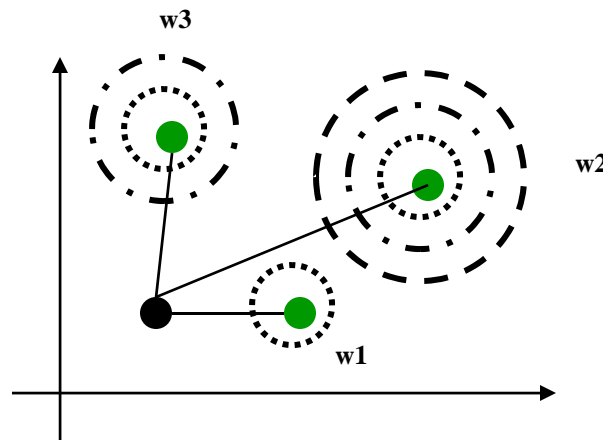


Radial-Basis Function Networks

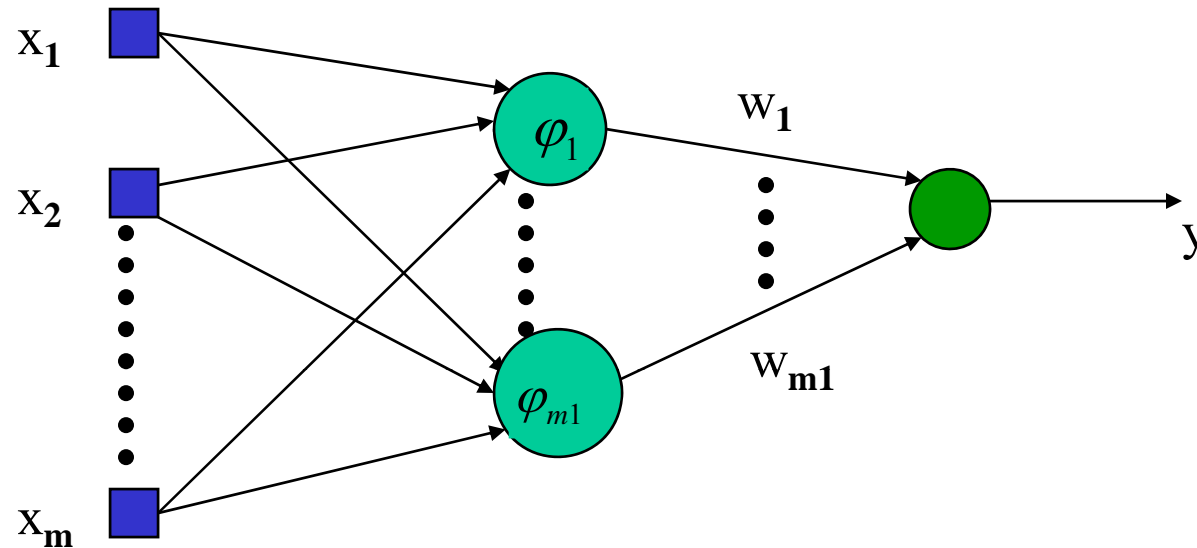
- A function is radial basis (RBF) if its output depends on (is a non-increasing function of) the distance of the input from a given stored vector.
- RBFs represent local receptors, as illustrated below, where each green point is a stored vector used in one RBF.
- In a RBF network one hidden layer uses neurons with RBF activation functions describing local receptors. Then one output node is used to combine linearly the outputs of the hidden neurons.



