LABIC

SCE-5809 - REDES NEURAIS
Redes Neurais Multi-Camadas – Parte 2

Profa. Roseli Ap. Francelin Romero

LABIC

Teorema de Aprox. Universal

- Qual é o número mínimo de camadas num PMC que fornece uma aproximação para qualquer mapeamento contínuo?
- Cybenko, 1989 – mostrou pela primeira vez que uma rede com uma única camada intermediária é suficiente para aproximar uniformemente qualquer função contínua definida num hipercubo unitário.

Teorema de Aprox. Universal

Teorema: Seja g(.) uma função contínua limitada, monótona estritamente crescente. Seja Ip, um hipercubo unitário p-dimensional e C(Ip): o espaço das funções contínuas em Ip. Então, dado qualquer função $f \in C(Ip)$ e $\varepsilon>0$, existe um inteiro M e constantes reais α_i, θ_i, w_{ji}, onde i=1,2,...,M e j=1,...,p, tal que se pode definir:

$$F(x_1,...,x_p) = \sum \alpha_i \, g \left(\sum w_{ji} \, x_j - \theta_i\right) \qquad (1)$$

com

$$| F(x_1,...,x_p) - f(x_1,...,x_p) | < \varepsilon \quad \{x_1,...,x_p\} \in Ip$$

Teorema de Aprox. Universal

- a função sigmoid ou logística é contínua, estritamente cresc. e limitada e portanto satisfaz as condições impostas na função g(.).
- A equação (1) representa a saída de PMC:

  - a rede tem p nós de entrada e uma única camada intermediária de M nós.
- O neurônio i tem pesos: w_{1i}, ..., w_{pi} e limiar θ_i
- A saída da rede é uma C.L. das saídas dos neurônios intermediários, com α_i

LABIC

Teorema de Aprox. Universal

- O teorema é um teorema de Existência, pois ele fornece uma justificativa para a aprox. de funções contínuas. SUFICIENTE
- Entretanto, o teorema não afirma que uma única camada é um número OTIMO.

LABIC

Teorema de Aprox. Universal

- Na prática, nem sempre se dispõe de uma função contínua e nem de uma camada intermediária de tamanho qualquer.
- Chester, 1990 e Funahashi, 1989, defendem o uso de duas camadas interm., pois, torna a aprox. mais maleável.

Teorema de Aprox. Universal

- Caracter. Locais são extraídas na primeira camada. Alguns neurônios na primeira camada são usados para particionar o espaço em várias regiões, e outros aprendem as caract. locais daquelas regiões.
- Caracter. Globais são extraídas na segunda camada. Um neurônio na 2a. Camada combina as saídas de neurônios da primeira que estão operando numa região particular do espaço de entrada e assim aprende caract. globais daquela região.

LABIC

Velocidade de Aprendizado

- O algoritmo **BP** fornece uma “aproximação” para a trajetória no espaço dos pesos
- Quanto menor o valor de η, menor as mudanças nos pesos e mais suave será a trajetória. APRENDIZADO LENTO
- Se, η é muito grande, o aprendizado torna-se rápido então a rede pode tornar-se INSTÁVEL

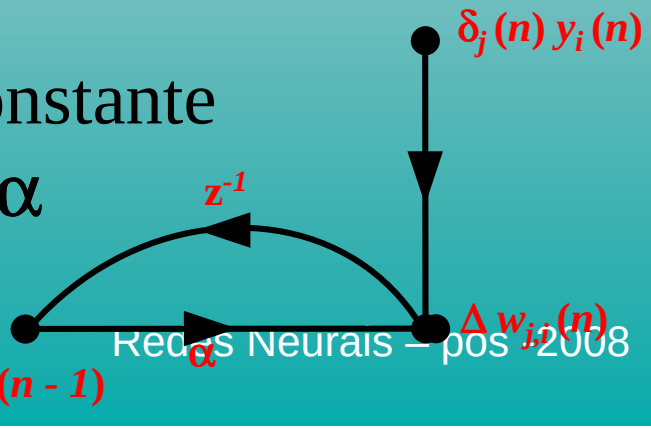
Termo Momentum

É um método simples de aumentar a velocidade de aprendizado e evitar o perigo de instabilidade, como mostrado por Rumelhart et al., l986.

$$\Delta w_{j,i}(n) = \eta\, \delta_j(n)\, out_i(n) + \alpha\, \Delta w_{j,i}(n-1) \quad (\alpha)$$

onde α é geralmente um número positivo chamado CONSTANTE MOMENTUM

Efeito da Constante Momentum α



A equação (α) é chamada *REGRA DELTA GENERALIZADA*. Se α =0 $\Rightarrow$ REGRA DELTA

Efeito da Constante α

Vamos considerar uma série de tempo com índice t (de **0** a n)

A equação (α) pode ser vista como uma equação diferença de primeira ordem em relação a $\Delta w_{j,i}(n)$. Resolvendo

$$\Delta w_{j,i}(n) = \eta \sum_{t=0}^{n} \alpha^{n-t} \delta_j(t)\, out_i(t)$$

que representa uma série de tempo comprimido $n+1$. Mas:

$$\delta_j(n)\, out_i(n) = -\frac{\partial E(n)}{\partial w_{j,i}(n)} \qquad \therefore \Delta w_{j,i}(n) = -\eta \sum_{t=0}^{n} \alpha^{n-t} \frac{\partial E(t)}{\partial w_{j,i}(t)}$$

1. O ajuste atual $\Delta w_{j,i}(n)$ representa a soma de uma série de tempo ponderada exponencialmente convergente $\Rightarrow 0 \leq |\alpha| < 1$

Efeito da Constante α

2. Quando $\frac{\partial E(t)}{\partial w_{j,i}(t)}$ tem o mesmo sinal algébrico em iterações consecutivas, então a série cresce em magnitude então os pesos são ajustados por uma quantia grande

Portanto **BP** tende a acelerar a "descida" nas regiões de descida da superfície do erro.

3. Quando $\frac{\partial E(t)}{\partial w_{j,i}(t)}$ tem sinais opostos em iterações sucessivas então a série diminui em magnitude e $\Delta w_{j,i}(n)$ é atualizado por uma quantia pequena. Então a inclusão do termo momentum tem o ***"efeito de estabilização"*** nas direções em que o sinal oscila.

Efeito da Constante α

Portanto o termo momentum pode ter efeitos benéficos no comportamento do aprendizado do algoritmo, ele pode evitar que o processo termine num mínimo local na superfície de erro.

OBS.: o parametro η foi considerado constante

(1) n_{ji} **:** dependente da conexão. Fatos interessantes ocorrem se n_{ji} é tomado diferente em diferentes partes do algoritmo

(2) restringir o número de pesos a serem ajustados $n_{ji} = 0$ para o peso $w_{j,i}$

(3) Modo no qual as camadas ocultas são interconectadas. No procedimento, supomos que cada camada recebe

Efeito da Constante α

entradas apenas das unidades da **camada anterior.** Mas não existe nenhuma RAZÃO para isto. Se este não for o caso, existem dois tipos de sinais de erro:

- Um sinal de erro que resulta de comparação direta do sinal saída daquele neurônio com uma resposta desejada.
- Um sinal de erro que é passado através de outras unidades cuja ativação ele afeta.

