2. Redes Neurais Artificiais

Prof. Renato Tinós

Depto. de Computação e Matemática (FFCLRP/USP)

2.3. Perceptron Multicamadas - MLP

2.3.1. Introdução ao MLP

2.3.2. Treinamento

2.3.3. Projeto de MLPs

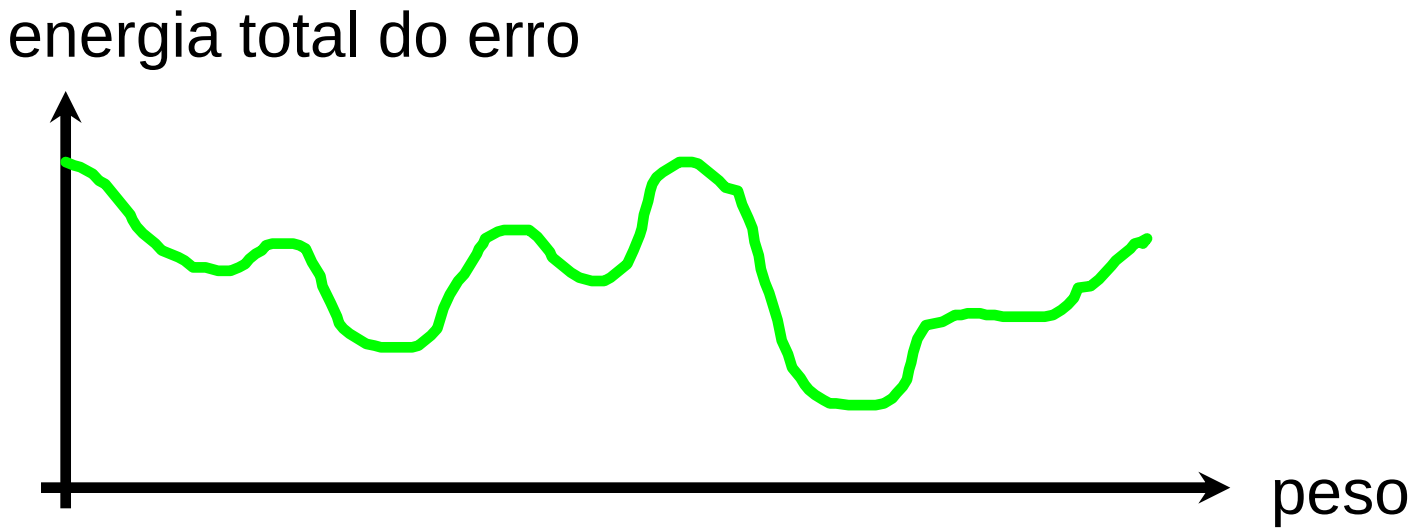
2.3.4. Reconhecimento de Padrões

2.3.5. Aproximação de Funções

2.3.6. Generalização

2.3.3. Projeto de MLPs

- **Taxa de aprendizagem**
  - Fornece o tamanho do ajuste dos pesos de uma iteração para a outra
    - » Tamanho do passo na superfície de erro
      - ❑Exemplo:



2.3.3. Projeto de MLPs

- **Taxa de aprendizagem**
  - Taxas de aprendizagem pequenas
    - » Pequenas variações dos pesos sinápticos de uma iteração para outra
    - » Aprendizagem lenta
    - » Pode provocar o aprisionamento dos pesos em um mínimo local
  - Taxas de aprendizagem altas
    - » Podem tornar o aprendizado instável (oscilatório)
  - Então, qual é a taxa de aprendizagem que deve ser utilizada?

2.3.3. Projeto de MLPs

- **Taxa de aprendizagem**
  - Uma solução para este problema é modificar a regra delta (Eq. 2.38) para incluir um termo de momento

$$\Delta w_{ji}(n) = \alpha \, \Delta w_{ji}(n-1) + \eta \, \delta_j(n) \, o_i(n) \qquad (2.43)$$

  - Para um termo de momento restrito a $0 \leq |\alpha| < 1$
    - » A Eq. (2.43) torna-se uma série temporal ponderada exponencialmente e convergente
  - Quando dois ajustes de peso têm o mesmo sinal
    - » $\Delta w_{ji}(n)$ cresce em magnitude
    - » Aceleração do aprendizado
  - Quando dois ajustes de peso têm sinais opostos
    - » $\Delta w_{ji}(n)$ diminui em magnitude
    - » Efeito estabilizador

