Redes RBF
Radial Basis Function

Computação Natural

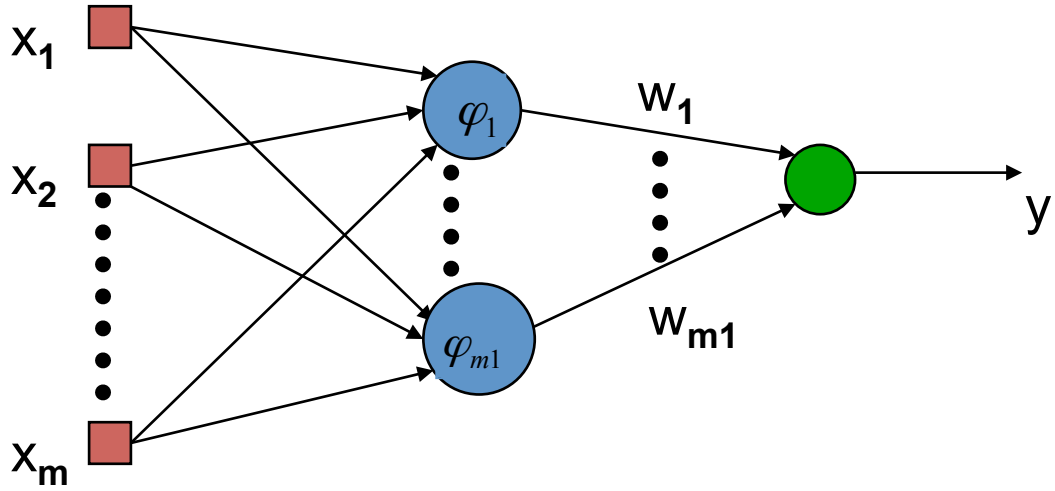
Gisele L. Pappa

Introdução

- São redes de 3 camadas
- Os neurônios da camada escondida implementam funções de base radial
- Os nós de saída implementam funções lineares
- O treinamento é dividido em 2 estágios:
  - Determina os pesos da camada de entrada para a escondida
  - Determina os pesos da camada escondida para a de saída
- O aprendizado é rápido

Arquitetura

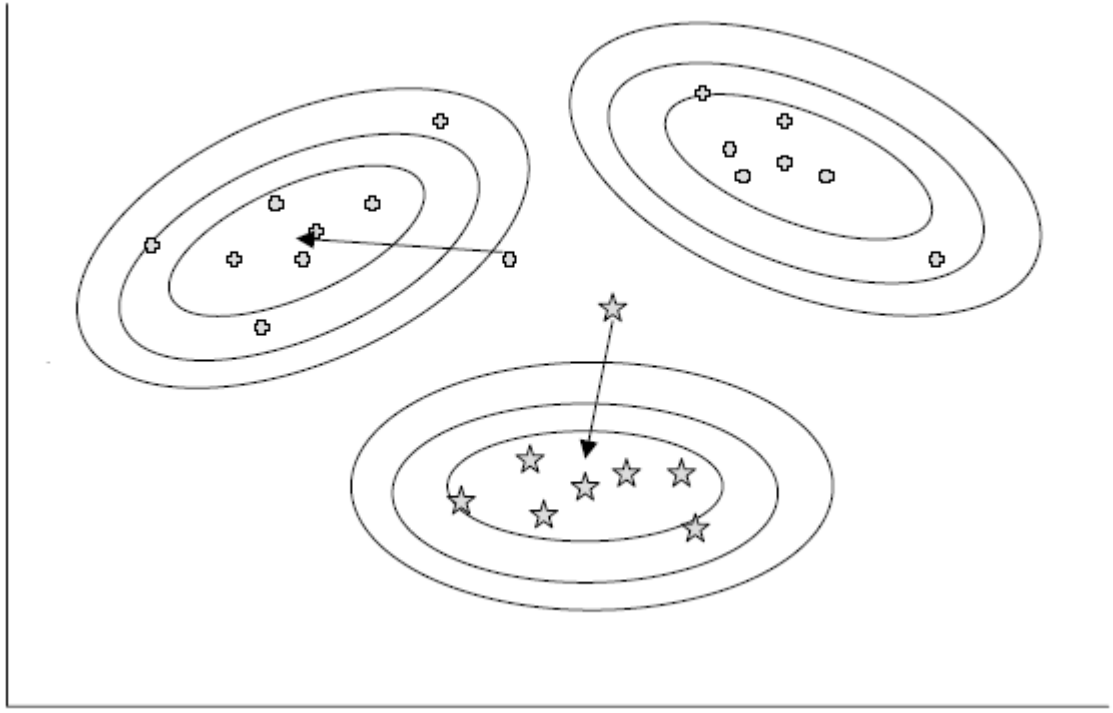
- **Camada de entrada**

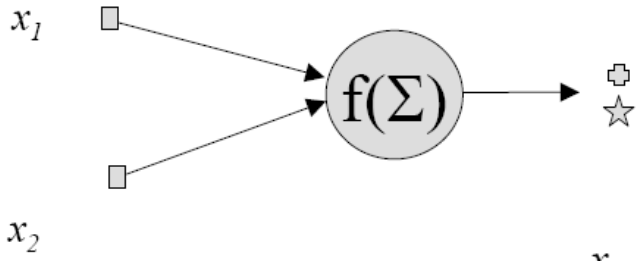


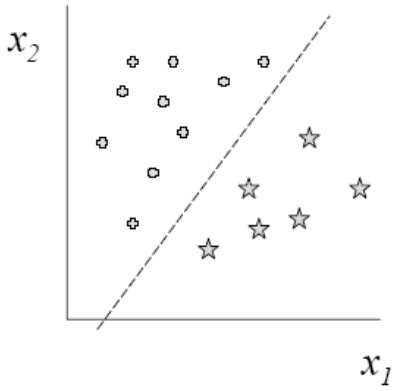
- **Camada escondida**
  - **Aplica uma transformação não linear do espaço de entrada para o espaço escondido**
- **Camada de saída**
  - **Aplica uma transformção linear do espaço escondido para o espaço de saída**

