UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
Departamento de Ciências de Computação

SCC-5809 - Capítulo 6 Redes de Função de Base Radial

João Luís Garcia Rosa[1]

[1]SCC-ICMC-USP - joaoluis@icmc.usp.br

2012

Sumário

Sumário

Redes RBF: abordagem diferente

- O algoritmo BP pode ser visto como uma aplicação de uma técnica recursiva conhecida em estatística como *aproximação estocástica*.
- Neste capítulo, uma abordagem completamente diferente vê o projeto da RNA como um *problema de ajuste (aproximação) à curva* em um espaço de alta dimensão.
- Aprender é equivalente a achar uma superfície de um espaço multidimensional que proveja um melhor ajuste aos dados de treinamento.
- Generalização é equivalente ao uso dessa superfície multidimensional para interpolar os dados de teste.
- As unidades escondidas provêem um conjunto de "funções" que constitui uma "base" arbitrária para os padrões (vetores) de entrada quando são expandidos no espaço escondido: *funções de base radial*.

Rede de função de base radial

- Uma *rede de função de base radial (RBF)* tem três camadas com papéis diferentes:
  1. A camada de entrada contém nós fonte que conectam a rede ao seu ambiente.
  2. A segunda camada, a *única* camada escondida, aplica uma transformação não-linear do espaço de entrada para o espaço escondido (alta dimensionalidade).
  3. A camada de saída é linear, fornecendo a resposta da rede ao padrão (sinal) de ativação aplicado na entrada.
- De acordo com Cover [3], um problema de classificação de padrões que "cai" num espaço de alta dimensão é mais provável ser linearmente separável do que em espaço de baixa dimensão (*teorema de Cover da separabilidade de padrões*): razão porquê a dimensão do espaço escondido de uma rede RBF ser alta.

Sumário

Teorema de Cover

- Quando uma rede RBF é usada para realizar uma tarefa de classificação de padrões complexos: o problema deve ser transformado um um espaço de alta dimensão de uma maneira não-linear.
- Justificativa está no teorema de Cover da separabilidade de padrões.
- Quando se tem padrões linearmente separáveis, o problema de classificação torna-se relativamente simples.
- Considere uma família de superfícies onde cada uma divide um espaço de entrada em duas regiões.
- Seja $\mathscr{X}$ um conjunto de N padrões (vetores) $\mathbf{x}_1$, $\mathbf{x}_2$, ..., $\mathbf{x}_N$, cada um atribuído a uma de duas classes $\mathscr{X}_1$ e $\mathscr{X}_2$.
- Essa *dicotomia* (partição binária) dos pontos é separável com respeito à família de superfícies se uma superfície existe na família que separa os pontos na classe $\mathscr{X}_1$ dos pontos na classe $\mathscr{X}_2$.

Teorema de Cover

- Para cada padrão $\mathbf{x} \in \mathscr{X}$, defina um vetor construído de um conjunto de funções de valor real $\{\varphi_i(\mathbf{x}) \mid i = 1, 2, ..., m_1\}$, como mostrado por

$$\Phi(\mathbf{x}) = [\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), ..., \varphi_{m_1}(\mathbf{x})]^T \quad (1)$$

- Suponha que o padrão $\mathbf{x}$ seja um vetor em um espaço de entrada m_0-dimensional.
- O vetor $\Phi(\mathbf{x})$ mapeia pontos de um espaço de entrada m_0-dimensional em um novo espaço de dimensão m_1.
- Refere-se a $\varphi_i(\mathbf{x})$ como uma *função escondida*, pois desempenha papel semelhante a uma unidade escondida em uma RNA *feedforward*.
- *Espaço escondido* ou *espaço de características*: espaço coberto pelo conjunto de funções escondidas $\{\varphi_i(\mathbf{x})\}_{i=1}^{m_1}$.

Teorema de Cover

- Uma dicotomia $\{\mathscr{X}_1, \mathscr{X}_2\}$ de $\mathscr{X}$ é φ-*separável* se existe um vetor m_1-dimensional $\mathbf{w}$ tal que pode-se escrever:

$$\mathbf{w}^T \Phi(\mathbf{x}) > 0, \mathbf{x} \in \mathscr{X}_1 \tag{2}$$

$$\mathbf{w}^T \Phi(\mathbf{x}) < 0, \mathbf{x} \in \mathscr{X}_2 \tag{3}$$

- O hiperplano definido pela equação

$$\mathbf{w}^T \Phi(\mathbf{x}) = 0 \tag{4}$$

  descreve a superfície de separação no espaço φ (isto é, espaço escondido).
- O teorema de Cover diz (em outras palavras) que a probabilidade de separação linear aumenta, a medida que m_1 aumenta.

Sumário

O problema do XOR

- Para ilustrar a significância da ideia da separabilidade de padrões φ: considere o problema XOR.
- Construção de um classificador de padrões que produza 0 para entradas (1,1) ou (0,0) e 1 para (0,1) e (1,0).
- Pontos pertos no espaço de entrada (distância de Hamming) mapeiam para regiões maximamente separadas no espaço de saída.
- Escolhendo funções escondidas gaussianas:

$$\varphi_1(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{t}_1\|^2}, \mathbf{t}_1 = [1,1]^T \tag{5}$$

