** these incoming signals.
- The **axon** is a single long fiber that **carries the signal** from the cell body out to other neurons.
- The point of **contact between** an **axon** of one cell and a **dendrite** of another cell is called a **synapse**.

Neuron Models

- **Single-Input Neuron**:
- P is the input,
  w is the weight,
  b is the bias or offset,
  f is called the activation
  function or transfer function.



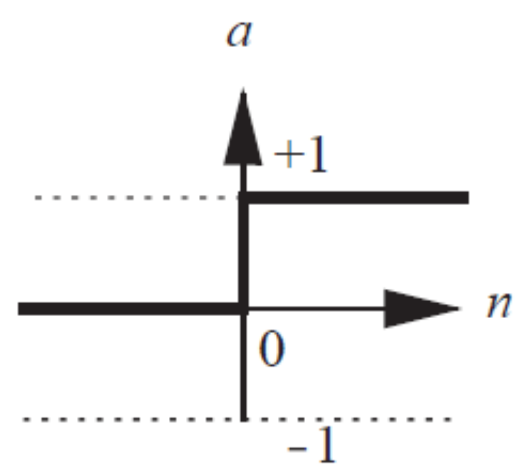
- The summer output , often ref
  goes

into a transfer function , which produces the scalar neuron output .

- If we take p = 2, w = 3, b = -1.5
  a = f (wp + b) = f[2(3)-1.5)]=f(4.5)
- The actual output depends on the particular transfer function that is chosen.

Neuron Models

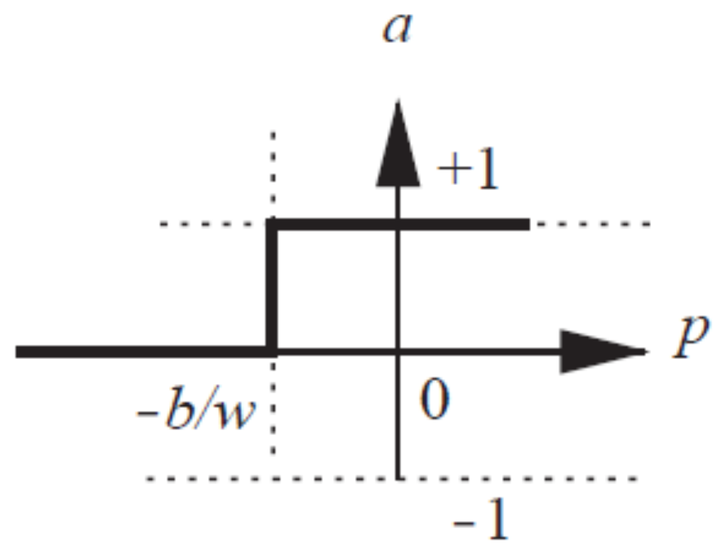
- **Transfer Functions:** The transfer function may be a linear or a nonlinear function of net input ‘n’.
- A variety of transfer functions are used depending on the required application of the neural network.
- **Hard Limit Transfer Function:**
- The hard limit transfer function sets the output of the neuron to 0 if the function argument is less than 0, or 1 if its argument is greater than or equal to 0.



Hard Limit Transfer Function

Neuron Models

- **a = f (wp + b)** → 



$$a = f\left(\sum_i w_i p_i\right) = f(W.P) = f(W^T P)$$

- The above figure illustrates the input/output characteristic of a single-input neuron that uses a hard limit transfer function.

Transfer functions

Name	Input/Output Relation	Icon	MATLAB Function
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$		hardlim
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$		hardlims
Linear	$a = n$		purelin
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$		satlin
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$		satlins

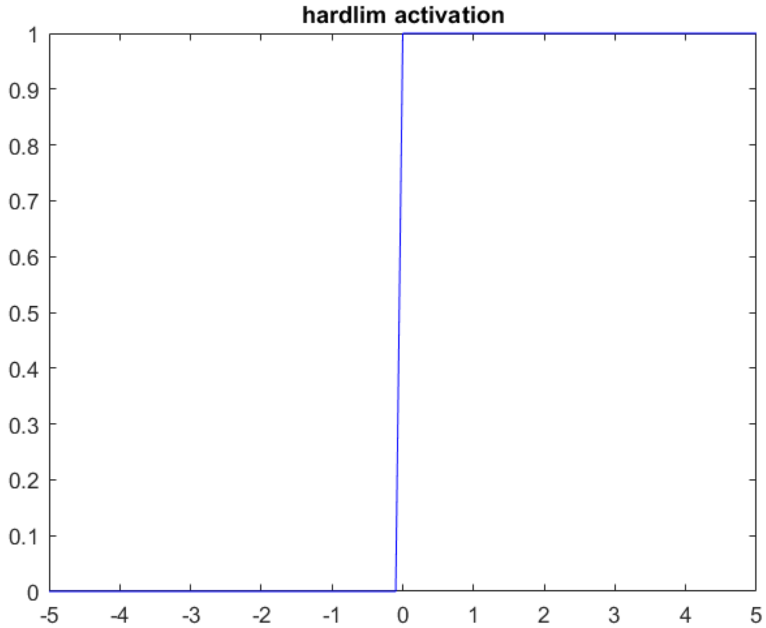
Transfer functions

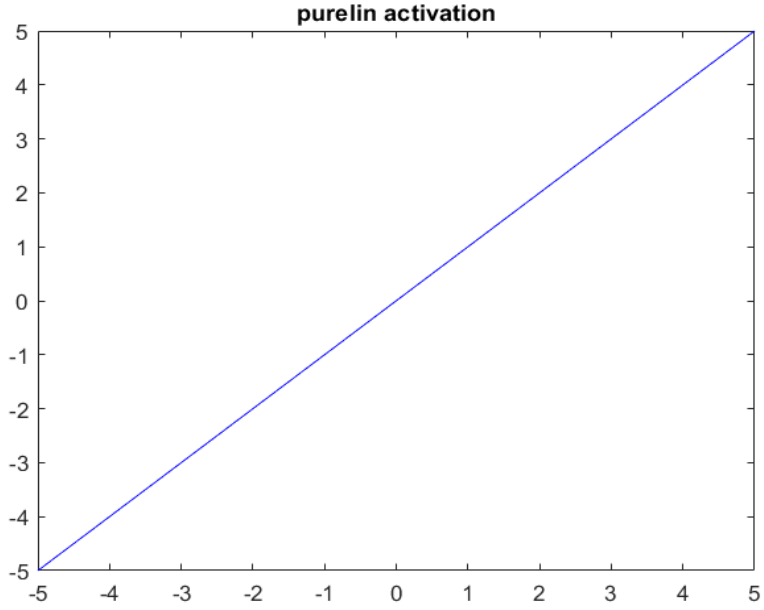
Log-Sigmoid	$a = \dfrac{1}{1+e^{-n}}$		logsig
Hyperbolic Tangent Sigmoid	$a = \dfrac{e^{n}-e^{-n}}{e^{n}+e^{-n}}$		tansig
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \le n$		poslin
Competitive	$a = 1$ neuron with max n $a = 0$ all other neurons		compet

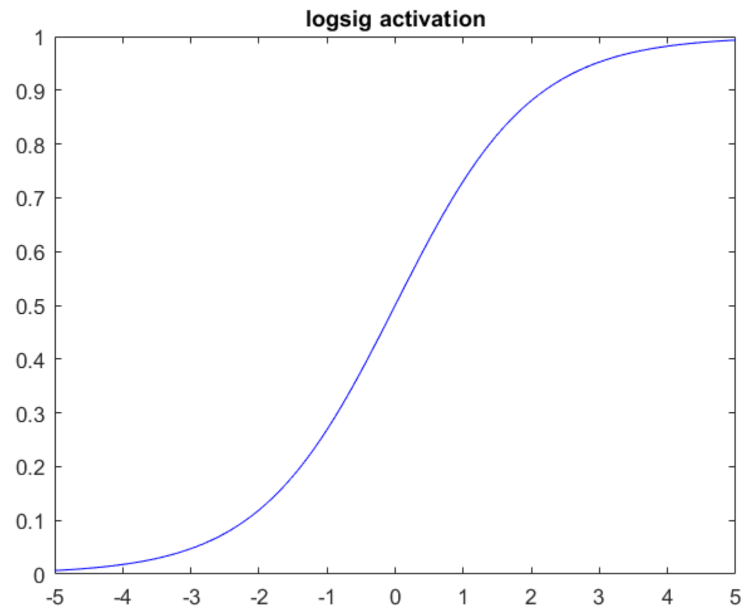
Matlab Implementation

- n = -5:0.1:5;
- figure()
- plot(n,hardlim(n),'b');
- title('hardlim activation')
- figure();
- plot(n,purelin(n),'b');
- title('purelin activation')
- figure();
- plot(n,logsig(n),'b');
- title('logsig activation')

