Radial Basis Function (RBF) Neural Networks

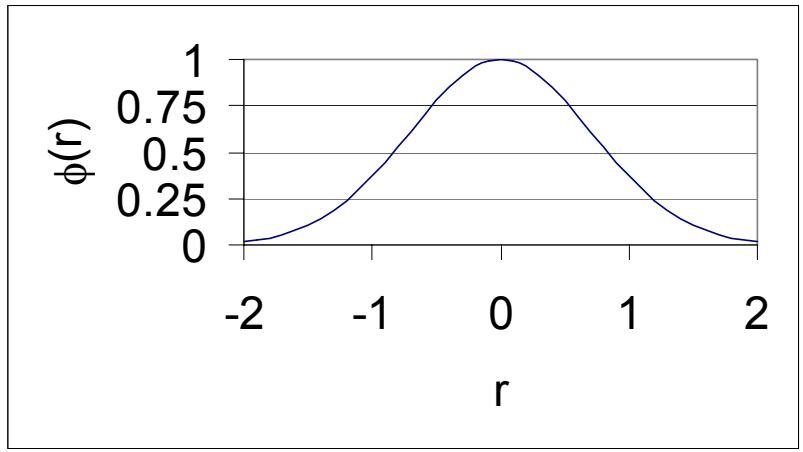
for the Senior Design Project

Radial Basis Functions

- Functions, $\phi(r)$, whose output (ϕ) depends on the distance (r) from some center point
  - Output is large ($\approx$ 1) for input points near the center (i.e., for small r)
  - Output falls off rapidly ($\rightarrow$ 0) as input points move away from the center (i.e., as r increases)
- Used to form "clusters" in pattern recognition (assuming clusters exist in the pattern)

Example: Gaussian

$$\Phi(r) = \exp\left\{\frac{-r^2}{2\sigma^2}\right\}$$



σ is called the **radius** of the basis function. (It is a measure of the width of the Gaussian pulse.)

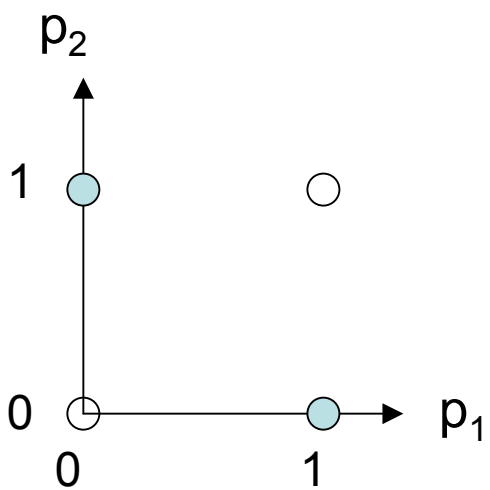
RBF Architecture

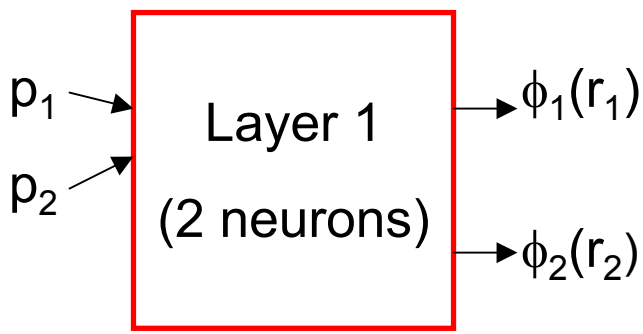
- RBF Neural Networks are 2-layer, feed-forward networks.
- The 1st layer (hidden) is **not** a traditional neural network layer.
- The function of the 1st layer is to **transform** a **non-linearly separable** set of input vectors to a **linearly separable** set.
- The second layer is then a simple feed-forward layer (e.g., of Perceptron or ADALINE type neurons) that draws the hyperplane to separate the classes.