2.3.3. Projeto de MLPs

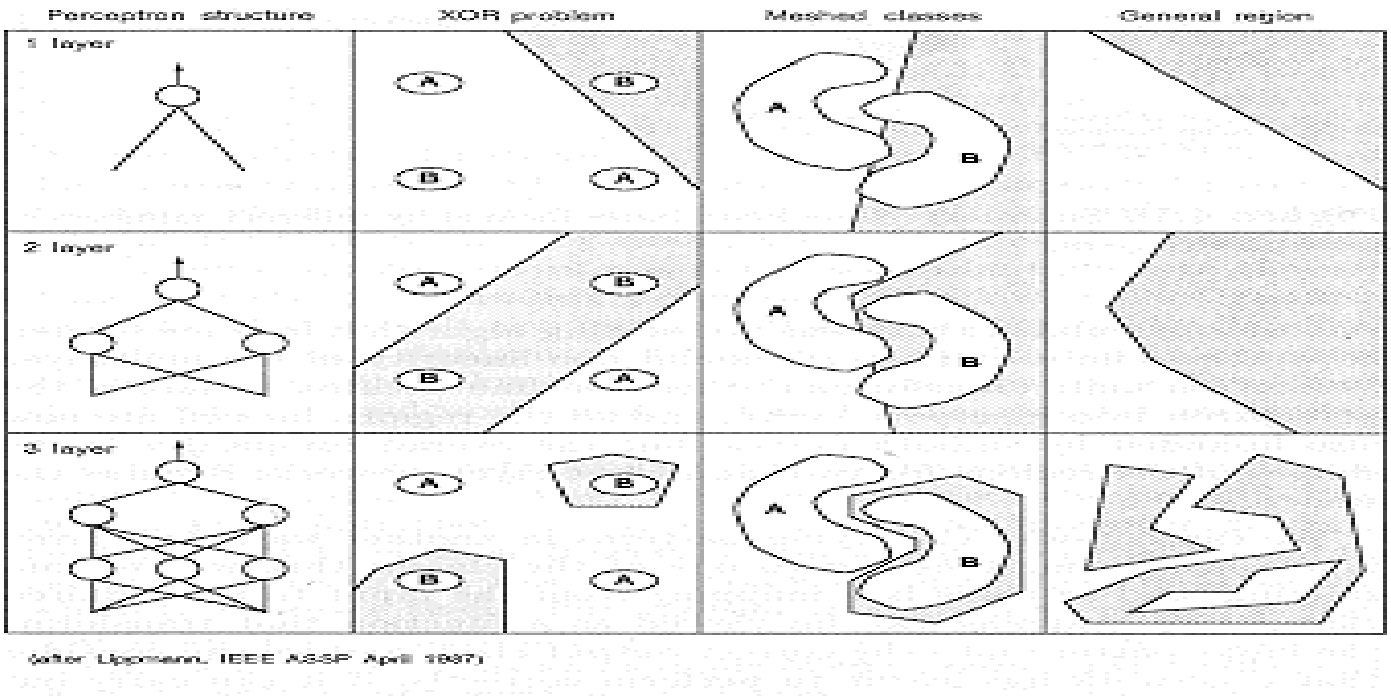
- **Critérios de parada**
  - Não se pode demonstrar que o algoritmo *backpropagation* convergiu
  - No entanto, alguns critérios de parada podem ser utilizados
    1. Considera-se que o algoritmo tenha convergido quando a norma euclidiana do ajuste do vetor de pesos alcançar um limiar suficientemente pequeno
    2. Considera-se que o algoritmo tenha convergido quando a taxa absoluta de variação do erro quadrático médio (Eq. 2.36) for suficientemente pequena
    3. Considera-se que o algoritmo tenha convergido quando o desempenho da rede para um conjunto de padrões não vistos durante o treinamento for satisfatório (generalização)

2.3.3. Projeto de MLPs

- **Número de camadas ocultas**
  - Função implementada por cada neurônio é formada pela combinação das funções implementadas por neurônios da camada anterior
    - » Camada oculta 1: hiperplanos no espaço de decisão
    - » Camada 2 (oculta ou de saída): regiões convexas
      - ❑Número de lados = número de unidades na camada anterior
    - » Camada oculta 3 (oculta ou de saída): Combinações de figuras convexas, produzindo formatos abstratos
      - ❑Número de figuras convexas = número de unidades da camada anterior

2.3.3. Projeto de MLPs

- **Número de camadas ocultas**



http://playground.tensorflow.org

2.3.3. Projeto de MLPs

- **Número de camadas ocultas**
  - O MLP pode ser visto como um veículo prático para realizar mapeamentos não-lineares de entrada-saída de maneira geral
  - Teorema da aproximação universal
    - » *Uma única camada oculta é suficiente para o MLP realizar o mapeamento entrada-saída com um erro ε >0 para um dado conjunto de treinamento*

2.3.3. Projeto de MLPs

- **Número de neurônios por camada oculta**
  - Em geral não é conhecido
  - Depende de
    - » Número de entradas (dimensão do espaço de entradas)
    - » Número de saídas
    - » Geometria e número das FD em problemas de classificação
    - » Número de bases necessárias para aproximar a função mapeada

2.3.3. Projeto de MLPs

- **Número de neurônios por camada oculta**
  - Porque não utilizar então um grande número de neurônios em cada camada
    - » Quanto mais neurônios, mais lentos são o treinamento e a operação do MLP, e maior é a exigência de *hardware*
  - Em geral, o número de neurônios em cada camada oculta deve ser maior que o número de entradas
    - » Aumento da dimensão do problema
  - No entanto, em problemas que exigem a compactação dos dados, o número de neurônios em cada camada oculta deve ser menor que o número de entradas

2.3.3. Projeto de MLPs

- **Número de neurônios por camada oculta**
  - Algumas regra heurísticas (para uma única camada oculta)

$$n_h = 2\, n_a + 1$$

$$n_h = n_a + n_c$$

$$n_h = (n_a + n_c) / 2$$

$$n_h = n_c$$

n_a : número de atributos (entradas)

n_c : número de classes

n_h : número de neurônios na camada oculta

2.3.3. Projeto de MLPs

- **Funções de ativação**
  - Deve ser diferenciável
  - Normalmente utiliza-se funções sigmóides
    - » Função logística ou *logsig* (saídas entre 0 e 1)
    - » Função tangente hiperbólica (saídas entre -1 e 1)
  - Normalização das entradas e saídas desejadas entre os valores máximos e mínimos das saídas das funções de ativação
    - » Entre outras vantagens, evita a saturação dos neurônios nas camadas ocultas
    - » Exemplo: normalização entre 0 e 1 dos dados de entrada e entre 0,2 e 0,8 (ou 0,1 e 0,9) dos dados de saídas desejadas para a função logística