Modos de Treinamento

Aprendizado **BP** resulta de muitas apresentações de um conjunto de treinamento de exemplos. Uma apresentação COMPLETA do conjunto de treinamento inteiro ⇒ 1 CICLO (1 epoch)

O processo de aprendizagem é repetido CICLO após CICLO até que os pesos sinápticos e níveis threshold se **estabilizam**.

Tomar os pesos numa forma ALEATÓRIA ⇒ pesquisa no espaço dos pesos ESTOCÁSTICA.

1) MODO PADRÃO/PADRÃO

Atualização nos pesos é feita após a apresentação de cada exemplo de treinamento. Um ciclo consistindo de ***N*** exemplos

Modos de Treinamento

de treinamento arranjados na ordem:

$$\{[\, x(1), d(1)], \ldots\ldots, [\, x(N), d(N)]\}$$

$[\, x(1), d(1)] \rightarrow$ Cálculos forward/backward atualização pesos/threshold

$[\, x(2), d(2)] \rightarrow$ Cálculos forward/backward atualização pesos/threshold

.
.

Desta forma, a variação média nas mudanças dos pesos é :

$$\Delta \hat{w}_{j,i} = 1/N \sum_{n=1}^{N} \Delta w_{j,i}(n) = -\eta/N \sum_{n=0}^{N} \frac{\partial E(n)}{\partial w_{j,i}(n)} = -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j(n)}{\partial w_{j,i}(n)}.$$

Modos de Treinamento

2) MODO BATCH

Atualização dos pesos é feita depois da apresentação de todos os exemplos de treinamento que constituem um ciclo.

Para um ciclo particular, função custo com o erro quadrático médio: $\mathcal{E}_{av} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n)$ onde **C** denota o conjunto de índices correspondentes aos neurônios da camada de saída. e_j é o sinal do erro neurônio j correspondente ao exemplo de treinamento w.

$$\Delta w_{j,i} = -\eta \frac{\partial \mathcal{E}_{av}}{\partial w_{j,i}} = -\frac{\eta}{N} \sum_{n=1}^{N} e_j(n) \frac{\partial e_j(n)}{\partial w_{j,i}}$$

Conjunto inteiro ter sido apresentado

Modos de Treinamento

Claramente $\Delta \hat{w}_{j,i}$ é diferente de $\Delta w_{j,i}$. $\Delta \hat{w}_{j,i}$ representa uma ***estimativa*** $\Delta w_{j,i}$. Do ponto de vista "on-line" o **MODO PADRÃO** é o mais preferido. Além disso, os exemplos de treinamento são aleatoriamente apresentados $\Rightarrow$ atualização nos pesos é ESTOCÁSTICA $\Rightarrow$ menos provável o algoritmo BP estacionar num **MÍNIMO LOCAL**

Por outro lado, **MODO BATCH** fornece uma estimativa mais precisa do ***VETOR GRADIENTE***. De qualquer forma, a eficiência dos dois modos depende do problema que se tem em mãos(Hertz,1991)

CRITÉRIO DE PARADA

Não se pode, em geral, mostrar a CONVERGÊNCIA do algoritmo **BP** e nem existem critérios bem definidos para parar seu processamento. Para formular um critério ⇒ propriedades de mínimo local ou global da superfície de erro.

Seja ***w**** denotar o vetor mínimo local ou global

1) Uma condição necessária para ***w**** ser mínimo:

gradiente $\boldsymbol{g(w)} \rightarrow$ (derivada de 1ª ordem) da superfície de erro com relação a ***w*** seja ***zero*** em $\boldsymbol{w = w^*}$.

"O algoritmo **BP** é considerado ter convergido quando a norma do vetor gradiente é menor que um certo **ε**, **ε** peq. arbitrário."

CRITÉRIO DE PARADA

2) Função custo $\varepsilon_{av}(w)$ é estacionária em $w = w^*$.

"O algoritmo **BP** é considerado ter convergido quando a taxa de mudança no erro quadrático médio por ciclo é suficientemente pequena"

Tipicamente, é considerado pequena uma taxa de mudança no erro quadrático de **0.1** a **1%** por ciclo

ou **0.01%** por ciclo

Kramer e Sangiovanni-Vicentelli(1989) sugere um critério de convergência:

O algoritmo BP termina no vetor peso w_{final} *quando* $\| g(w_{final}) \| \leq \varepsilon$, onde ε é suficiente pequeno, ou $\| \varepsilon_{av}(\text{ final }) \| \leq \tau$ onde τ é suficiente pequeno.

LABIC

INICIALIZAÇÃO

O primeiro passo do algoritmo **BP** é a inicialização da rede.

Uma boa escolha para os parâmetros livres(pesos sinápticos e threshold) podem contribuir significativamente no sucesso do aprendizado.

- **Informação disponível**
- **Nenhuma informação é disponível ?**

  Pesos aleatoriamente, isto é, inicializar os pesos com valores uniformemente distribuídos num intervalo pequeno.
- **Escolha Errada ⇒ Saturação Prematura**

  Esse fenômeno refere-se a uma situação onde o erro quadrático permanece constante por um período de tempo, mas depois continua a diminuir depois que este período é concluído.

LABIC

INICIALIZAÇÃO

Vários fatos interessantes podem ocorrer:

1) Supor que para um particular padrão de treinamento, o nível de ativação interna de um neurônio saída tem um valor cuja **magnitude é grande**. Como a função é a "sigmoid" $\Rightarrow$ ***y* = 1 *ou* *y* = -1.** Em tal caso, diz-se que o neurônio está em **saturação.**

2) Se ***y*** está mais próximo de **1** quando a saída desejada é **-1** ou vice-versa, o neurônio está **incorretamente saturado.** Quando isso ocorre, o ajuste nos pesos será pequeno, embora o erro seja de magnitude grande e a rede levará um longo tempo para corrigir isto (Lee,1991).

3) No estágio inicial de **BP** podem existir neurônios não-saturados e incorretamente saturados.

INICIALIZAÇÃO

Para os não-saturados ⇒ os pesos mudam rapidamente.

Incorretamente saturados ⇒ permanecem saturados por algum tempo

⇒ **Fenômeno de Saturação Prematura** pode ocorrer com **ε** permanecendo constante. Em **LEE(1991)** uma fórmula para a probabilidade de **Saturação Prematura** foi obtida para o **modo Batch**

A essência desta fórmula pode ser: [**Haykin, 1994**]

1) Saturação Incorreta: é evitada escolhendo valores iniciais dos pesos sinápticos e níveis threshold, uniformente distribuídos num intervalo pequeno.

2) Saturação Incorreta: é menos provável quando o número de neurônios intermediários é mantido baixo.