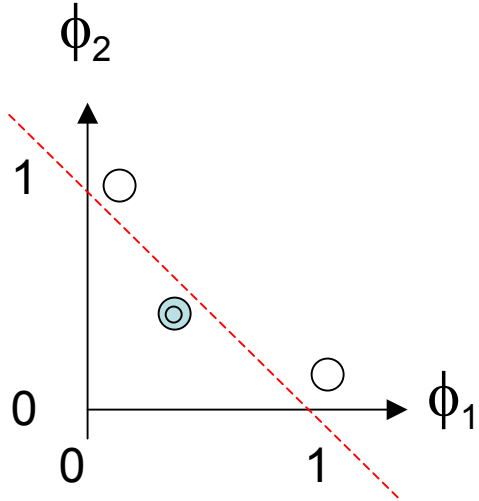
1st-Layer Functionality

- Example: X-OR Problem

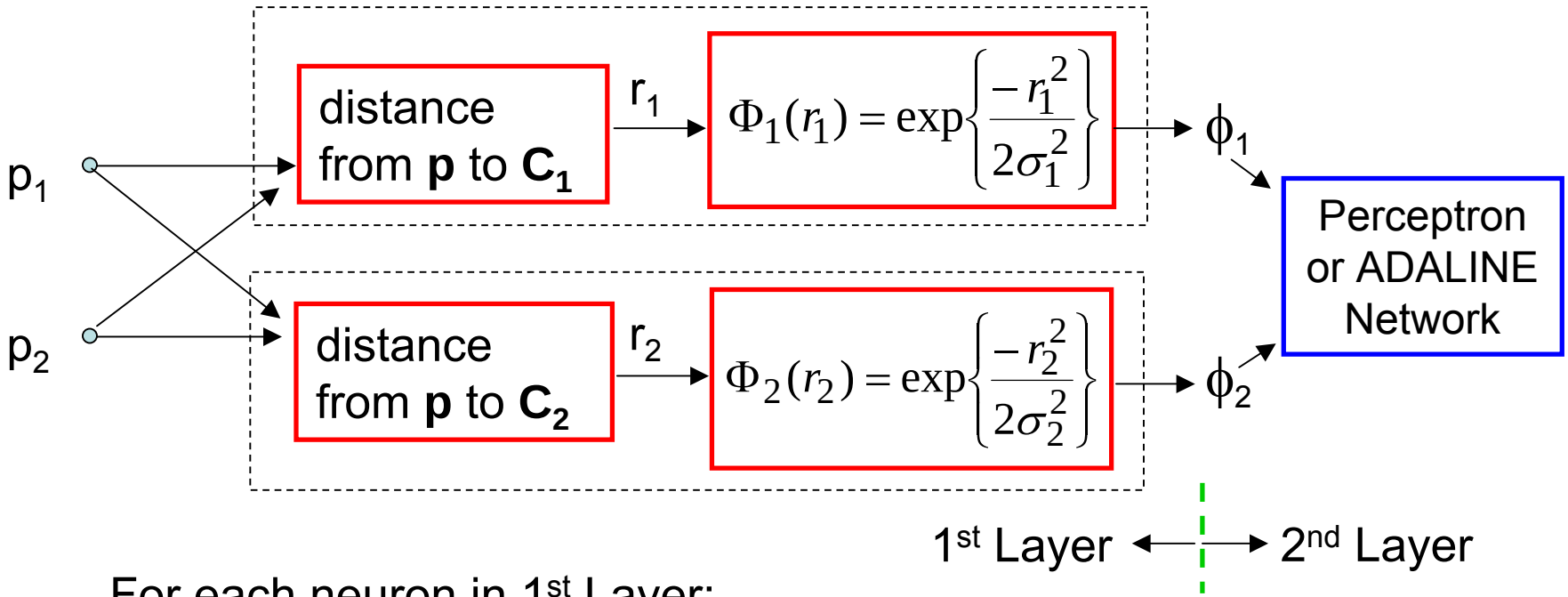
p_1	p_2	X-OR
0	0	0
0	1	1
1	0	1
1	1	0







1st Layer Architecture
Example: 2 Inputs, 2 Neurons

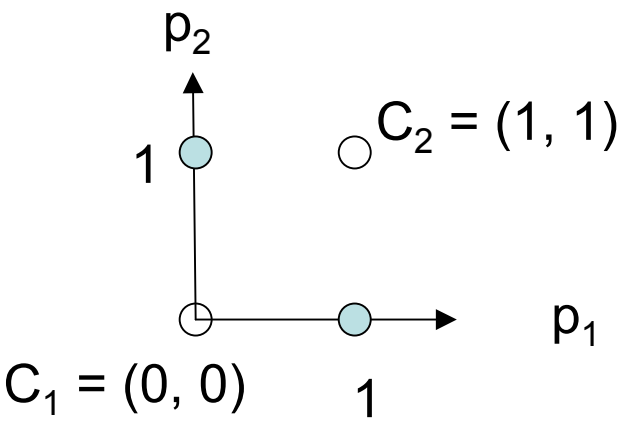


For each neuron in 1st Layer:

1. Compute the distance of the input "point" (**p**) from the center of the cluster (**C**) representing that neuron
2. Find the radial basis function ϕ as of function of the distance r between the input and the cluster center.

X-OR Example (Mechanics Only)

- **Assume** 2 neurons in Layer 1, with centers: $\mathbf{C_1} = (0, 0)$, $\mathbf{C_2} = (1,1)$



p_1	p_2	r_1^2	r_2^2	$\phi(r_1)$	$\phi(r_2)$
0	0	0	2	1	.135
0	1	1	1	.368	.368
1	0	1	1	.368	.368
1	1	2	0	.135	1

- Find r_1 for each point $\mathbf{p} = (p_1, p_2)$ (distance to $\mathbf{C_1}$)

- Find r_2 for each point $\mathbf{p} = (p_1, p_2)$ (distance to $\mathbf{C_2}$)

- **Set σ**: $2\sigma^2 = 1 \Rightarrow \sigma^2 = ½$

(In this example $\phi_1 = \phi_2 = \phi$)

- Calculate $\phi(r_1)$ and $\phi(r_2)$ using

$\phi(r) = \exp(-r^2)$

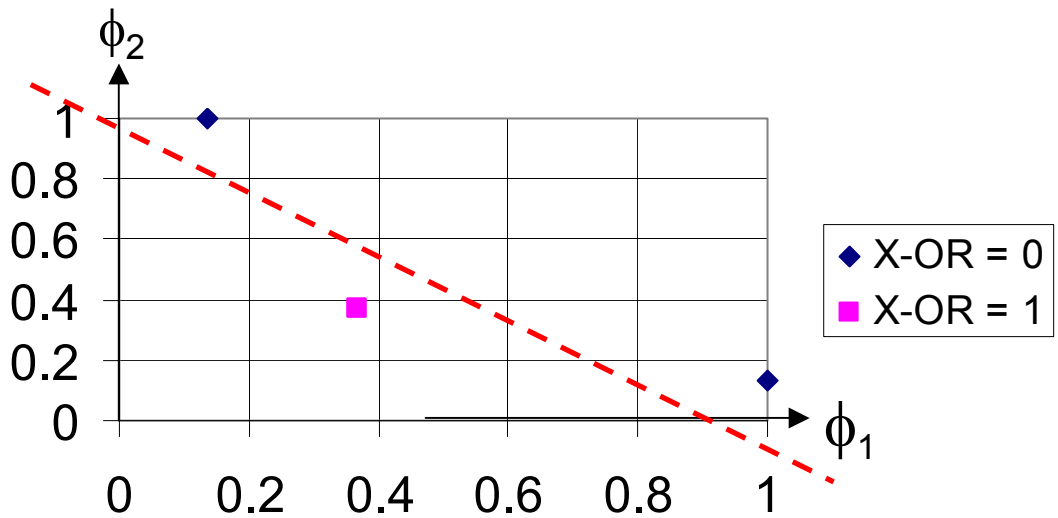
X-OR Example, Continued

- Result:

p_1	p_2	r_1^2	r_2^2	$\phi(r_1)$	$\phi(r_2)$
0	0	0	2	1	.135
0	1	1	1	.368	.368
1	0	1	1	.368	.368
1	1	2	0	.135	1

- Plot resulting points (ϕ_1, ϕ_2)

(Linearly Separable)



Problems with X-OR Example

- # of neurons in the hidden layer, the centers of the neurons, and the radius (σ) of the RBF's were assumed known
- In most pattern recognition problems, the centers for the neurons must be learned*
- Training of the RBF (i.e., finding the centers of the clusters) can be done via
  - the k-means Clustering Algorithm; or
  - the Kohonen Algorithm

  } (types of unsupervised learning)
- We will use k-means Clustering, where the number of clusters, k, is set in advanced

---

* Since, in traditional NN's, the weights are the parameters that must be learned, the centers are sometimes called the weights of RBF neurons.

k-Means Clustering Algorithm

1. Randomly choose k points from the input training data to be initial cluster centers

2. For each training data point, find the distance to each of the k cluster centers

3. Choose the closest center for each training data point, and assign the corresponding cluster number from 1 to k for the data point.