2.3.4. Reconhecimento de Padrões

- **Formulação Estatística de Classificadores**

  - Considere um classificador com um único atributo de entrada. Assuma que
    - » Exemplo *x* é uma variável aleatória
    - » Para cada classe é assumida uma função densidade de probabilidades (pdf)
      - ❑Por exemplo, a distribuição normal
    - » Segundo Fisher, o **classificador ótimo** é aquele que escolhe a classe c_i que maximiza a probabilidade *a posteriori* $P(c_i \mid x)$ de que o exemplo *x* pertença à classe c_i , i. e,
      - ❑Exemplo *x* pertence à classe c_i se

$$P(c_i \mid x) > P(c_j \mid x) \text{ para todo } j \neq i$$

2.3.4. Reconhecimento de Padrões

- **Formulação Estatística de Classificadores**

  - Apesar de a probabilidade *a posteriori* não pode ser medida diretamente, ela pode ser estimada através da regra de Bayes

$$P(c_i \mid x) = \frac{p(x \mid c_i)\, P(c_i)}{P(x)} \qquad (2.44)$$

  - sendo
    - » $P(c_i)$: probabilidade *a priori* da classe c_i
    - » $p(x|c_i)$: probabilidade de que x foi produzido pela classe c_i
    - » $P(x)$: probabilidade *a priori* do exemplo x

2.3.4. Reconhecimento de Padrões

[PRINCIPE *et al.*, 2000]

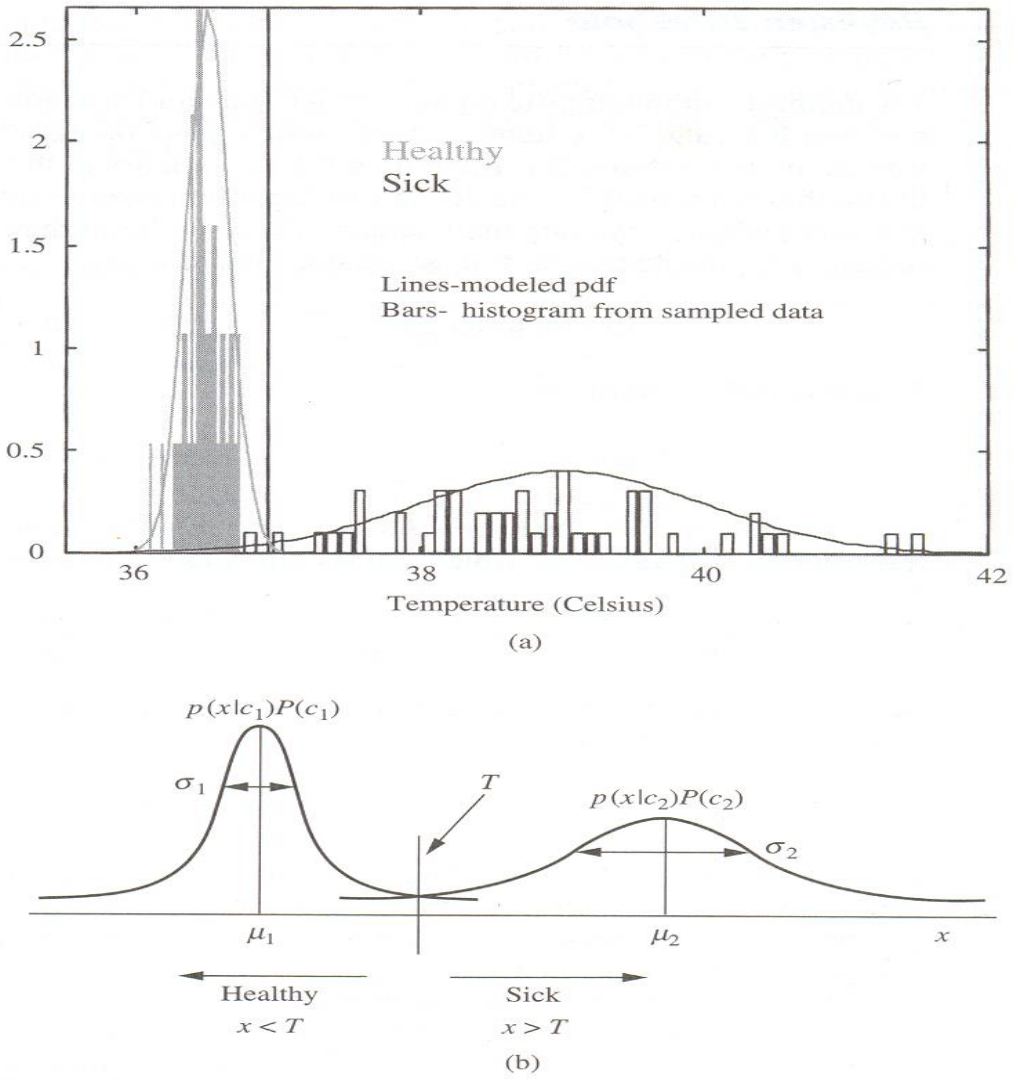


FIGURE 2-2 (a) Sampled data distributions, (b) Bayes threshold l

2.3.4. Reconhecimento de Padrões

- **Treinamento de classificadores**
  - Paramétrico
    - » Uso de funções densidade de probabilidades e seus parâmetros,
      - ❑Ex.: Distribuição Normal
  - Não-paramétrico
    - » Utilização de algoritmos iterativos para alocar funções discriminantes
      - ❑Deve-se decidir a forma da função discriminante
  - Semi-paramétrico
    - » Fronteiras de decisão são formadas através de várias funções discriminantes simples
      - ❑Ex.: MLP e redes RBF