INICIALIZAÇÃO

3) Saturação Incorreta: raramente ocorre quando os neurônios da rede operam em sua regiões lineares

Segundo [**Haykin,1994**], para o **Modo Padrão** de atualização dos pesos, os resultados mostram uma tendência similar ao **modo Batch**

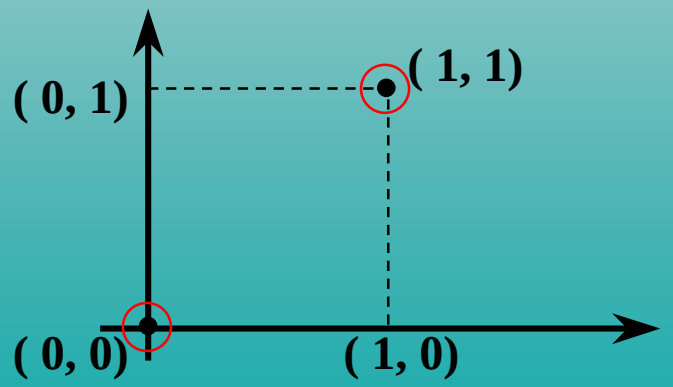
O PROBLEMA XOR

$(0, 0) \rightarrow 0$

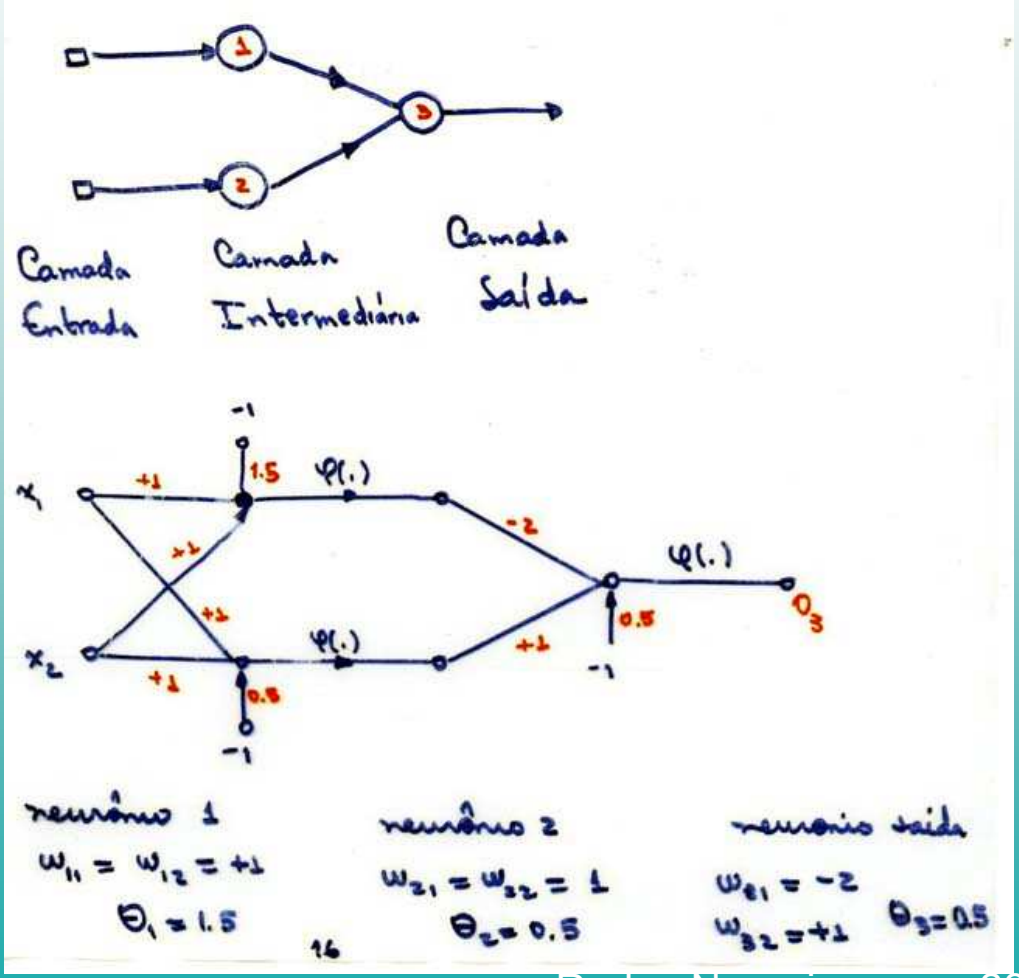
$(1, 0) \rightarrow 1$

$(0, 1) \rightarrow 1$

$(1, 1) \rightarrow 0$



INICIALIZAÇÃO



Verificar o comportamento da rede XOR durante o aprendizado

LABIC

Sugestões de melhoria de desempenho

Existem algumas sugestões para o algoritmo **BP** trabalhar melhor (**RUSSO-1991, GUYON-1991**)

1) Função Ativação: Impar $\varphi(-v) = -\varphi(v)$ Esta condição não é satisfeita pela função **LOGÍSTICA,** mas sim pela função sigmoid não-linear na forma de tangente hiperbólica. Esta função é dada por:

$$\varphi(-v) = a\ tanh\ (\ bv\ )$$

onde ***a*** e ***b*** são constantes. Esta função nada mais é que a função logística transladada:

$$a\ tanh(\ bv\ ) = a\ [\frac{1 - e^{-bv}}{1 + e^{-bv}}] = \frac{2a}{1 + e^{-bv}} - a$$

Valores adequados para ***a*** e ***b*** são: ***a*** **= 1.716** e ***b*** **= 2/3 (Guyon,1991)**

Sugestões de melhoria de desempenho

2) É importante que os valores alvo (resposta desejada **d**) sejam escolhidos no mesmo intervalo onde se encontram os valores assumidos pela função de ativação. Se o valor limite da função sigmoid é: $+a \Rightarrow d_j = a - \varepsilon$, onde ε é uma constante positiva apropriada.

Por exemplo, se a função ativação for a tangente hiperbólica com

$$a=1.716 \Rightarrow \quad \varepsilon = 0.716 \quad \text{e assim} \quad d_j = \pm 1$$

Se isso não for feito o algoritmo **BP** tende a dirigir os parâmetros livres da rede $\rightarrow$: $\Rightarrow$ **processo de aprendizado muito lento.**

3) Inicialização dos parametros livres deve ser aleatória num intervalo pequeno. Contudo, a magnitude do intervalo não deve ser tão pequena, o que pode causar que os gradientes sejam também muito

Sugestões de melhoria de desempenho

pequenos e o aprendizado, portanto, inicialmente muito lento.

Para uma função de ativação impar na forma de uma tangente hiperbólica, com constantes ***a*** e ***b*** dadas acima, uma escolha possivel é:

$$\left(-\frac{2.4}{F_i} , \frac{2.4}{F_i} \right)$$

onde $\mathbf{F_i}$ é o número total de entradas na rede.

4) É desejável que todos os neurônios numa rede multi-camadas ***aprendam na mesma velocidade.*** Tipicamente, as últimas camadas→ gradientes locais maiores que as camadas iniciais. Então, η deve ser menor para as últimas camadas do que para as primeiras camadas. Neurônios com muitas entradas teriam um valor de η menor que com poucas entradas.