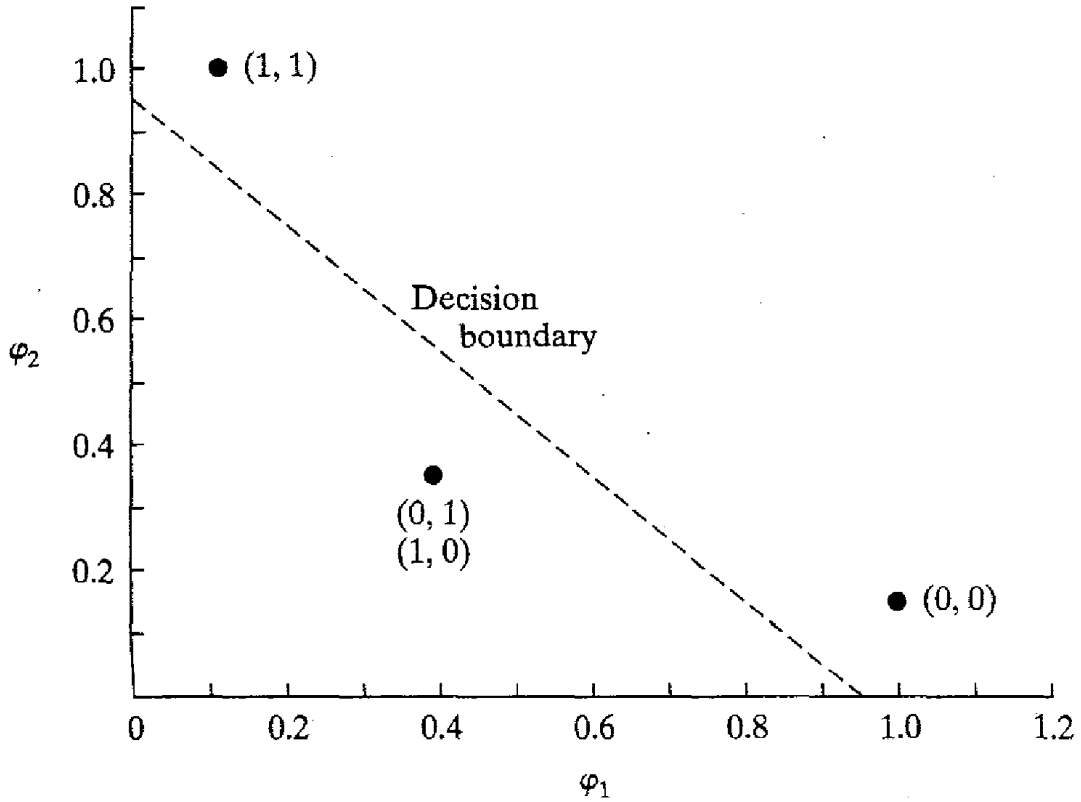
$$\varphi_2(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{t}_2\|^2}, \mathbf{t}_2 = [0,0]^T \tag{6}$$

$\mathbf{x}$	$\varphi_1(\mathbf{x})$	$\varphi_2(\mathbf{x})$
(1,1)	1	0.1353
(0,1)	0.3678	0.3678
(0,0)	0.1353	1
(1,0)	0.3678	0.3678

Notas: $\mathbf{x}$ = padrão de entrada, $\varphi_1(\mathbf{x})$ = Primeira Função Escondida, $\varphi_2(\mathbf{x})$ = Segunda Função Escondida.

O problema do XOR

Figure 1 : Diagrama de decisão do problema XOR [4].



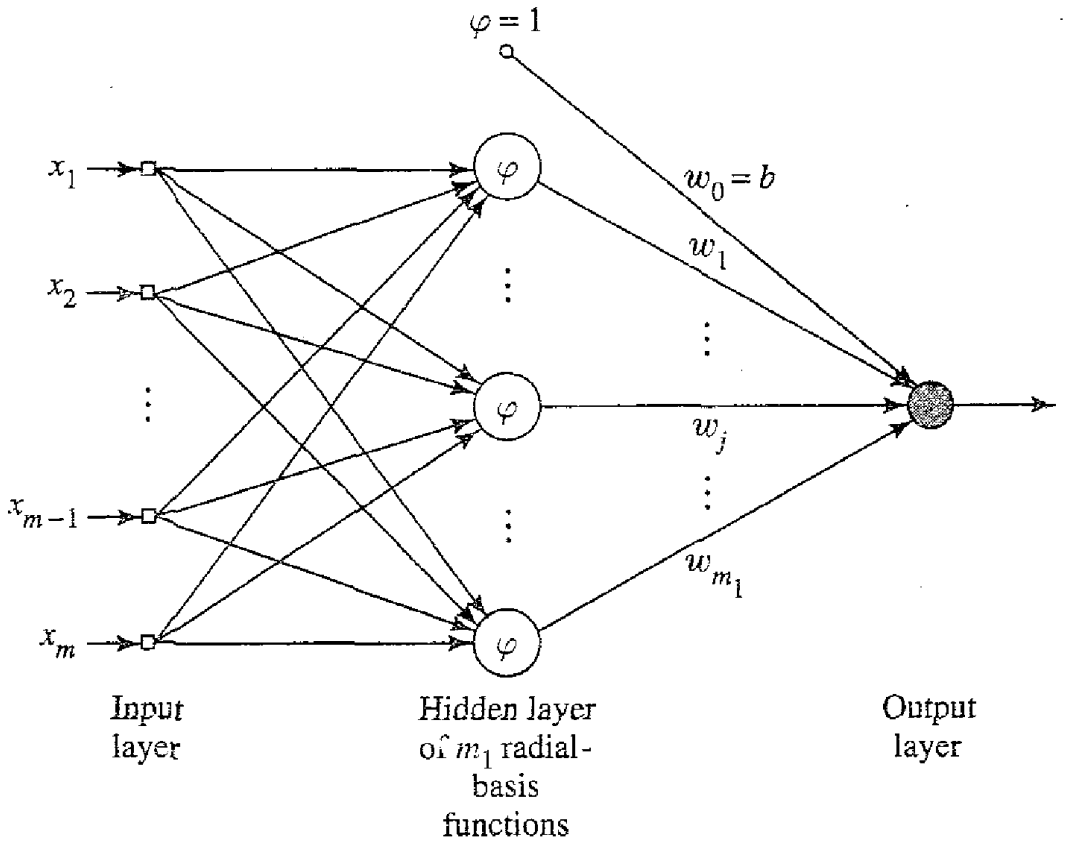
Sumário

RBF

- A maioria das RNAs multicamadas computa uma função não-linear do produto escalar dos vetores de entrada e de pesos.
- Nas redes de Função de Base Radial (RBF), a função de ativação de cada neurônio da camada escondida é função da distância entre seus vetores de peso e de entrada.
- Evolução das redes MLP.
- Redes de duas camadas:
  1. Primeira camada: Utiliza funções de ativação não lineares (funções de base radial).
  2. Segunda camada: Utiliza funções de ativação lineares.

