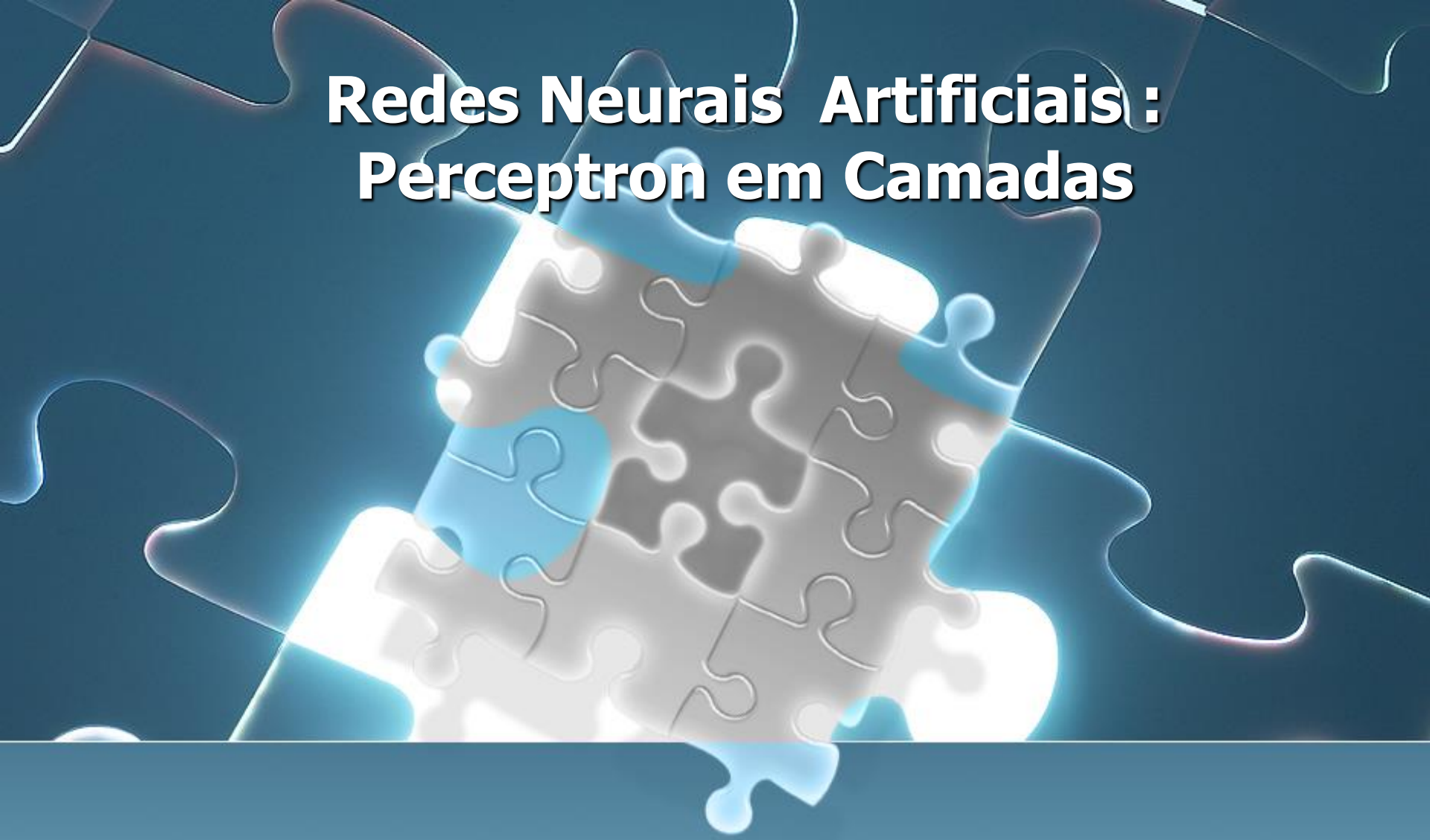


Redes Neurais Artificiais : Perceptron em Camadas

Prof. Dr. Ernane José Xavier Costa – LAFAC- FZEA- USP

Topologia das Redes Neurais Artificiais

Introdução

Interligação dos neurônios

As camadas intermediárias (ou ocultas)