The output of the red vector is “interpolated” using the three green vectors, where each vector gives a contribution that depends on its weight and on its distance from the red point. In the picture we have

$$w_1 < w_3 < w_2$$

RBF ARCHITECTURE



- **One hidden layer with RBF activation functions**

$$\varphi_1 \dots \varphi_{m1}$$

- **Output layer with linear activation function.**

$$y = w_1 \varphi_1(\|x - t_1\|) + \dots + w_{m1} \varphi_{m1}(\|x - t_{m1}\|)$$

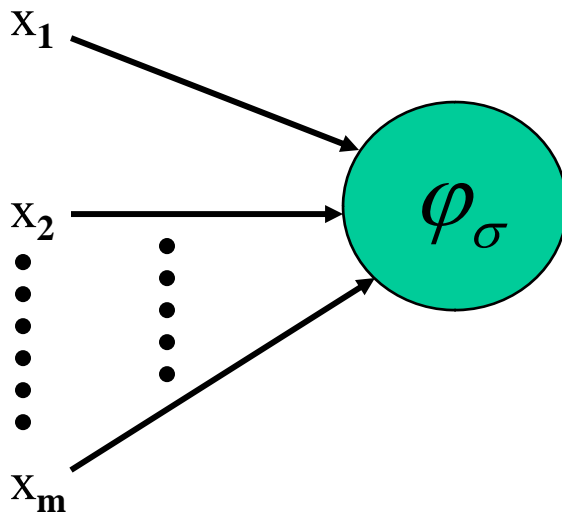
$\|x - t\|$ distance of $x = (x_1, \dots, x_m)$ from vector t

HIDDEN NEURON MODEL

- **Hidden units:** use radial basis functions

$$\varphi_{\sigma}(\|x - t\|)$$

the output depends on the distance of the input x from the center t



$$\varphi_{\sigma}(\|x - t\|)$$

t is called **center**

σ is called **spread**

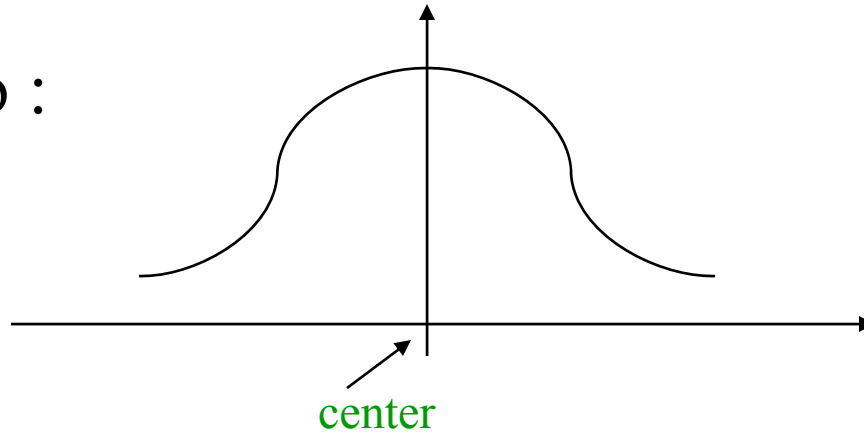
center and spread are parameters

Hidden Neurons

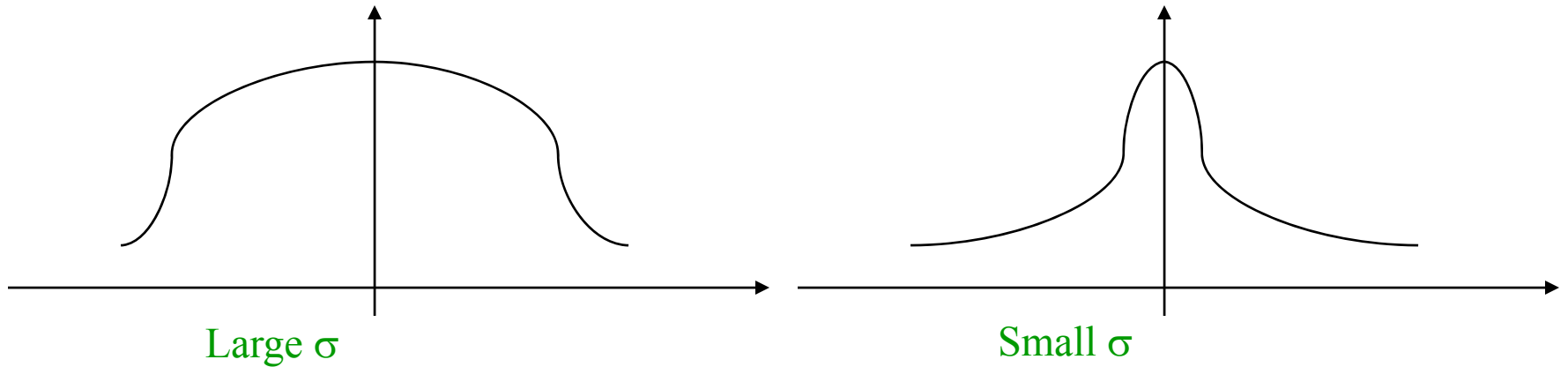
- A hidden neuron is more sensitive to data points near its center.
- For Gaussian RBF this sensitivity may be tuned by adjusting the spread σ , where a larger spread implies less sensitivity.
- **Biological example:** cochlear stereocilia cells (in our ears ...) have locally tuned frequency responses.