LABIC

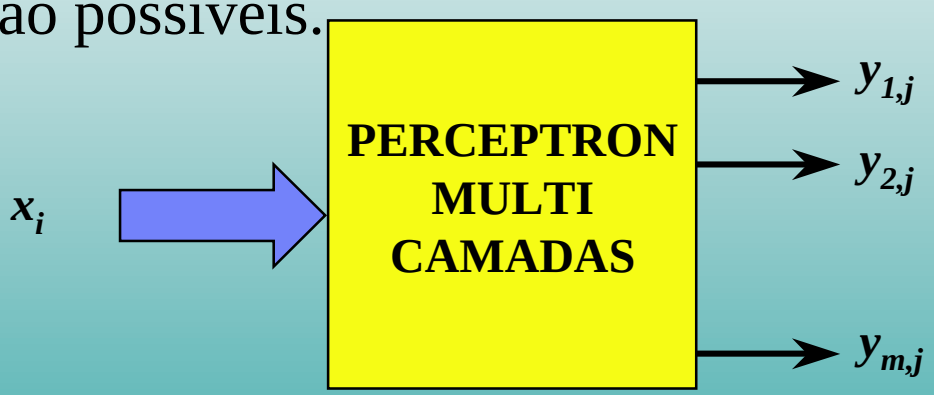
Sugestões de melhoria de desempenho

O modo **PADRÃO** é mais indicado que o modo **BATCH**, na atualização dos pesos. Em **PROBLEMAS CLASSIF.** envolvendo dados grandes e redundantes o modo **PADRAO é mais rápido.**

Entretanto, a atualização padrão é mais dificil para ser **PARALELIZADA.**

Representação 1-de-m da Saida e Regra de Decisão

Na teoria, para um problema de classificação envolvendo ***m-classes***, para o qual a união das ***m*** classes distintas forma o espaço inteiro, precisamos de ***m saidas*** para representar todas as decisões de classificação possíveis.



$$yj = [y_{1,j}, y_{2,j}, \ldots, y_{m,j}]^T = [F_1(x_j), F_2(x_j), \ldots, F_m(x_j)]^T = F(x_j)$$

onde ***F*** e uma função vetorial. " Depois do treinamento do **PMC,** qual seria a regra de decisão para classificar as ***m*** saídas da rede?"

Representação da Saida e Regra de Decisao

Supondo, agora, que a rede seja treinada com valores alvo binários,

isto é: $d_{k,j} = \begin{cases} \textbf{1 quando } x_j \in C_k \\ \textbf{0 quando } x_j \notin C_k \end{cases}$. Baseada, nesta notação, a classe C_k é representada por um vetor alvo m-dimensional **[0,0,0, …, 1,0,0]**

K-ésimo elemento

Podemos classificar o vetor aleatório x como pertencente

a classe C_k se $F_k(x) > F_j(x) \; \forall \; j \neq k$ **(2)** onde $F_k(x)$ e $F_j(x)$ são elementos da função vetorial $F(x) = [F_1(x), F_2(x), \ldots, F_m(x)]$

OBS: Um valor maior único existe com probabilidade **1** quando as distribuições a posteriori são distintas (veremos mais tarde!!!).

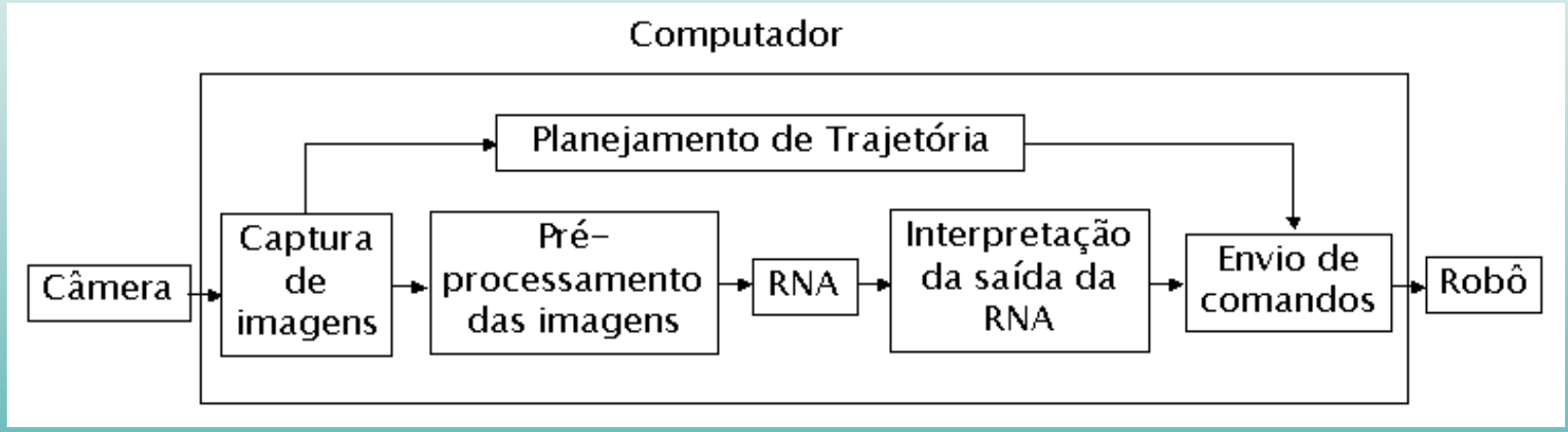
Esta regra de decisão tem a vantagem de produzir decisões não-ambíguas sobre a regra comum que designa x à classe C_k se $F_k(x)$ é maior que algum threshold, o que pode conduzir a múltiplas classes.

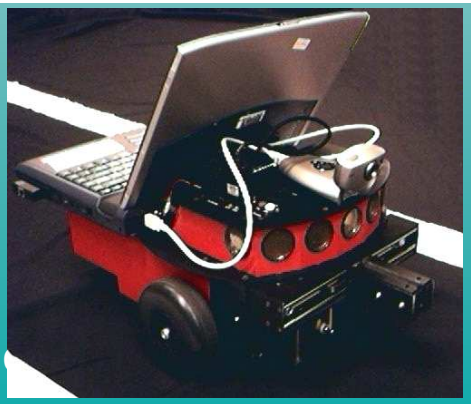
LABIC

Aplicação

- Deseja-se ensinar um robô a andar em um caminho delimitado por faixas brancas.
- Utilizar uma Rede Neural Multi-camadas para ensinar a reconhecer curvas abertas, fechadas e caminhos para seguir em frente.

Estrutura do Sistema





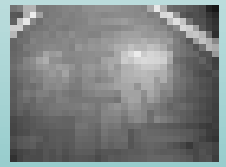
Red

Pré-processamento das Imagens

- Redimensionamento para 32x24 (*pixels*)
  - Tempo de processamento da RNA
  - Informações importantes
  - Média

Pré-processamento das Imagens

- Tons de cinza
  - Informações sobre cor
  - Média das componentes RGB
  - Normalização dos *pixels*
    - Variações de luminosidade
  - Chão escuro e faixas brancas



