Aprendizagem de Máquina

Alceu Britto

Programa de Pós-Graduação em Informática
Pontifícia Universidade Católica do Paraná (PUCPR)

Redes Neurais Artificiais

Referências

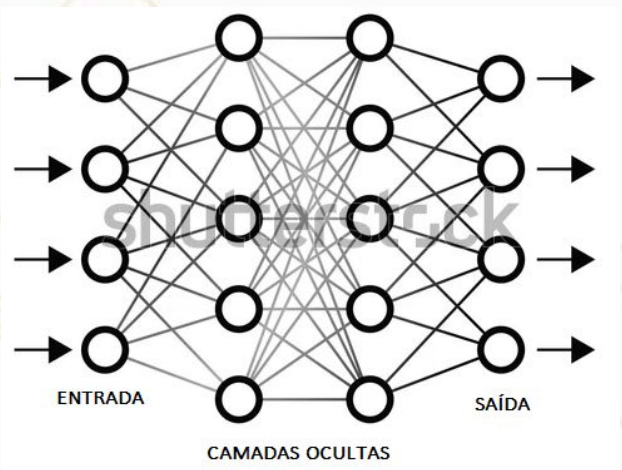
- Duda R., Hart P., Stork D. Pattern Classification 2ed. Willey Interscience, 2002. Capítulo 6
- Mitchell T. Machine Learning. WCB McGraw–Hill, 1997. Capítulo 4.
- Haykin S. Neural Networks: A Comprehensive Foundation (2nd Edition) 842 pages Prentice Hall; 2nd edition (July 6, 1998) ISBN: 0132733501
- Bishop C. Neural Networks for Pattern Recognition. 504 pages. Oxford University Press (January 1996) ISBN: 0198538642

Introdução

- Redes Neurais Artificiais (RNAs) fornecem um método geral e prático para a aprendizagem de funções de valor real e de valor discreto a partir de exemplos.
- Algoritmos como o *backpropagation* (retro-propagação) utilizam a "descida do gradiente" para ajustar os parâmetros das redes para melhor adaptar um conjunto de treinamento de pares entrada – saída (ou vetor de atributos – valor do conceito alvo).
- A aprendizagem de redes neurais é robusta a erros e ruídos nos dados de treinamento.

Introdução

- Modelo inspirado na aprendizagem de sistemas biológicos → redes complexas de neurônios interconectados.



- Entrada: neurônios de entrada (conforme a quantidade de atributos de entrada do problema)
- Saída: neurônios de saída (em um problema de classificação normalmente um para cada classe do problema)
- Camadas ocultas: quantidade e tamanho (número de neurônios) são definidos experimentalmente.

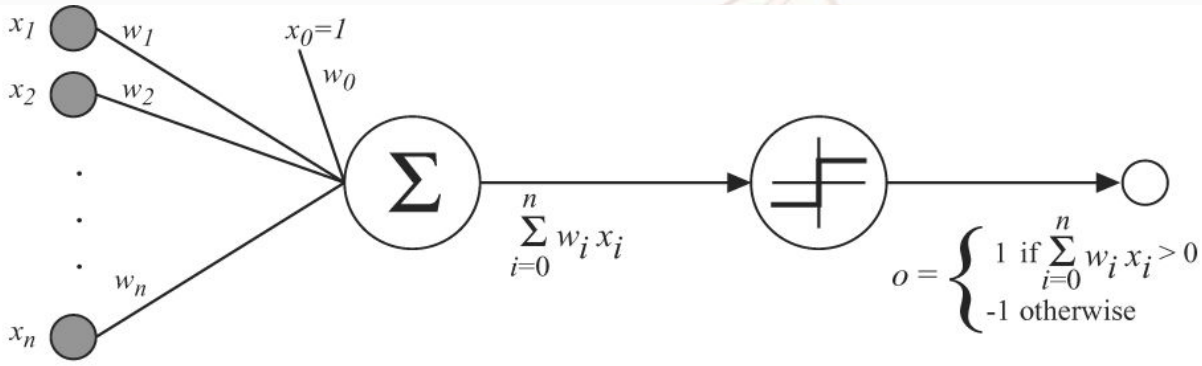
Perceptron

- Rede neural elementar baseada em uma unidade chamada *Perceptron* criada por Rosenblatt em 1958.