Gaussian RBF ϕ

ϕ :



σ is a measure of how spread the curve is:



Types of φ

- Multiquadrics:

$$\varphi(r) = (r^2 + c^2)^{1/2} \quad c > 0$$

- Inverse multiquadrics:

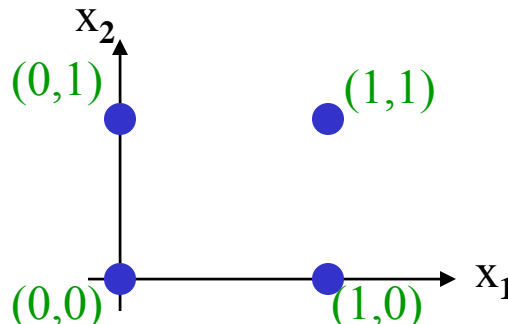

$$\varphi(r) = \frac{1}{(r^2 + c^2)^{1/2}} \quad c > 0$$

- Gaussian functions (most used):

$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad \sigma > 0$$

$$r = \|x - t\|$$

Example: the XOR problem

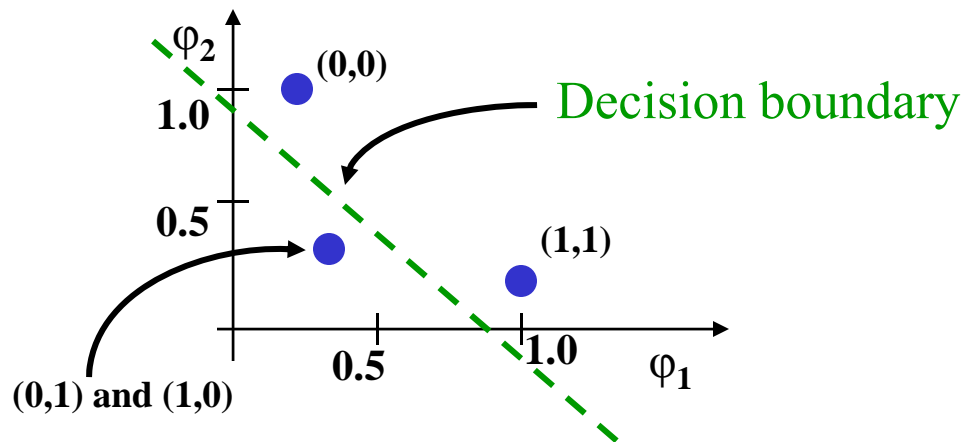
- Input space: 
- Output space: 
- Construct an RBF pattern classifier such that:
 - (0,0) and (1,1) are mapped to 0, class C1
 - (1,0) and (0,1) are mapped to 1, class C2

Example: the XOR problem

- In the feature (hidden layer) space:

$$\varphi_1(\|x - t_1\|) = e^{-\|x - t_1\|^2}$$

$$\varphi_2(\|x - t_2\|) = e^{-\|x - t_2\|^2} \quad \text{with } t_1 = (1,1) \text{ and } t_2 = (0,0)$$