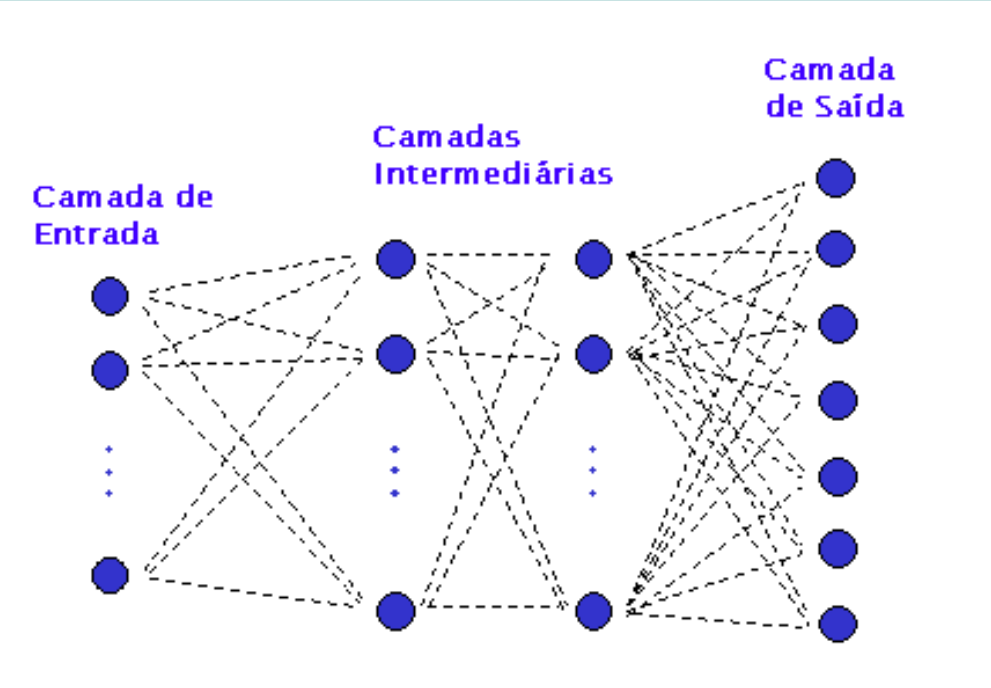
Pré-processamento das Imagens

- Branco e Preto
  - Peculiaridades do piso
  - Elimina dependência de cor
  - Faixa e chão
  - Eliminar ruídos

Redes Neurais Artificiais

Conhecimento: pesos das conexões

- Determinados na fase de treinamento



LABIC

Redes Neurais Artificiais

- Treinamento
  - Supervisionado
    - Pares de treinamento (entrada e saída desejada)
- RNAs *Multilayer Perceptron* (MLP)
  - Diversas camadas
  - Algoritmo de treinamento *Back Propagation*
    - Supervisionado

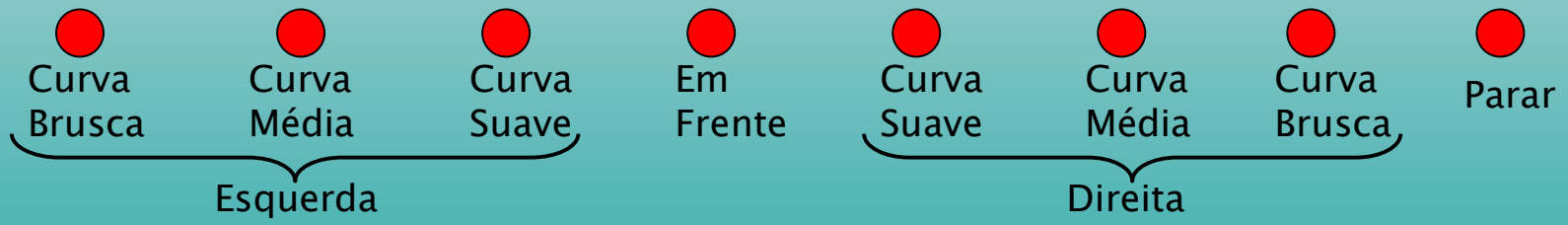
Redes Neurais Artificiais

- Treinamento
  - Ferramenta de simulação SNNS
- 768 elementos de entrada
  - *Pixels* das imagens da pista (32x24)
- Elementos de saída: direções do robô
  - Representação 1-de-n
  - Representação Gaussiana

Redes Neurais Artificiais

Representação 1-de-n

- Definição:
  - N elementos de saída
  - Ativação de apenas um elemento: resposta
- 8 elementos de saída
  - Direções possíveis:



- Cada intensidade de curva – um raio de curvatura
- Saída parar – imagens críticas

Redes Neurais Artificiais

Representação 1-de-n

- Pesquisa anterior
  - Coleta de conjunto de imagens para treinamento
    - Pares de treinamento

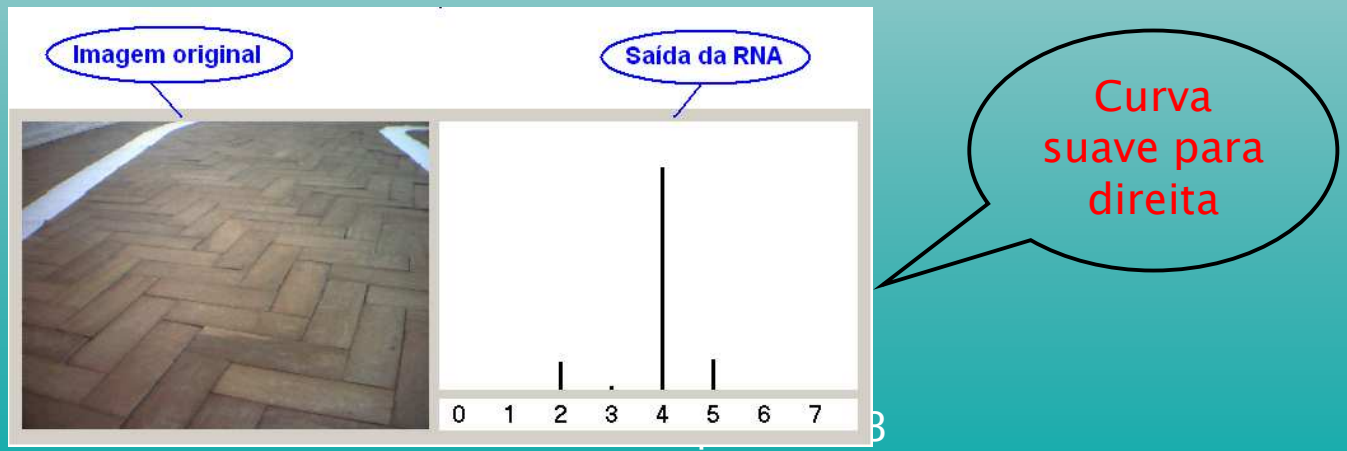
Entrada -> Saída desejada

0 0 0 0 1 0 0 0

Redes Neurais Artificiais

Representação 1-de-n

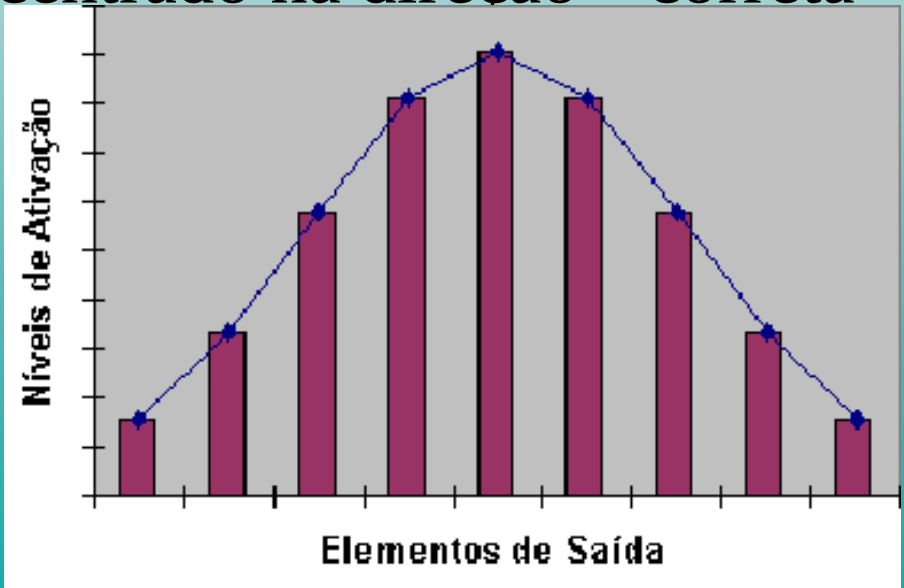
- Interpretação da Saída
  - Maior valor entre os elementos da saída
  - Saída confiável:
    - Maior que um certo limite
    - Suficientemente maior que outros elementos