Matlab Implementation



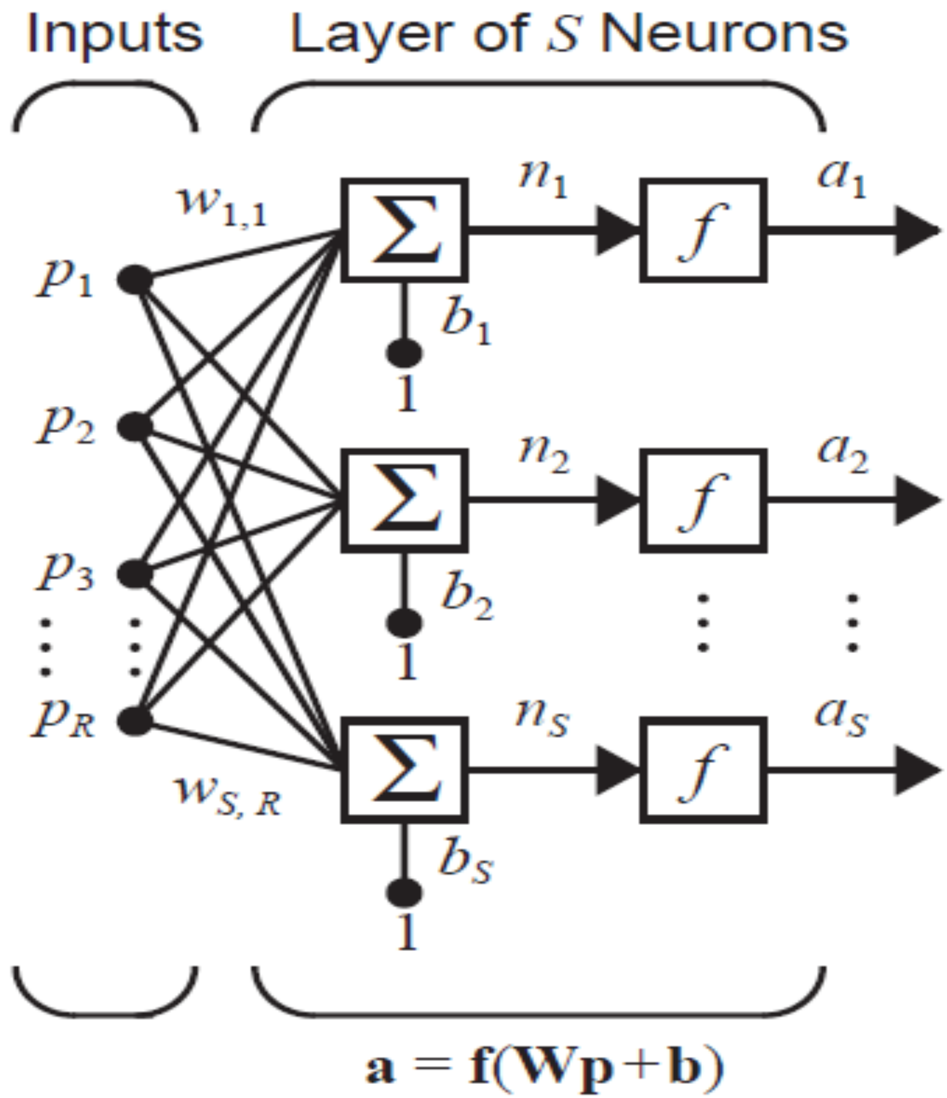




A single-layer network of S neurons

- Layers of n neurons:
- Each of the R inputs is connected to each of the neurons.

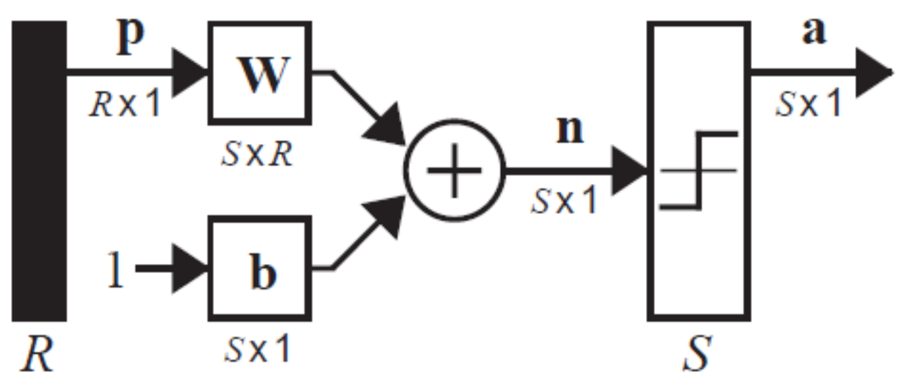
$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$



Layer of S Neurons

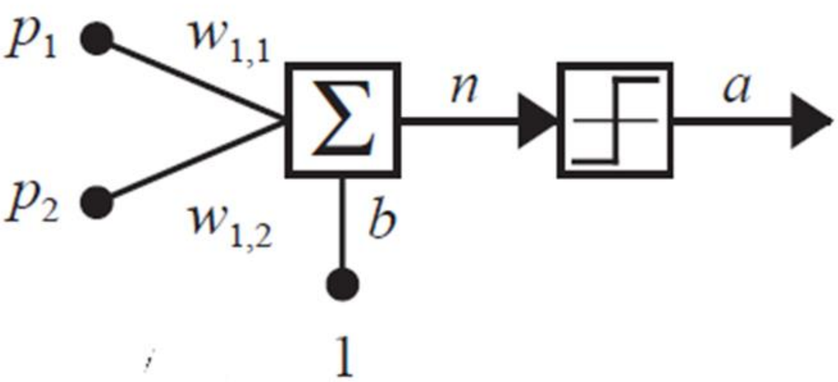
Perceptron

- Figure shows a single-layer perceptron with symmetric hard limit transfer function (Single-neuron perceptron).
- Single-neuron perceptron can classify input vectors into two categories.



Perceptron

- For a two-input perceptron,



$$a = f(wp + b)$$

$$a = hardlim(n) = hardlim(\mathbf{W}\mathbf{p} + b)$$

$$= hardlim({}_1\mathbf{w}^T\mathbf{p} + b) = hardlim(w_{1,1}p_1 + w_{1,2}p_2 + b)$$

Perceptron

- The decision boundary between the categories is determined by $wp + b = 0$
- The decision boundary is determined by the input vectors for which the net input 'n' is zero.
- To draw the line, we can find the points where it intersects the and axes
- To draw the line, we can find the points where it intersects the p_1 and p_2 axes.

Perceptron Learning Rule

- Let $w_{11} = 1, w_{12} = 1 \text{ and } b = \text{-}1$
- The decision boundary is then given by $n = w_i^T p + b = 0$

$$w_{11} p_1 + w_{12} p_2 + b = 0$$

- This defines a line in the input space.
- On the line and on one side of the line the network output will be 0 , whereas on the other side of the line the output will be 1.
- We can find the intercepts as

$$p_1 = -\frac{b}{w_{11}} \qquad p_2 = -\frac{b}{w_{12}}$$

Perceptron Learning Rule

- To find the p_1 intercept, set $p_2 = 0$

$$p_1 = -\frac{b}{w_{11}} = -\frac{-1}{1} = 1$$

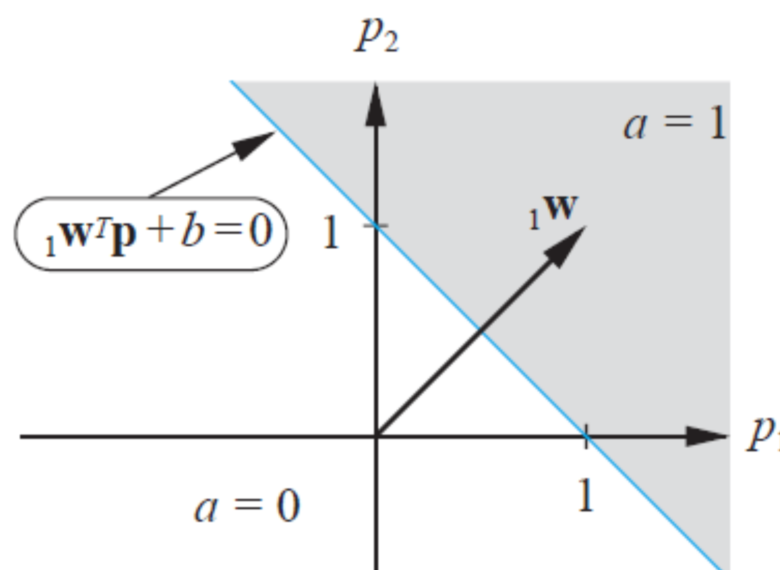
- To find the p_2 intercept, set $p_1 = 0$

$$p_2 = -\frac{b}{w_{12}} = -\frac{-1}{1} = 1$$

- The resulting decision boundary is
- For the input $p = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$

$$a = activation(w_i^T p + b)$$

$$= activation([1,1]\begin{bmatrix} 2 \\ 0 \end{bmatrix} - 1) = 1$$



Perceptron Learning Rule

- For the input $p = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$

$$a = activation(w_i^T p + b) \qquad w_{11} = 1, w_{12} = 1 \text{ and } b = \text{-}1$$

$$a = activation\left([1,1]\begin{bmatrix} -2 \\ 0 \end{bmatrix} - 1 \right) = activation(-3) = 0$$