RBF

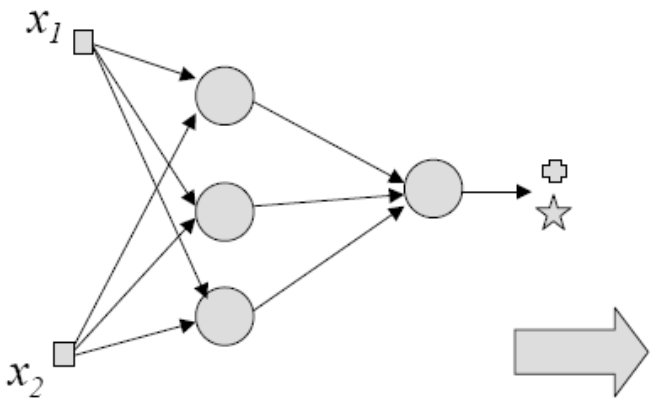
- A saída de rede é aproximadamente a combinação linear de funções de base radial
- RBFs capturam o comportamento local das funções

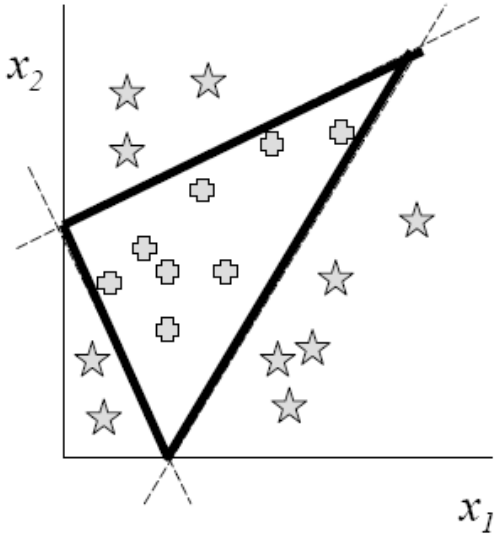






- Perceptron





- Perceptron multi-camdas

O que é uma RBF ?

- Funções de base radial
  - Radial: Simétrica em torno do centro
  - Funções base
    - Também conhecidas como kernels
    - Conjunto de funções cuja combinação linear pode gerar uma função arbitrária em um dado espaço de funções

Qual a função de um kernel?

- Digamos que você tenha um problema em que deseja separar 2 classes, mas a borda que você tem que utilizar se confunde
- Você pode encontrar um algoritmo que separe os pontos, mas ele irá demorar para convergir
- Funções de kernel mapeam esses dados em espaços de maior dimensão, na esperança de que os dados possam ser mais facilmente separados

RBFs

- Qual a motivação para utilizar uma função não-linear seguida de uma linear?
  - Teorema de Cover sobre a separabilidade de padrões:
    - "Um problema de classificação de padrões complexos moldado não-linearmente em um espaço com muitas dimensões tem maior probabilidade de ser linearmente separável que quando moldado em um espaço com poucas dimensões"
  - Quando a camada escondida aplica uma transformação não-linear no espaço de entrada, ela cria um novo espaço tipicamente com mais dimensões que o espaço de entrada

* Mesmo argumento para trabalhar com SVM

Tipos de Função RBF

- Multiquadráticas:

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

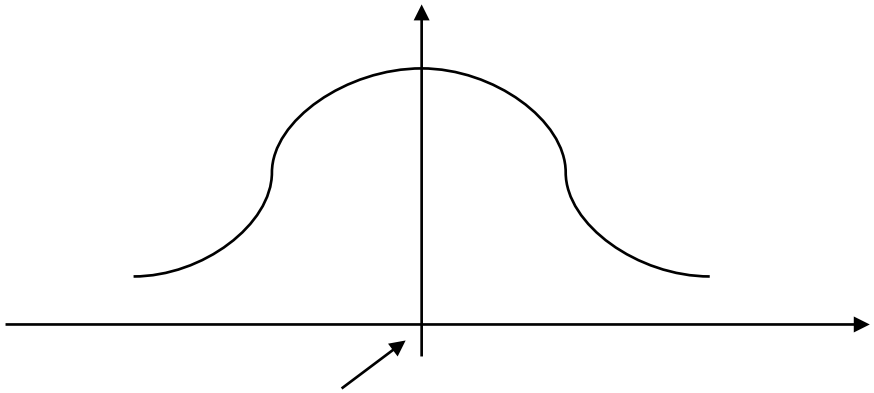
- Multiquadráticas inversas:

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

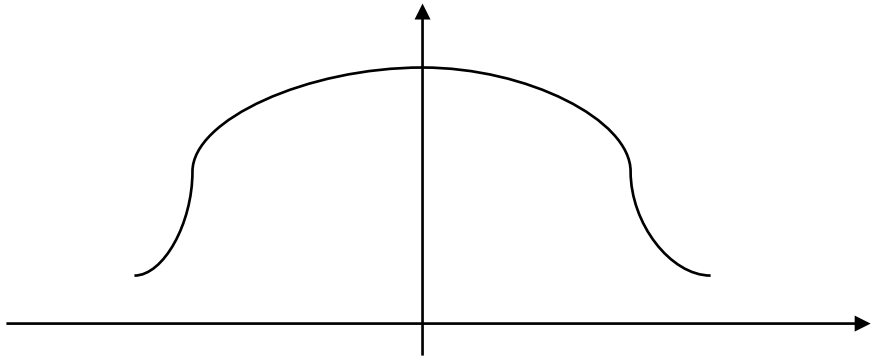
- Gaussianas:

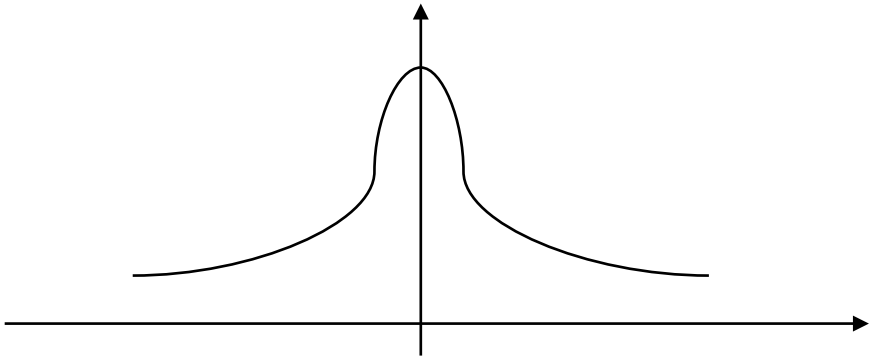
$$\phi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \qquad \sigma > 0$$