Redes Neurais Artificiais

Representação Gaussiana

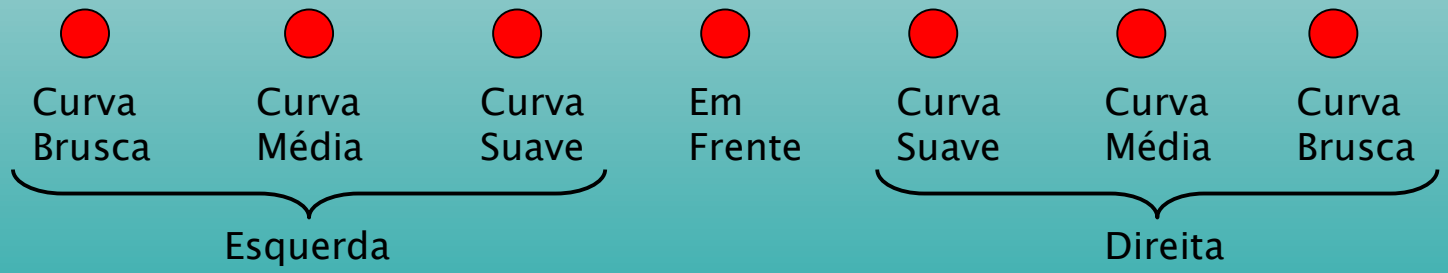
- Todas as saídas ativadas
  - Em forma de função gaussiana
  - Pico centrado na direção correta



Redes Neurais Artificiais

Representação Gaussiana

- Número de elementos de saída: 7
  - Elementos representando ações parecidas níveis de ativação próximos
  - Não há saída representando movimento parar:



Redes Neurais Artificiais
Representação Gaussiana

- Pares de treinamento
  - Saída desejada calculada com equação gaussiana:

$$x_i = e^{-\frac{d_i^2}{5}}$$

  - xi: valor da saída i
  - di: distância entre a saída i e o centro
  - 5: controla abertura
    - Pares de treinamento

Entrada -> Saída desejada

0,16 0,44 0,81 1 0,81 0,44 0,16

Representação Gaussiana

- Seis topologias verificadas
  - Mais adequada com duas camadas intermediárias
    - 1ª camada intermediária: 100 neurônios
    - 2ª camada intermediária: 30 neurônios
    - Melhor resultado comparado com representação 1-de-n

Redes Neurais Artificiais

Representação Gaussiana

- Interpretação da saída
  - Ajustar valores de saída a uma função gaussiana
    - Minimização dos quadrados dos erros
  - Posição do pico – direção correta
    - Várias intensidades de curva

LABIC

Redes Neurais Artificiais

Representação Gaussiana

- Vantagens:
  - Pequenas variações nas imagens – pequenas modificações nas saída da RNA
  - Mapeamento suave entre as direções
    - Respostas arbitrárias de direção
      - Muitos valores possíveis para o raio de curvatura
    - Representação 1-de-n
      - Conjunto limitado de direções

Generalização

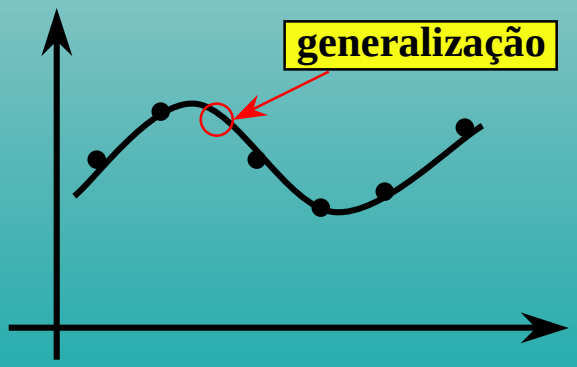
Aprendizado **BP**

conjunto treinamento + algoritmo BP ⇒ pesos sinápticos

GENERALIZAR

"GENERALIZAÇÃO": termo da psicologia.

Processo de aprendizado pode ser visto como um Método de Aproximação de Funcões



: efeito de uma boa aproximação não linear dos dados de entrada, tamanho e eficiência do conjunto treinamento, arquitetura da rede, complexidade física do problema

LABIC

Complexidade da Rede

Problema: Determinar o melhor número de nós na camada intermediária.

Estatisticamente, esse problema é equivalente a determinar o tamanho do conjunto de parâmetros usado para modelar o conjunto de dados. Existe um limite no tamanho da rede.

Esse limite deve ser tomado lembrando que é melhor treinar a rede para **produzir a melhor generalização** do que treinar a rede para representar perfeitamente um conjunto de dados.

Isso pode ser feito usando validação cruzada.

LABIC

Validação Cruzada

- Conjunto de dados:

  treinamento (~75%)

  teste (~25%)

- Conjunto de Treinamento

  1 subconjunto: validação do modelo

  1 subconjunto: treinamento

Validar o modelo num conj. diferente do usado para estimar o modelo

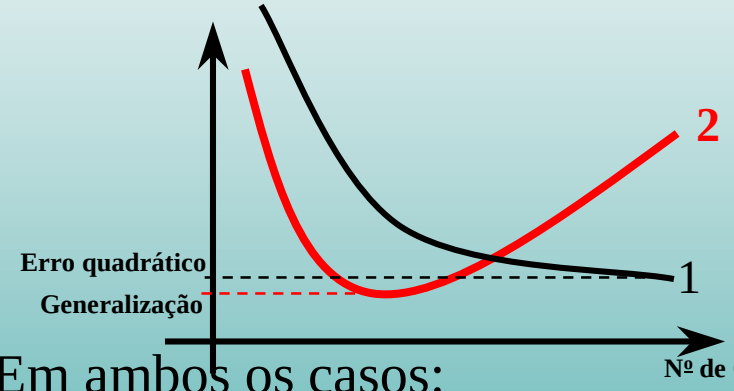
LABIC

Validação Cruzada

- Usa-se o subconjunto de validação para avaliar o desempenho de diferentes candidatos do modelo (dif. Topologias) e então escolhe-se uma delas.
- O modelo escolhido é treinado sobre o conj. treinamento inteiro e a capacidade de generalização é medida no conjunto de teste.

Tamanho do Conjunto Treinamento

A validação cruzada pode ser usada para decidir quando o treinamento de uma rede deveria ser cessado.



Curva 1: poucos parâmetros (under fitting)

Curva 2: muitos parâmetros (over fitting)

Em ambos os casos:

1) O desempenho do erro na generalização exibe um mínimo

2) O mínimo no caso over fitting é menor e mais definido.

Pode-se obter boa generalização se a rede é projetada com muitos neurônios desde que o treinamento é cessado num número de ciclos correspondente ao mínimo da curva do ERRO obtida na Validação Cruzada.

LABIC

Validação Cruzada

- É melhor treinar uma rede para produzir a melhor generalização, usando validação cruzada, do que treinar uma rede para representar perfeitamente um dado conjunto de dados.