RBF

Figure 2 : Rede de função de base radial [4].



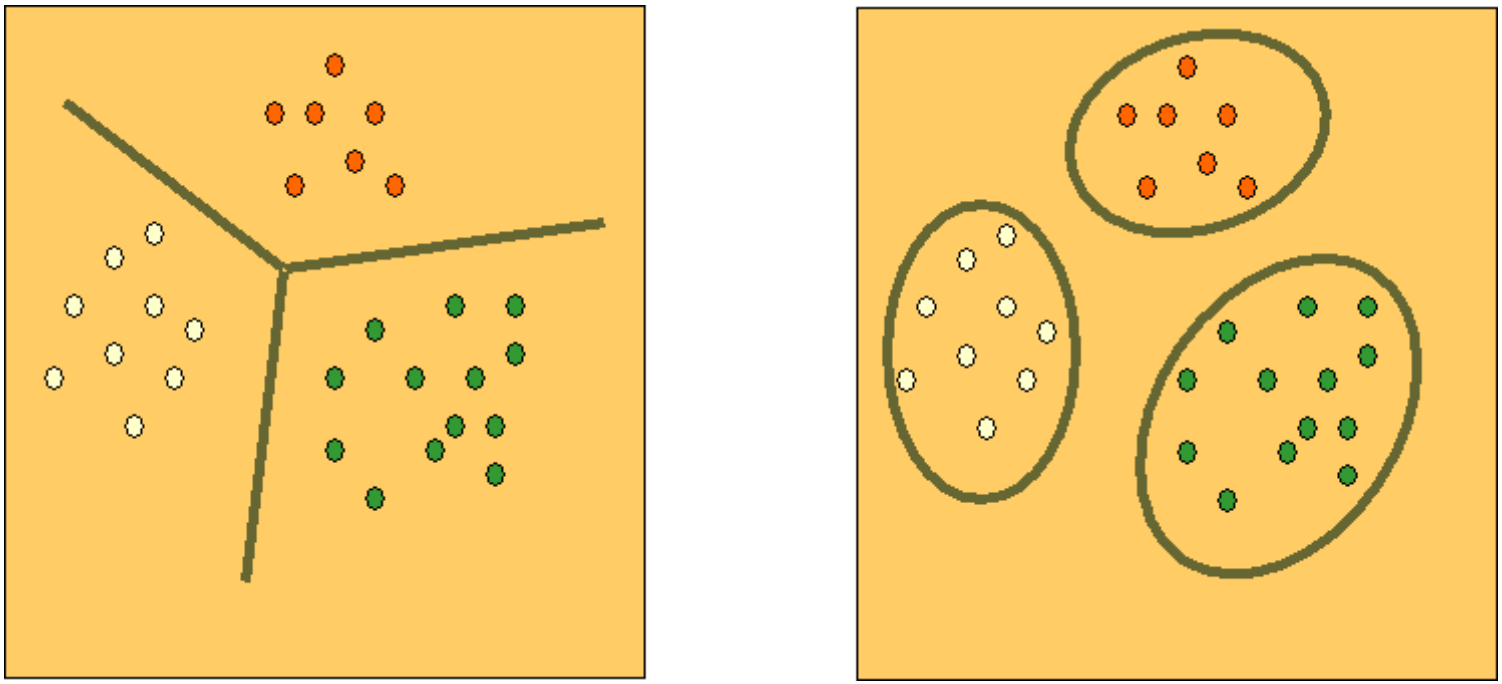
Sumário

Redes RBF × MLP

- Redes RBF e MLP são exemplos de redes com camadas *feedforward* não-lineares.
- Ambas são aproximadores universais.
- MLP utiliza hiperplanos para particionar espaço de entradas (camada escondida):
  - Definidos por funções da forma $f(\sum w_i x_i)$.
- Rede RBF utiliza hiper-elipsóides para particionar o espaço de entradas (camada escondida):
  - Definidos por funções da forma $\Phi(\| x_i - t_i \|)$,
  - Distância do vetor de entrada ao centro de um *cluster*.

Fronteiras de decisão

Figure 3 : MLP a esquerda, RBF a direita [2, 5].



Redes RBF × MLP

- Apesar das semelhanças, há diferenças importantes:
  1. Uma rede RBF (na maioria dos casos) tem uma única camada escondida.
  2. Os nós de computação num MLP, localizados nas camadas escondidas e de saída, compartilham um modelo neuronal comum: na rede RBF os nós de computação da camada escondida são diferentes e têm um propósito diferente dos de saída.
  3. A camada escondida de uma rede RBF é não-linear e a saída é linear. No MLP, camadas escondidas e de saída são usualmente não-lineares.
  4. O argumento da função de ativação de cada unidade escondida numa rede RBF computa a *norma (distância) euclidiana* entre o vetor de entrada e o centro da unidade. No caso do MLP, computa-se a *produto interno* do vetor de entrada e do vetor de pesos sinápticos da unidade.
  5. MLP constróem aproximações *globais* ao mapeamento entrada-saída não-linear. Redes RBF que usam não-linearidades localizadas exponencialmente decrescentes (por exemplo, funções gaussianas) contróem aproximações *locais*.

Redes RBF × MLP

- Para a aproximação de um mapeamento entrada-saída não-linear, o MLP pode requerer um número menor de parâmetros que a rede RBF para o mesmo grau de acurácia.
- As características lineares da rede RBF pressupõem que tal rede está mais próxima do perceptron de Rosenblatt do que do MLP.
- Entretanto, a rede RBF difere do perceptron no que diz respeito a ser capaz de implementar transformações não-lineares arbitrárias do espaço de entrada.
- Isso é ilustrado pelo problema do XOR, que não pode ser resolvido por nenhum perceptron linear mas pode ser resolvido pela rede RBF.