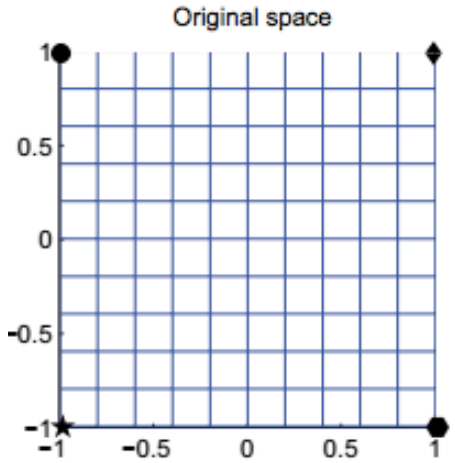
Função RBF Gaussiana

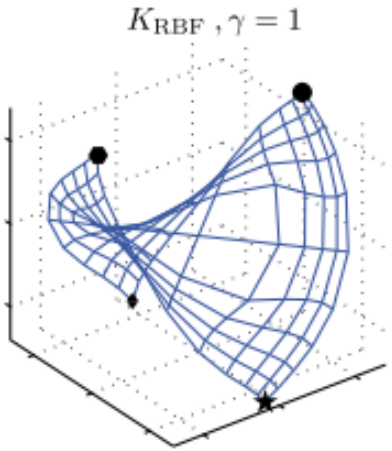


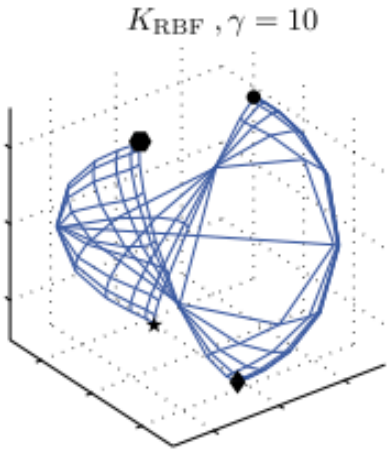
σ é a medida do quão "achatada" a curva é
(dentro de um raio, define a influência de um ponto):