Superfícies de decisão mais complexas

Topologia das Redes Neurais Artificiais

Redes diretas e recorrentes

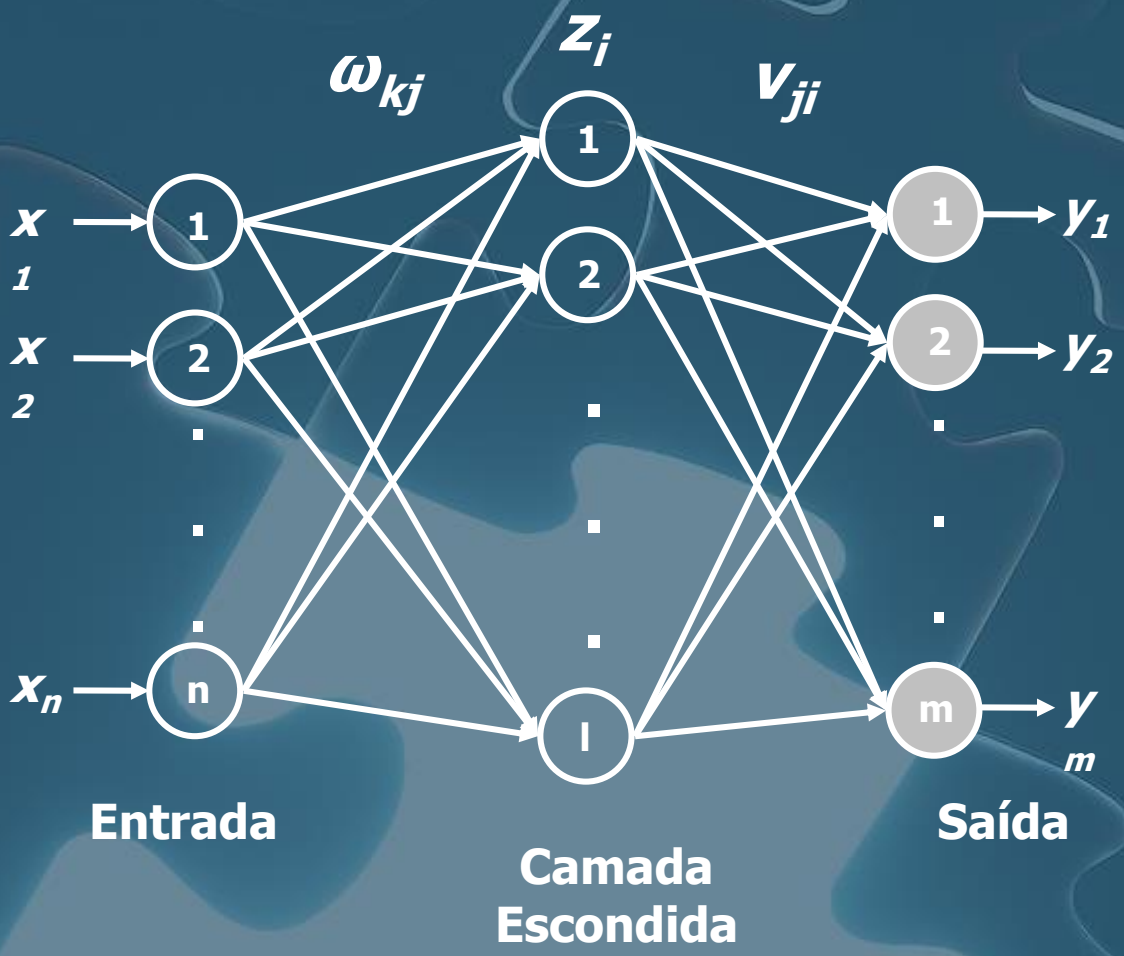
A principal diferença entre elas é que na rede direta **não há retroalimentação** em suas entradas, ou seja, seu grafo não tem ciclos

Rede direta popularizada com o ***backpropagation***

Considerada um aproximador universal de funções

Topologia das Redes Neurais Artificiais

Redes Diretas



Treinamento por Correção de Erros

Introdução

- Perceptron simples de uma camada (1958)

  1º modelo de RNA que envolveu o conceito de aprendizado

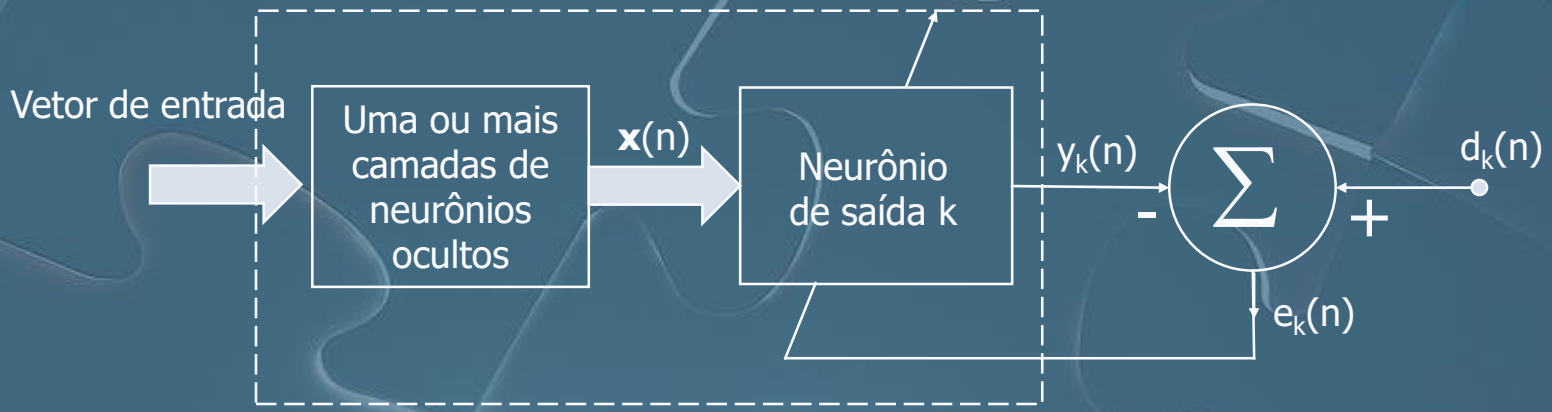
- Problemas mais complexos demandariam redes mais complexas

  Treinamento de redes torna-se um problema em aberto

  Backpropagation (1986)

Treinamento por Correção de Erros

Introdução



Sinal de erro $e_k(n)$ aciona um **mecanismo de controle**, cujo objetivo é aplicar uma seqüência de **ajustes corretivos** aos pesos ω do neurônio k

$$e_k = d_k(n) - y_k(n)$$

Treinamento por Correção de Erros

Ajuste dos pesos

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \Delta\omega_{kj}(n)$$

Ajustes corretivos → aproximar a saída y da resposta desejada d

Isto é feito **minimizando-se** uma função de custo ou índice de desempenho E(n) em relação aos pesos ω

$$E(n) = \frac{1}{2} e_k^2(n)$$ ➡ Valor instantâneo da energia do erro

Ou seja

$$\Delta\omega_{ji}(n) \propto \frac{\partial E(n)}{\partial \omega_{ji}(n)}$$

e_j

Treinamento por Correção de Erros

A Regra Delta

A minimização de E(n) resulta na regra de aprendizagem conhecida como regra delta ou regra de Widrow-Hoff

A Regra Delta

$$\Delta\omega_{kj}(n) = \eta e_k x_j(n)$$

η taxa de aprendizagem
e_k sinal de erro ($e_k = d_k(n) - y_k(n)$)
x_k elemento do vetor de sinal **x**(n)