2.3.4. Reconhecimento de Padrões

- **Interpretação probabilística da função de ativação logística**
  - No caso de duas classes, c_1 e c_2, e do vetor de entradas **x**, o denominador da Eq. (2.44) pode ser escrito como $P(\mathbf{x})=p(\mathbf{x}|c_1)P(c_1)+p(\mathbf{x}|c_2)P(c_2)$ e a Eq. (2.44) fica

$$P(c_1 \mid \mathbf{x}) = \frac{1}{1+ p(\mathbf{x} \mid c_2) P(c_2) / p(\mathbf{x} \mid c_1) P(c_1)} \qquad (2.45)$$

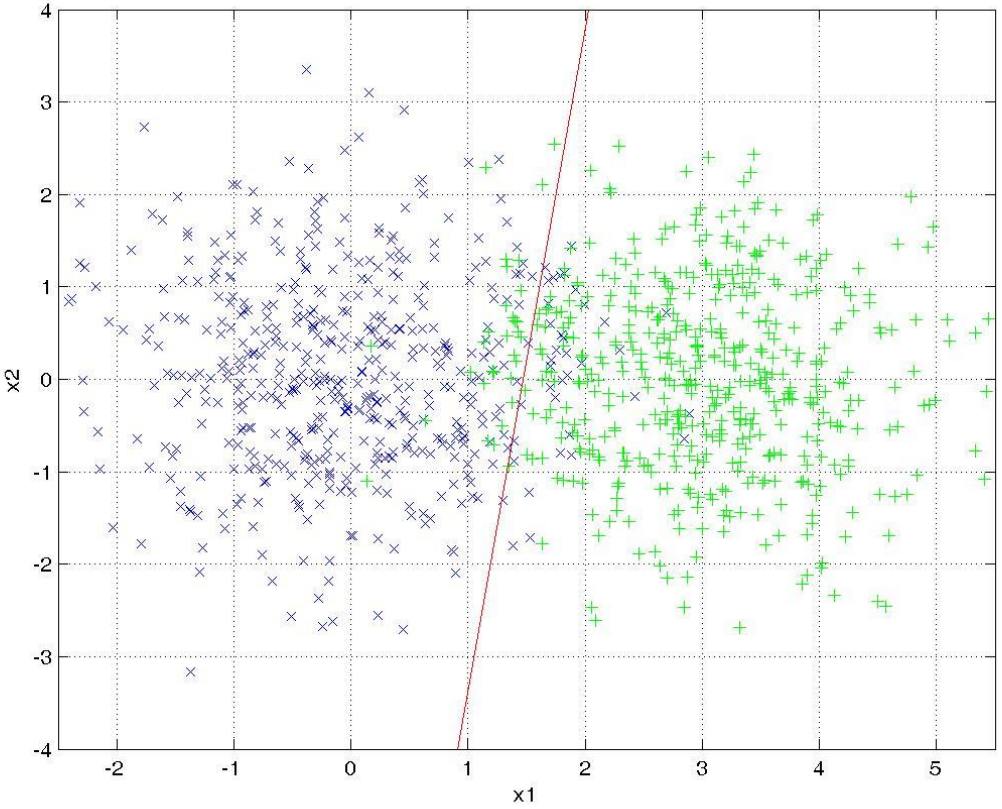
  - Considerando que as distribuições de probabilidades são do tipo normal e com variâncias iguais, então:
    - » A equação anterior reproduz a equação logística
    - » A saída do neurônio pode ser interpretada como a probabilidade *a posteriori* da classe c_1 dado o vetor de entrada

2.3.4. Reconhecimento de Padrões

- **MLP como classificador ótimo**
  - Se assumirmos que um MLP com uma única camada oculta tem um número suficiente de neurônios ocultos, que o Erro Médio Quadrático (EMQ) é otimizado durante o treinamento, que o conjunto de treinamento é **representativo** do problema real e que usamos uma função de ativação que resulta em soma igual a 1 de todas as saídas (*softmax*), então (ver pag. 163-165 de [PRINCIPE *et al*., 2000]) :
    - » A saída y_k fornece uma estimativa para a probabilidade condicional da saída desejada d_k, dados os exemplos de entrada
    - » A saída do MLP fornece a probabilidade *a posteriori* da classe, dado o exemplo de entrada **x**
      - ❑Reflexo da minimização do EMQ