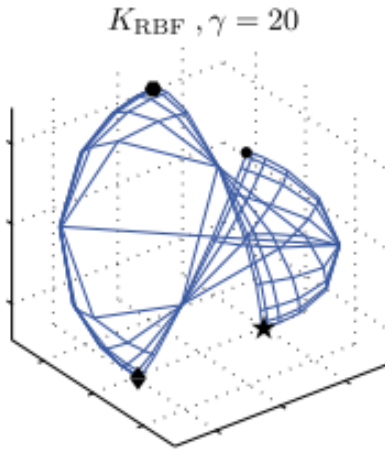





Original space
1
0.5
0
−0.5
−1
−1
−0.5
0
0.5
1

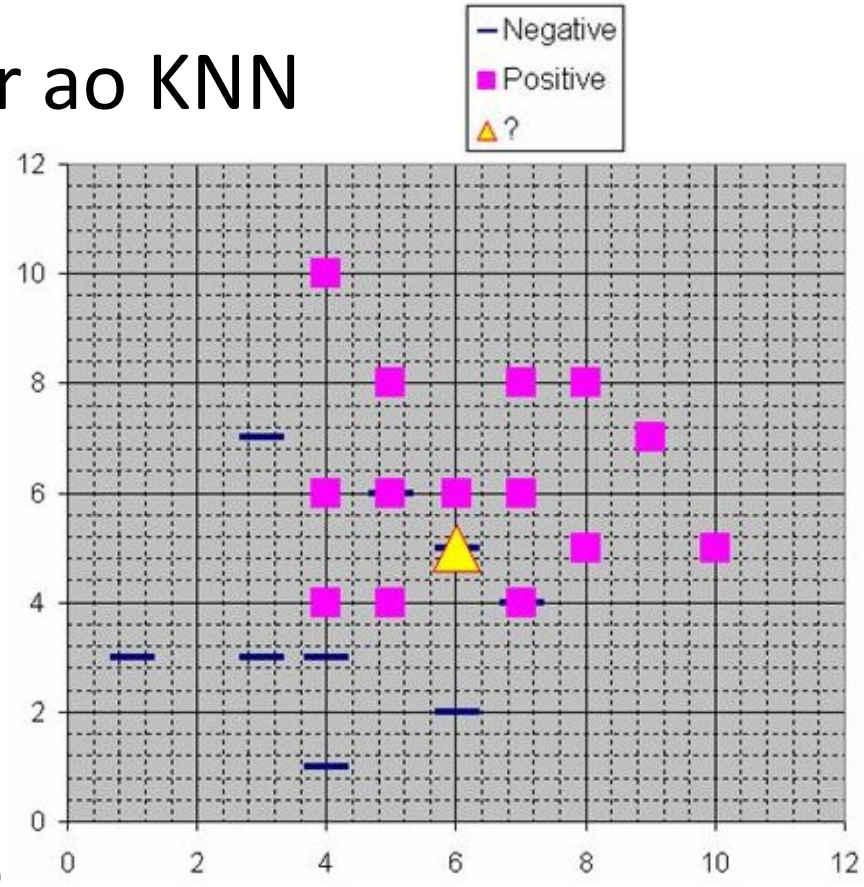

$K_{\mathrm{RBF}}\,, \gamma = 1$


$K_{\mathrm{RBF}}\,, \gamma = 10$


$K_{\mathrm{RBF}}\,, \gamma = 20$

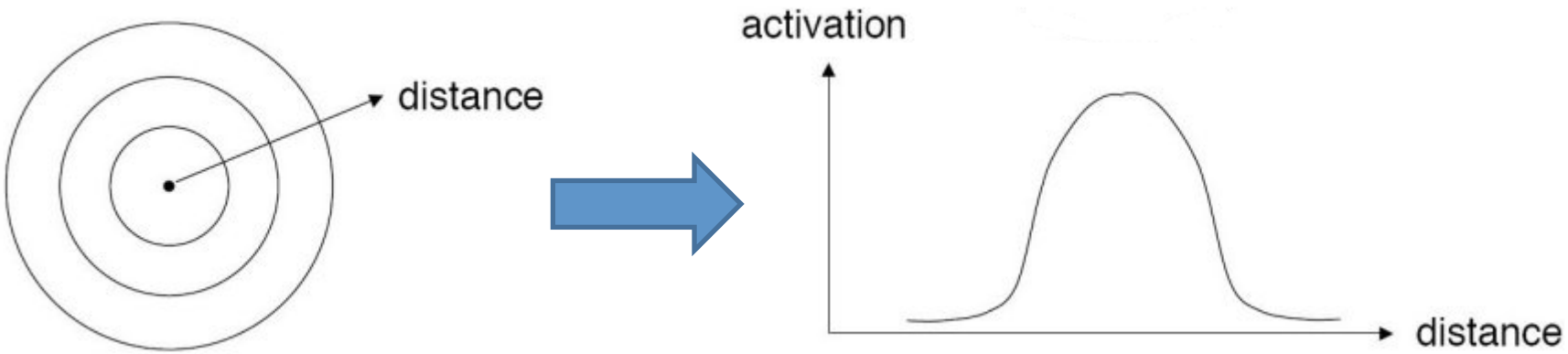
Como uma Rede RBF funciona?

- Conceitualmente, similar ao KNN
- A rede posiciona 1 ou mais neurônios no espaço n-D descrito pelos n atributos que descrevem o exemplo
- Calcula a distância Euclidiana do ponto sendo avaliado para o centro de cada neurônio

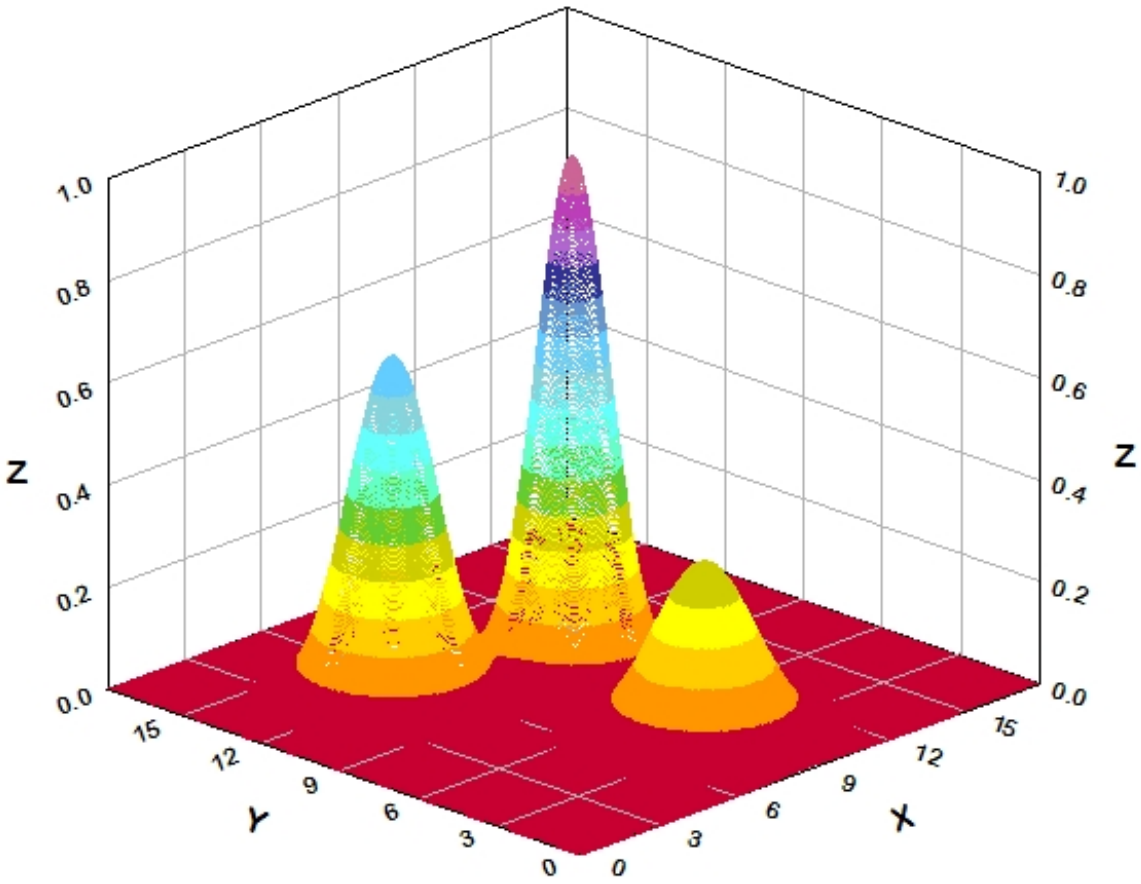


Como uma Rede RBF funciona?

- Aplica uma função RBF a cada distância ponto-neurônio, para computar o peso (influência) de cada neurônio
  - Peso(neurônio) = RBF (distância)
- Quanto mais longe o neurônio está do ponto sendo avaliado, menor seu peso.



Como uma rede RBF funciona?