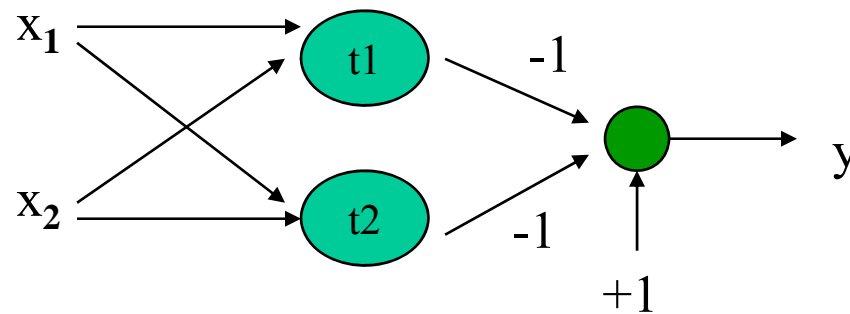


- When mapped into the feature space $\langle \varphi_1, \varphi_2 \rangle$ (hidden layer), C1 and C2 become *linearly separable*. So a linear classifier with $\varphi_1(x)$ and $\varphi_2(x)$ as inputs can be used to solve the XOR problem.

RBF NN for the XOR problem

$$\varphi_1(\|x - t_1\|) = e^{-\|x - t_1\|^2} \quad \text{with } t_1 = (1,1) \text{ and } t_2 = (0,0)$$

$$\varphi_2(\|x - t_2\|) = e^{-\|x - t_2\|^2}$$



$$y = -e^{-\|x - t_1\|^2} - e^{-\|x - t_2\|^2} + 1$$

If $y > 0$ then class 1 otherwise class 0

RBF network parameters

- **What do we have to learn for a RBF NN with a given architecture?**
 - The centers of the RBF activation functions
 - the spreads of the Gaussian RBF activation functions
 - the weights from the hidden to the output layer
- Different learning algorithms may be used for learning the RBF network parameters. We describe three possible methods for learning centers, spreads and weights.

Learning Algorithm 1

- **Centers: are selected at random**
 - **centers** are chosen randomly from the training set
- **Spreads: are chosen by normalization:**

$$\sigma = \frac{\text{Maximum distance between any 2 centers}}{\sqrt{\text{number of centers}}} = \frac{d_{\max}}{\sqrt{m_1}}$$

- Then the activation function of hidden neuron i becomes:

$$\varphi_i \left(\| \mathbf{x} - \mathbf{t}_i \|^2 \right) = \exp \left(- \frac{m_1}{d_{\max}^2} \| \mathbf{x} - \mathbf{t}_i \|^2 \right)$$

Learning Algorithm 1

- **Weights:** are computed by means of the **pseudo-inverse method**.

– For an example (x_i, d_i) consider the output of the network

$$y(x_i) = w_1 \varphi_1(\|x_i - t_1\|) + \dots + w_{m_1} \varphi_{m_1}(\|x_i - t_{m_1}\|)$$

– We would like $y(x_i) = d_i$ for each example, that is

$$w_1 \varphi_1(\|x_i - t_1\|) + \dots + w_{m_1} \varphi_{m_1}(\|x_i - t_{m_1}\|) = d_i$$

Learning Algorithm 1

- This can be re-written in matrix form for one example

$$[\varphi_1(\|x_i - t_1\|) \dots \varphi_{m_1}(\|x_i - t_{m_1}\|)] [w_1 \dots w_{m_1}]^T = d_i$$

and

$$\begin{bmatrix} \varphi_1(\|x_1 - t_1\|) \dots \varphi_{m_1}(\|x_1 - t_{m_1}\|) \\ \dots \\ \varphi_1(\|x_N - t_1\|) \dots \varphi_{m_1}(\|x_N - t_{m_1}\|) \end{bmatrix} [w_1 \dots w_{m_1}]^T = [d_1 \dots d_N]^T$$

for all the examples at the same time

Learning Algorithm 1

let

$$\Phi = \begin{bmatrix} \varphi_1(\|x_1 - t_1\|) & \dots & \varphi_{m_1}(\|x_N - t_{m_1}\|) \\ \dots & \dots & \dots \\ \varphi_1(\|x_N - t_1\|) & \dots & \varphi_{m_1}(\|x_N - t_{m_1}\|) \end{bmatrix}$$

then we can write

$$\Phi \begin{bmatrix} w_1 \\ \dots \\ w_{m_1} \end{bmatrix} = \begin{bmatrix} d_1 \\ \dots \\ d_N \end{bmatrix}$$