- For the input

$$p = \begin{bmatrix} -2 \\ +4 \end{bmatrix} \quad a = activation\left([1,1]\begin{bmatrix} -2 \\ 4 \end{bmatrix} - 1 \right) = activation(1) = 1$$

Perceptron

- If the vector inputs are three dimensional with a single neuron then

$$y = activation\left(\left[w_{11}, w_{12}, w_{13}\right]\begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} + b\right)$$

Perceptron Learning Rule

- By learning rule we mean a procedure for updating the weights and biases of a network to perform some specific task.
- Perceptron learning algorithm converges to a correct W within a finite number of iterations, k, over the data if the classes are linearly separable.
- There are many types of neural network learning rules. They fall into three broad categories: **supervised learning**, **unsupervised learning** and **reinforcement (or graded) learning.**

Perceptron Learning Rule

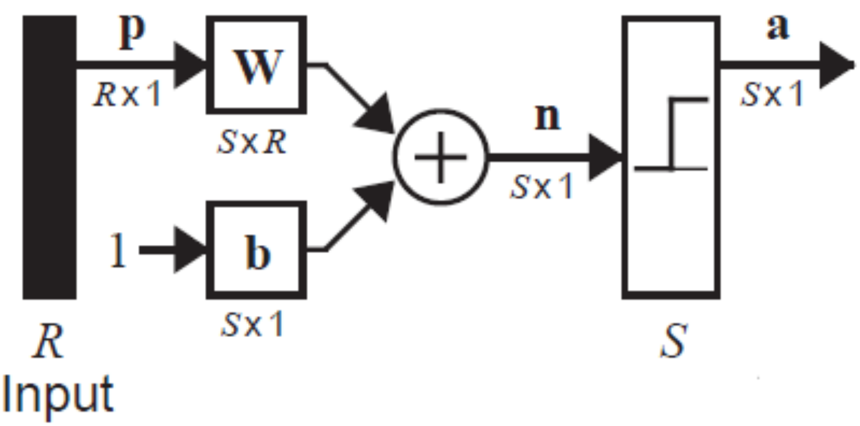
- In **supervised learning**, the learning rule is provided with a set of examples (the training set) of proper network behavior: $\{p_1, t_1\}, \{p_2, t_2\}, \ldots \{p_n, t_n\}$

'p_n' is the input to the model and 't_n' is corresponding correct (target) output.

- **In unsupervised learning**, the weights and biases are modified in response to network inputs only. There are no target outputs available.

Perceptron Learning Rule

- Perceptron Architecture:



$$ {}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix}. $$

$$ \mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}. $$

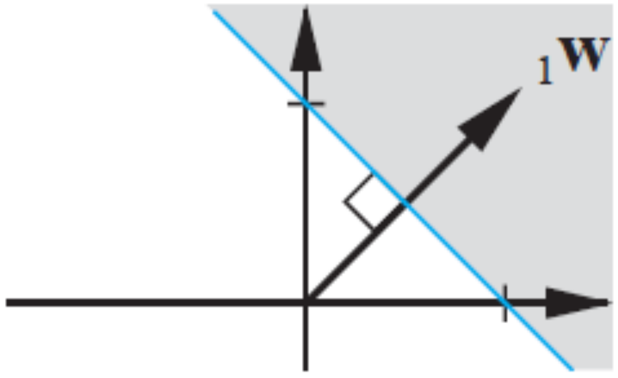
$$ \mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix}. $$

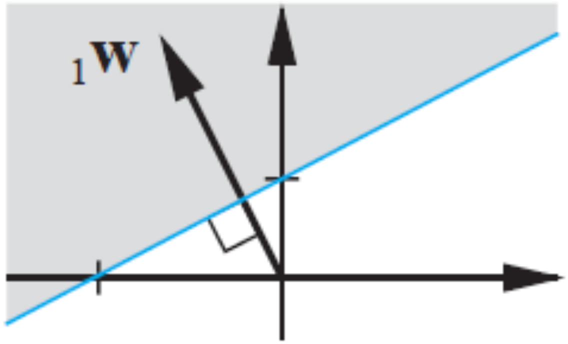
$$ a_i = Activation(WP + b) $$

$$ a_i = Activation(w_i^T p + b) $$

Perceptron Learning Rule

- For all points on the boundary, the inner product of the input vector with the weight vector is the same.
- This implies that these input vectors will all have the same projection onto the weight vector, so they must lie on a line orthogonal to the weight vector.
- Therefore the weight vector will always point toward the region where the neuron output is 1.



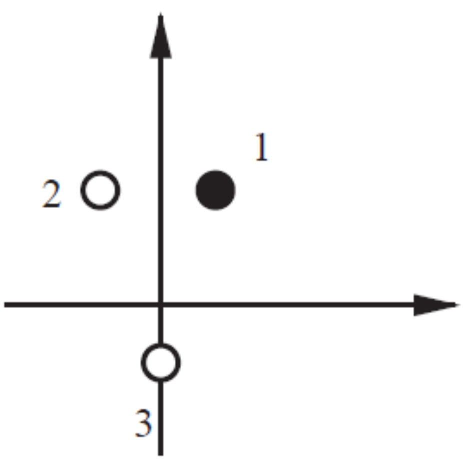


Perceptron Learning Rule

- Perceptron learning rule we will begin with a simple test problem.
- Let the input/target pairs are

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, t_1 = 1\right\} \left\{\mathbf{p}_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, t_2 = 0\right\} \left\{\mathbf{p}_3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, t_3 = 0\right\}.$$

- There must be an allowable decision boundary that can separate the vectors p_2 and p_3 from p_1.



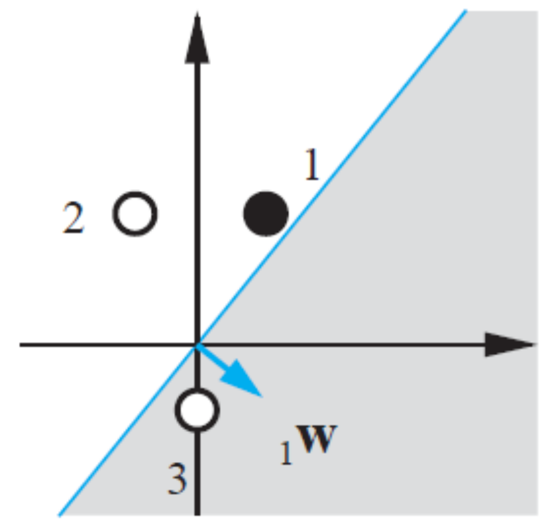
Constructing Learning Rules

- Let set the weight vectors randomly $w_{11} = \begin{bmatrix} 1 \\ -0.8 \end{bmatrix}$
- We begin with P1.

$$a = hardlim({}_1\mathbf{w}^T\mathbf{p}_1) = hardlim\left(\begin{bmatrix} 1.0 & -0.8 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right)$$

$$a = hardlim(-0.6) = 0 .$$

- The network has not returned
- the correct value.
- **The network output is 0**,
- while the target response is $t_1 = 1$



Perceptron Learning Rule

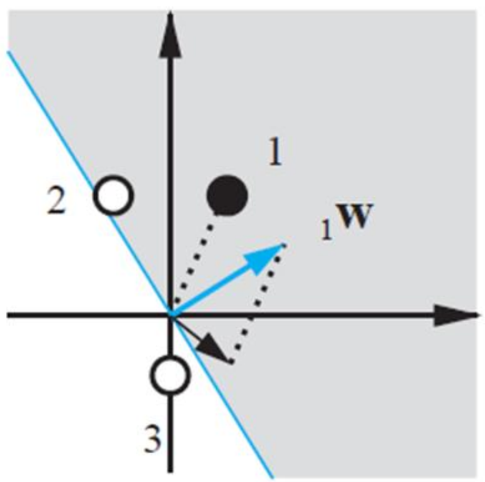
- Let add p_1 to w_{11} with the condition

$$\text{If } t = 1 \text{ and } a = 0, \text{ then } \mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} + \mathbf{p}.$$

$$\mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} + \mathbf{p}_1 = \begin{bmatrix} 1.0 \\ -0.8 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix}.$$

- We now move on to the next input vector and will continue making changes to the weights and cycling through the inputs until they are all classified correctly.
- The next input vector is p_2.

$$a = activation\left([2, 1.2] \begin{bmatrix} -1 \\ 2 \end{bmatrix} \right)$$
$$= activation(0.4) = 1$$



- A class 0 vector was misclassified as a 1.