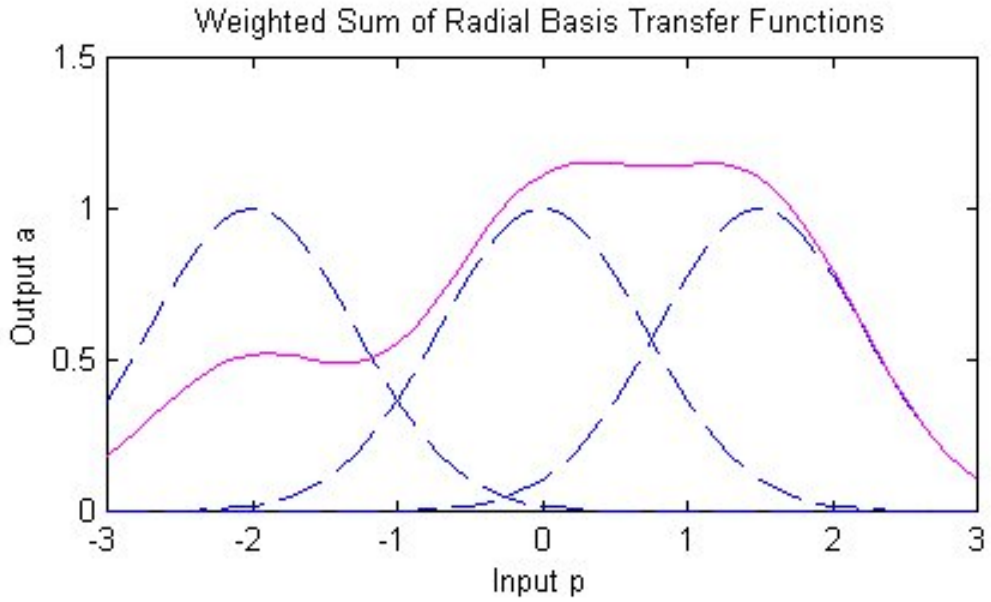


- 3 neurônos em uma espaço 2D, determinado pelas variáveis x e y.

- Z é o valor de saída de função RBF

Como uma rede RBF funciona?

- O valor predito para um novo ponto é dado pela soma dos valores da função RBF multiplicado pelos pesos encontrados para cada neurônio



Definição do Problema

- Dado um vetor de entrada de D dimensões $x^p = \{x^p_i : i = 1,..,D\}$, queremos encontrar um vetor de saída correspondente no espaço de c dimensões, $t^p = \{t^p_k : k = 1,..,c\}$
- Essas saída serão geradas por um conjunto de funções $g_k(x)$. A ideia é aproximar $g_k(x)$ com funções $y_k(x)$ da seguinte forma

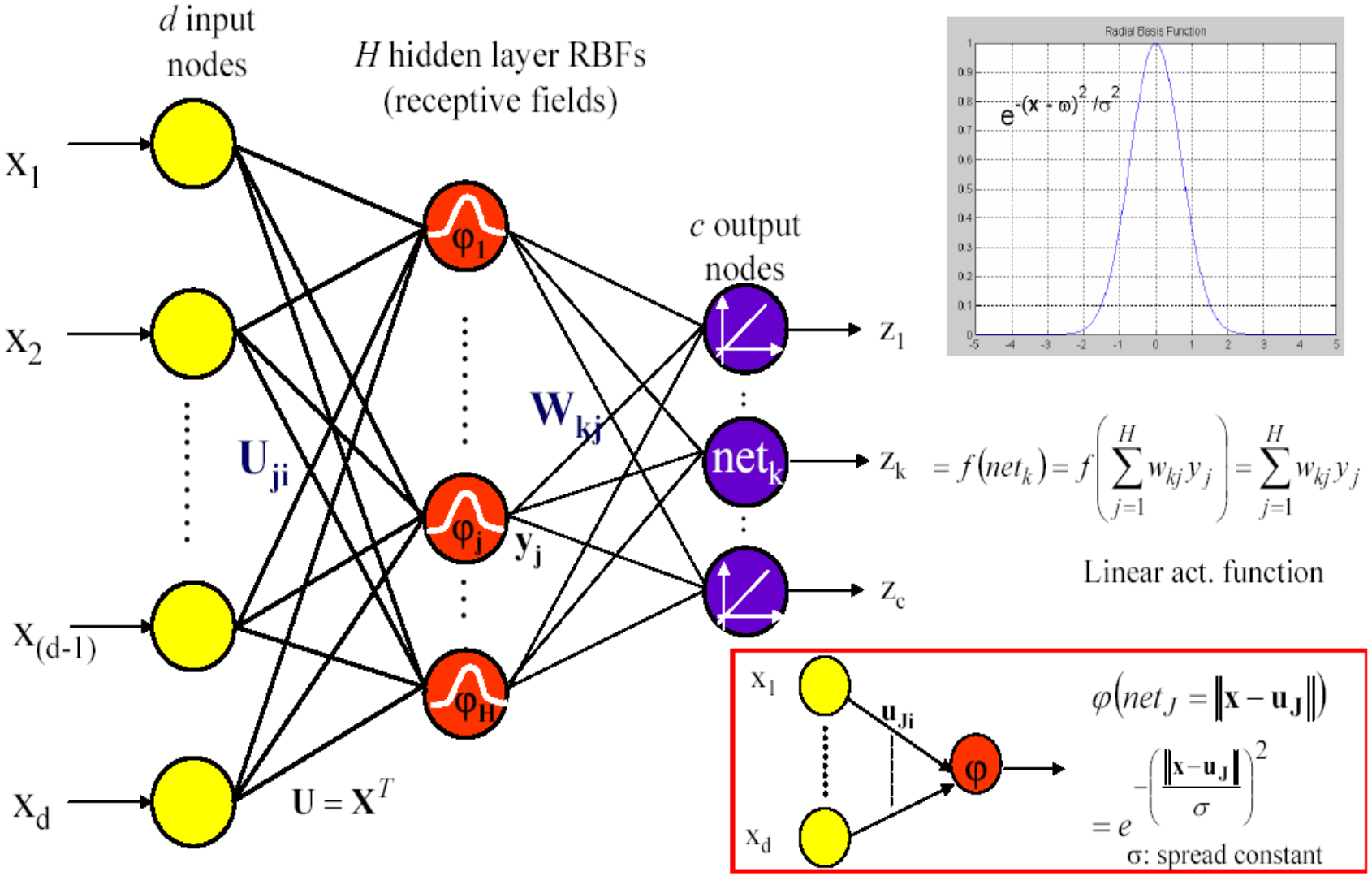
$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x})$$