4. For each cluster i (i = 1, …, k), find the average of the assigned training data points in that cluster. Make the average the new center for the cluster.

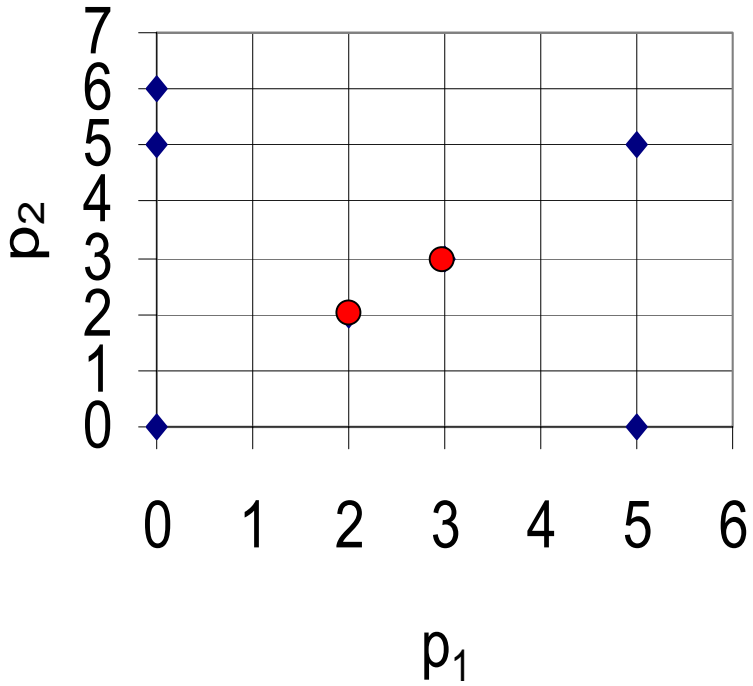
5. Repeat steps 2 – 4, until the clusters have converged (i.e., until none of the training data points are changing clusters.)

Finding the Radius (σ) for the RBF's

- Usually found with P-nearest-neighbor algorithm (often with P = 2)
- P-nearest-neighbor algorithm:

  1. For each cluster center, find the P nearest cluster centers
  2. For each neuron/cluster, Set σ = RMS distance between the cluster center and its P nearest cluster centers.

Practice Problem: k-Means Clustering, k = 5



Vectors	p_1 coord	p_2 coord
$\mathbf{p_1}$	0	5
$\mathbf{p_2}$	0	6
$\mathbf{p_3}$	3	3
$\mathbf{p_4}$	5	0
$\mathbf{p_5}$	0	0
$\mathbf{p_6}$	2	2
$\mathbf{p_7}$	5	5

Step 1: k = 5 randomly selected cluster centers: $\mathbf{C_1} = \mathbf{p_1}$, …, $\mathbf{C_5} = \mathbf{p_5}$

K-Means Clustering, Steps 2 -4 (2 Iterations)