Perceptron Learning Rule

- Since we would now like to move the weight vector w_{11} away from the input, we can simply change the addition in to subtraction.

$$\text{If } t = 0 \text{ and } a = 1, \text{ then } \mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} - \mathbf{p}.$$

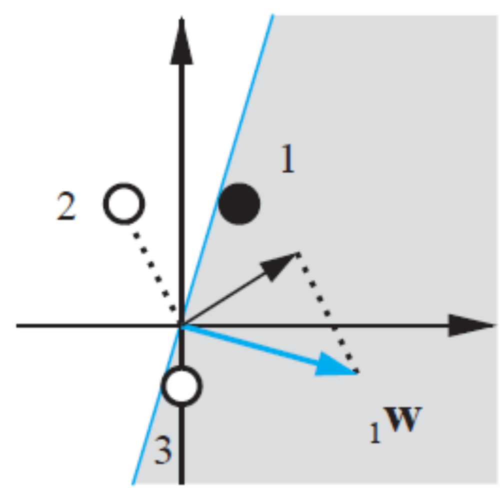
- If we apply this to the test problem we find

$$\mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} - \mathbf{p}_2 = \begin{bmatrix} 2.0 \\ 1.2 \end{bmatrix} - \begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix}$$

- The next input vector is p_3

$$a = activation\left([3.0, -0.8] \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right)$$

$$= activation(0.8) = 1$$



Perceptron Learning Rule

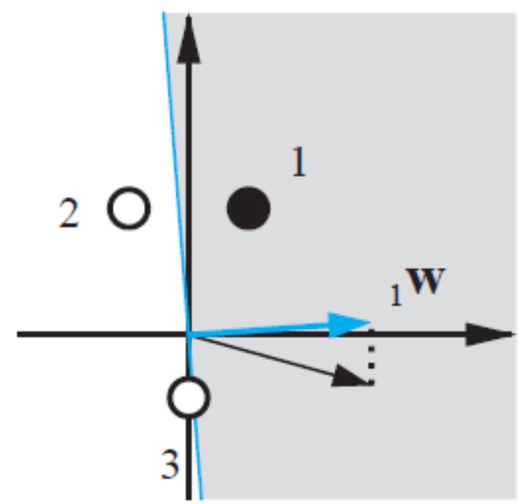
- The current weight w_{11} results in a decision boundary that misclassifies p_3 .
- So w_{11} will updated using the same condition.

$$\mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} - \mathbf{p}_3 = \begin{bmatrix} 3.0 \\ -0.8 \end{bmatrix} - \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 3.0 \\ 0.2 \end{bmatrix}$$

$$a = activation\left([3.0, 0.2] \begin{bmatrix} 0 \\ -1 \end{bmatrix} \right)$$

$$= activation(-0.2) = 0$$

If $t = a$, then $\mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old}$



Perceptron Learning Rule

- Here are the three rules, which cover all possible combinations of output and target values:

$$\text{If } t = 1 \text{ and } a = 0 \text{, then } \mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} + \mathbf{p}.$$

$$\text{If } t = 0 \text{ and } a = 1 \text{, then } \mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} - \mathbf{p}.$$

$$\text{If } t = a \text{, then } \mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old}.$$

- Now if we take **e = t – a**

Perceptron Learning Rule

- We have

$$\text{If } e = 1, \text{ then } \mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} + \mathbf{p}.$$

$$\text{If } e = -1, \text{ then } \mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} - \mathbf{p}.$$

$$\text{If } e = 0, \text{ then } \mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old}.$$

- we can see that the sign of **P** is the same as the sign on the error, e. Furthermore, the absence of in the third rule corresponds to an error e of 0.

$$\mathbf{w}_{11}^{new} = \mathbf{w}_{11}^{old} + e\mathbf{p} = \mathbf{w}_{11}^{old} + (t - a)\mathbf{p}.$$

$$b^{new} = b^{old} + e.$$

Perceptron Learning Rule

- The perceptron rule can be written conveniently in matrix notation

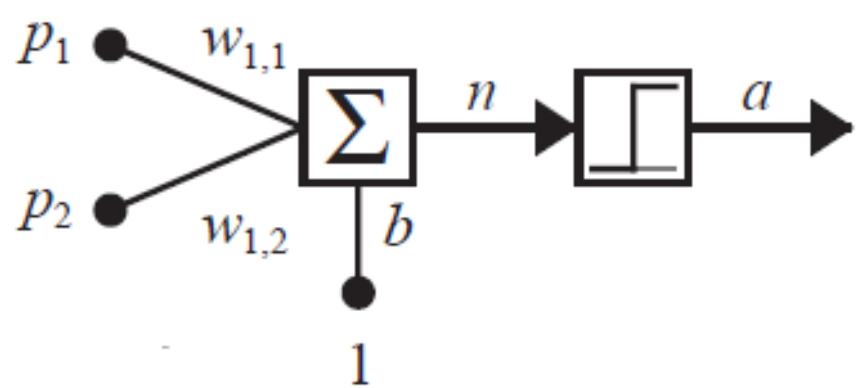
$$W^{new} = W^{old} + eP^{T}$$

$$b^{new} = b^{old} + e$$

- **Training Multiple-Neuron Perceptron:**

$$W_i^{new} = W_i^{old} + e_i P$$

$$b_i^{new} = b_i^{old} + e_i$$



Perceptron Learning Rule

- Example: $$\left\{\mathbf{p}_1 = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}, t_1 = \begin{bmatrix} 0 \end{bmatrix}\right\} \qquad \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, t_2 = \begin{bmatrix} 1 \end{bmatrix}\right\}$$
- Typically the weights and biases are initialized to small random numbers. Let

$$W = \begin{bmatrix} 0.5 \text{ -1 -0.5} \end{bmatrix} \qquad b = 0.5$$

- The first step is to apply the first input vector P1

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + 0.5\right)$$

$$= hardlim(2.5) = 1$$

Perceptron Learning Rule

- Then we calculate the error: $e = t_1 - a = 0 - 1 = -1$
- The weight update is

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix} 0.5 & -1 & -0.5 \end{bmatrix} + (-1)\begin{bmatrix} 1 & -1 & -1 \end{bmatrix}$$
$$= \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix}.$$

- The bias update is

$$b^{new} = b^{old} + e = 0.5 + (-1) = -0.5.$$

- This completes the first iteration.

Perceptron Learning Rule

- The second iteration of the perceptron rule is:

$$a = hardlim(\mathbf{W}\mathbf{p}_2 + b) = hardlim\left(\begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + (-0.5)\right)$$

$$= hardlim(-0.5) = 0$$

$$e = t_2 - a = 1 - 0 = 1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} + 1\begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 0.5 & 1 & -0.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = -0.5 + 1 = 0.5$$

Perceptron Learning Rule

- The third iteration begins again with the first input

$$a = hardlim(\mathbf{W}\mathbf{p}_1 + b) = hardlim\left(\begin{bmatrix} 0.5 & 1 & -0.5 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix} + 0.5\right)$$

$$= hardlim(0.5) = 1$$

$$e = t_1 - a = 0 - 1 = -1$$

$$\mathbf{W}^{new} = \mathbf{W}^{old} + e\mathbf{p}^T = \begin{bmatrix} 0.5 & 1 & -0.5 \end{bmatrix} + (-1)\begin{bmatrix} 1 & -1 & -1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5 & 2 & 0.5 \end{bmatrix}$$

$$b^{new} = b^{old} + e = 0.5 + (-1) = -0.5 .$$

Perceptron Learning Rule

- This weight and bias is applied to 2nd input

$$a = \text{hardlim}\left([-0.5\ 2\ 0.5] \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} + -0.5 \right) = \text{hardlim}(.5) = 1$$

$$e = t_2 - a = 1 - 1 = 0$$

$$\mathrm{W}^{\text{new}} = \mathrm{W}^{\text{old}} + \mathrm{e}\mathrm{P}^{\mathrm{T}} \qquad \mathrm{b}^{\text{new}} = \mathrm{b}^{\text{old}} + \mathrm{e}$$

$$\mathrm{W}^{\text{new}} = [-0.5\ 2\ 0.5] \qquad \mathrm{b}^{\text{new}} = -0.5$$

Matlab implementation

- % You can create a perceptron with the following:
- %
- % net = perceptron;
- % net = configure(net,P,T);
- % where input arguments are as follows:
- %
- % P is an R-by-Q matrix of Q input vectors of R elements each.
- %
- % T is an S-by-Q matrix of Q target vectors of S elements each.