M : número de neurônios na camada escondida

Definição do Problema

- O problema se transforma em encontrar valores apropriados para:
  - M : número de neurônios na camada escondida
  - U_{ij} : centros (valores dos neurônios da camada escondida)
  - σ_j, *spread* da função
  - W_{kj}:pesos da camada escondida para camada de saída

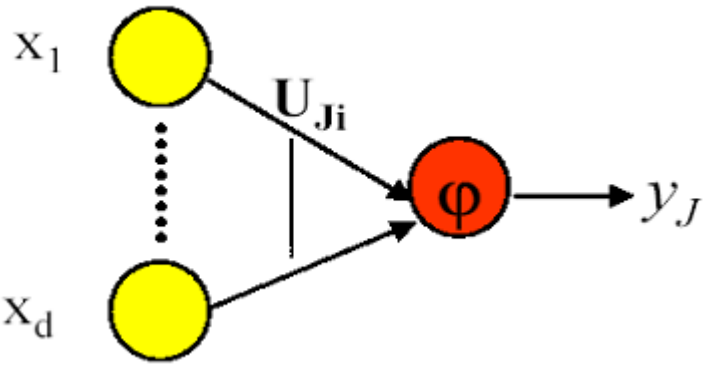
RBF


d input nodes
H hidden layer RBFs (receptive fields)
c output nodes
x1
x2
x(d-1)
xd
U_ji
W_kj
φ1
φj
φH
y_j
net_k
z1
zk
zc
U = X^T
Radial Basis Function
e^-(x - ω)^2 /σ^2
= f(net_k) = f(Σ_{j=1}^{H} w_kj y_j) = Σ_{j=1}^{H} w_kj y_j
Linear act. function
x1
xd
u_Ji
φ
φ(net_J = ||x − u_J||)
= e^-(||x−u_J|| / σ)^2
σ: spread constant

Neurônios da Camada Escondida

- Utilizam uma função de base radial

$\phi(|| x - u||^2)$ A saída depende da distância da entrada x e do centro u



Norma euclidiana

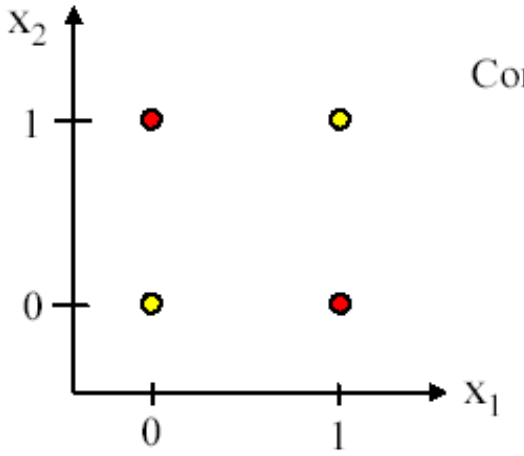
$$\phi_j(|| x - u||^2) = e^{-\left(\frac{\|\mathbf{x}-\mathbf{u}\|}{\sigma}\right)^2}$$

u é o centro da função
σ é o *spread* da função
* Ambos são parâmetros

Neurônios da Camada Escondida

- Os neurônios da camada escondida são mais sensíveis as entradas de dados próximas ao seu centro. Essa sensibilidade pode ser controlada pelo parâmetro σ.
  - Quanto maior o valor de σ, menor a sensibilidade

Exemplo -XOR



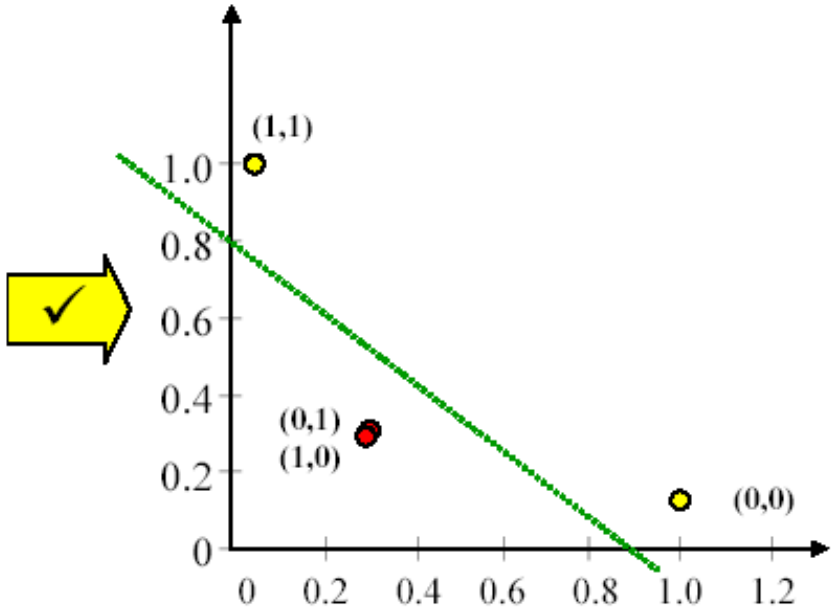
Consider the nonlinear functions to map the input vector **x** to the φ_1- φ_2 space

$\mathbf{x}=[x_1\ x_2]$

$$\varphi_1(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{u}_1\|^2} \qquad \mathbf{u}_1 = [1\ \ 1]^T$$

$$\varphi_2(\mathbf{x}) = e^{-\|\mathbf{x}-\mathbf{u}_2\|^2} \qquad \mathbf{u}_2 = [0\ \ 0]^T$$

Input x	$\varphi_1(x)$	$\varphi_2(x)$
(1,1)	1	0.1353
(0,1)	0.3678	0.3678
(1,0)	0.3678	0.3678
(0,0)	0.1353	1



The nonlinear φ function transformed a nonlinearly separable problem into a linearly separable one !!!