Os ajustes passo a passo persistem até o sistema atingir um **estado estável**

Treinamento por Correção de Erros

Redes com uma camada

Para redes com uma camada como o Perceptron ou o ADALINE, o ajuste de pesos assume uma forma simples

$$\Delta\omega_{kj}(n) = \eta e_k x_j(n)$$

Treinamento por Correção de Erros

Redes com múltiplas camadas

A extensão do método do gradiente ou regra delta para redes com múltiplas camadas é conhecida **Regra Delta Generalizada** ou ***Backpropagation***

Na 1ª fase do treinamento o sinal é propagado para a frente (fase *foward*) das entradas até a saída da rede

Como o valor desejado d para a camada de saída é conhecido, o erro pode ser calculado e os pesos da camada de saída ajustados

Treinamento por Correção de Erros

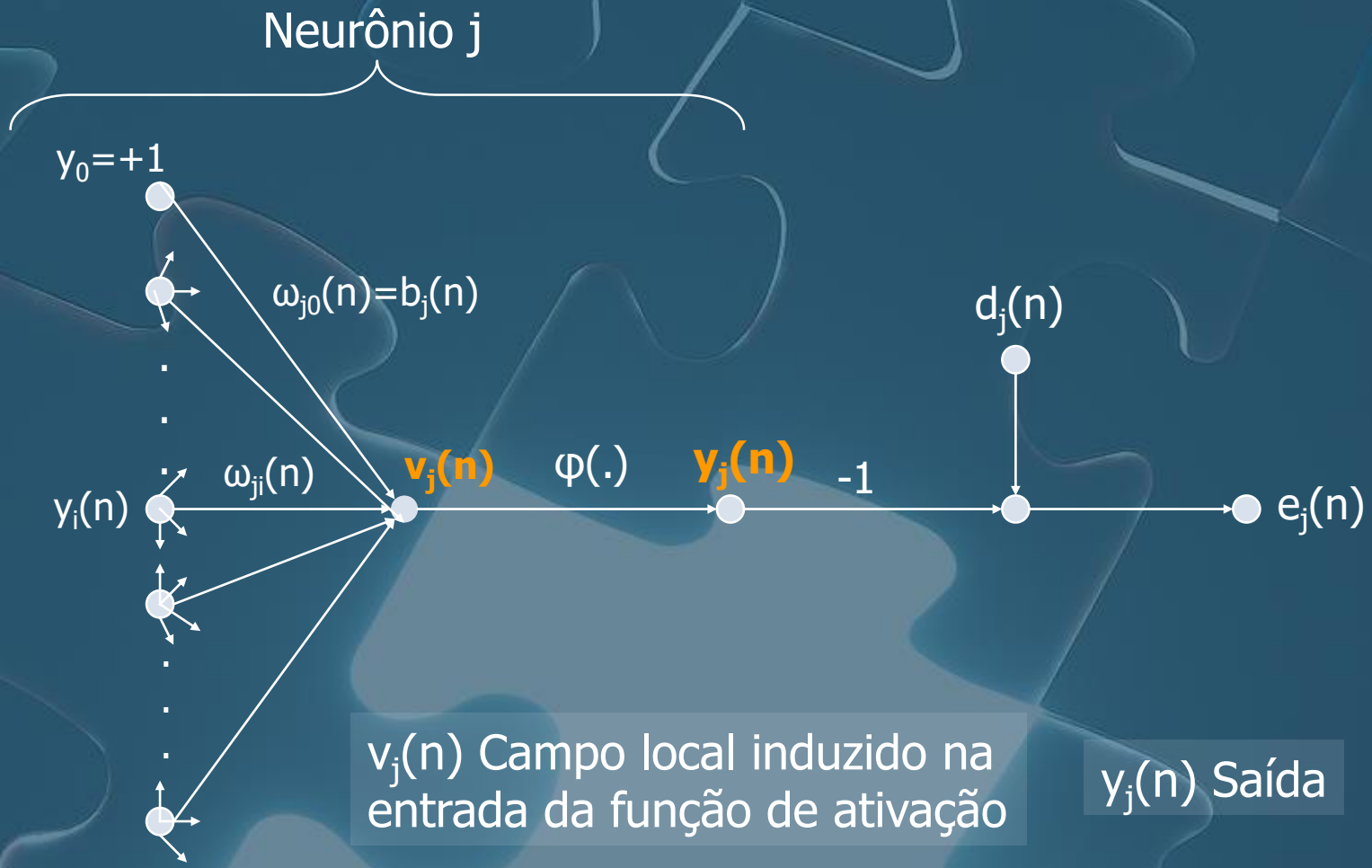
Redes com múltiplas camadas

Para as camadas intermediárias não existem valores desejados

O ajuste dos pesos das camadas intermediárias é feito através da propagação do erro para trás, o que caracteriza o treinamento com *backpropagation*

O algoritmo de retropropagação

Introdução



$v_j(n)$ Campo local induzido na entrada da função de ativação

$y_j(n)$ Saída

O algoritmo de retropropagação

Introdução

Para um neurônio

$e_k = d_k(n) - y_k(n)$ Sinal de erro

$E(n) = \frac{1}{2} e_k^2(n)$ Valor instantâneo da energia do erro

Para todos os neurônios

$$E(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

Se utilizarmos N padrões para treinamento da rede

$E_{med} = \frac{1}{N} \sum_{n=1}^{N} E(n)$ Energia média do erro

O algoritmo de retropropagação

Introdução

Para um total de m entradas (excluindo o bias)

$$v_j(n) = \sum_{i=0}^{m} \omega_{ij}(n) y_i(n)$$

$$y_j(n) = \varphi_j(v(n))$$

Ajuste dos pesos

$$\Delta W_{ij}(n) \propto \frac{\partial E(n)}{\partial W_{ij}}$$

O algoritmo de retropropagação

Introdução

Pela regra da cadeia

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial \omega_{ji}(n)}$$

$$\frac{\partial E(n)}{\partial e_j(n)} = \frac{\partial}{\partial e_j(n)} \left(\frac{1}{2} \sum_{k \in C} e_k^2(n) \right) = e_j(n)$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)} (d_j(n) - y_j(n)) = -1$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \frac{\partial}{\partial v_j(n)} \varphi(v_j(n)) = \varphi'(v_j(n))$$

$$\frac{\partial v_j(n)}{\partial \omega_{ji}(n)} = \frac{\partial}{\partial \omega_{ji}(n)} \left(\sum_{i=0}^{m} \omega_{ij}(n) y_i(n) \right) = y_i(n)$$