Sumário

RBF

- Cada neurônio da camada escondida computa uma função base radial
  - Centro
    - Protótipo para um *cluster*.
  - Largura
    - Área de influência de um *cluster*.
- Podem ser bem mais rápidas que redes MLP.
- Entrada total

$$u = \| x_i - t_i \| \ (camada\ escondida) \tag{7}$$

$$u = \sum w_i \varphi_i \| x_i - t_i \| \ (camada\ de\ saida) \tag{8}$$

- Medida de distância: geralmente é utilizada a distância Euclidiana

$$\| x_i - t_i \| = \sqrt{\sum_{i=1}^{N} (x_i - t_i)^2} \tag{9}$$

RBF

- Processamento à frente (*forward*).
- Estados de ativação:
  - 1 (+1) = ativo.
  - 0 (-1) = inativo.
- Função da primeira camada é transformar conjunto de dados não-linearmente separáveis em linearmente separáveis.
- Função de ativação das unidades escondidas:
  - Não linear.
  - Valor aumenta ou diminui com relação à distância a um ponto central.

Funções de ativação

- Funções de ativação típicas:
  - Função Gaussiana

$$\varphi(v) = e^{(-\frac{v^2}{2\sigma^2})} \tag{10}$$

onde $v = \| \mathbf{x} - \mathbf{t} \|$
x: vetor de entrada,
t: centro da função radial,
σ: largura da função radial.

  - Função Multiquadrática

$$\varphi(v) = \sqrt{v^2 + \sigma^2} \tag{11}$$

  - Função *Thin-Plate-Spline*

$$\varphi(v) = v^2 \log(v) \tag{12}$$

Funções de ativação

- Escolha da função depende de:
  - Nível de conhecimento sobre os dados
    - Funções devem cobrir pontos uniformemente distribuídos do espaço de entradas.
  - Conhecimento da estrutura geral das entradas
    - Levar a estrutura em consideração na escolha das funções.
- Função de saída
  - Função de identidade $y_i = a_i$
- Treinamento
  - Geralmente dois estágios
    - Primeiro estágio: camada escondida - **não supervisionado**.
    - Segundo estágio: camada de saída - **supervisionado**.

Primeiro Estágio

- Determina parâmetros das funções de base radial:
  - Número de bases,
  - Centros das bases,
  - Larguras das bases.
- Número de funções base:
  - Geralmente definido por tentativa e erro.
  - Sejam M o número de funções base e N o número de vetores de entrada: $M << N$ (M = N pode levar a *overfitting*).
  - Deve ser determinado pela complexidade dos dados.
  - Número de funções base radial = número de *clusters*.
- Definições de centros:
  - Existem várias abordagens:
    - Seleção aleatória dos centros,
    - *Clustering*: *K-means*, SOM, Algoritmos Genéticos.

Definição dos centros

- Seleção aleatória de centros (**t**):
  - Seleciona centros aleatoriamente a partir dos padrões de treinamento,
  - Assume que padrões estão distribuídos de uma maneira representativa para o problema.
    - Padrões são capazes de capturar as principais características do problema.
  - Problema: distribuição dos dados pode não ser representativa.
- *K-means clustering*:
  - Centros são colocados no meio de agrupamentos de vetores de entrada (*clusters*).
    - Utiliza aprendizado não-supervisionado.
    - Divide os vetores de entrada em K conjuntos disjuntos S_j: cada conjunto S_j tem N_j vetores.
  - Objetivo: minimizar distâncias entre vetores de S_j e seu centro.

Definição das larguras

- Heurísticas para definir larguras (raios) das funções radiais (σ):
  1. todas as larguras iguais à média sobre todas as distâncias Euclidianas entre o centro de cada unidade N_i e o centro da unidade N_j mais próxima

$$\sigma = \frac{1}{m} \sum_{i=1}^{m} \| t_i - t_j \| \tag{13}$$

     t_i é o centro mais próximo de t_j.
  2. atribuir a cada unidade uma largura diferente baseada na distância do seu centro ao centro da unidade mais próxima

$$\sigma_i = \alpha \| t_i - t_j \| \tag{14}$$

     t_i é o centro mais próximo de t_j e $1.0 < \alpha < 1.5$.
  3. atribuir a cada σ_j a distância média de seu centro aos N vetores de entrada mais próximos.
  4. atribuir a σ_j um valor constante (geralmente 1).

Segundo Estágio

- Determina pesos da camada de saída:
  - Recebe vetores linearmente separáveis,
  - Supervisionado,
  - Classificação/regressão dos vetores de entrada.
- Métodos para ajustar pesos:
  - Decomposição em valores singulares,
  - Regra delta.

Sumário

XOR revisitado

- Solução do problema do XOR usando uma rede RBF.
- Par de funções gaussianas:

$$G(\| \mathbf{x} - \mathbf{t}_i \| = \exp(- \| \mathbf{x} - \mathbf{t}_i \|^2), i = 1, 2 \qquad (15)$$