- Um *perceptron*:
  1. Recebe um vetor de entradas de valor real
  2. Calcula uma combinação linear destas entradas
  3. Fornece na saída:
     - "+1" se o resultado é maior que algum limiar
     - "–1" caso contrário.

- Mais precisamente, fornecidas as entradas $x_1 ... x_n$, a saída $o(x_1 ... x_n)$ calculada pelo *perceptron* é . . .

Perceptron



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Algumas vezes utilizaremos notação vetorial simplificada:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

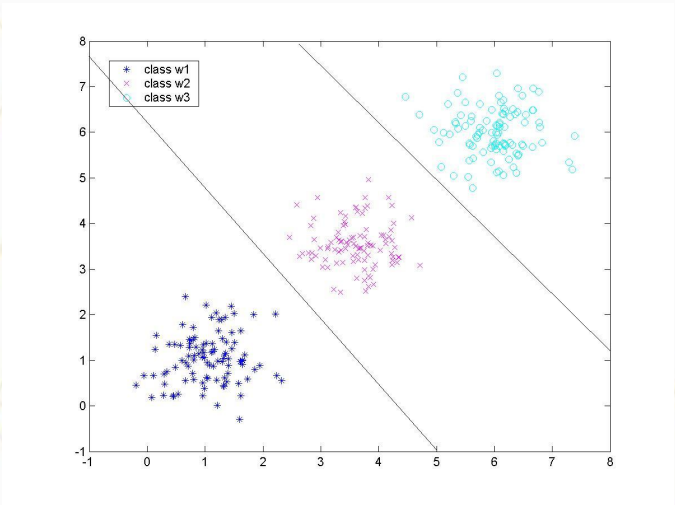
Perceptron

- onde:
  - Cada elemento w_i é uma constante de valor real, ou peso, que determina a contribuição da entrada x_i na saída do *perceptron*.
- A aprendizagem do *perceptron* envolve:
  - A escolha dos valores dos pesos w_o a w_n.
- A camada de entrada deve possuir uma unidade especial conhecida como *bias*, que é um termo constante que não depende de nenhum valor de entrada.

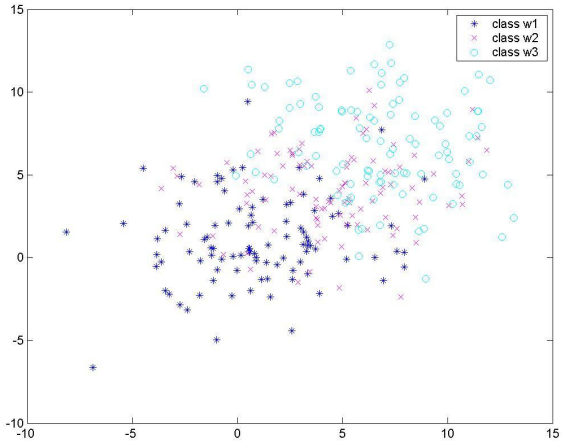
Superfícies de Decisão

- Podemos "ver" o *perceptron* como uma superfície de separação em um espaço n-dimensional de instâncias.
- O *perceptron* fornece "1" para instâncias dispostas em um lado do hiperplano e "-1" para instâncias dispostas no outro lado.
- Um único *perceptron* consegue separar somente conjuntos de exemplo linearmente separáveis.

Superfícies de Decisão



Linearmente Separável



Linearmente Não–Separável

Regra de Treinamento *Perceptron*

- Os pesos do *perceptron* são modificados a cada passo de acordo com a regra de treinamento do *perceptron*, que modifica o peso w_i associado a entrada x_i de acordo com a regra:

$$w_i \leftarrow w_i + \Delta w_i$$

onde

$$\Delta w_i = \eta (t - o) x_i$$

  - t é o valor alvo para o exemplo de treinamento.
  - o é a saída gerada pelo *perceptron*.
  - η é uma constante pequena (0.1) chamada de taxa de aprendizagem.