O algoritmo de retropropagação

Introdução

$$\frac{\partial E(n)}{\partial \omega_{ji}(n)} = -e_j(n)\varphi_j'(v_j(n))y_i(n)$$

Assim a correção no peso pode ser escrita como:

$$\Delta\omega_{ji}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{ji}(n)}$$

$$\Delta\omega_{ji}(n) = -\eta \frac{\partial E(n)}{\partial \omega_{ji}(n)} = -\eta e_j(n)\varphi_j'(v_j(n))y_i(n)$$

$$= -\eta\delta_j(n)y_i(n)$$

Gradiente local

O algoritmo de retropropagação

O gradiente local

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial e_j(n)}\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n)\varphi'_j(v_j(n))$$

Assim

(Correção de peso $\Delta\omega_{ji}(n)$) = (Parâmetro da taxa de aprendizagem η) · (Gradiente local $\delta_j(n)$) · (Sinal de entrada do neurônio j $y_i(n)$)

O fator chave é calcular o sinal de erro e_j!!!!!!!

O algoritmo de retropropagação

Casos 1 e 2

Existem 2 casos a serem considerados:

Caso 1

O neurônio j é um neurônio da **camada de saída** e

$e_j = d_j(n) - y_j(n)$

Fácil!!!!!!!!

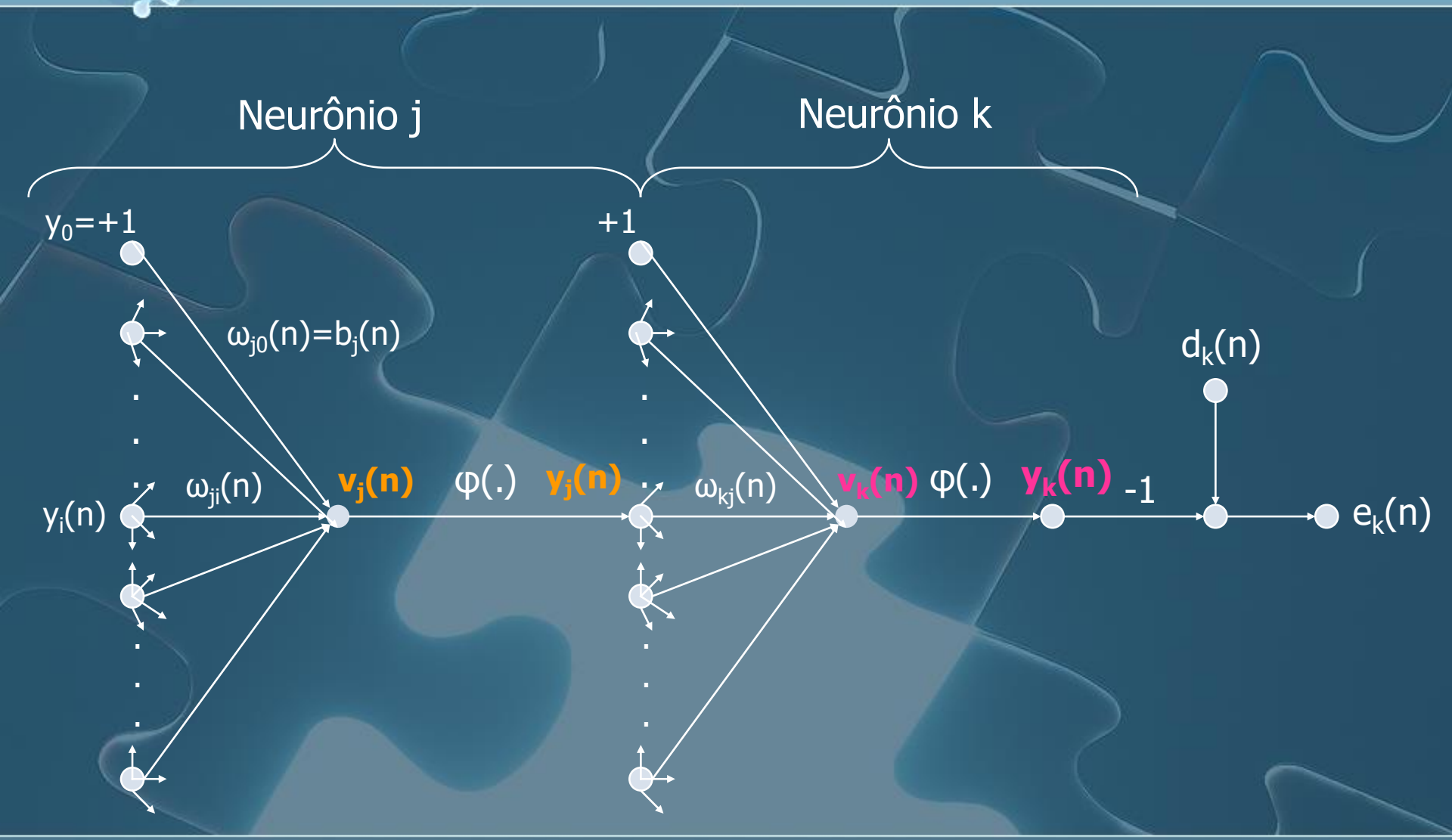
Caso 2

O neurônio j é um neurônio da **camada escondida**

Olhar melhor para este caso!!!!!!!

O algoritmo de retropropagação

Caso 2



O algoritmo de retropropagação

Gradiente local para o **neurônio oculto j**

Como calcular????????