onde os centros $\mathbf{t}_1$ e $\mathbf{t}_2$ são

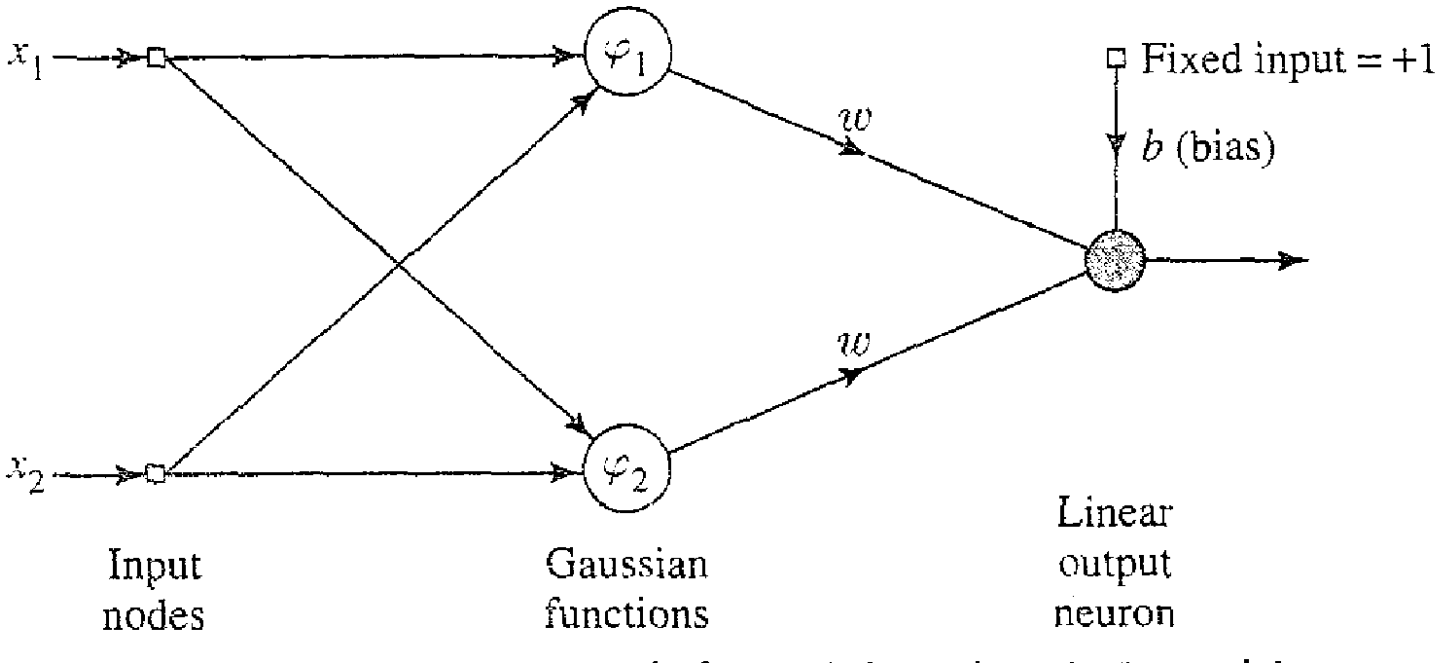
$$\mathbf{t}_1 = [1, 1]^T \qquad (16)$$

$$\mathbf{t}_2 = [0, 0]^T \qquad (17)$$

- Para a caracterização da unidade de saída, assume-se que
  1. A unidade de saída usa *compartilhamento de pesos*, justificado pela simetria do problema. Com duas unidades escondidas, há apenas um peso w.
  2. A unidade de saída inclui um *bias* b.

XOR revisitado

Figure 4 : Rede RBF para resolver o problema XOR [4].



XOR revisitado

- A relação entrada-saída da rede é definida por

$$y(\mathbf{x}) = \sum_{i=1}^{2} wG(\| \mathbf{x} - \mathbf{t}_i \|) + b \tag{18}$$

- Para ajustar os dados de treinamento da tabela 33 é necessário que

$$y(\mathbf{x}_j) = d_j, j = 1, 2, 3, 4 \tag{19}$$

onde $\mathbf{x}_j$ é um vetor de entrada e d_j é o valor correspondente da saída desejada.

j	$\mathbf{x}_j$	d_j
1	(1,1)	0
2	(0,1)	1
3	(0,0)	0
4	(1,0)	1

Notas: j = ponto de dados, $\mathbf{x}_j$ = padrão de entrada, d_j = Saída desejada.

XOR revisitado

- Seja

$$g_{ji} = G(\| \mathbf{x}_j - \mathbf{t}_i \|), j = 1, 2, 3, 4; i = 1, 2 \tag{20}$$

- Usando os valores da tabela 33 na equação 20, tem-se o seguinte conjunto de equações na forma de matriz:

$$\mathbf{Gw} = \mathbf{d} \tag{21}$$

onde

$$\mathbf{G} = \begin{bmatrix} 1 & 0.1353 & 1 \\ 0.3678 & 0.3678 & 1 \\ 0.1353 & 1 & 1 \\ 0.3678 & 0.3678 & 1 \end{bmatrix} \tag{22}$$

$$\mathbf{d} = \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}^T \tag{23}$$

$$\mathbf{w} = \begin{bmatrix} w & w & b \end{bmatrix}^T \tag{24}$$

XOR revisitado

- Problema super-determinado, uma vez que há mais pontos de dados que parâmetros livres.
- Isso explica porque a matriz **G** não é quadrada.
- Consequentemente, não existe inversa única para a matriz **G**.
- Para superar essa dificuldade, usa-se a solução da *norma mínima* da equação

$$\mathbf{G}^{+} = (\mathbf{G}^{T}\mathbf{G})^{-1}\mathbf{G}^{T} \tag{25}$$

onde $\mathbf{G}^{+}$ é a pseudo-inversa da matriz **G**.

$$\mathbf{w} = \mathbf{G}^{+}\mathbf{d} = (\mathbf{G}^{T}\mathbf{G})^{-1}\mathbf{G}^{T}\mathbf{d} \tag{26}$$

- $\mathbf{G}^{T}\mathbf{G}$ é uma matriz quadrada com única inversa.

XOR revisitado

- Substituindo a equação 22 em 25:

$$\mathbf{G}^{+} = \begin{bmatrix} 1.8292 & -1.2509 & 0.6727 & -1.2509 \\ 0.6727 & -1.2509 & 1.8292 & -1.2509 \\ -0.9202 & 1.4202 & -0.9202 & 1.4202 \end{bmatrix} \quad (27)$$

- Substituindo a equação 23 e 27 em 25:

$$\mathbf{w} = \begin{bmatrix} -2.5018 \\ -2.5018 \\ +2.8404 \end{bmatrix} \quad (28)$$

que completa a especificação da rede RBF.

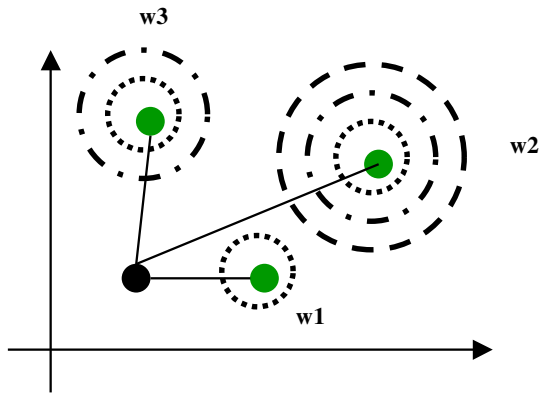
Sumário

Material do Marco Botta

Os próximos 29 slides contêm material de Marco Botta, disponíveis em [1].