Regra de Treinamento *Perceptron*

- Se o exemplo de treinamento é classificado corretamente:

$$(t - o) = \text{zero} \quad \rightarrow \quad \Delta w_i = 0$$

- Se o exemplo de treinamento é classificado incorretamente, o valor de Δw_i é alterado:
  - Se $x_i = 0.8$, $\eta = 0.1$, $t = 1$, $o = -1$
  - A atualização do peso será:

$$\Delta w_i = \eta(t - o)x_i = 0.1(1 - (-1))0.8 = 0.16$$

Regra de Treinamento *Perceptron*

- Pode–se provar que este procedimento de aprendizagem converge dentro de um número finito de passos quando:
  - As classes dos dados de treinamento são linearmente separáveis;
  - η é suficientemente pequeno.

- Porém: falha em convergir se as classes forem linearmente não-separáveis.

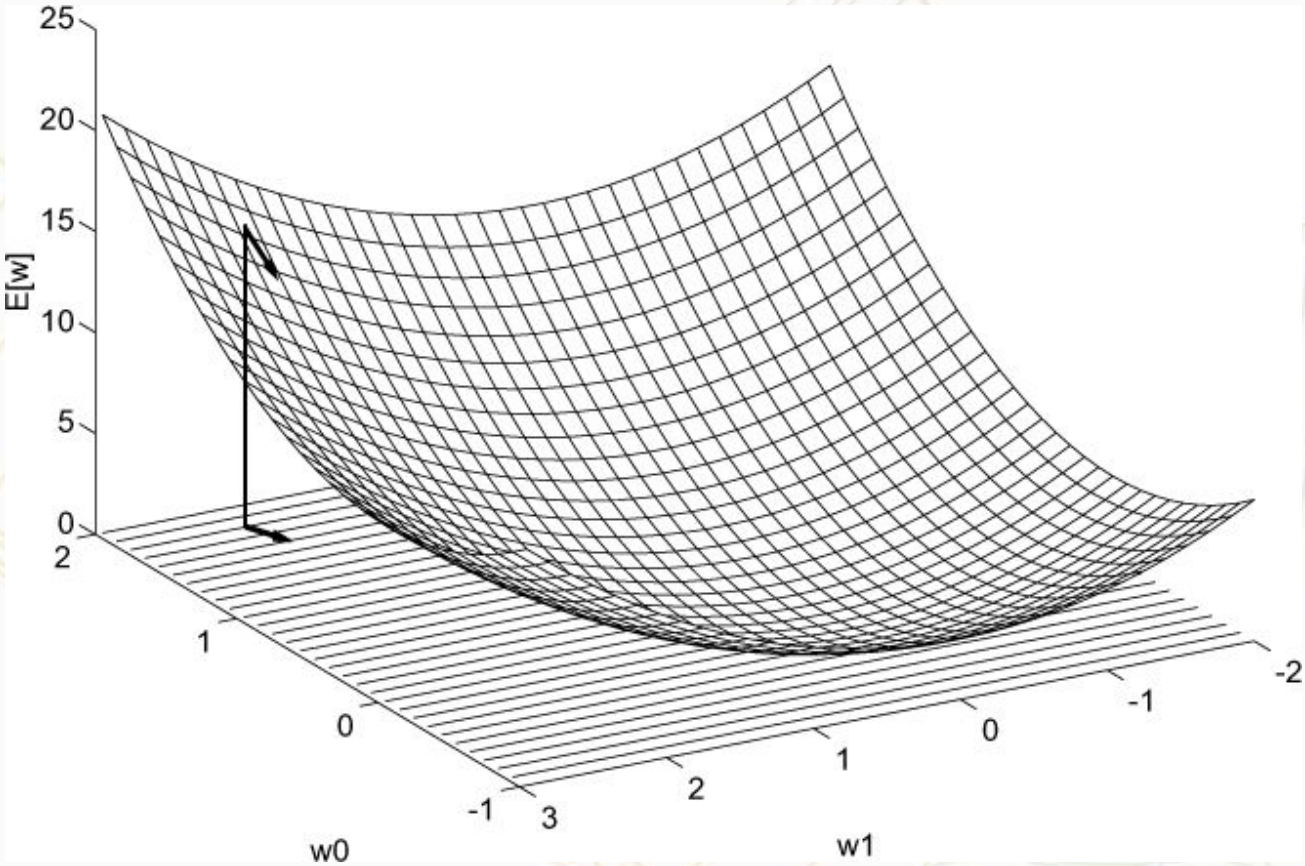
- Solução: algoritmo descida do gradiente