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\varphi'_j(v_j(n))$$

$$v_j(n) = \sum_{i=0}^{m} \omega_{ij}(n) y_i(n) \qquad y_j(n) = \varphi_j(v(n))$$

Só tenho informação para E(n) na camada de saída, ou seja:

$$E(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n)$$

k é um neurônio da camada de saída

O algoritmo de retropropagação

Gradiente local para o **neurônio oculto j**

$$\frac{\partial E(n)}{\partial y_j(n)} = \frac{\partial}{\partial y_j} E(n) = \frac{\partial}{\partial y_j}\left(\frac{1}{2}\sum_{k\in C} e_k^2(n)\right) = \frac{\partial E(n)}{\partial e_k}\frac{\partial e_k}{\partial y_j}$$

$$= \sum_k e_k \frac{\partial e_k(n)}{\partial y_j}$$

Como calcular????????

Pela regra da cadeia

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j} = \sum_k e_k \frac{\partial e_k}{\partial v_k}\frac{\partial v_k}{\partial y_j}$$

O algoritmo de retropropagação

Gradiente local para o **neurônio oculto j**

$$\frac{\partial e_k}{\partial v_k} = \frac{\partial}{\partial v_k}(d_k(n) - y_k(n)) = \frac{\partial}{\partial v_k}(d_k(n) - \varphi_k(v_k(n))) = -\varphi'_k(v_k(n))$$

$$\frac{\partial v_k}{\partial y_j} = \frac{\partial}{\partial y_j(n)} \sum_{j=0}^{m} \omega_{kj}(n) y_j(n) = \omega_{kj}(n)$$

Assim

$$\delta_k(n)$$

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j} = \sum_k e_k \frac{\partial e_k}{\partial v_k} \frac{\partial v_k}{\partial y_j} = -\sum_k e_k \varphi'_k(v_k(n)) \omega_{kj}(n)$$

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k \delta_k(n) \omega_{kj}(n)$$

O algoritmo de retropropagação

Gradiente local para o **neurônio oculto j**

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\varphi'_j(v_j(n))$$

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k \delta_k(n)\omega_{kj}(n)$$

$$\delta_j(n) = \varphi'_j(v_k(n))\sum_k \delta_k(n)\omega_{kj}(n)$$

j neurônio oculto

k neurônio saída

O erro na camada oculta depende do erro na camada de saída!!

O algoritmo de retropropagação

Correção nos pesos

Assim

Correção de peso $\Delta\omega_{ji}(n)$ = Parâmetro da taxa de aprendizagem η · Gradiente local $\delta_j(n)$ · Sinal de entrada do neurônio j $y_i(n)$

→ 2 casos

Caso 1 - j neurônio saída

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n))$$

Caso 2 - j neurônio oculto

$$\delta_j(n) = \varphi'_j(v_k(n))\sum_k \delta_k(n)\omega_{kj}(n)$$

O algoritmo de retropropagação

Equação de ajuste dos pesos

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \Delta\omega_{kj}(n) \qquad \Delta\omega_{kj}(n) = -\eta\delta_j(n)y_j(n)$$

Caso 1 - j neurônio saída

$$\delta_j(n) = e_j(n)\varphi'_j(v_j(n))$$

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \eta e_j(n)\varphi'_j(v_j(n))y_j(n)$$

Caso 2 - j neurônio oculto

$$\delta_j(n) = \varphi'_j(v_k(n))\sum_k \delta_k(n)\omega_{kj}(n)$$

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \eta\varphi'_j(v_j(n))\sum_k \delta_k(n)\omega_{kj}(k)y_j(n)$$

O algoritmo de retropropagação

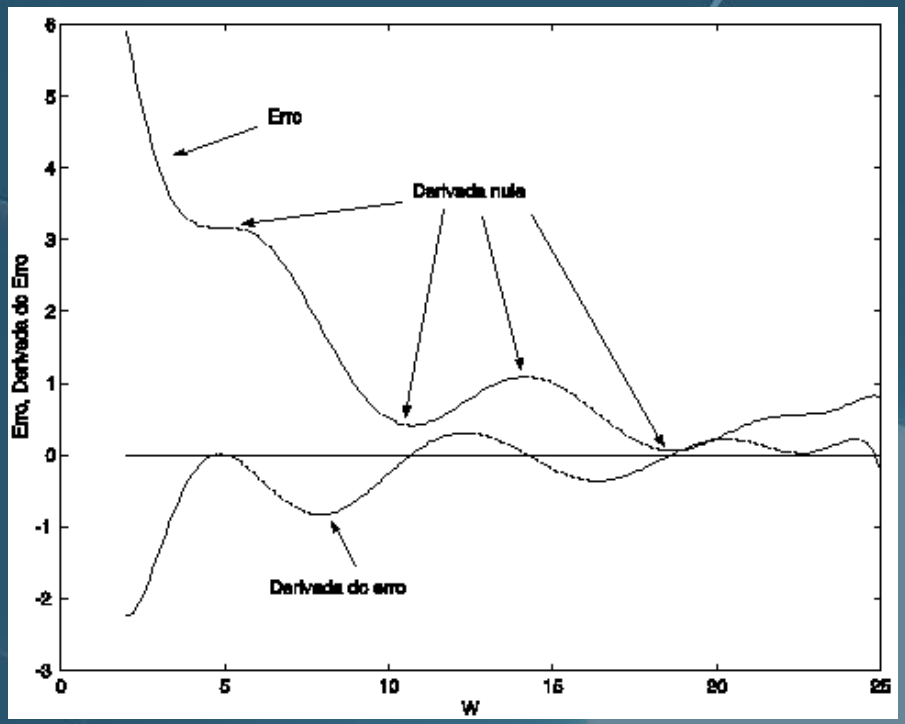
A superfície de erro

- A **forma irregular da superfície de erro** em razão das não-linearidades das funções de ativação causa **dificuldades para o treinamento** com algoritmos baseados no gradiente descendente, como o backpropagation.