Radial-Basis Function Networks

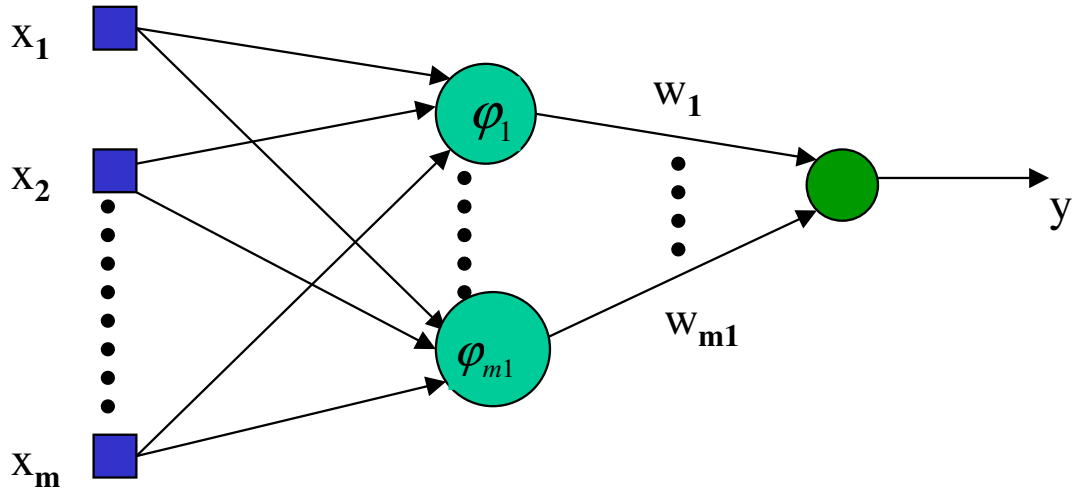
- A function is radial basis (RBF) if its output depends on (is a non-increasing function of) the distance of the input from a given stored vector.
- RBFs represent local receptors, as illustrated below, where each green point is a stored vector used in one RBF.
- In a RBF network one hidden layer uses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

$$w1 < w3 < w2$$

RBF ARCHITECTURE



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

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

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

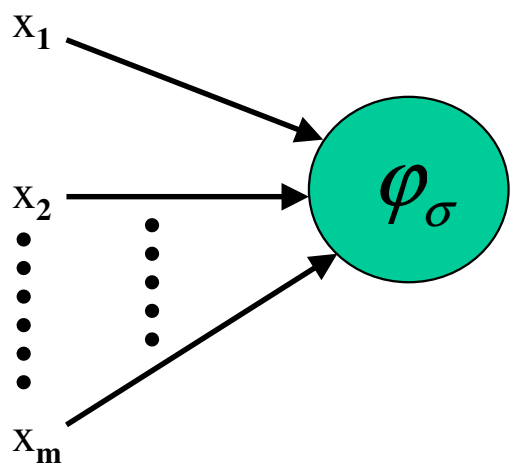
$$y = w_1\varphi_1(\| x - t_1 \|) + \dots + w_{m1}\varphi_{m1}(\| x - t_{m1} \|)$$

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

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



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

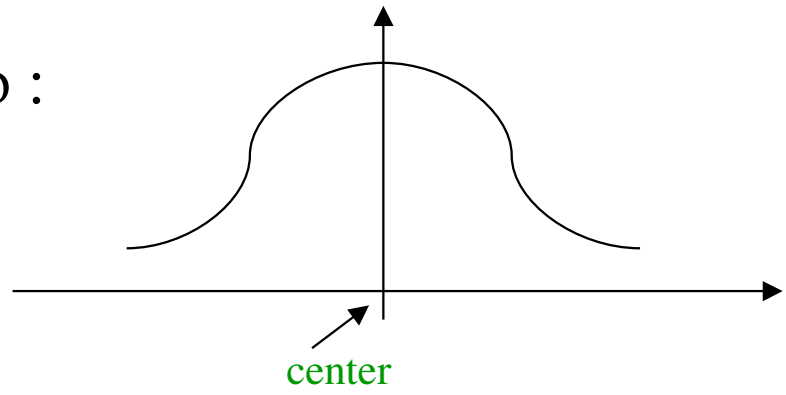
t is called center
σ is called spread
center and spread are parameters

Hidden Neurons

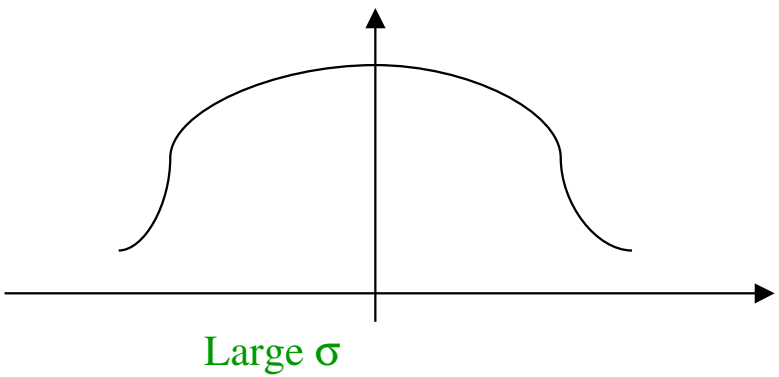
- A hidden neuron is more sensitive to data points near its center.
- For Gaussian RBF this sensitivity may be tuned by adjusting the spread σ, where a larger spread implies less sensitivity.
- Biological example: cochlear stereocilia cells (in our ears ...) have locally tuned frequency responses.