Descida do Gradiente

- Para dados não linearmente separáveis, a Regra Delta converge em direção a aproximação que melhor se ajusta ao conceito alvo.

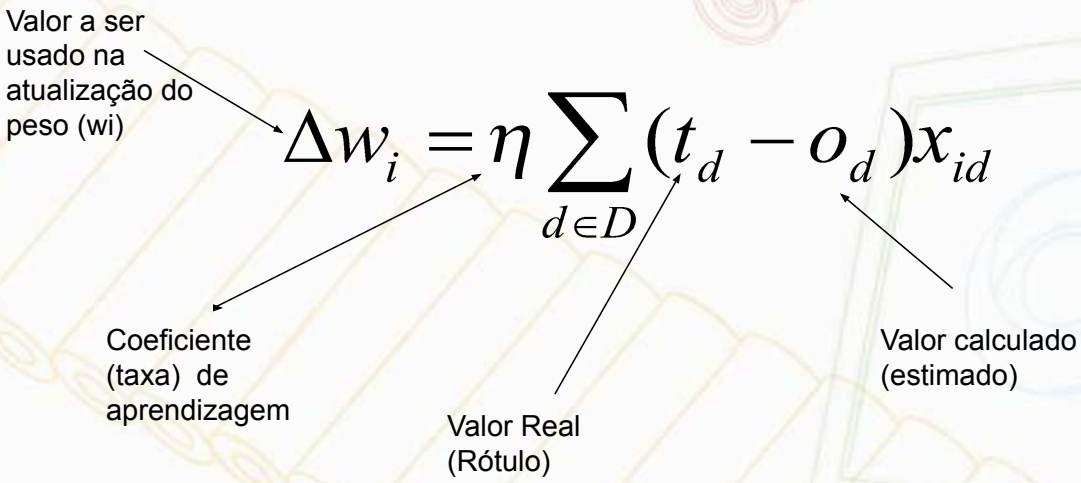
- Ideia chave: usar a descida do gradiente para procurar o melhor vetor de pesos.

Descida do Gradiente



Descida do Gradiente

- A regra para atualização dos pesos para o gradiente descendente é



Descida do Gradiente

- Resumindo, o algoritmo descida do gradiente para a aprendizagem de unidade lineares:

1. Pegar um vetor inicial aleatório de pesos;
2. Aplicar a unidade linear para todos os exemplos de treinamento e calcular Δw_i para cada peso de acordo com a equação anterior;
3. Atualizar cada peso w_i adicionando Δw_i e então repetir este processo.

- O algoritmo convergirá para um vetor de pesos com erro mínimo.

Descida do Gradiente

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, *where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i \quad \text{(T4.1)}$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i \quad \text{(T4.2)}$$

Redes Multicamadas

- *Perceptrons* expressam somente superfícies de decisão linear.
- Redes multicamadas treinadas pelo algoritmo *backpropagation* são capazes de expressar uma rica variedade de superfícies de decisão não lineares.

