Aprendizagem de Máquina

Redes Neurais Artificiais (RNAs)

Plano de Aula

- Introdução
- Motivação Biológica
- *Perceptron*
- Superfície de Decisão
- Descida do Gradiente
- Redes Multicamadas (*Multilayer*)
- Retropropagação (*Backpropagation*)
- Generalização e Sobreajuste
- Exemplo: Reconhecimento de Faces

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 tais como o *Backpropagation* (retropropagação) 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 nos dados de treinamento.

Introdução

- Modelo inspirado na aprendizagem de sistemas biológicos → redes complexas de neurônios interconectados.
- Modelo impreciso, pois não considera muitas complexidades.
- Também chamadas de *Artificial Neural Networks* (*ANN*) ou simplesmente *Neural Networks* (*NN*).

Introdução

- Diversas aplicações práticas:
  - Interpretação de cenas visuais.
  - Reconhecimento da fala.
  - Aprendizado de estratégias para controlar robôs.
  - Reconhecimento de caracteres manuscritos.
  - Reconhecimento de faces
  - Bioinformática
  - Predição financeira
  - Etc . . .

Motivação Biológica

- Considere os seres humanos:
  - Tempo de chaveamento do neurônios: 0.001 seg
  - Número de neurônios: 10.000.000.000 (10^{10})
  - Conexões por neurônio: 10.000 a 100.000 ($10^4 - 10^5$)
  - Tempo para o reconhecimento de uma cena: 0.1 seg

  muita computação paralela !!

- Propriedade de redes neurais artificiais (RNAs):
  - Muitas unidades de limiar similares aos neurônios
  - Muitas interconexões ponderadas entre unidades
  - Processo altamente paralelo e distribuído
  - Ênfase em ajustar pesos automaticamente

Motivação Biológica

- Apesar das RNAs serem motivadas pelos sistemas neurais biológicos, muitas complexidades não são modeladas pelas RNAs.
- Duas linhas de pesquisas:
  - Objetivo de utilizar RNAs para estudar e modelar o processo de aprendizagem biológico.
  - Objetivo de obter algoritmos de aprendizagem de máquina altamente eficientes.

Quando Considerar RNAs

- Entrada discreta ou de valor real de alta dimensão
- Saída discreta ou de valor real
- Saída for um vetor de valores
- Dados possivelmente ruidosos
- Forma da função alvo é desconhecida
- Leitura humana dos resultados não é importante

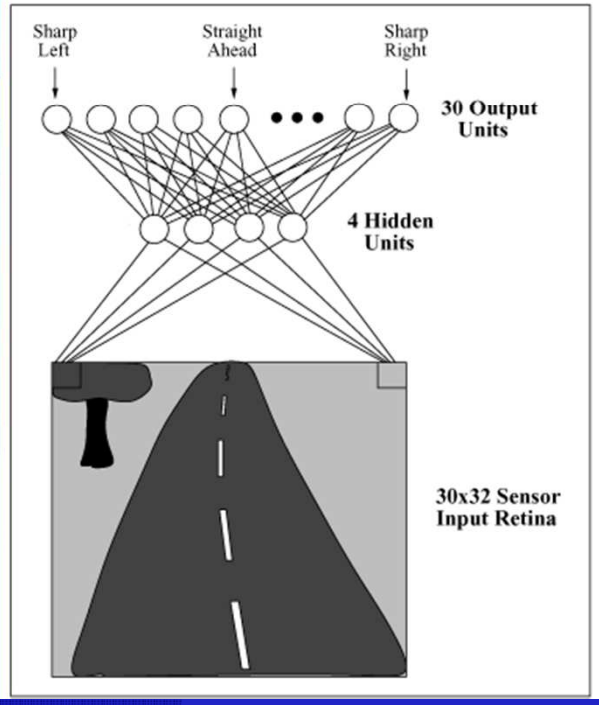
Exemplos:

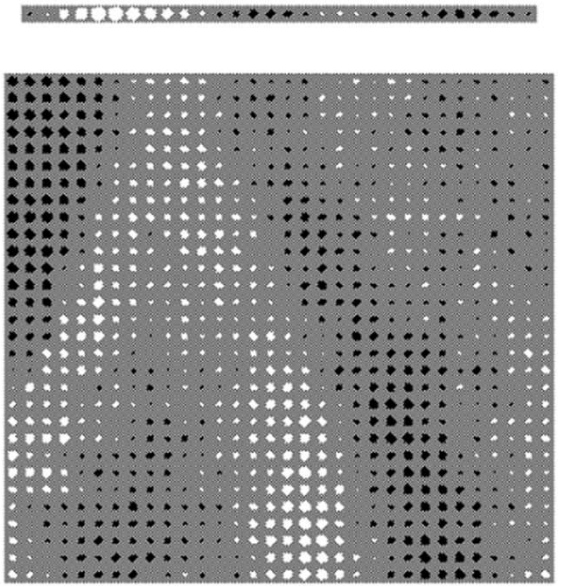
- Reconhecimento de fonemas na fala
- Classificação de imagens
- Predição financeira

Exemplo: Sistema ALVINN

- Sistema ALVINN: utiliza RNAs para guiar um veículo autônomo em velocidade normal em vias públicas.
  - Entrada: *Grid* de 30 x 32 pixel fornecidos por uma câmera montada na frente do veículo. 960 neurônios
  - Camada escondida (*hidden*): 4 neurônios
  - Saída: Direção para qual o veículo deve seguir. 30 neurônios de saída, cada um indicando uma direção particular para o veículo.

ALVINN Dirige a 112km/h





Problemas Apropriados para RNAs

- Aprendizagem de RNAs é bem adaptada para problemas onde os dados de treinamento correspondem a dados com ruído e dados complexos, como os obtidos através de câmeras e microfones.

Problemas Apropriados para RNAs

- Instâncias são representadas por pares atributo-valor.
  - A função alvo a ser aprendida é descrita por um vetor de características (números reais).
- O valor do conceito alvo (i.e. a saída da função alvo) pode ser valores discretos, valores reais ou vetores de valores discretos e reais.
  - Geralmente o valor de cada saída é um número real entre 0 e 1 que corresponde a confiança da predição.

Problemas Apropriados para RNAs

- Os exemplos de treinamento podem conter erros.
- Grandes tempos de treinamento são aceitáveis.
  - Algoritmos de treinamento de redes geralmente consomem muito tempo (de poucos segundos a muitas horas.
  - Depende do número de pesos, número de exemplos de treinamento e outros parâmetros.

Problemas Apropriados para RNAs

- Avaliação rápida da função alvo aprendida.
  - Uma vez treinada, a avaliação da rede, dada uma nova instância, é geralmente muito rápida.
- A habilidade dos humanos entenderem a função alvo aprendida não é importante.
  - Geralmente é difícil interpretarmos os pesos aprendidos pelas redes neurais.
  - Fazem menos sentido do que um conjunto de regras (C4.5).

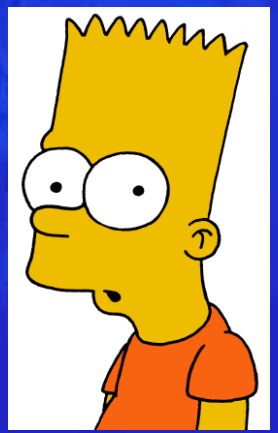
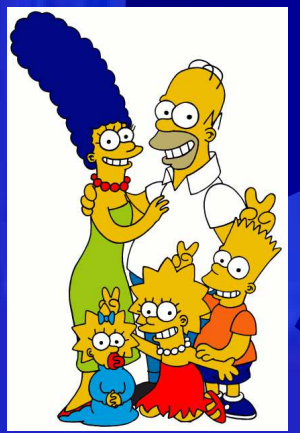
Exemplo