LABIC

Tamanho do Parâmetro de Velocidade

Quando o desempenho da rede sobre o conjunto de validação cruzada não melhora (0.5%) $\Rightarrow$ o tamanho de η é reduzido (geral , pela metade).

Depois de cada ciclo η é reduzido até não se conseguir mais melhorar o desempenho de classificação.

Neste ponto o treinamento é terminado.

- Sistema ALVINN

Usou uma RN com uma camada intermed.

-INPUT: Imagem da rodovia (32 x 30 pixels)

-SAÍDA: Direção

-Treinamento: ~10 minutos de direção humana

-Nenhum controle de velocidade

Processo de Aprendizado para RNA

Paradigmas de Aprendizado:

Aprendizado Supervisionado

Aprendizado Não Supervisionado

Aprendizado com Reforço

Aprendizado Hebbiano - Regra de Hebb:

Ajuste aplicado ao peso sináptico no tempo *n*

$$\Delta w_{kj}(n) = \eta y_k(n) x_j(n)$$

- $y_k(n)$ é a saída do neurônio k no tempo n
- $x_j(n)$ o j-ésimo elemento do vetor de entrada no tempo n.
- Regra Anti-Hebbiana:

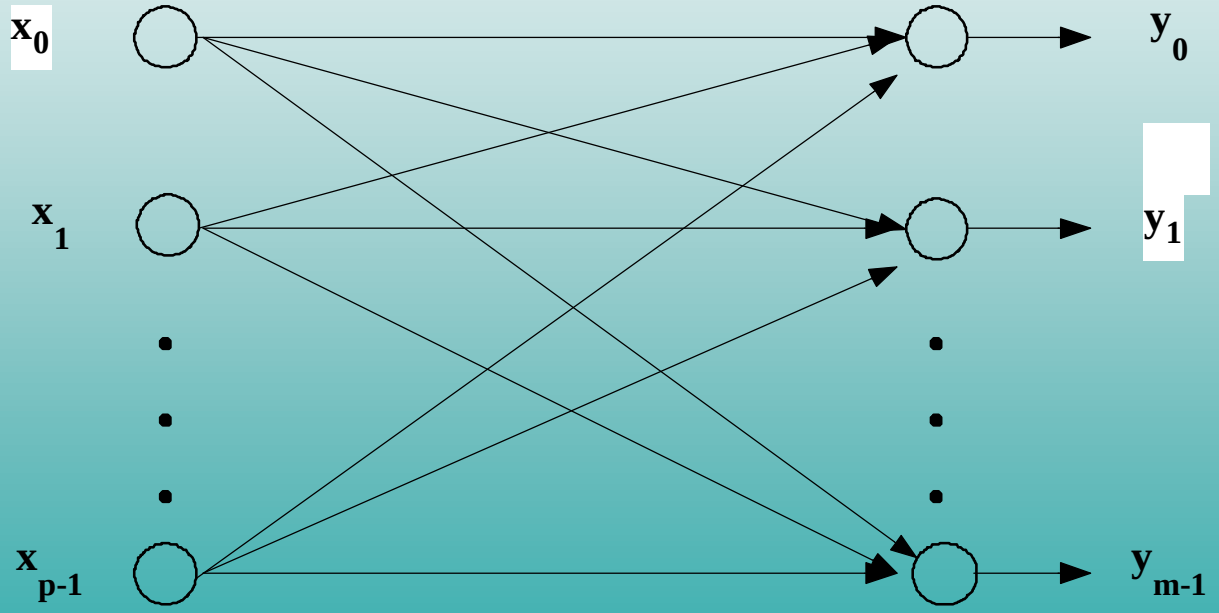
$$\Delta w_{kj}(n) = -\eta y_k(n) x_j(n)$$

- Regra Hebbiana Modificada:

$$\Delta w_{kj} = \eta y_k(n) x_j(n) - \alpha w_{kj}(n) y_k(n)$$

Redes Neurais que implementam PCA

- **Rede PCA Auto-Organizante**



Onde

Saída $y_j(n)$ do neurônio j no tempo n produzida em resposta ao conjunto de entradas $\{x_i(n) | i = 0,1,\ldots, p-1\}$:

$$y_j(n) = \sum_{i=0}^{p-1} w_{ij}(n) x_i(n), \qquad \text{j=0,1, ..., m-1}$$

Ajuste do peso sináptico $w_{ij}(n)$ (Regra generalizada de Hebb)

$$\Delta w_{ij}(n) = \eta [y_j(n) x_i(n) - y_j(n) \sum_{k=0}^{j} w_{ik}(n) y_k(n)]$$

$$\text{i=0,1,..., p-1} \qquad \text{j=0,1,...., m-1}$$

- No limite:

$$\Delta w_j(n) \to 0 \quad \text{e} \quad w_j \to a_j, \qquad \text{j = 0,1,...,m-1}$$

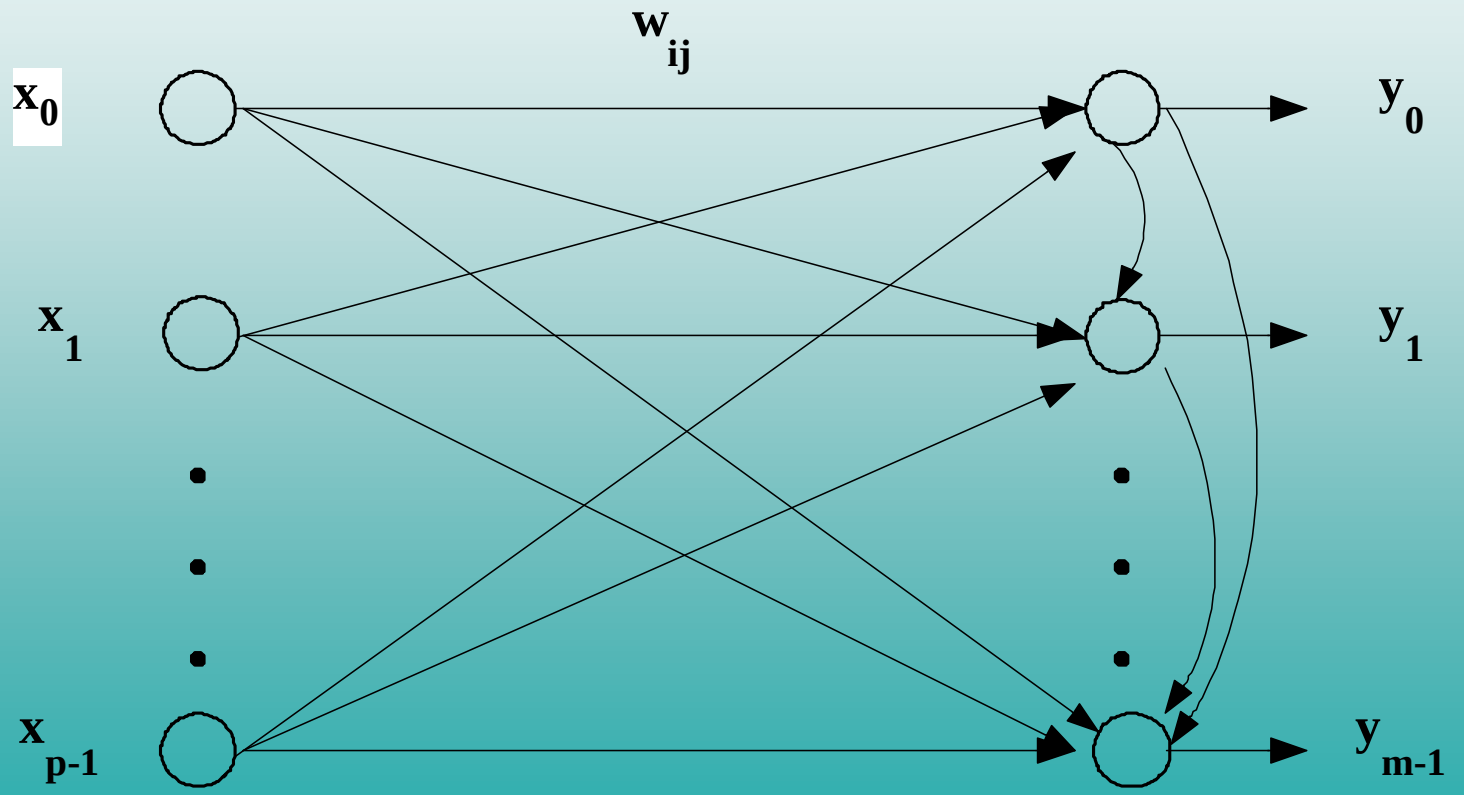
tal que $\|w_j(n)\| = 1$ para todo j.