Redes Multicamadas

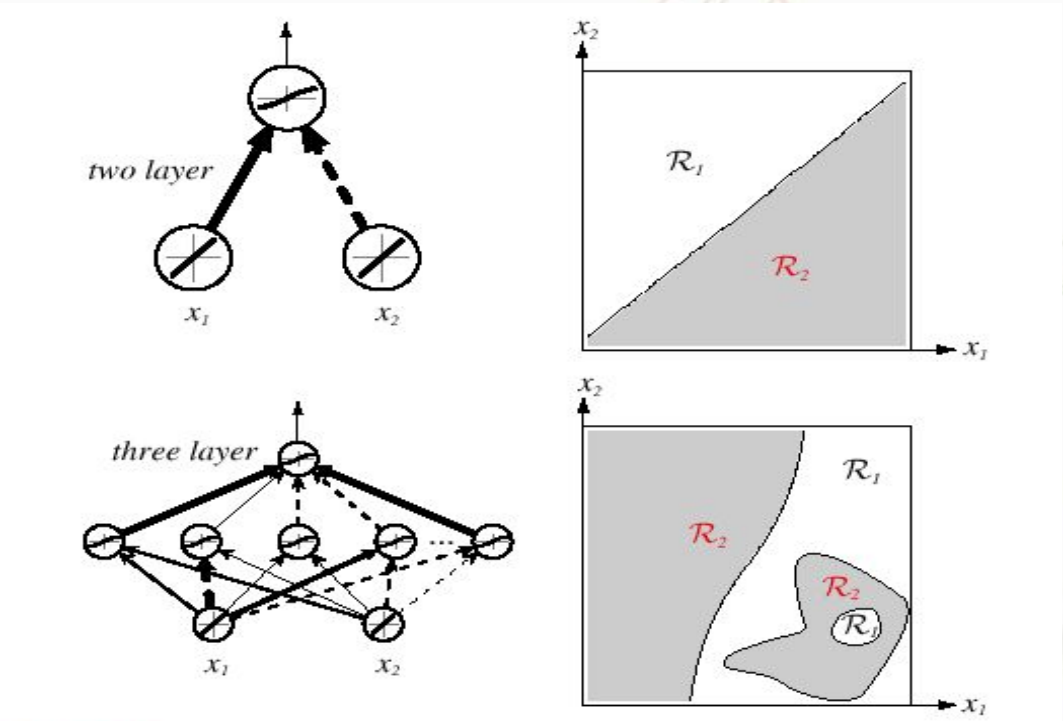
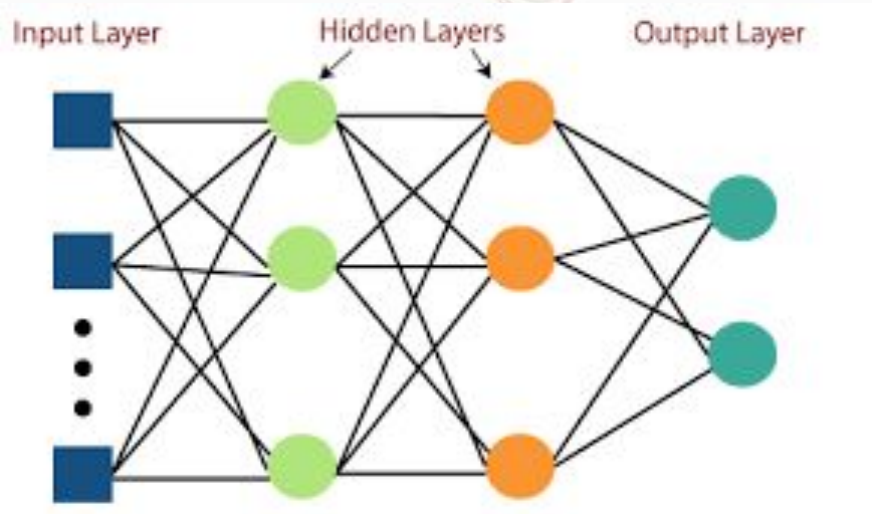


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Redes Multicamadas

- Redes são capazes de representar funções altamente não lineares.



Algoritmo *Backpropagation*

- Aprende os pesos para uma rede multicamadas, dada uma rede com um número fixo de unidades e interconexões.