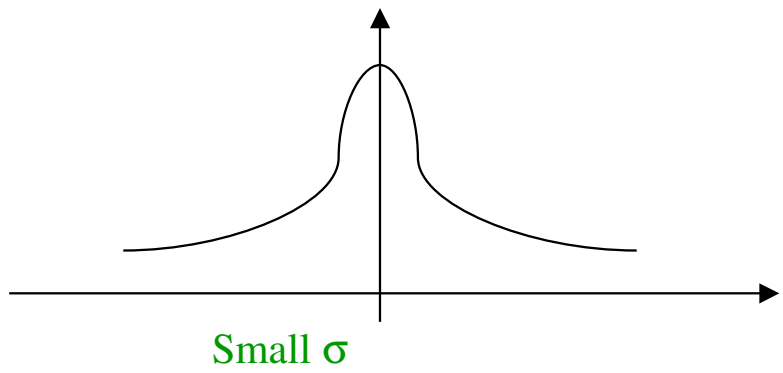
Gaussian RBF φ

φ :



σ is a measure of how spread the curve is:





Types of φ

- Multiquadrics:

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

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

- Inverse multiquadrics:

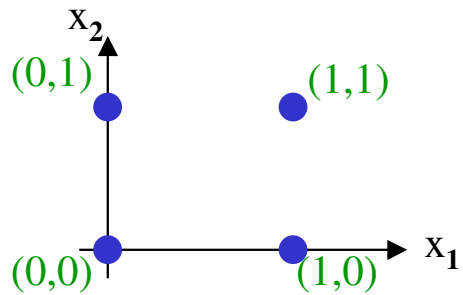
$$\varphi(r) = \frac{1}{(r^2 + c^2)^{1/2}} \qquad c > 0$$

- Gaussian functions (most used):

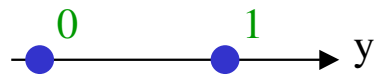
$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \qquad \sigma > 0$$

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

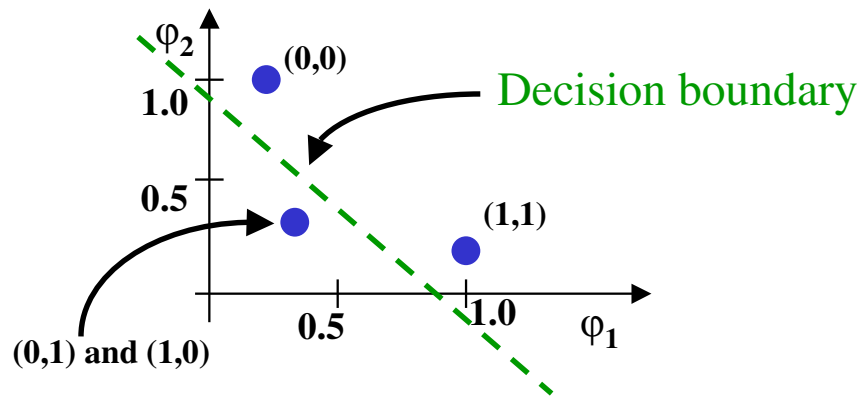
Example: the XOR problem

- In the feature (hidden layer) space:

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

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

with $t_1 = (1,1)$ and $t_2 = (0,0)$

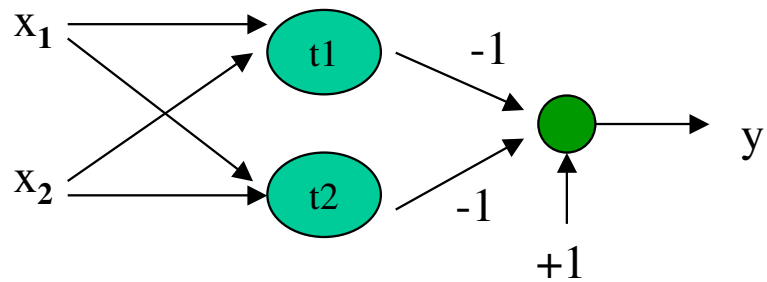


- When mapped into the feature space $< \varphi_1 , \varphi_2 >$ (hidden layer), C1 and C2 become *linearly separable.* So a linear classifier with $\varphi_1(x)$ and $\varphi_2(x)$ as inputs can be used to solve the XOR problem.

RBF NN for the XOR problem

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

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



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

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:

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

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**

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

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

$$w_1 \varphi_1(\| x_i - t_1 \|) + \ldots + w_{m1} \varphi_{m1}(\| x_i - t_{m1} \|) = d_i$$

Learning Algorithm 1

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

$$[\varphi_1(\| x_i - t_1 \|) \dots \varphi_{m1}(\| x_i - t_{m1} \|)][w_1 \dots w_{m1}]^T = d_i$$

and

$$\begin{bmatrix} \varphi_1(\| x_1 - t_1 \|) \dots \varphi_{m1}(\| x_1 - t_{m1} \|) \\ \dots \\ \varphi_1(\| x_N - t_1 \|) \dots \varphi_{m1}(\| x_N - t_{m1} \|) \end{bmatrix} [w_1 \dots w_{m1}]^T = [d_1 \dots d_N]^T$$

for all the examples at the same time

Learning Algorithm 1

let $$\Phi = \begin{bmatrix} \varphi_1(\| x_1 - t_1 \|) & \dots & \varphi_{m1}(\| x_N - t_{m1} \|) \\ & \dots & \\ \varphi_1(\| x_N - t_1 \|) & \dots & \varphi_{m1}(\| x_N - t_{m1} \|) \end{bmatrix}$$

then we can write $$\Phi \begin{bmatrix} w_1 \\ \dots \\ w_{m1} \end{bmatrix} = \begin{bmatrix} d_1 \\ \dots \\ d_N \end{bmatrix}$$