- As não-linearidades podem provocar o surgimento de **mínimos locais** e **regiões planas** de **gradiente nulo**, dificultando o treinamento

O algoritmo de retropropagação

A superfície de erro



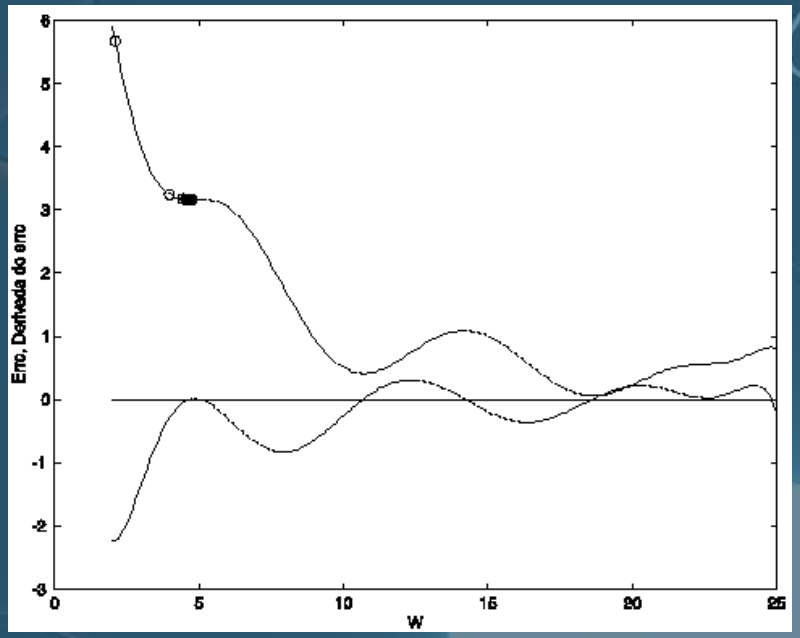
$$E(n) = \frac{1}{2} e_k^2(n)$$

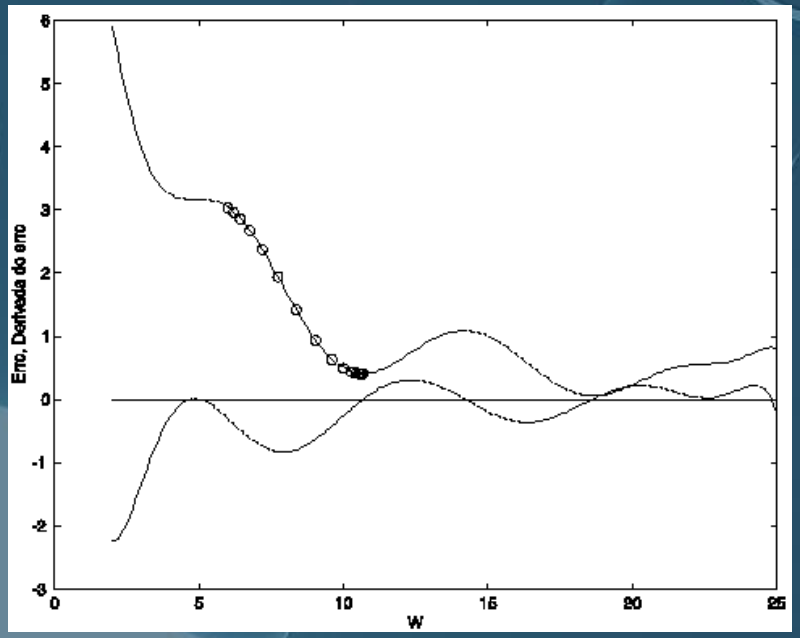
$$e_k = d_k(n) - y_k(n)$$

O algoritmo de retropropagação

A superfície de erro

Inicialização dos pesos



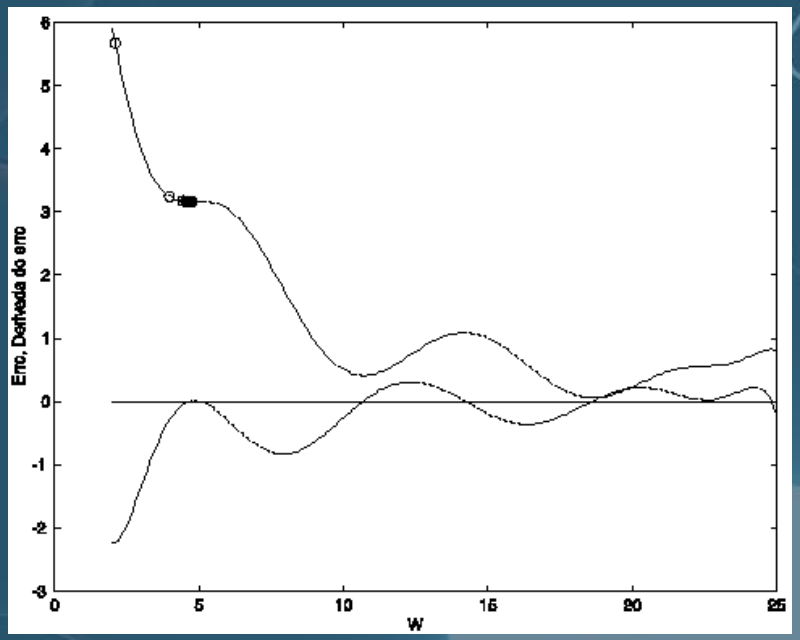


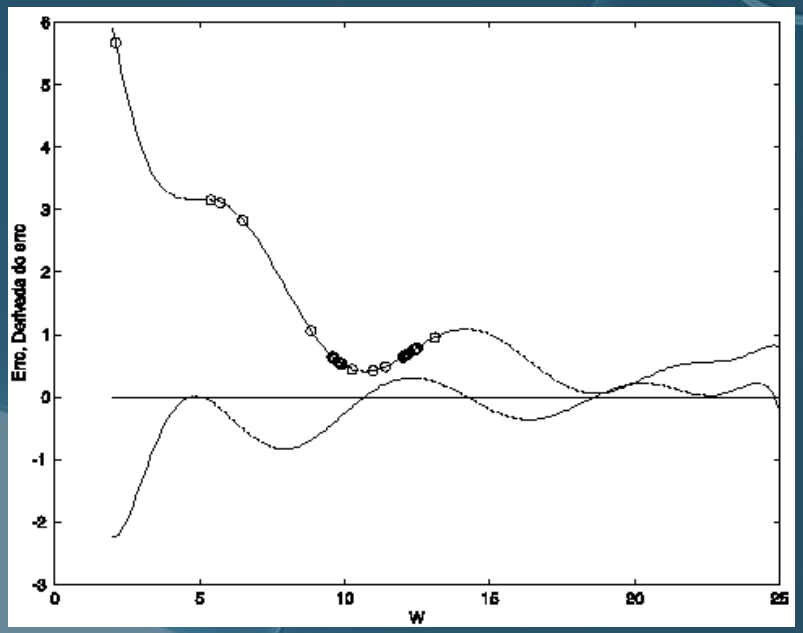
O desempenho depende das condições iniciais de treinamento