- O algoritmo backpropagation emprega a "descida do gradiente" para minimizar o erro quadrático entre a saída da rede e os valores alvos para estas saídas.

valor do conceito alvo na saída da rede → 0.119 0.059 0.253 0.246

0 0 1 0 ← valor do conceito alvo

Erro = (valor do conceito alvo real) – (valor do conceito alvo estimado)

Algoritmo *Backpropagation*

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

  4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

Algoritmo *Backpropagation*

- Notação:
  - Um índice é atribuído a cada nó da rede, onde nó pode ser uma entrada da rede ou a saída de alguma unidade da rede.
  - x_{ji} indica a entrada a partir do nó *i* para unidade *j* e w_{ji} indica o peso correspondente.
  - δ_n indica o termo do erro associado com a unidade *n*. Similar a (*t*–*o*).

Mais sobre *Backpropagation*

- Descida do gradiente sobre o vetor de pesos inteiro da rede
- Encontrará um erro mínimo local (não necessariamente global)
  - Na prática, geralmente funciona bem (pode executar múltiplas vezes).
- Geralmente inclui peso do momento α

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

Mais sobre *Backpropagation*

- Minimiza o erro sobre os exemplos de treinamento
  - Generalizará bem sobre exemplos subseqüentes?
- O treinamento pode levar milhares de iterações → vagaroso
- A utilização da rede após o treinamento → muito rápida

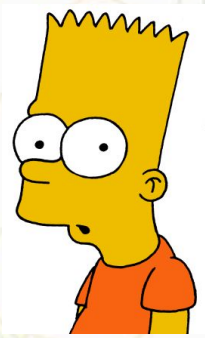
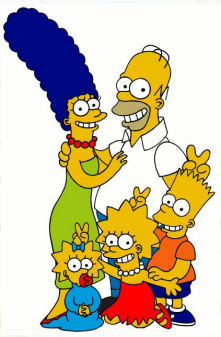
Exemplo

- Dada a imagem de um personagem, ele deve ser classificado corretamente, ou seja, se a imagem for do personagem Bart, ela deve ser classificada pelo algoritmo de aprendizagem como sendo o personagem Bart.

Classes / Valor do Conceito Alvo

Marge 0 0 0 1 **Homer 0 0 1 0** **Bart 0 1 0 0** **Família 1 0 0 0**



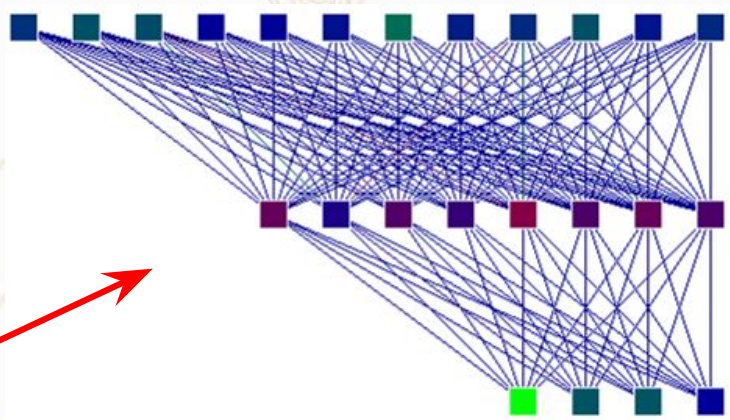



Exemplo

vetor de características

0.43 0.03 0.40 0.19 0.12 0.16 0.04 0.01 0.00 0.01 0.40 0.02
0 0 1 0

valor do conceito alvo associado ao vetor



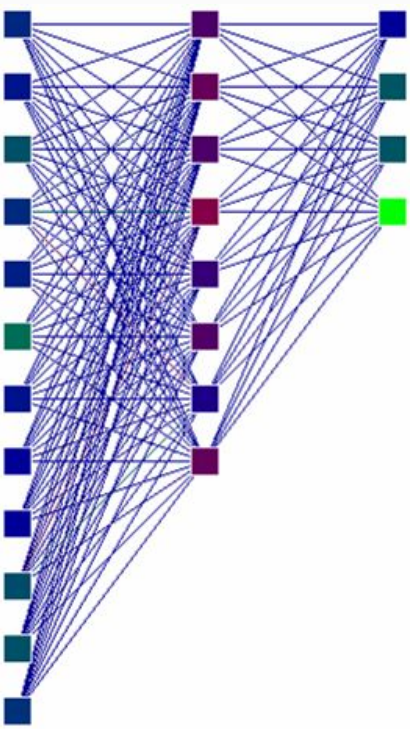
rede neural treinada

0.119 0.059 0.253 0.569

valor do conceito alvo estimado

Erro = (valor do conceito alvo real) – (valor do conceito alvo estimado)