- P = [0 2];
- T = [0 1];
- net = perceptron;
- net = configure(net,P,T);
- inputweights = net.inputweights{1,1}
- biases = net.biases{1}

Matlab implementation

- %Start with a single neuron having an input vector with just two elements.
- net = perceptron;
- net = configure(net,[0;0],0);
- net.b{1} = [0];
- w = [1 -0.8];
- net.IW{1,1} = w;
- % The input target pair is given by
- p = [1; 2];
- t = [1];
- %compute the output and error
- a = net(p)
- e = t-a
- dw = learnp(w,p,[],[],[],[],e,[],[],[],[],[])
- wnew = w + dw

Perceptron learning

- Solve the following classification problem with the perceptron rule.

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, t_1 = 0\right\} \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, t_2 = 1\right\} \left\{\mathbf{p}_3 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}, t_3 = 0\right\} \left\{\mathbf{p}_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, t_4 = 1\right\}$$

- Using initial weights and bias, we can start

$$\mathbf{W}(0) = \begin{bmatrix} 0 & 0 \end{bmatrix} \qquad b(0) = 0 .$$

- For the first input vector p1, using the initial weights and bias, the output a is

$$a = hardlim(\mathbf{W}(0)\mathbf{p}_1 + b(0))$$

$$= hardlim\left(\begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} + 0\right) = hardlim(0) = 1$$

Perceptron learning

- The output a does not equal the target value t1.

$$e = t_1 - a = 0 - 1 = -1$$

$$\mathbf{W}(1) = \mathbf{W}(0) + e\mathbf{p}_1^T = \begin{bmatrix} 0 & 0 \end{bmatrix} + (-1)\begin{bmatrix} 2 & 2 \end{bmatrix} = \begin{bmatrix} -2 & -2 \end{bmatrix}$$

$$b(1) = b(0) + e = 0 + (-1) = -1$$

- Apply the second input vector p2 , using the updated weights and bias.

$$a = hardlim(\mathbf{W}(1)\mathbf{p}_2 + b(1))$$

$$= hardlim\left(\begin{bmatrix} -2 & -2 \end{bmatrix}\begin{bmatrix} 1 \\ -2 \end{bmatrix} - 1\right) = hardlim(1) = 1$$

- The output a is equal to the target t2, hence no change in weight and bias.

Perceptron learning

- Hence $\mathbf{W}(2) = \mathbf{W}(1)$

$$b(2) = b(1)$$

- We now apply this to the third input vector p3.

$$a = hardlim(\mathbf{W}(2)\mathbf{p}_3 + b(2))$$

$$= hardlim\left(\begin{bmatrix} -2 & -2 \end{bmatrix} \begin{bmatrix} -2 \\ 2 \end{bmatrix} - 1\right) = hardlim(-1) = 0$$

- The output in response to input vector p3 is equal to the target t3. Hence no change.

$$\mathbf{W}(3) = \mathbf{W}(2)$$

$$b(3) = b(2)$$

Perceptron learning

- Move on to the last input vector p4 using this weight.

$$a = hardlim(\mathbf{W}(3)\mathbf{p}_4 + b(3))$$

$$= hardlim\left(\begin{bmatrix} -2 & -2 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix} - 1\right) = hardlim(-1) = 0$$

- This time the output a does not equal the appropriate target t4, update the weight

$$e = t_4 - a = 1 - 0 = 1$$

$$\mathbf{W}(4) = \mathbf{W}(3) + e\mathbf{p}_4^T = \begin{bmatrix} -2 & -2 \end{bmatrix} + (1)\begin{bmatrix} -1 & 1 \end{bmatrix} = \begin{bmatrix} -3 & -1 \end{bmatrix}$$

$$b(4) = b(3) + e = -1 + 1 = 0$$

Perceptron learning

- We now must check the first vector p1 again.

$$a = hardlim(\mathbf{W}(4)\mathbf{p}_1 + b(4))$$

$$= hardlim\left(\begin{bmatrix} -3 & -1 \end{bmatrix}\begin{bmatrix} 2 \\ 2 \end{bmatrix} + 0\right) = hardlim(-8) = 0$$

- Therefore there are no changes.

$$\mathbf{W}(5) = \mathbf{W}(4)$$

$$b(5) = b(4)$$

Now apply this to p2.

$$a = hardlim(\mathbf{W}(5)\mathbf{p}_2 + b(5))$$

$$= hardlim\left(\begin{bmatrix} -3 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ -2 \end{bmatrix} + 0\right) = hardlim(-1) = 0$$

Perceptron learning

- Calculating the error

$$e = t_2 - a = 1 - 0 = 1$$

$$\mathbf{W}(6) = \mathbf{W}(5) + e\mathbf{p}_2^T = \begin{bmatrix} -3 & -1 \end{bmatrix} + (1)\begin{bmatrix} 1 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -3 \end{bmatrix}$$

$$b(6) = b(5) + e = 0 + 1 = 1.$$

- Apply this to p3

$$a = hardlim(\mathbf{W}(6)\mathbf{p}_3 + b(6)) = hardlim\left(\begin{bmatrix} -2 & -3 \end{bmatrix}\begin{bmatrix} -2 \\ 2 \end{bmatrix} + 1\right) = 0 = t_3$$

$$a = hardlim(\mathbf{W}(6)\mathbf{p}_4 + b(6)) = hardlim\left(\begin{bmatrix} -2 & -3 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \end{bmatrix} + 1\right) = 1 = t_4$$

$$a = hardlim(\mathbf{W}(6)\mathbf{p}_1 + b(6)) = hardlim\left(\begin{bmatrix} -2 & -3 \end{bmatrix}\begin{bmatrix} 2 \\ 2 \end{bmatrix} + 1\right) = 0 = t_1$$

Perceptron learning

$$a = hardlim(\mathbf{W}(6)\mathbf{p}_2 + b(6)) = hardlim\left(\begin{bmatrix}-2 & -3\end{bmatrix}\begin{bmatrix}1 \\ -2\end{bmatrix} + 1\right) = 1 = t_2$$

- Therefore the algorithm has converged. The final solution is

$$\mathbf{W} = \begin{bmatrix}-2 & -3\end{bmatrix} \qquad b = 1 .$$

- Now we can graph the training data and the decision boundary of the solution.

$$n = \mathbf{W}\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b = -2p_1 - 3p_2 + 1 = 0 .$$

To find the p_2 intercept of the decision boundary, set $p_1 = 0$:

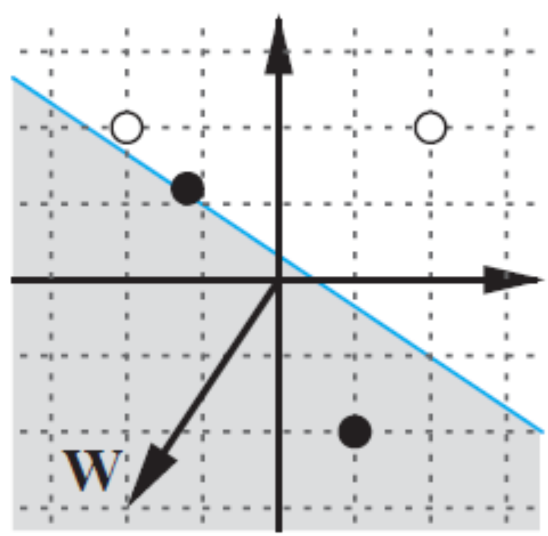
$$p_2 = -\frac{b}{w_{1,2}} = -\frac{1}{-3} = \frac{1}{3} \qquad \text{if } p_1 = 0$$

Perceptron learning

To find the p_1 intercept, set $p_2 = 0$:

$$p_1 = -\frac{b}{w_{1,1}} = -\frac{1}{-2} = \frac{1}{2} \qquad \text{if } p_2 = 0$$

- The resulting decision boundary is like



Matlab implementation

- Now

$$\left\{\mathbf{p}_1 = \begin{bmatrix}2\\2\end{bmatrix}, t_1 = 0\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}1\\-2\end{bmatrix}, t_2 = 1\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}-2\\2\end{bmatrix}, t_3 = 0\right\}\left\{\mathbf{p}_4 = \begin{bmatrix}-1\\1\end{bmatrix}, t_4 = 1\right\}$$

- Let the initial values are W(0) and b(0)

$$\mathbf{W}(0) = [0 \quad 0] \quad b(0) = 0$$

- Then starting with initial weights

$$\alpha = hardlim(\mathbf{W}(0)\mathbf{p}_1 + b(0))$$

$$= hardlim\left([0 \quad 0]\begin{bmatrix}2\\2\end{bmatrix} + 0\right) = hardlim(0) = 1$$

- The output **a** does not equal the target value **t1.**

Matlab implementation

- We get the error $e = t_1 - \alpha = 0 - 1 = -1$

$$\Delta \mathbf{W} = e\mathbf{p}_1^T = (-1)[2 \quad 2] = [-2 \quad -2]$$

$$\Delta b = e = (-1) = -1$$

- Updated weights

$$\mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T = [0 \quad 0] + [-2 \quad -2] = [-2 \quad -2] = \mathbf{W}(1)$$

$$b^{new} = b^{old} + e = 0 + (-1) = -1 = b(1)$$

$$\alpha = hardlim(\mathbf{W}(1)\mathbf{p}_2 + b(1))$$

$$= hardlim\left([-2 \quad -2]\begin{bmatrix} 1 \\ -2 \end{bmatrix} - 1\right) = hardlim(1) = 1$$

- No change in weight in next iteration. The final values are W(6) = [−2 −3] and b(6) = 1.