onde $a_0, a_1, \ldots, a_{m-1}$ são os autovetores normalizados associados aos m maiores autovalores da matriz C de correlação dos vetores de entrada $x(n)$,estando esses autovalores arranjados em ordem decrescente.

PCA X PCA Auto-Organizante

Rede PCA Auto-Organizante				PCA Clássica			
a_0	a_1	a_2	a_3	a_0	a_1	a_2	a_3
0.4889	0.3702	-0.7210	-0.2570	0.5223	0.3723	-0.7210	-0.2620
-0.2825	0.9274	0.2371	0.1165	-0.2633	0.9255	0.2420	0.1241
0.5904	0.0158	0.1437	0.8005	0.5812	0.0210	0.1409	0.8011
0.5767	0.0507	0.6350	-0.5287	0.5656	0.0654	0.6338	-0.5235

PCA - Adaptativa



- Saída $y_j(n)$ do neurônio j no tempo n produzida em resposta ao conjunto de entradas : $\{x_i(n) \mid i = 0,1,..., p-1\}$

$$y_j(n) = \sum_{i=0}^{p-1} w_{ij}(n) x_i(n) + \sum_{l<j} u_{lj}(n) y_l(n),$$

j=0,1,...,m-1

- Ajuste dos pesos das conexões (Regra de Hebb):

$$\Delta w_{ij}(n) = \eta x_i(n) y_j(n)$$

$$i = 0,1,...,p\text{-}1 \qquad j = 0,1,...,m\text{-}1$$

- Ajuste dos pesos sinápticos laterais (Regra Anti-Hebbiana

$$\Delta u_{lj}(n) = -\mu y_l(n) y_j(n) \qquad l<j$$

LABIC

Teorema de Convergência[Sang-89]

"Se a matriz de pesos sinápticos *W(n)* for associada a valores aleatórios no tempo *n* = 0, então, com probabilidade 1, a regra generalizada de Hebb irá convergir na média, e , no limite, irá se aproximar de uma matriz cujas colunas serão os primeiros *m* autovetores da matriz *C* de covariancia dos vetores de entrada *x(n)*,ordenados por ordem decrescente de autovalor."

Teorema

Portanto, no limite, pode-se escrever:

$$\Delta w_j(n) \to 0 \quad w_j \longrightarrow a_j \qquad j = 0,1,...,m-1$$

tal que $\| w_j(n) \| = 1$ para todo *j*. Os valores representam os autovetores normalizados associados aos *m* maiores autovalores da matriz *C* de covariancia dos vetores de entrada *x(n)*, estando esses autovalores ordenados em ordem decrescente.

Aceleração da Convergência

$$\Delta w_{ij}(n+1) = \eta(n) x_i y_j + \beta(n) \Delta w_{ij}(n) \quad (1)$$

$$\Delta u_{lj}(n+1) = -\mu(n) y_l y_j + \beta(n) \Delta u_{lj}(n) \quad (2)$$

Onde $\eta(n+1) = \max(\alpha\eta(n), 0.0001)$,
$\mu(n+1) = \max(\alpha\,\mu(n), 0.0002)$,
$\beta(n+1) = \max(\alpha\beta(n), 0.0001)$,
e α é o fator de diminuição

Algoritmo PCA Adptativa-[Rubn-89]

Inicio

1. Inicialize todos os pesos de conexões com pequenos valores aleatórios e escolha os valores para os parâmetros de aprendizagem.Normalize-os em [0,1]. Se normalizar em [-1,1] pode mudar os sinais dos auto-vetores.

2. Repita

2.2 Selecione aleatoriamente um padrão *p*-dimensional e apresente-o à rede.

2.3. Ajuste os pesos das conexões entre a camada de entrada e a camada de saída de acordo com a Eq. (1)

2.4. Normalize os vetores-peso (em colunas).

2.5. Atualize os pesos laterais de acordo com a equacao (2)(não precisa normalizar)

2.6. Modifique os parâmetros β, η, e μ.

Até que {todos os pesos laterais sejam suficientemente pequenos (a soma de seus valores absolutos seja menor que algum *threshold* ε)} ou {um número de iterações máximo seja atingido}.

Fim.

LABIC

PCA X PCA Adaptativa

Rede PCA Adaptativa				PCA Clássica			
a_0	a_1	a_2	a_3	a_0	a_1	a_2	a_3
0.5222	0.3833	-0.7231	-0.2696	0.5223	0.3723	-0.7210	-0.2620
-0.2692	0.9209	0.2384	0.1185	-0.2633	0.9255	0.2420	0.1241
0.5807	0.0265	0.1419	0.8004	0.5812	0.0210	0.1409	0.8011
0.5636	0.0652	0.6325	-0.5249	0.5656	0.0654	0.6338	-0.5235

LABIC

Referencias

Livro do Haykin, 1999.

D. Pormeleau, “ALVINN: An Autonomous Land Vehicle in a Neural Network. NIPS 1988: 305-31

MEDEIROS, D. M. R. ; ROMERO, R. A. F. . Utilização de Redes Neurais para Navegação de Robôs móveis em um Chão de Fábrica. Revista de Iniciação Científica (USP), v. 6, p. 45-51, 2004.

WALDHERR, S. ; ROMERO, R. A. F. ; THRUN, S. . Gesture-Based Interface For Human-Robot Interaction. Journal Autonomous Robots, Netherlands, v. 9, n. 2, p. 151-173, 2000.

Referencias

Livro do Haykin

- D. Pormeleau, “ALVINN: An Autonomous Land Vehicle in a Neural Network. NIPS 1988: 305-313

- MEDEIROS, D. M. R. ; ROMERO, R. A. F. . Utilização de Redes Neurais para Navegação de Robôs móveis em um Chão de Fábrica. Revista de Iniciação Científica (USP), v. 6, p. 45-51, 2004.

- 2.

- MEDEIROS, D. M. R. ; ROMERO, R. A. F. . Sistema de Controle Autônomo de Robôs Móveis com Utilização de Redes Neurais Artificiais. RIC - Revista de Iniciação Científica - No. 5, São Carlos - SP - Brasil, v. 5, p. 47-51, 2003