Representação da RNA



unit definition section :

no.	typeName	unitName	act	bias	st	position	act func	out func	sites
1			0.15710	0.00200	i	2, 2, 0	Act_Identity		
2			0.08250	0.00492	i	2, 3, 0	Act_Identity		
3			0.31630	0.00955	i	2, 4, 0	Act_Identity		
4			0.16530	0.00616	i	2, 5, 0	Act_Identity		
5			0.11860	0.00476	i	2, 6, 0	Act_Identity		
6			0.43310	0.00818	i	2, 7, 0	Act_Identity		
7			0.06930	0.00605	i	2, 8, 0	Act_Identity		
8			0.00890	0.00587	i	2, 9, 0	Act_Identity		
9			0.00380	0.00916	i	2,10, 0	Act_Identity		
10			0.29860	0.00922	i	2,11, 0	Act_Identity		
11			0.31760	0.00948	i	2,12, 0	Act_Identity		
12			0.19330	0.00649	i	2,13, 0	Act_Identity		
13			-0.30391	-46.08251	h	5, 2, 0	Act_Identity		
14			-0.40381	-101.31063	h	5, 3, 0	Act_Identity		
15			-0.30793	97.62634	h	5, 4, 0	Act_Identity		
16			-0.52309	160.65987	h	5, 5, 0	Act_Identity		
17			-0.21414	-79.82547	h	5, 6, 0	Act_Identity		
18			-0.32417	135.45871	h	5, 7, 0	Act_Identity		
19			-0.10986	-53.94949	h	5, 8, 0	Act_Identity		
20			-0.39891	-55.78927	h	5, 9, 0	Act_Identity		
21			-0.00000	-0.02777	o	8, 2, 0	Act_Identity		
22			0.33768	165.30469	o	8, 3, 0	Act_Identity		
23			0.33482	380.65833	o	8, 4, 0	Act_Identity		
24			1.03949	260.54959	o	8, 5, 0	Act_Identity		

Generalização e Sobreajuste

- A condição de parada do algoritmo *backpropagation* foi deixada em aberto.

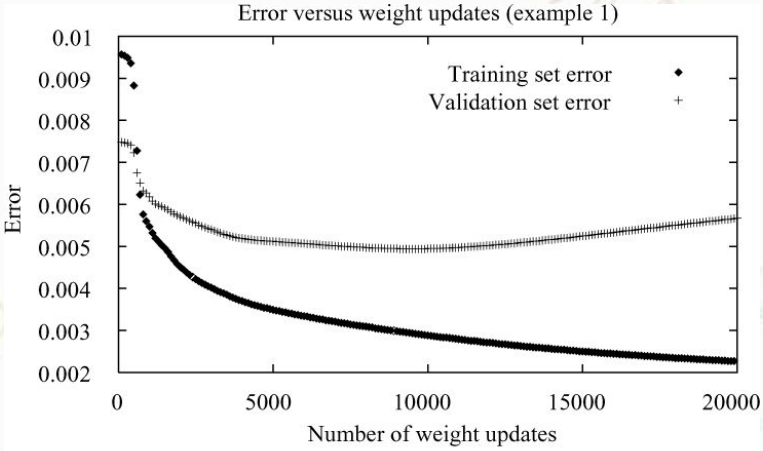
- Quando devemos parar o treinamento, i.e. parar de atualizar os pesos?
  - Escolha óbvia: continuar o treinamento até que o erro (E) seja menor do que um valor pré-estabelecido.
  - Porém, isto implica em sobre ajuste (*overfitting*) !!!