Matlab implementation

- Matlab:
- net = perceptron;
- p = [2; 2];
- t = [0];
- net.trainParam.epochs = 1;
- net = train(net,p,t);
- w = net.iw{1,1}, b = net.b{1}
- w =
- -2 -2
- b =
- -1

Matlab implementation

- p = [[2;2] [1;-2] [-2;2] [-1;1]]
- t = [0 1 0 1]
- net = perceptron;
- net.trainParam.epochs = 10;
- net = train(net,p,t);
- w = net.iw{1,1}, b = net.b{1}

- w = -2 -3
- b = 1

Matlab implementation

- Now classify with the perceptron rule

$$\left\{\mathbf{p}_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, t_1 = 0\right\} \left\{\mathbf{p}_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, t_2 = 1\right\} \left\{\mathbf{p}_3 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}, t_3 = 0\right\} \left\{\mathbf{p}_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, t_4 = 1\right\}$$

Use the initial weights and bias:

$$\mathbf{W}(0) = \begin{bmatrix} 0 & 0 \end{bmatrix} \qquad b(0) = 0\,.$$

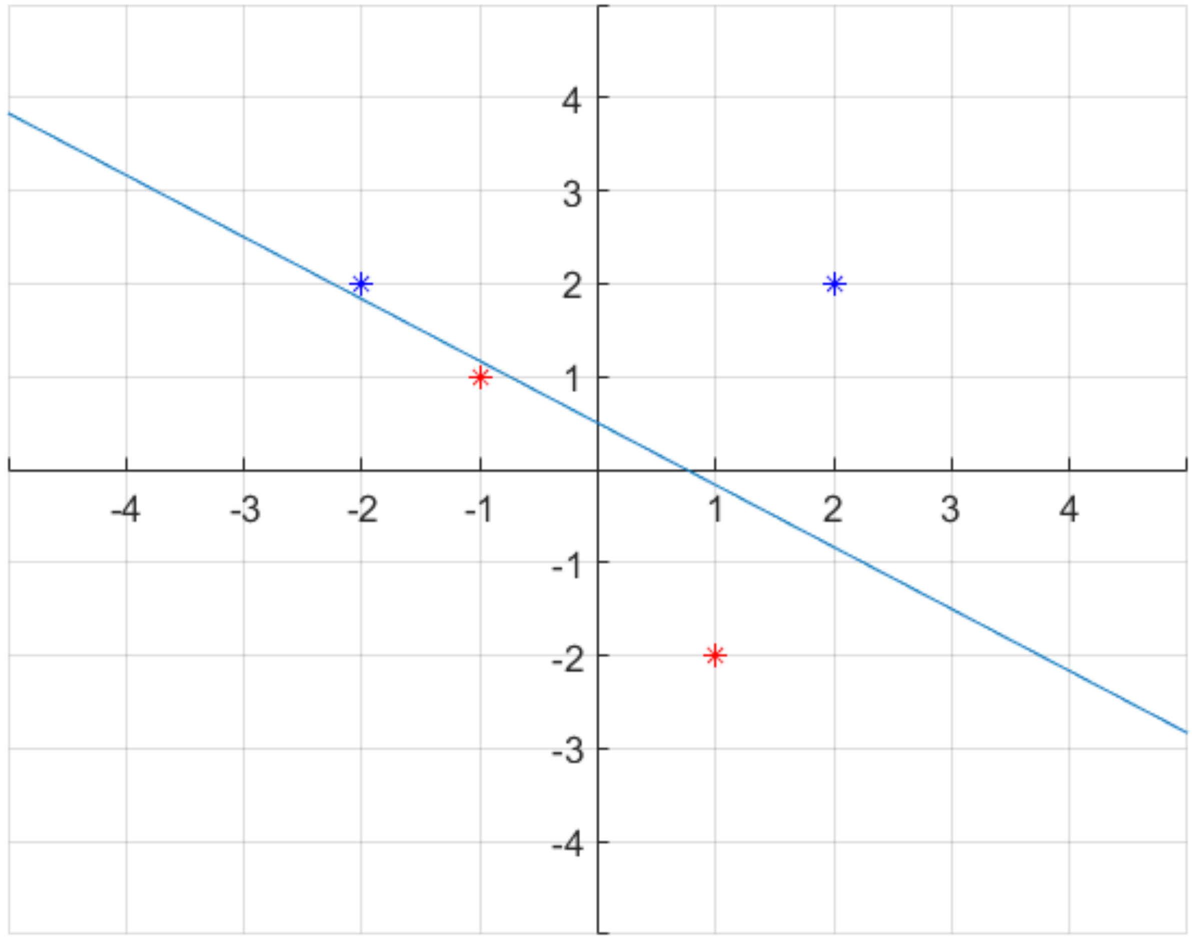
Matlab implementation

- p = [[2;2] [1;-2] [-2;2] [-1;1]]
- t = [0 1 0 1]
- net = perceptron;
- net.trainParam.epochs = 1;
- net = train(net,p,t);
- a = net(p);
- net.trainParam.epochs = 10;
- net = train(net,p,t);
- w = net.iw{1,1}, b = net.b{1}
- %plotting the line
- p1=-b/w(1),p2=-b/w(2),
- %plot([p1 0], [0 p2])
- A = [p1 0];
- B = [0 p2];
- %plot(A,B,'*')
- hold on
- plot(2,2,'*','color','blue')
- hold on
- plot(1,-2,'*','color','red')
- hold on
- plot(-2,2,'*','color','blue')
- hold on

Matlab implementation

- plot(-1,1,'*','color','red')
- axis([-5 5 -5 5])
- ax = gca;
- ax.XAxisLocation = 'origin';
- ax.YAxisLocation = 'origin';
- hold on
- %line(A,B)
- %hold off
- xlim = get(gca,'XLim');
- m = (0-p2)/(p1-0);
- n = p1;
- y1 = m*xlim(1) + n;
- y2 = m*xlim(2) + n;
- grid on, hold on
- line([xlim(1) xlim(2)],[y1 y2])
- hold off
-