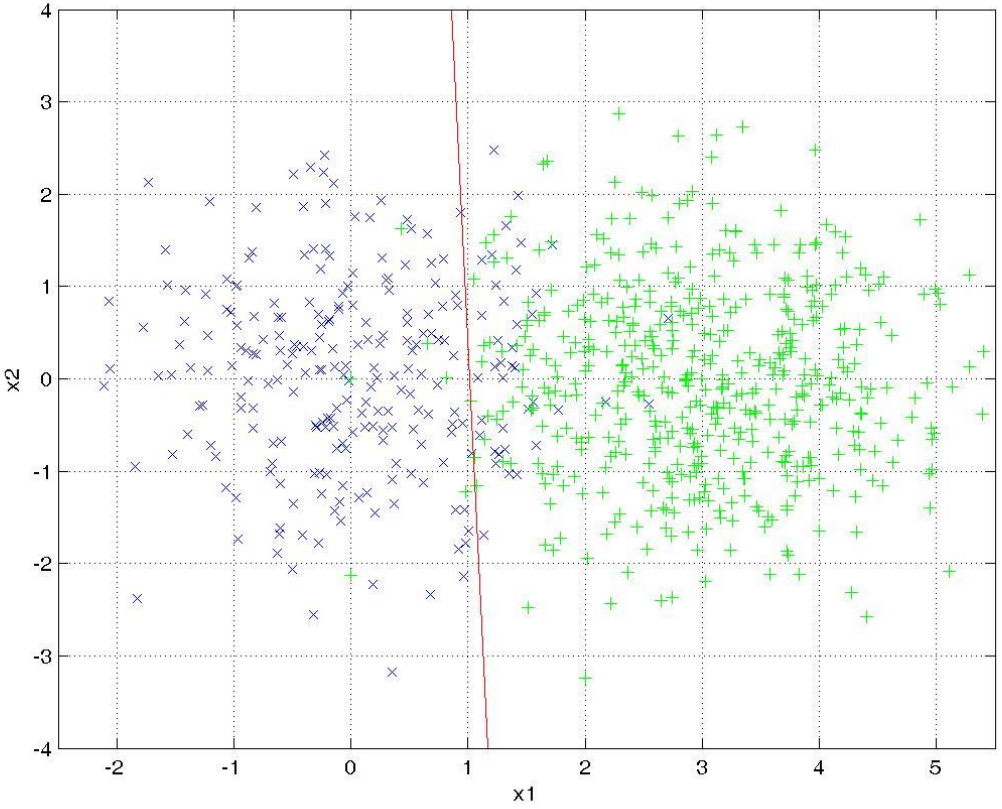
2.3.4. Reconhecimento de Padrões

- **Conjunto de Treinamento deve ser representativo**
  - Exemplo
    - » $P(C_1)= P(C_2)$
    - » Perceptron
      - ❑Classe 1 (‘x’)
        - ➢N_1=500
      - ❑Classe 2 (‘+’)
        - ➢N_2=500



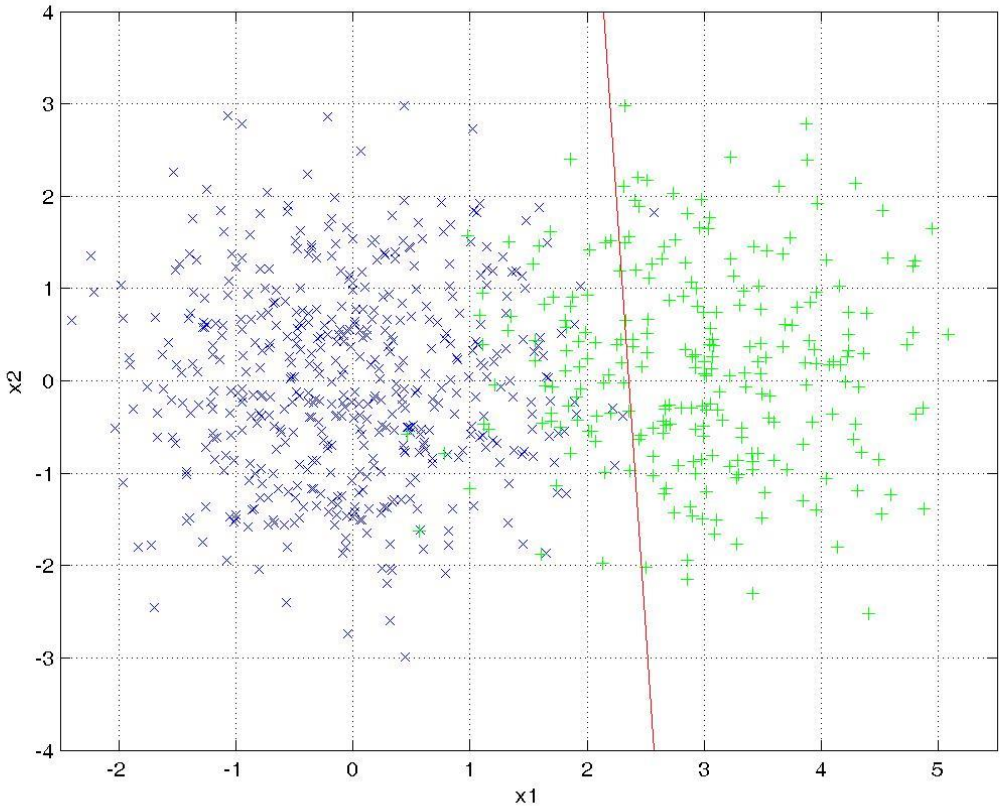
2.3.4. Reconhecimento de Padrões

- **Conjunto de Treinamento deve ser representativo**
  - Exemplo
    - » $P(C_1)<P(C_2)$
    - » Perceptron
      - ❑Classe 1 (‘x’)
        - ➢N_1=250
      - ❑Classe 2 (‘+’)
        - ➢N_2=500