Treinamento

- Neurônios da camada de entrada para escondida tem funções bem diferentes dos da camada escondida para de saída
- Em um passo inicial, otimizamos os parâmetros das funções RBF
  - Os centros (protótipos dos dados)
  - Os *spreads*
- Em um segundo passo, deixamos esses parâmetros fixos e treinamos a segunda parte da rede

Algoritmos de Aprendizado

- Estudaremos 3:
  - Centros aleatórios + método da pseudo-inversa
  - K-means + LMS
  - Descida do gradiente (funciona em uma única fase)

Algoritmo de Aprendizagem 1

Fase 1

- Centros são selecionados aleatoriamente entre os dados de treinamento

- *Spreads* são escolhidos por normalização:

$$\sigma = \frac{\text{Distância máxima entre 2 centros}}{\sqrt{\text{número de centros}}} = \frac{d_{max}}{\sqrt{m_1}}$$

Treinamento – Fase 2

- Pesos da camada de entrada para escondida já determinados
  - Encontrar pesos para uma rede de uma camada com função linear

$$y_k(\mathbf{x}^p) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x}^p) \quad .$$

M - número de neurônios na camada escondida

- A primeira forma de treinamento é utilizar o método da pseudo-inversa

Treinamento – Fase 2

- **Método da pseudo-inversa.**
  - Para um exemplo (x_i, d_i) , considere a saída da rede como sendo

$$y(x_i) = w_1\varphi_1(\| x_i - \mathsf{U}_1 \|) + \ldots + w_{m1}\varphi_{m1}(\| x_i - \mathsf{U}_{m1} \|)$$

  - Gostaríamos que, para cada exemplo, $y(x_i) = d_i$

$$w_1\varphi_1(\| x_i - \mathsf{U}_1 \|) + \ldots + w_{m1}\varphi_{m1}(\| x_i - \mathsf{U}_{m1} \|) = d_i$$

Treinamento – Fase 2

- Essa equação pode ser reescrita em forma de matriz

$$[\varphi_1(\| x_i - u_1 \|) \ldots \varphi_{m1}(\| x_i - u_{m1} \|)][w_1 \ldots w_{m1}]^T = d_i$$

o que resulta em

$$\begin{bmatrix} \varphi_1(\| x_1 - u_1 \|) \ldots \varphi_{m1}(\| x_1 - u_{m1} \|) \\ \ldots \\ \varphi_1(\| x_N - u_1 \|) \ldots \varphi_{m1}(\| x_N - u_{m1} \|) \end{bmatrix} [w_1 \ldots w_{m1}]^T = [d_1 \ldots d_N]^T$$

quando consideramos todos os exemplos de treinamento

Algoritmo de Aprendizagem 1

Reescrevendo

$$\Phi = \begin{bmatrix} \varphi_1(\| x_1 - u_1 \|) & \cdots & \varphi_{m1}(\| x_N - u_{m1} \|) \\ & \cdots & \\ \varphi_1(\| x_N - u_1 \|) & \cdots & \varphi_{m1}(\| x_N - u_{m1} \|) \end{bmatrix}$$

como

$$\Phi \begin{bmatrix} w_1 \\ \cdots \\ w_{m1} \end{bmatrix} = \begin{bmatrix} d_1 \\ \cdots \\ d_N \end{bmatrix}$$

Seja Φ^+ a pseudo-inversa da matriz Φ , podemos obter os pesos usando:

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

Algoritmo de Aprendizagem 2

FASE 1

- Utiliza um algoritmo de agrupamento para encontrar os centros, como o K-means
- Tem a vantagem de fazer com que os centros reflitam a distribuição dos dados
- Calcula os *spreads* por normalização

Algoritmo de Aprendizagem 2

- Fase 2: LMS (*Least Mean Square*)

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha\, e_i\, \mathrm{x} \quad \text{(peso)}$$

$$\mathbf{b}(t+1) = \mathbf{b}(t) + \alpha\, e_i \quad \textit{(bias)}$$

α é a taxa de aprendizagem

e_i é o erro entre a saída encontrada e a desejada

Algoritmo de Aprendizado 3

- Aplica o método da descida do gradiente para encontrar simultaneamente o centro, spread e pesos, minimizando o erro quadrático médio

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

- Atualizações:

centros

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

spread

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

pesos

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

RBF vs MLP

- RBF tem apenas uma camada escondida
- Na RBF os neurônios da camada escondida tem função diferente dos neurônios de saída
- Na RBF a camada escondida tem função de ativação não linear e a camada de saída linear

RBF para Reconhecimento de Face

- Problema:
  - Reconhecimento de face de pessoas conhecidas em um ambiente fechado
- Abordagem:
  - Aprender as classes de faces utilizando uma variedade de fotos usando RBF
- Base de dados
  - **100 imagens de 10 pessoas** (em preto e branco, resolução 384 x 287)

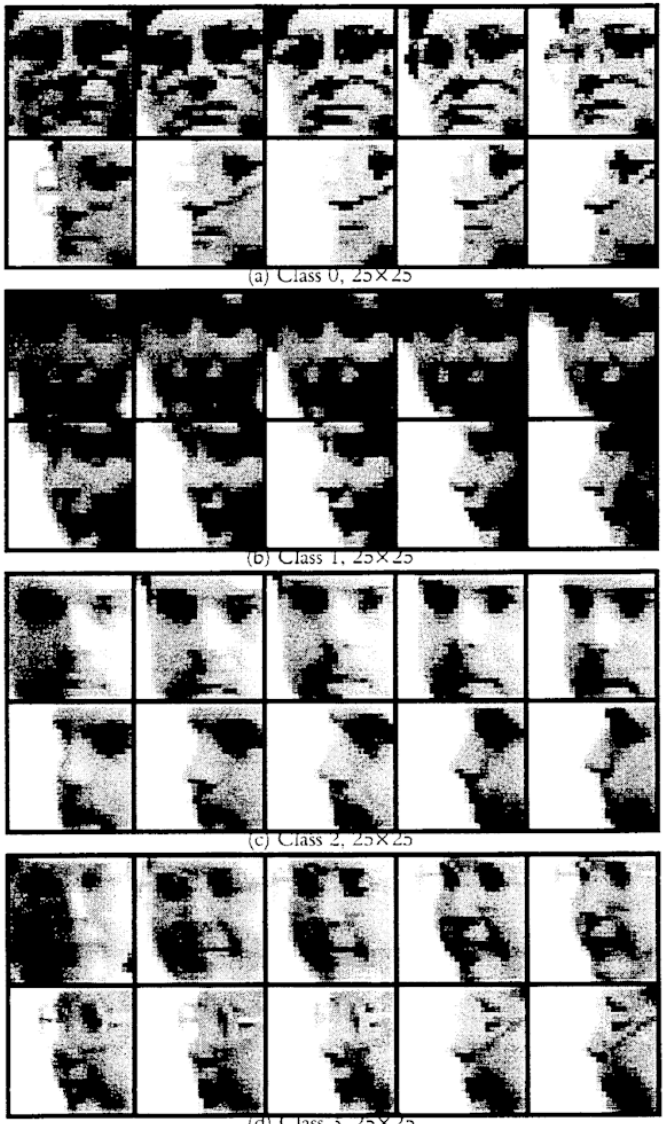
RBF para Reconhecimento de Face

Figure 2: **Shift-varying** data for the 'face on' view of one individual: (a) top left (b) top right (c) normal view (d) bottom left (e) bottom right