If Φ^+ is the pseudo-inverse of the matrix Φ
obtain the weights using the following formula

$$[w_1 \dots w_{m_1}]^T = \Phi^+ [d_1 \dots d_N]^T$$

Learning Algorithm 1: summary

1. Choose the centers randomly from the training set.
2. Compute the spread for the RBF function using the normalization method.
3. Find the weights using the pseudo-inverse method.

Learning Algorithm 2: Centers

- **clustering algorithm for finding the centers**

1 **Initialization**: $t_k(0)$ random $k = 1, \dots, m_1$

2 **Sampling**: draw x from input space

3 **Similarity matching**: find index of center closer to x

$$k(x) = \arg \min_k \|x(n) - t_k(n)\|$$

4 **Updating**: adjust centers

$$t_k(n+1) = \begin{cases} t_k(n) + \eta[x(n) - t_k(n)] & \text{if } k = k(x) \\ t_k(n) & \text{otherwise} \end{cases}$$

5 **Continuation**: increment n by 1, goto 2 and continue until no noticeable changes of centers occur

Learning Algorithm 2: summary

- *Hybrid Learning Process:*
 - **Clustering** for finding the **centers**.
 - **Spreads** chosen by normalization.
 - **LMS algorithm (see Adaline)** for finding the **weights**.

Learning Algorithm 3

- Apply the gradient descent method for finding centers, spread and weights, by minimizing the (instantaneous) squared error

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

- Update for:

centers

$$\Delta t_j = -\eta_{t_j} \frac{\partial E}{\partial t_j}$$

spread

$$\Delta \sigma_j = -\eta_{\sigma_j} \frac{\partial E}{\partial \sigma_j}$$

weights

$$\Delta w_{ij} = -\eta_{ij} \frac{\partial E}{\partial w_{ij}}$$

Comparison with FF NN

RBF-Networks are used for regression and for performing complex (non-linear) pattern classification tasks.

Comparison between **RBF networks** and **FFNN**:

- Both are examples of *non-linear layered feed-forward* networks.
- Both are *universal approximators*.

Comparison with multilayer NN

- Architecture:
 - RBF networks have one *single* hidden layer.
 - FFNN networks may have *more* hidden layers.
- Neuron Model:
 - In RBF the neuron model of the hidden neurons is *different* from the one of the output nodes.
 - Typically in FFNN hidden and output neurons share a *common neuron model*.
 - The hidden layer of RBF is *non-linear*, the output layer of RBF is *linear*.
 - Hidden and output layers of FFNN are usually *non-linear*.

Comparison with multilayer NN

- Activation functions:
 - The argument of activation function of each hidden neuron in a RBF NN computes the *Euclidean distance* between input vector and the center of that unit.
 - The argument of the activation function of each hidden neuron in a FFNN computes the *inner product* of input vector and the synaptic weight vector of that neuron.
- Approximation:
 - RBF NN using Gaussian functions construct *local* approximations to non-linear I/O mapping.
 - FF NN construct *global* approximations to non-linear I/O mapping.

Application: FACE RECOGNITION

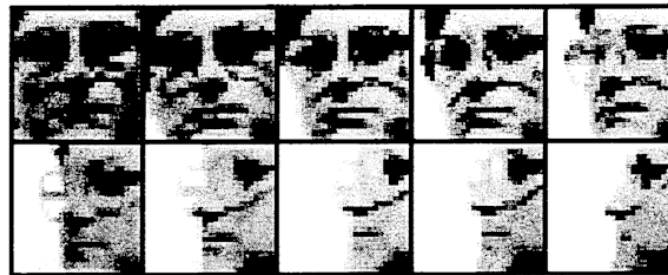
- The problem:
 - Face recognition of persons of a known group in an indoor environment.
- The approach:
 - Learn face classes over a wide range of poses using an RBF network.