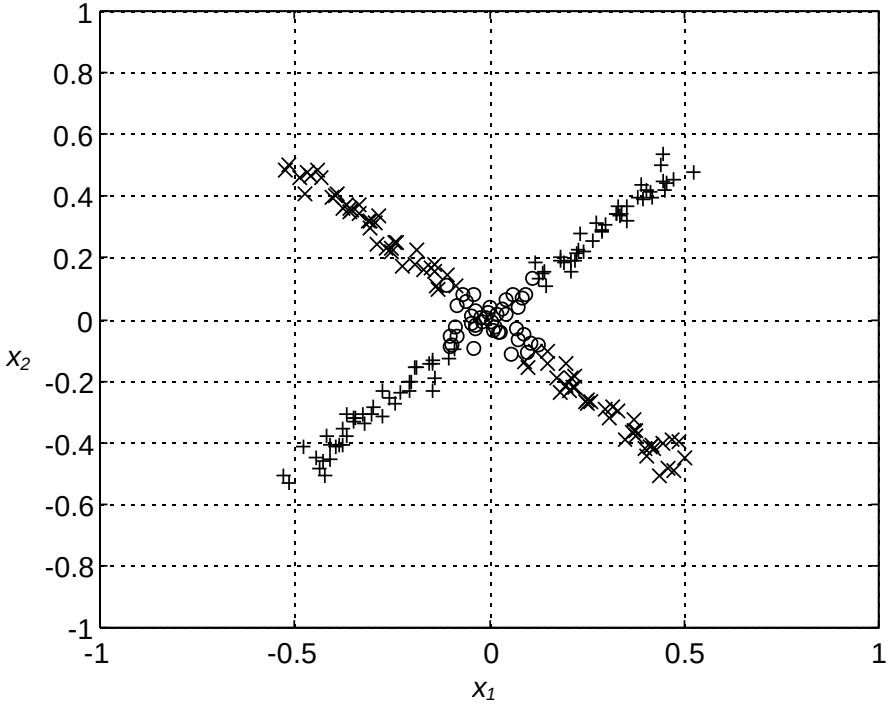
2.3.4. Reconhecimento de Padrões

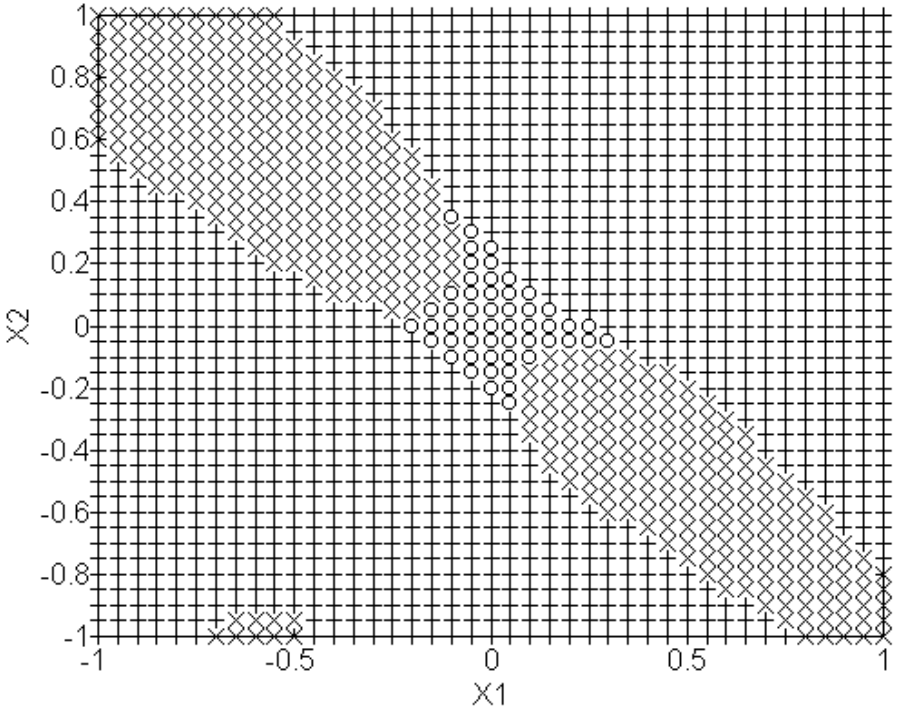
- **Conjunto de Treinamento deve ser representativo**
  - Exemplo
    - » $P(C_1) > P(C_2)$
    - » Perceptron
      - ❑Classe 1 (‘x’)
        - ➢N_1=500
      - ❑Classe 2 (‘+’)
        - ➢N_2=250



2.3.4. Reconhecimento de Padrões

- **Conjunto de Treinamento deve ser representativo**
  - As fronteiras de decisão são otimizadas de acordo com o EMQ sobre o Conjunto de Treinamento
    - » O que pode gerar fronteiras não-robustas e não-intuitivas
    - » Exemplo: espaço de classificação gerado pelo MLP





2.3.5. Aproximação de Funções

- **A tarefa de aproximação de funções procura descrever o comportamento de funções complicadas através da composição de funções simples**
  - Exemplos
    - » Uso de polinômios
    - » Séries de Taylor
      - ❑Calcula o valor de uma função perto de uma vizinhança

2.3.5. Aproximação de Funções

- **Projeção Linear**
  - Combinação linear de funções elementares (bases)