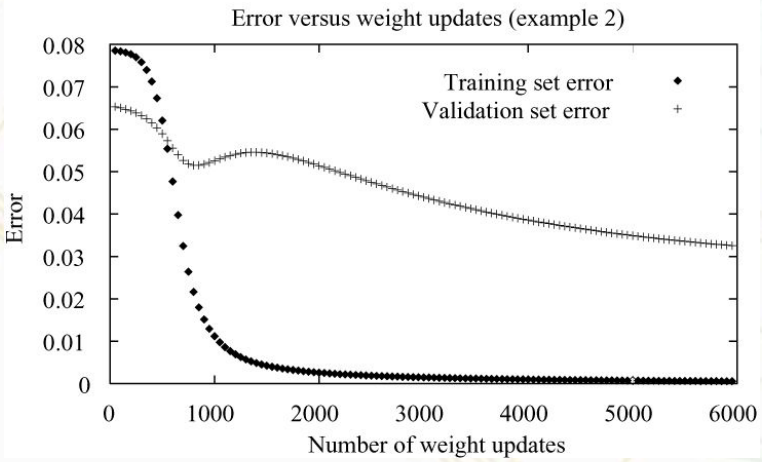
Generalização e Sobreajuste

- O algoritmo *backpropagation* é susceptível a sobre ajustar a rede aos exemplos de treinamento ao preço de reduzir a generalização sobre exemplos novos.

- A Figura a seguir ilustra o perigo de minimizar o erro sobre os dados de treinamento em função do número de iterações (atualização dos pesos).

Generalização e Sobreajuste





Generalização e Sobreajuste

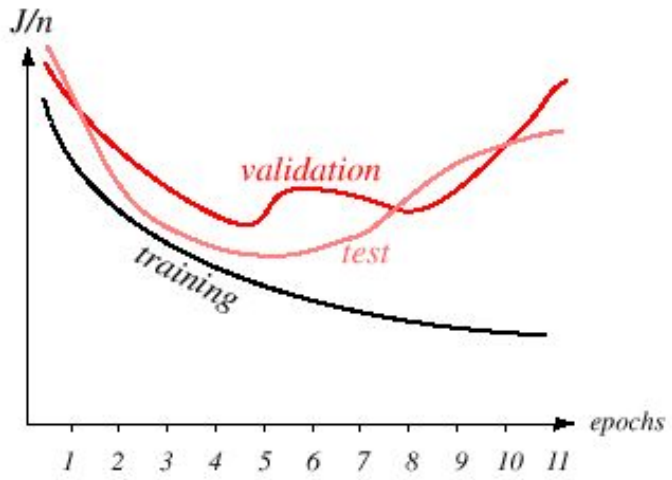


FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n \sum_{p=1}^{n} J_p$. The validation error and the test or generalization error per pattern are virtually always higher than the training error. In some protocols, training is stopped at the first minimum of the validation set. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*.

Generalização e Sobreajuste

- A linha inferior mostra o decréscimo do erro sobre os exemplos de treinamento em função do número de iterações de treinamento.
  - Esta linha mede o “Erro de Aprendizagem”
- A linha superior mostra o erro medido sobre exemplos de validação (não utilizados para atualizar os pesos !!!)
  - Esta linha mede a “Precisão da Generalização”
  - A precisão que a rede classifica corretamente exemplos diferentes dos utilizados no treinamento.

Resumo

- Redes Neurais: um método prático para aprendizagem de funções de valor real e vetorial sobre atributos de valor contínuo e discreto.
- Robustez a ruídos nos dados de treinamento.
- O espaço considerado pelo algoritmo *backpropagation* é o espaço de todas as funções que podem ser representadas pelos pesos.
- O *backpropagation* busca o espaço de todos os pesos possíveis usando a descida do gradiente para reduzir iterativamente o erro em uma rede (ajustar aos dados de treinamento).

Resumo

- Sobre ajuste resulta em redes que não generalizam bem. Utilizar um conjunto de validação para avaliar a generalização

- *Backpropagation* é o algoritmo de aprendizagem mais comum, porém existem muitos outros . . .