O algoritmo de retropropagação

A superfície de erro

A taxa de aprendizagem η





O desempenho depende da taxa de aprendizado

O algoritmo de retropropagação

A taxa de aprendizagem η

η peq. $\rightarrow$ maior sensibilidade às variações da superfície

η maior $\rightarrow$ evita regiões planas e mínimos locais

η influencia a velocidade de convergência

- η gde. $\rightarrow$ os ajustes de peso são de maior magnitude e o erro tende a diminuir mais rapidamente
- η peq. $\rightarrow$ convergência é lenta e o treinamento pode ficar presos a mínimos locais

Alguns algoritmos usam uma taxa de aprendizado adaptativa dependente do erro, começa em um valor grande e diminui próximo às regiões de mínimo

O algoritmo de retropropagação

O *momentum* α

- Alternativa para evitar regiões de gradiente nulo

$$\Delta\omega_{ij}(n) = -\eta \frac{\partial E}{\partial \omega_{ij}} + \alpha \Delta \omega_{ij}(n-1)$$

- Responsável pelo acumulo do histórico dos ajustes anteriores

  Resulta em um termo residual quando o ajuste é nulo por causa do gradiente

- Mínimos locais e regiões planas podem ser evitados

O algoritmo de retropropagação

O *momentum* α

- Se o **treinamento** passa por um **min. local** o **termo residual** forçará os ajustes na mesma direção dos ajustes **anteriores á passagem pelo mínimo**

- até que o gradiente se torne maior que o termo residual

- os ajustes passam a ser feitos novamente em direção ao mínimo, podendo passar por ele em direção contrária à anterior

- o treinamento permanecerá neste movimento em torno do mínimo até a convergência

O algoritmo de retropropagação

O *momentum* α

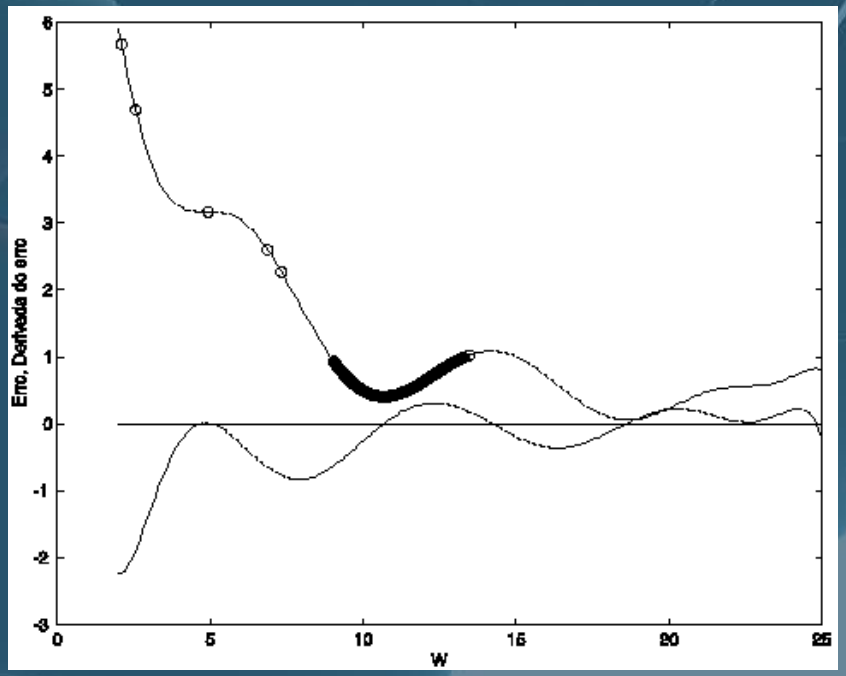
- Se o valor de α for grande o suficiente não haverá convergência para o mínimo local mas sim para o mínimo global

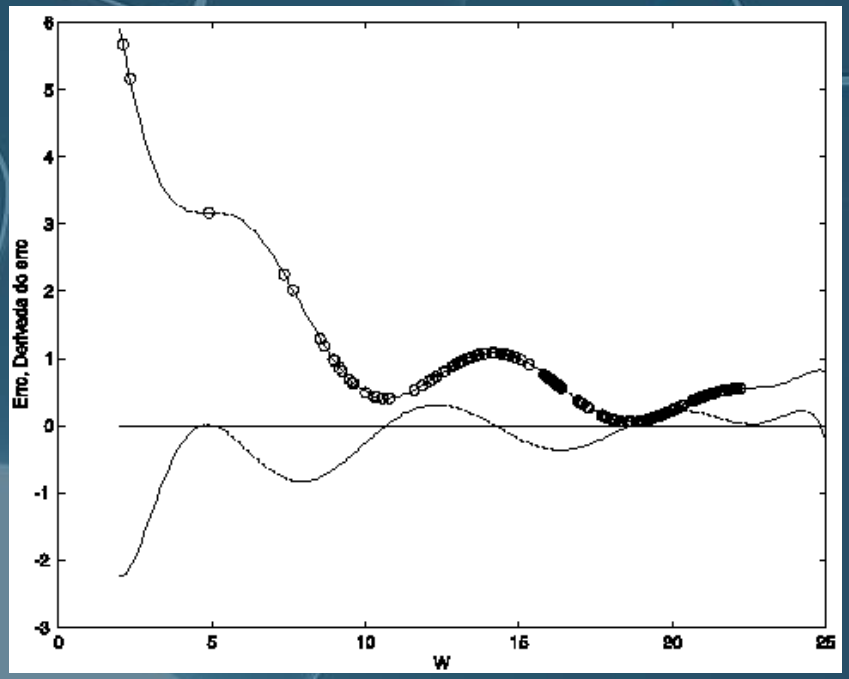
- pode resultar em melhor qualidade de treinamento