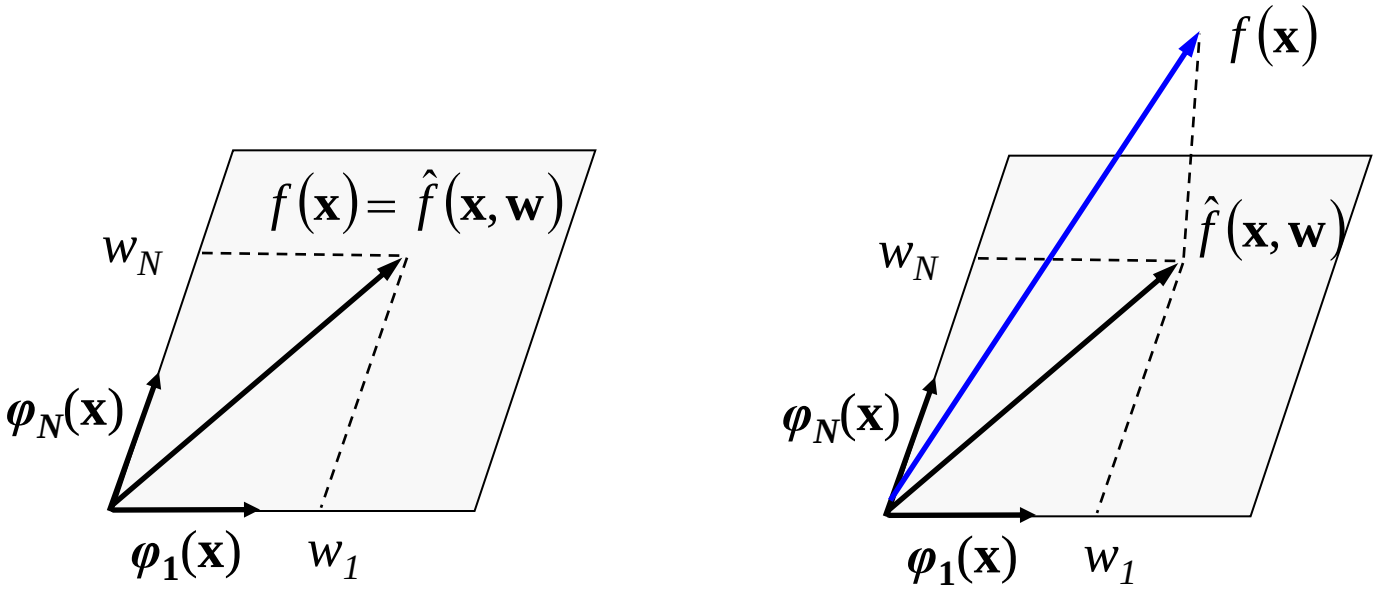
$$\hat{f}(\mathbf{x},\mathbf{w}) = \sum_{i=1}^{p} w_i \varphi_i(\mathbf{x}) \qquad (2.46)$$

sendo **w** um vetor de números reais escolhido de tal forma que, para um valor real ε suficientemente pequeno,

$$\left| f(\mathbf{x}) - \hat{f}(\mathbf{x},\mathbf{w}) \right| < \varepsilon \qquad (2.47)$$

2.3.5. Aproximação de Funções

- **Interpretação geométrica da projeção linear [PRINCIPE *et al*., 2000])**



2.3.5. Aproximação de Funções

- **Exemplos de técnicas de projeção linear**
  - Análise de Fourier
    - » Funções elementares: exponenciais complexas
      - ❑ $\varphi_i = e^{-j\,(2\pi/T)\,i\,t}$
      - ❑ Ou funções trigonométricas
  - *Wavelets*
    - » Funções elementares: funções *wavelet*

2.3.5. Aproximação de Funções

- **Exemplos de técnicas de projeção linear**
  - MLP com
    - » Camada de saída: ativação linear
    - » Camada oculta (uma): ativação não-linear
      - ❑ Ex.: função logística
    - » Funções elementares globais
  - Redes RBF
    - » Funções elementares locais

2.3.5. Aproximação de Funções

- **Projeção linear para o caso do MLP**
  - Utiliza um conjunto de bases adaptativas
    - » As bases não são pré-determinadas
      - ❑Como na Análise de Fourier ou nas Wavelets
    - » Bases são determinadas a partir do conjunto de dados entrada/saída (treinamento)
    - » Bases dependem dos pesos da camada oculta e das entradas
      - ❑Bases mudam com as entradas
    - » Pesos da camada de saída são ajustados para achar a melhor projeção da saída
    - » Treinamento é difícil porque deve-se achar as melhores
      - ❑Bases
      - ❑Projeções

2.3.5. Aproximação de Funções

- **MLP: aproximador universal de funções**
  - *Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function", Mathematics of Control, Signals, and Systems, 2, p.303-312.3.":*
    - » *Seja φ qualquer função de ativação sigmoidal contínua. Então, dada qualquer função contínua com valores reais f (.) em um subespaço compacto s $\subset \Re^p$ e $\varepsilon > 0$, existem vetores $\boldsymbol{\omega_1}$, $\boldsymbol{\omega_2}$,..., $\boldsymbol{\omega_m}$, $\boldsymbol{\alpha}$, $\boldsymbol{\theta}$ e uma função parametrizada $G(.,\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\theta}):\rightarrow\Re$ tal que*

$$\left|G(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\theta}) - f(\mathbf{x})\right| < \varepsilon$$

$$G(\mathbf{x},\boldsymbol{\omega},\boldsymbol{\alpha},\boldsymbol{\theta}) = \sum_{j=1}^{m} \alpha_j \varphi\left(\boldsymbol{\omega}_j^{\mathrm{T}}\mathbf{x} + \theta_j\right)$$

nas quais $\boldsymbol{x} \in \Re^p$, $\boldsymbol{\omega_j}=[\omega_{j1}...\omega_{jp}]^T$, $\alpha=[\alpha_1...\alpha_m]^T$ e $\boldsymbol{\theta}=[\theta_1...\theta_m]^T$.