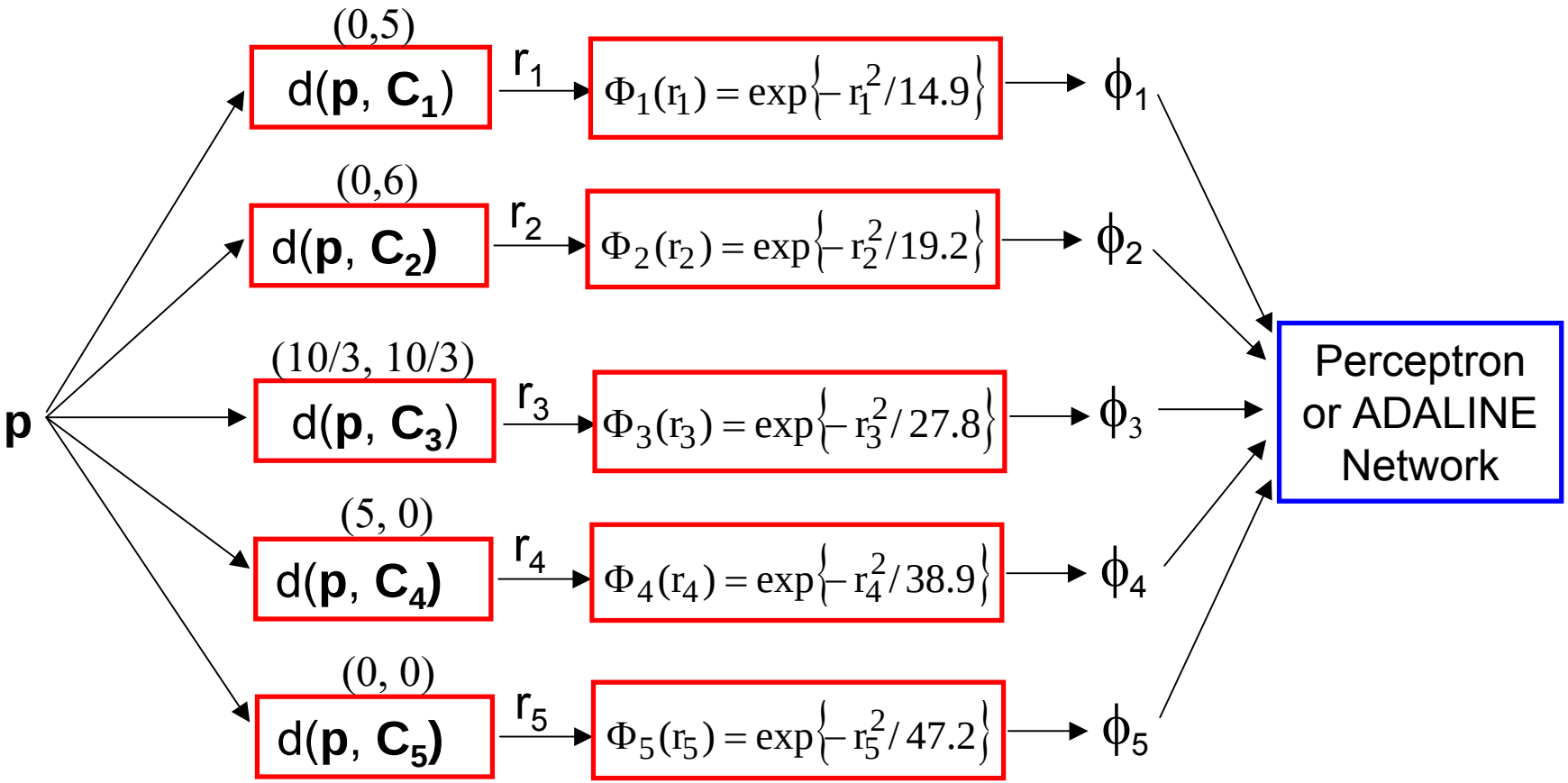
				distances							new ave.
1st Iteration	i	$C_i(x)$	$C_i(y)$	p_1	p_2	p_3 (3,3)	p_4	p_5	p_6 (2,2)	p_7 (5,5)	
	1	0	5	0	1						
	2	0	6		0						
	3	3	3			0			*	*	(10/3,10/3)
	4	5	0				0				
	5	0	0					0			
2nd Iteration	1	0	5	0	1						
	2	0	6		0						
	3	3.33	3.33			*			*	*	
	4	5	0				0				
	5	0	0					0			

Done

Finding the Radius (σ) for Each Neuron/Cluster

			distances squared					nearest	MS-dst	RMS-dst	$2\sigma^2$
i	$C_i(x)$	$C_i(y)$	C_1	C_2	C_3	C_4	C_5				
1	0	5	0	1	13.89	50	25	C2, C3	7.44	2.73	14.89
2	0	6	1	0	18.22	61	36	C1, C3	9.61	3.10	19.22
3	3.33	3.33	13.89	18.22	0	13.89	22.22	C1, C4	13.89	3.73	27.78
4	5	0	50	61	13.89	0	25	C3, C5	19.44	4.41	38.89
5	0	0	25	36	22.22	25	0	C1, C3	23.61	4.86	47.22