Figure 3: **Scale-varying** data for the 'face on' view of one individual: (a) +25% (uses 111×111 window) (b) +12.5% (107×107) (c) normal view (100×100) (d) –12.5% (94×94) (e) –25% (87×87)

Pré-processamento

- Imagens são centralizadas
- Amostragem
  - 25x25
- Filtros
  - Diferença de gaussianas
  - Gabor



Pré-processamento

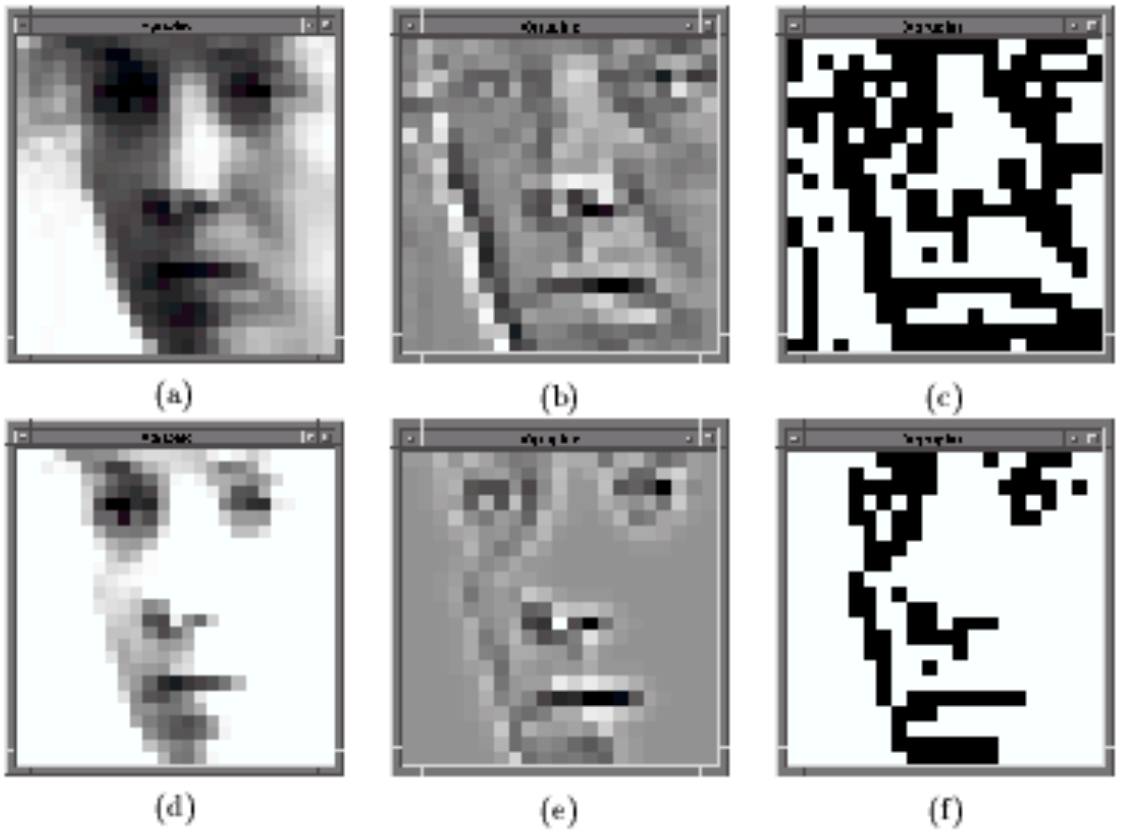


Figure 3: Effect of reducing range of grey-levels on 25×25 image (a) full range of grey-levels (b) after non-thresholded DoG (c) after thresholded DoG (d) reduced range of grey-levels (e) after non-thresholded DoG (f) after thresholded DoG

Abordagem

- RBF é treinada para reconhecer uma pessoa
- Imagens da pessoa em questão são usadas como exemplos positivos e a das outras 9 na base como exemplos negativos

Arquitetura da rede

- Camada de entrada
  - 25x 25 entradas (representando a imagem após pré-processamento
- Camada escondida tem p+a neurônios
  - p recebem apenas evidências para classe positiva
  - a recebem apenas evidências para classe negativa
- Camada de saída tem 2 neurônios
  - Uma para pessoa em questão
  - Outra para as outras 9 pessoas

Camada Escondida

- Como mencionado, neurônios podem ser:
  - Pró-neurônios: Evidência para pessoa
  - Anti-neurônios : Evidência negativa
- O número de pró-neurônios é igual ao número de exemplos positivos no conjunto de treinamento
- Para cada pró-neurônio, existem 1 ou 2 anti-neurônios.
- A função Gaussiana é utilizada

Aprendizagem

- Centros
  - De um pró-neurônio: correspondem a um exemplo positivo
  - De um anti-neurônio: o exemplo negativo mais próximo do exemplo positivo utilizado no pró-neurônio, utilizando a distância Euclidiana
- Spread
  - normalizado
- Pesos
  - Determinados utilizando o método da pseudo-inversa

Resultados

- Uma rede com 6 pró-neurônios e 12 anti-neurônios classificou corretamente 96% dos dados no conjunto de teste.

Leitura recomendada

- Tese de doutorado
  - Jonathan Howelland, Sussex University
    http://www.cogs.susx.ac.uk/users/jonh/index.html