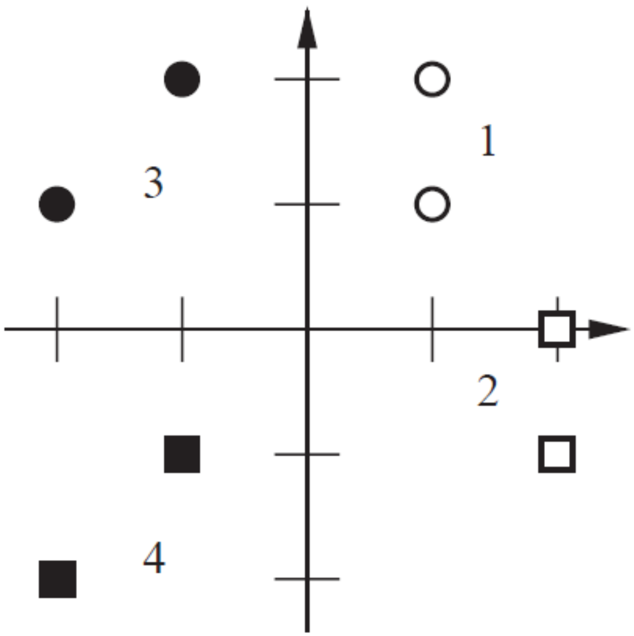
Matlab implementation



Example with two neurons

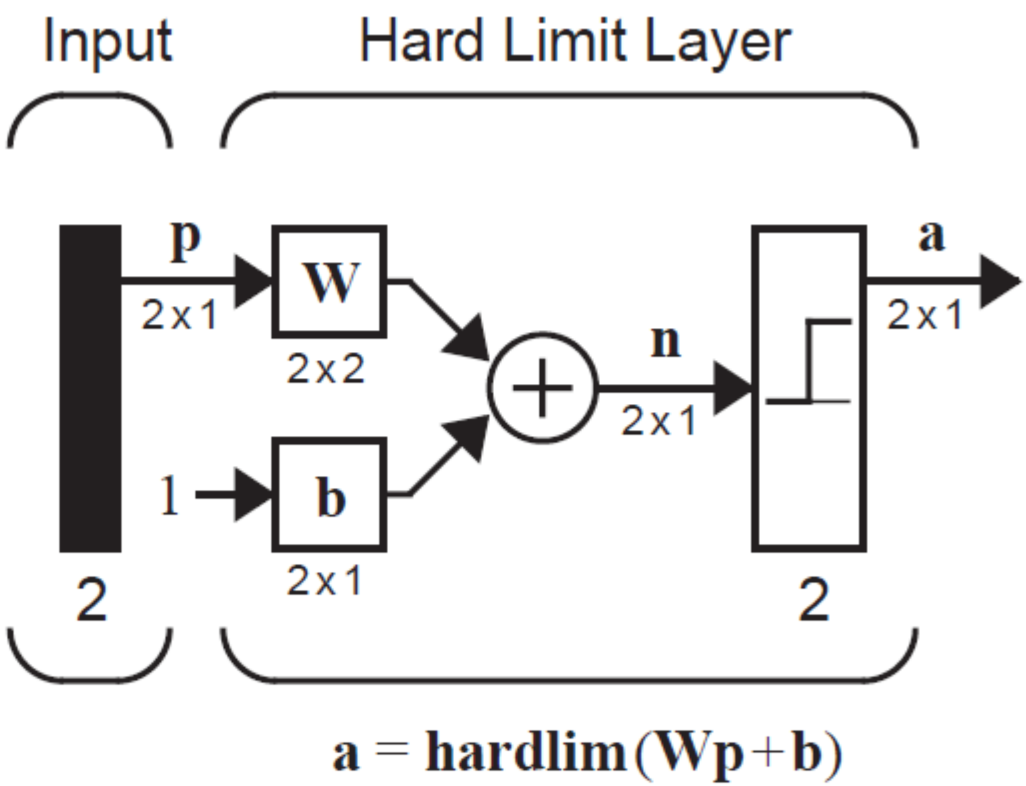
- $$\text{class 1:}\left\{\mathbf{p}_1 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{p}_2 = \begin{bmatrix}1\\2\end{bmatrix}\right\}, \text{class 2: }\left\{\mathbf{p}_3 = \begin{bmatrix}2\\-1\end{bmatrix}, \mathbf{p}_4 = \begin{bmatrix}2\\0\end{bmatrix}\right\},$$

$$\text{class 3:}\left\{\mathbf{p}_5 = \begin{bmatrix}-1\\2\end{bmatrix}, \mathbf{p}_6 = \begin{bmatrix}-2\\1\end{bmatrix}\right\}, \text{class 4: }\left\{\mathbf{p}_7 = \begin{bmatrix}-1\\-1\end{bmatrix}, \mathbf{p}_8 = \begin{bmatrix}-2\\-2\end{bmatrix}\right\}.$$



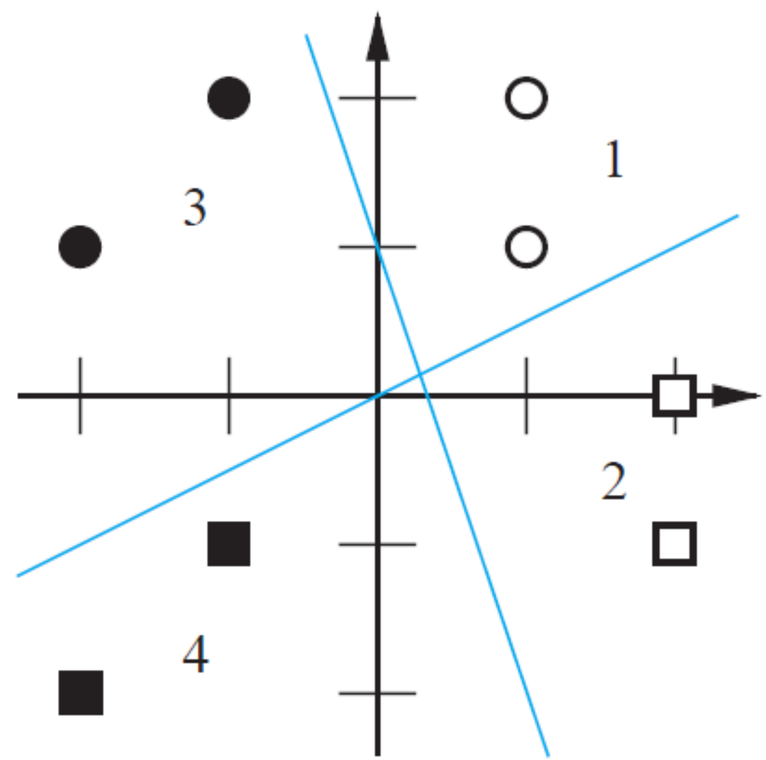
Example

- To solve a problem with four classes of input vector we will need a perceptron with at least two neurons.



Example

- A two-neuron perceptron creates two decision boundaries.
- we need to have one decision boundary divide the four classes into two sets of two. The remaining boundary must then isolate each class.
- The weight vectors should be orthogonal to the decision boundaries and should point toward the regions where the neuron outputs are 1



Example

- This solution corresponds to target values of

$$\text{class 1:}\left\{\mathbf{t}_1 = \begin{bmatrix}0\\0\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}0\\0\end{bmatrix}\right\}, \text{class 2: }\left\{\mathbf{t}_3 = \begin{bmatrix}0\\1\end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix}0\\1\end{bmatrix}\right\},$$

$$\text{class 3:}\left\{\mathbf{t}_5 = \begin{bmatrix}1\\0\end{bmatrix}, \mathbf{t}_6 = \begin{bmatrix}1\\0\end{bmatrix}\right\}, \text{class 4: }\left\{\mathbf{t}_7 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{t}_8 = \begin{bmatrix}1\\1\end{bmatrix}\right\}.$$

Let initialized the weight vectors as $_1\mathbf{w} = \begin{bmatrix}-3\\-1\end{bmatrix}$ and $_2\mathbf{w} = \begin{bmatrix}1\\-2\end{bmatrix}$.

- Now we can calculate the bias by picking a point on a boundary.

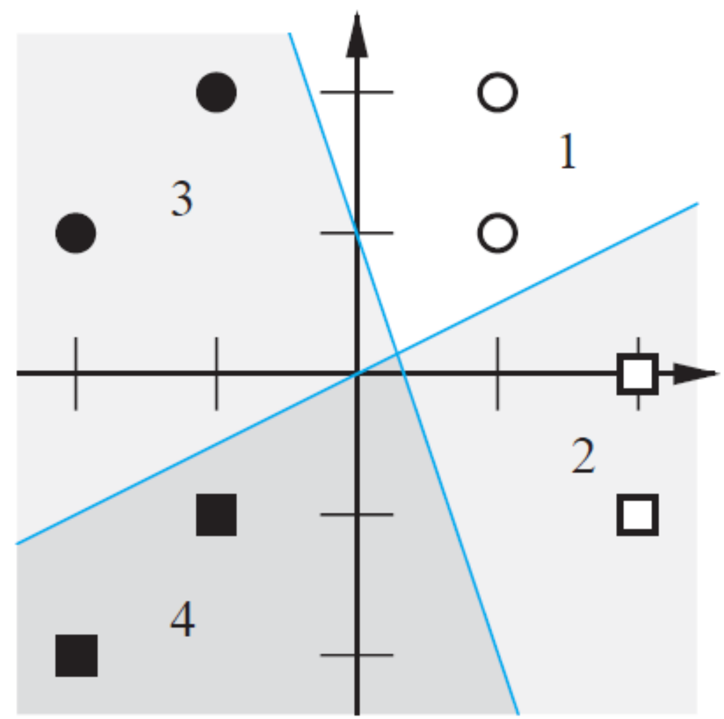
$$b_1 = -_1\mathbf{w}^T\mathbf{p} = -\begin{bmatrix}-3 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = 1\,, \qquad b_2 = -_2\mathbf{w}^T\mathbf{p} = -\begin{bmatrix}1 & -2\end{bmatrix}\begin{bmatrix}0\\0\end{bmatrix} = 0\,.$$

Example

- In matrix form we have

$$\mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \end{bmatrix} = \begin{bmatrix} -3 & -1 \\ 1 & -2 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

- With this the neural network
- Can be iterated.