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

$$[w_1 \dots w_{m1}]^T = \Phi^+ [d_1 \dots d_N]^T$$

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 $\quad k = 1, \ldots, m_1$

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

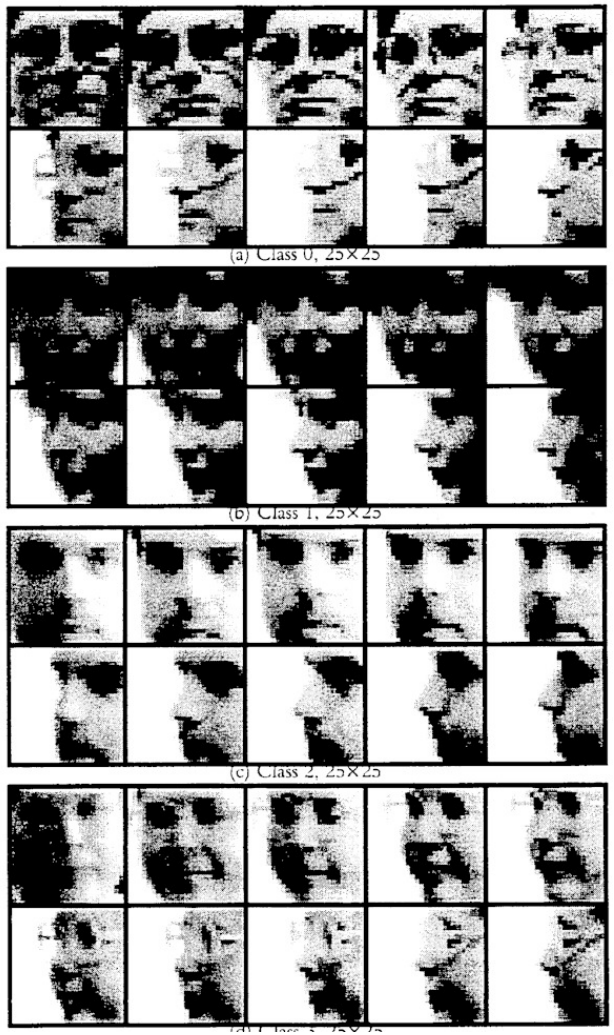
- The problem:
  - Face recognition of persons of a known group in an indoor environment.
- The approach:
  - Learn face classes over a wide range of poses using an RBF network.

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 asses 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



Approach: Face unit RBF

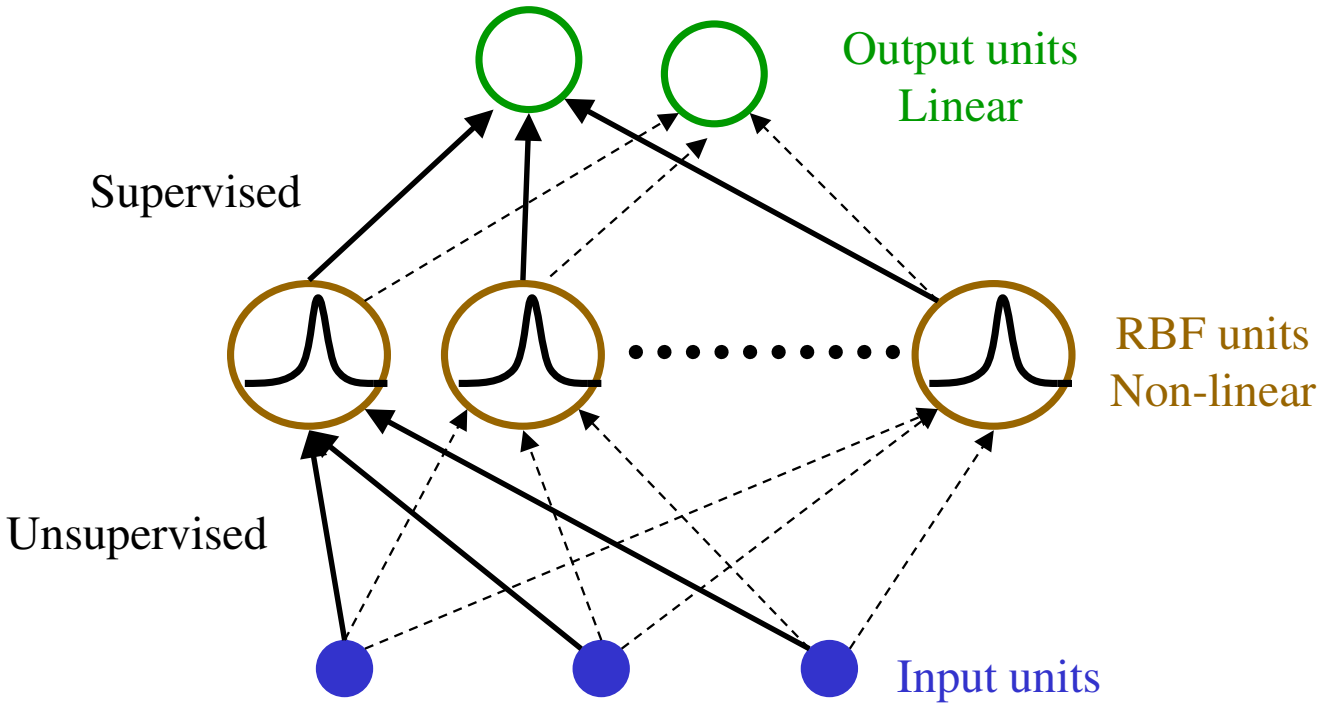
- A face recognition unit RBF neural networks is trained to recognize a single person.
- Training uses examples of images of the person to be recognized as positive evidence, together with selected confusable images of other people as negative evidence.

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.

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

Sumário

Aplicações e Conclusões

- Aplicações
  - Reconhecimento de caracteres
  - Reconhecimento de alvos
  - Análise de crédito
  - Processamento de sinais
  - Processamento de voz
- Conclusões
  - Redes RBF: alternativa a MLP
  - Funções de base radial
  - Função de ativação leva em conta distância entre vetores de entrada e protótipos
  - Aprendizado geralmente em dois estágios
    - Supervisionado
    - Não supervisionado

Bibliografia I

[1] M. Botta
Slides, Dipartimento di Informatica, Università di Torino.
http://www.di.unito.it/~botta/didattica/RBF_1.pdf/

[2] A. C. P. L. F. de Carvalho
Redes Neurais do tipo RBF.
Slides. Redes Neurais. LABIC - ICMSC - USP, 2008.

[3] T. M. Cover
Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition.
IEEE Transactions on Electronic Computers, vol. EC-14, pp. 326–334.

Bibliografia II

[4] S. Haykin
Neural networks - a comprehensive foundation.
2nd. edition. Prentice Hall, 1999.

[5] R. A. F. Romero
SCC-5809 Redes Neurais.
Slides e listas de exercícios. Programa de Pós-Graduação em Ciência de Computação e Matemática Computacional. ICMC/USP, 2010.