2.3.6. Generalização

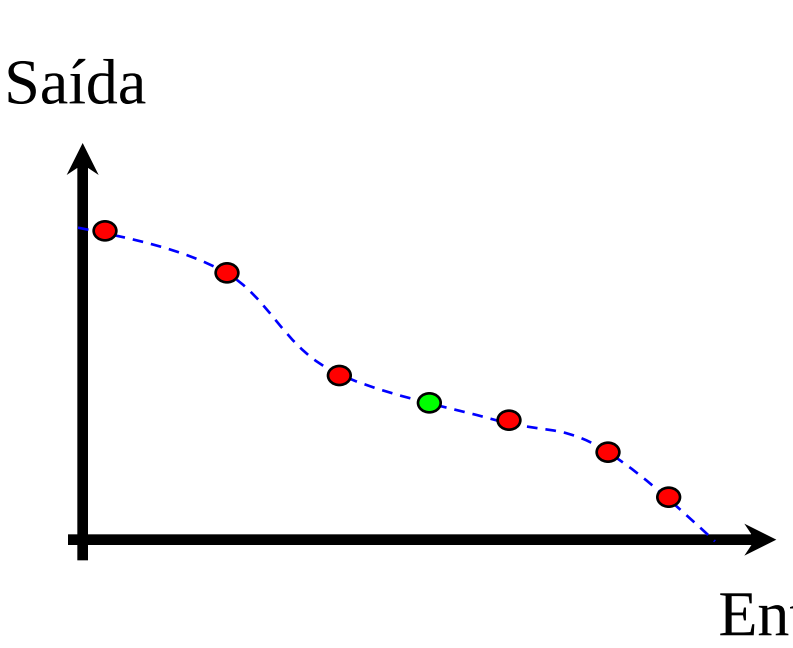
- **Uma RNA generaliza quando seu desempenho for aceitável para dados de teste não apresentados durante o treinamento**
- **O processo de aprendizagem pode ser visto com um problema de "ajuste de curva"**
  - A generalização passa então a ser vista como o efeito de uma interpolação não-linear adequada sobre os dados de entrada

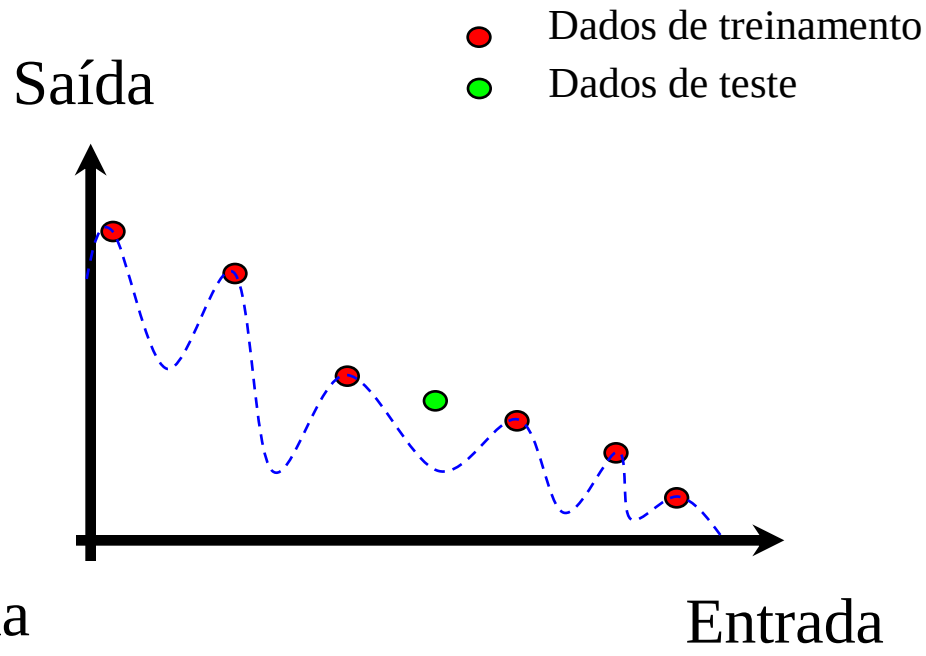
2.3.6. Generalização

- **Excesso de ajuste ou de treinamento (*Overfitting*)**
  - Quando a RNA é treinada em excesso, o mapeamento “memoriza” as saídas desejadas para os padrões utilizados no treinamento
    - » Memoriza, desta forma, os dados espúrios e as variações nos dados de entrada (por exemplo, decorrentes de ruído) presentes do conjunto de treinamento

2.3.6. Generalização

Exemplo





2.3.6. Generalização

- **A generalização é influenciada por**
  - Tamanho do conjunto de treinamento (*N*)
  - Arquitetura da RNA
  - Complexidade física do problema tratado
- **Tamanho suficiente do conjunto de treinamento para uma generalização válida**
  - Heurística

$$N = O\left(\frac{W}{\varepsilon}\right) \qquad (2.48)$$

Na qual *W* é o número de parâmetros livres da rede, ε representa a fração de erros de classificação permitida sobre os dados de teste, e *O*(.) representa a ordem da quantidade entre parênteses

Comentários

- **Referências**
  - **Haykin, S. S.. *Redes neurais: princípios e prática.* 2ª ed., Bookman, 2001.**
    - » Capítulo 4
  - **Braga, A.P.; Carvalho, C.P.L.F. & Ludermir, T.B.. *Redes neurais artificiais: Teoria e Aplicações.* LTC, 2000.**
    - Capítulo 4
  - **Principe, J. C.; Euliano, N. R. & Lefebvre, W. C. *Neural and Adaptive Systems: Fundamentals Through Simulations*. John Wiley & Sons, Inc. 2000**
    - Capítulos 2, 3, 4 e 5