Dataset

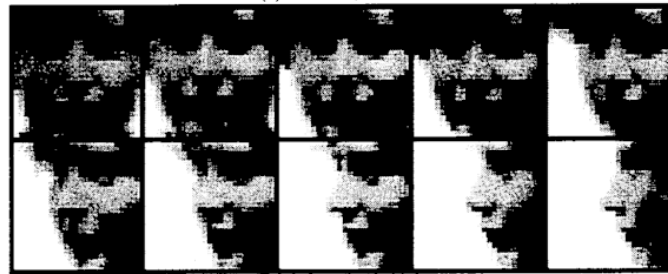
- **database**
 - **100 images of 10 people** (8-bit grayscale, resolution 384 x 287)
 - for each individual, 10 images of head in different pose **from face-on to profile**
 - Designed to assess performance of **face recognition techniques** when pose variations occur

Datasets

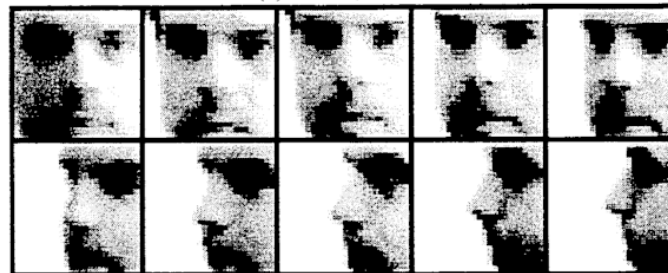
All ten images for classes 0-3 from the Sussex database, nose-centred and subsampled to 25x25 before preprocessing