Example

- Let $\left\{\mathbf{p}_1 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}0\\0\end{bmatrix}\right\} \left\{\mathbf{p}_2 = \begin{bmatrix}1\\2\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}0\\0\end{bmatrix}\right\} \left\{\mathbf{p}_3 = \begin{bmatrix}2\\-1\end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix}0\\1\end{bmatrix}\right\}$

$$\left\{\mathbf{p}_4 = \begin{bmatrix}2\\0\end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix}0\\1\end{bmatrix}\right\} \left\{\mathbf{p}_5 = \begin{bmatrix}-1\\2\end{bmatrix}, \mathbf{t}_5 = \begin{bmatrix}1\\0\end{bmatrix}\right\} \left\{\mathbf{p}_6 = \begin{bmatrix}-2\\1\end{bmatrix}, \mathbf{t}_6 = \begin{bmatrix}1\\0\end{bmatrix}\right\}$$

$$\left\{\mathbf{p}_7 = \begin{bmatrix}-1\\-1\end{bmatrix}, \mathbf{t}_7 = \begin{bmatrix}1\\1\end{bmatrix}\right\} \left\{\mathbf{p}_8 = \begin{bmatrix}-2\\-2\end{bmatrix}, \mathbf{t}_8 = \begin{bmatrix}1\\1\end{bmatrix}\right\}.$$

with initial weights

$$\mathbf{W}(0) = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}, \quad \mathbf{b}(0) = \begin{bmatrix}1\\1\end{bmatrix}.$$

Example

- The first iteration is

$$\mathbf{a} = hardlim\,(\mathbf{W}(0)\mathbf{p}_1 + \mathbf{b}(0)) = hardlim\,\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \end{bmatrix},$$

$$\mathbf{e} = \mathbf{t}_1 - \mathbf{a} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix},$$

$$\mathbf{W}(1) = \mathbf{W}(0) + \mathbf{e}\mathbf{p}_1^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix},$$

$$\mathbf{b}(1) = \mathbf{b}(0) + \mathbf{e} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Example

- The second iteration is

$$\mathbf{a} = hardlim\,(\mathbf{W}(1)\mathbf{p}_2 + \mathbf{b}(1)) = hardlim\,\left(\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\mathbf{e} = \mathbf{t}_2 - \mathbf{a} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

$$\mathbf{W}(2) = \mathbf{W}(1) + \mathbf{e}\mathbf{p}_2^T = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix},$$

$$\mathbf{b}(2) = \mathbf{b}(1) + \mathbf{e} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Example

- The third iteration is

$$\mathbf{a} = hardlim\,(\mathbf{W}(2)\mathbf{p}_3 + \mathbf{b}(2)) = hardlim\,\left(\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} 2 \\ -1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

$$\mathbf{e} = \mathbf{t}_3 - \mathbf{a} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix},$$

$$\mathbf{W}(3) = \mathbf{W}(2) + \mathbf{e}\mathbf{p}_3^T = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\begin{bmatrix} 2 & -1 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ 1 & -1 \end{bmatrix},$$

$$\mathbf{b}(3) = \mathbf{b}(2) + \mathbf{e} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}.$$

Example

- Iterations four through eight produce no changes in the weights.

$$\mathbf{W}(8) = \mathbf{W}(7) = \mathbf{W}(6) = \mathbf{W}(5) = \mathbf{W}(4) = \mathbf{W}(3)$$

$$\mathbf{b}(8) = \mathbf{b}(7) = \mathbf{b}(6) = \mathbf{b}(5) = \mathbf{b}(4) = \mathbf{b}(3)$$

- The ninth iteration produces the result:

Example

$$\mathbf{a} = hardlim\,(\mathbf{W}(8)\mathbf{p}_1 + \mathbf{b}(8)) = hardlim\,\left(\begin{bmatrix} -2 & 0 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

$$\mathbf{e} = \mathbf{t}_1 - \mathbf{a} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

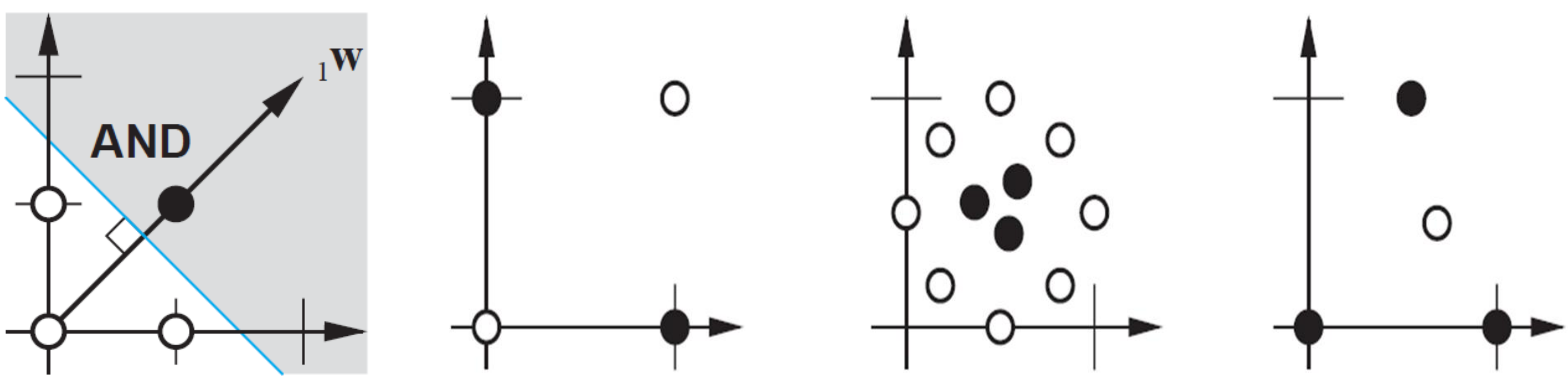
$$\mathbf{W}(9) = \mathbf{W}(8) + \mathbf{e}\mathbf{p}_1^T = \begin{bmatrix} -2 & 0 \\ 1 & -1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 0 \\ 0 & -2 \end{bmatrix},$$

$$\mathbf{b}(9) = \mathbf{b}(8) + \mathbf{e} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}.$$

- At this point the algorithm has converged.

Limitations of perceptron

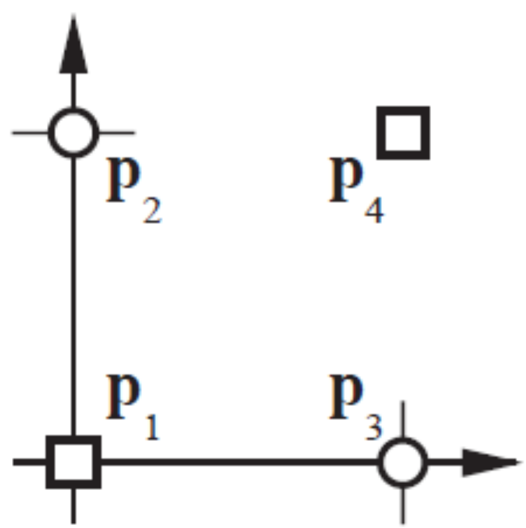
- What problems can a perceptron solve?
- The perceptron can be used to classify input vectors that can be separated by a linear boundary. We call such (linearly separable).
- The logical AND gate example is a two-dimensional example of a linearly separable problem.



Limitations of perceptron

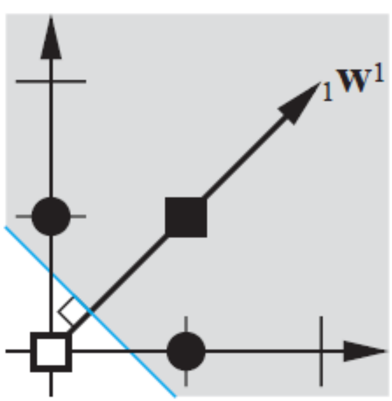
- Unfortunately, many problems are not linearly separable. The classic example is the XOR gate. The input/target pairs for the XOR gate are

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\} \left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\} \left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\} \left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 0\right\}$$

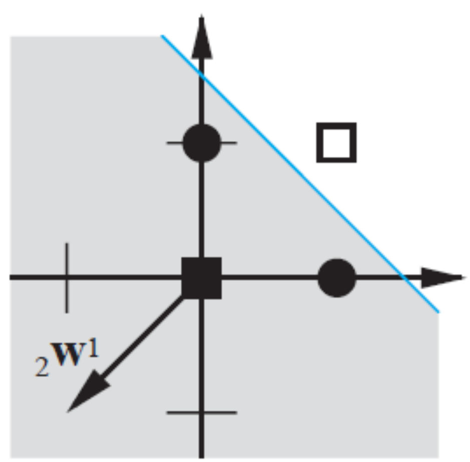


Limitations of perceptron

- A two-layer network can solve the XOR problem.
- One solution is to use two neurons in the first layer to create two decision boundaries.
- The first boundary separates p1 from the other patterns, and the second boundary separates p4.
- Then the second layer is used to combine the two boundaries together using an AND operation



Layer 1/Neuron 1



Layer 1/Neuron 2