- Problema: é mais um parâmetro a ser ajustado pelo usuário

O algoritmo de retropropagação

O *momentum* α





Redes Neurais Artificiais

Aprendizado e Generalização

O objetivo principal do aprendizado em Redes Neurais Artificiais é a **obtenção de modelos com boa capacidade de generalização** tendo como base um conjunto de dados

Redes Neurais Artificiais

Aprendizado e Generalização

Problemas de aproximação, classificação e predição

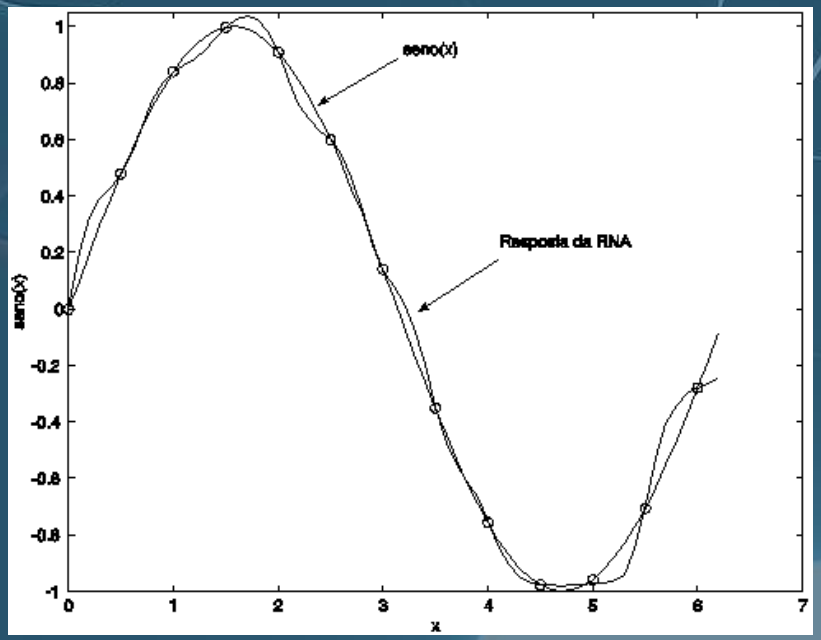
Conjunto de treinamento: pares (x,d)

Ajuste de pesos deve minimizar (y – d)

Minimização do erro pode não levar a resultados satisfatórios

Redes Neurais Artificiais

Aprendizado e Generalização



O erro foi minimizado, pois **a resposta da rede passa por todos os pontos**, mas a aproximação está distante da função *seno(x)*

É preciso mais que minimizar o erro para se obter boas respostas de generalização!!!!

Redes Neurais Artificiais

Aprendizado e Generalização

Treinamento

Minimizar o erro

Aproximar as funções geradoras dos dados

overfiting

A rede tem mais parâmetros (pesos) do que ela precisa

underfiting

A rede tem menos parâmetros (pesos) do que ela precisa

Encontrar o ajuste ideal

Redes Neurais Artificiais

Aprendizado e Generalização

Algoritmos que busquem o "*fit*" ideal

Métodos construtivos

Constroem a rede gradualmente. Iniciam em situação de *underfiting* e param em uma situação próxima ao *overfiting*

Algoritmos de poda

Visa a diminuição da estrutura da rede pela eliminação gradativa de pesos e neurônios

Redes Neurais Artificiais

Aprendizado e Generalização

Métodos construtivos e Algoritmos de poda

- Problemas
  - Não garantem a convergência da função geradora
  - Requerem o ajuste de parâmetros adicionais

Como encontrar o equilíbrio???

Redes Neurais Artificiais

Aprendizado e Generalização

Dilema entre polarização e variância

Modelos sobreparametrizados

Maior variabilidade na resposta

Modelos subparametrizados

Tem menos variabilidade e são polarizados para alguma resposta

O problema do treinamento de Redes Neurais é encontrar o equilíbrio entre polarização e variância

Redes Neurais Artificiais

Aprendizado e Generalização

Como encontrar este equilíbrio?

Associar algum outro fator (objetivo) além da minimização do erro ao treinamento

Sugestões

- Algoritmo *Weight Decay*

Um termo de custo que envolve a norma dos pesos é associado ao termo de energia do erro

$$E(n) = \frac{1}{2}(d_k(n) - y_k(n))^2 + \frac{\lambda}{2}\|\boldsymbol{\omega}\|^2$$

λ é uma parâmetro de treinamento que determina o peso da penalidade na obtenção da solução

Sugestões (cont.)

- SVMs (*Support Vector Machines*)

  Mapeiam os vetores de entrada em um espaço de mais alta dimensão (espaço de características), onde um hiperplano de separação é obtido para a resolução de problemas de classificação

  Um conceito fundamental no projeto de SVMs é o de **margem de separação** (associada ao erro permitido na classificação)

Redes Neurais Artificiais

Aprendizado e Generalização

Conclusão

O aprendizado de RNAs deve levar em conta algum outro objetivo, além do erro do conjunto de treinamento

Weight decay → norma dos pesos

SVMs → margem de classificação

O algoritmo de retropropagação

Simulação do backpropagation

Bibliografia

BRAGA AP, CARVALHO ACPLF, LUDERMIR, TB. Redes neurais artificiais. In: REZENDE SO. Sistemas inteligentes: princípios e aplicações. Barueri: Manole, 2003.

HAYKIN S. Redes Neurais: Princípios e prática. Porto Alegre: Bookman, 2001.