1st Layer of RBF Network
for the Practice Problem



Outputs from Layer 1 = Inputs for Layer 2

5-dim. Output vectors from Layer 1

i	$p_i(x)$	$p_i(y)$	r_1^2	r_2^2	r_3^2	r_4^2	r_5^2	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5
1	0	5	0	1	13.89	50	25	1	0.95	0.61	0.28	0.59
2	0	6	1	0	18.22	61	36	0.94	1	0.52	0.21	0.47
3	3	3	13	18	0.22	13	18	0.42	0.39	0.99	0.72	0.68
4	5	0	50	61	13.89	0	25	0.03	0.04	0.61	1	0.59
5	0	0	25	36	22.22	25	0	0.19	0.15	0.45	0.53	1
6	2	2	13	20	3.56	13	8	0.42	0.35	0.88	0.72	0.84
7	5	5	25	26	5.56	25	50	0.19	0.26	0.82	0.53	0.35

5-dim. Input vectors for Layer 2

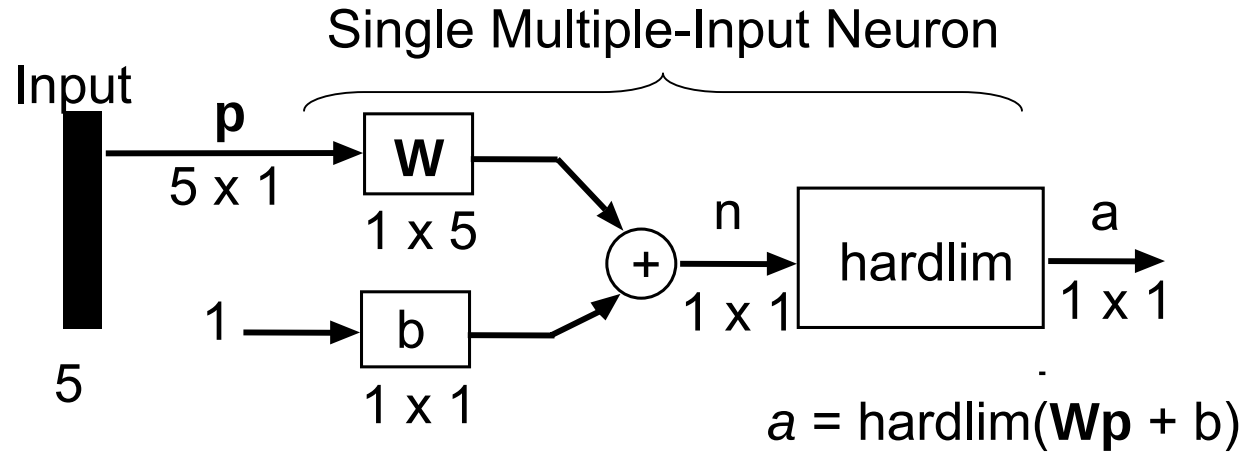
Layer 2: ADALINE or Perceptron

- Perceptron
  1. Initialize **W**, **b** with 0's
  2. Apply first input vector:
     output a = hardlim(**Wp**+**b**)
  3. Update: $\mathbf{W}_{new} = \mathbf{W}_{old} + (t-a)\mathbf{p}^t$
     $\mathbf{b}_{new} = \mathbf{b}_{old} + (t-a)$
  4. Repeat steps 2 and 3 for other input vectors, until weights & biases converge
  5. Check: a = hardlim(**Wp** + b) for all inputs

- ADALINE (ADAptive Linear Neuron)
  1. Initialize **W**, **b** with 0's
  2. Apply first input vector:
     output a = **Wp**+**b**
  3. Update: $\mathbf{W}_{new} = \mathbf{W}_{old} + (t-a)\mathbf{p}^t$
     $\mathbf{b}_{new} = \mathbf{b}_{old} + (t-a)$
  4. Repeat steps 2 and 3 for other input vectors, until weights & biases converge
  5. Find outputs: a = hardlims(**Wp** + b) for all inputs

Layer 2: Perceptron or ADALINE

Perceptron



ADALINE