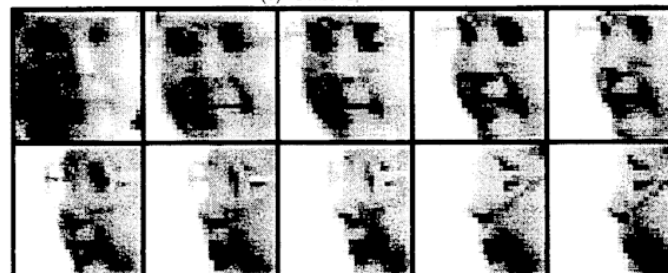
(a) Class 0, 25x25



(b) Class 1, 25x25



(c) Class 2, 25x25



(d) Class 3, 25x25

Approach: Face unit RBF

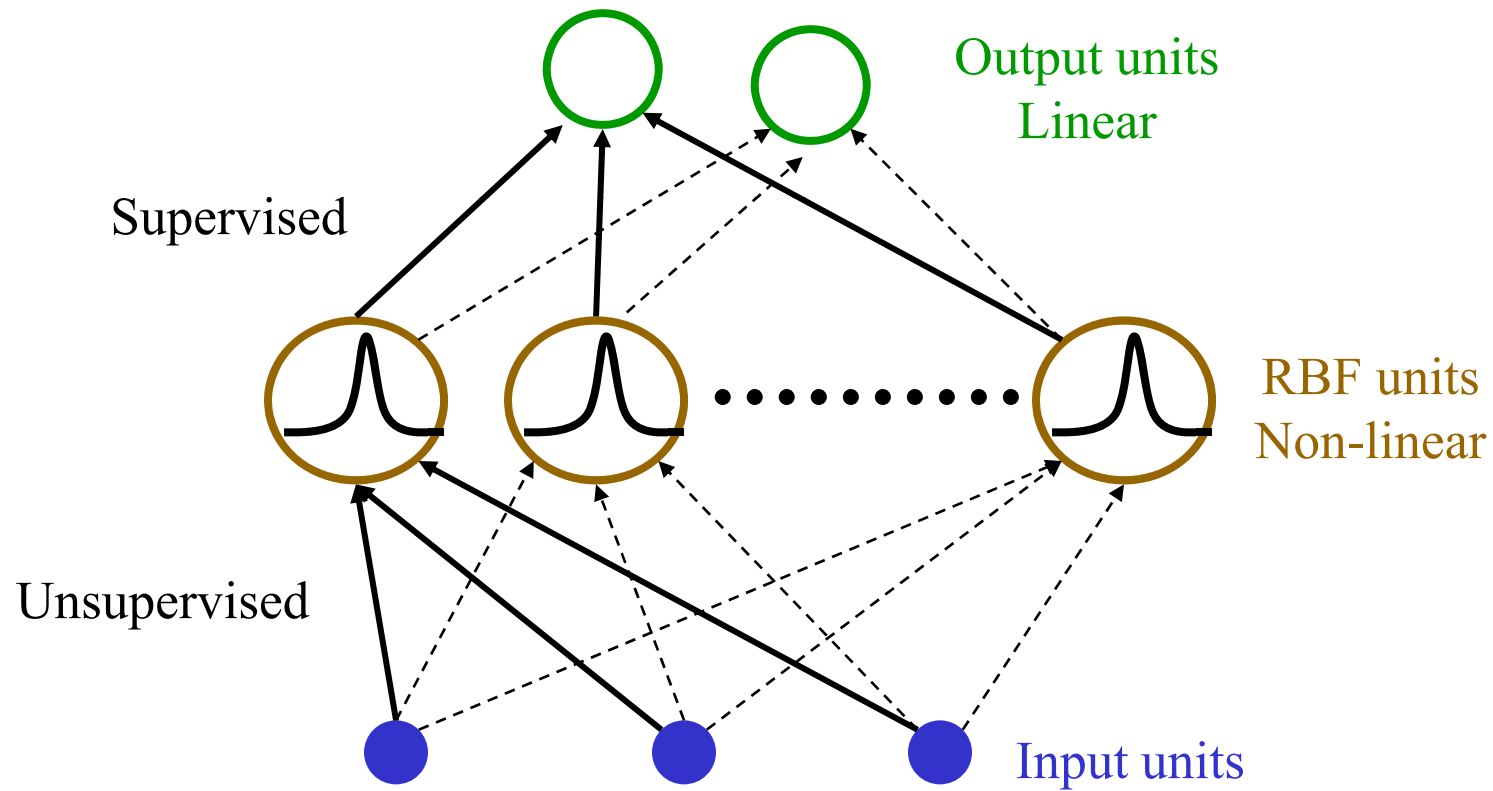
- A **face recognition** unit RBF neural networks is trained to recognize a single person.
- Training uses examples of images of the person to be recognized as positive evidence, together with selected confusable images of other people as negative evidence.

Network Architecture

- Input layer contains $25*25$ inputs which represent the pixel intensities (normalized) of an image.
- Hidden layer contains $p+a$ neurons:
 - p hidden pro neurons (receptors for positive evidence)
 - a hidden anti neurons (receptors for negative evidence)
- Output layer contains two neurons:
 - One for the particular person.
 - One for all the others.

The output is discarded if the absolute difference of the two output neurons is smaller than a parameter R .

RBF Architecture for one face recognition



Hidden Layer

- Hidden nodes can be:
 - Pro neurons: Evidence for that person.
 - Anti neurons: Negative evidence.
- The number of pro neurons is equal to the positive examples of the training set. For each pro neuron there is either one or two anti neurons.
- Hidden neuron model: Gaussian RBF function.

Training and Testing

- **Centers:**

- of a pro neuron: the corresponding positive example
- of an anti neuron: the negative example which is most similar to the corresponding pro neuron, with respect to the Euclidean distance.

- **Spread:** average distance of the center from all other centers. So the spread σ_n of a hidden neuron n is

$$\sigma_n = \frac{1}{H\sqrt{2}} \sum_h \|t^n - t^h\|$$

where H is the number of hidden neurons and t^i is the center of neuron i.

- **Weightsⁱ:** determined using the pseudo-inverse method.
- A RBF network with 6 pro neurons, 12 anti neurons, and R equal to 0.3, discarded 23 per cent of the images of the test set and classified correctly 96 per cent of the non discarded images.