- 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**



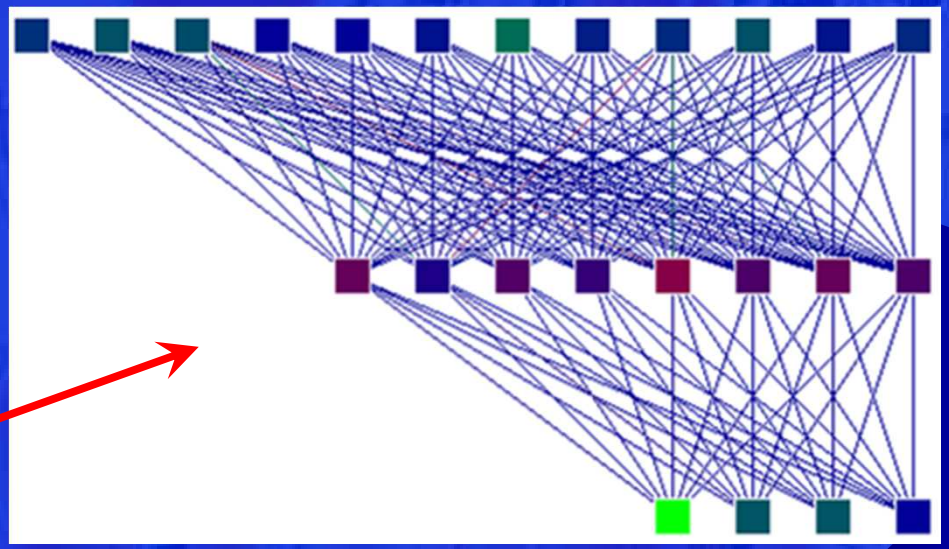



Exemplo

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)

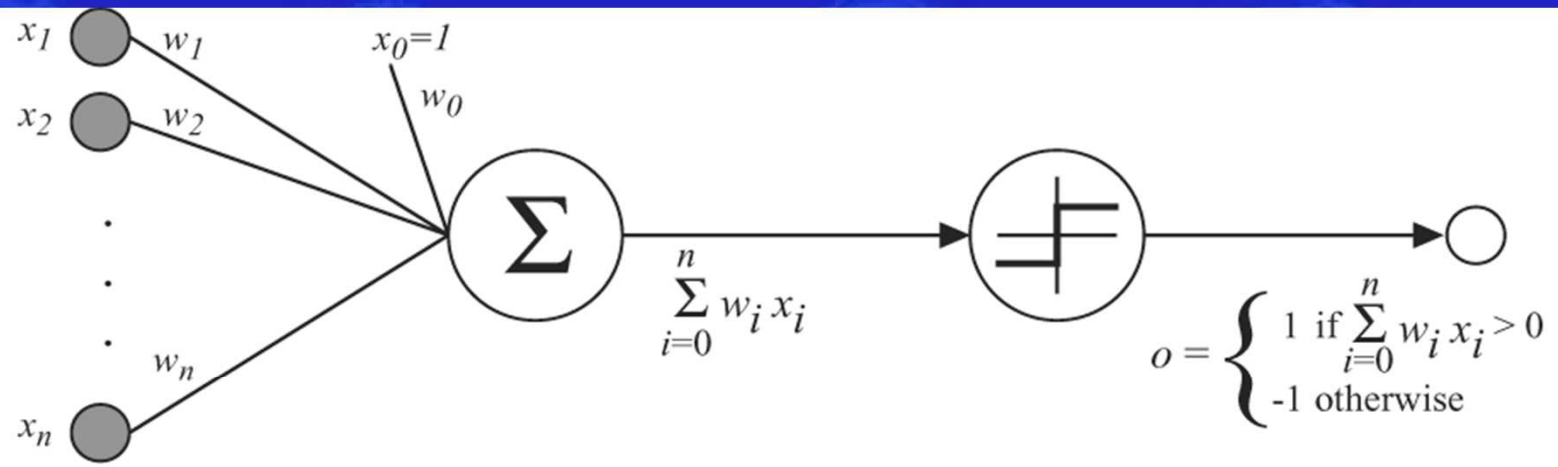
Perceptron

- Rede neural elementar baseada em uma unidade chamada *Perceptron*

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

- Mais precisamente, fornecidas as entradas x_1 a x_n, a saída $o(x_1, \ldots, x_n)$ computada 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:

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

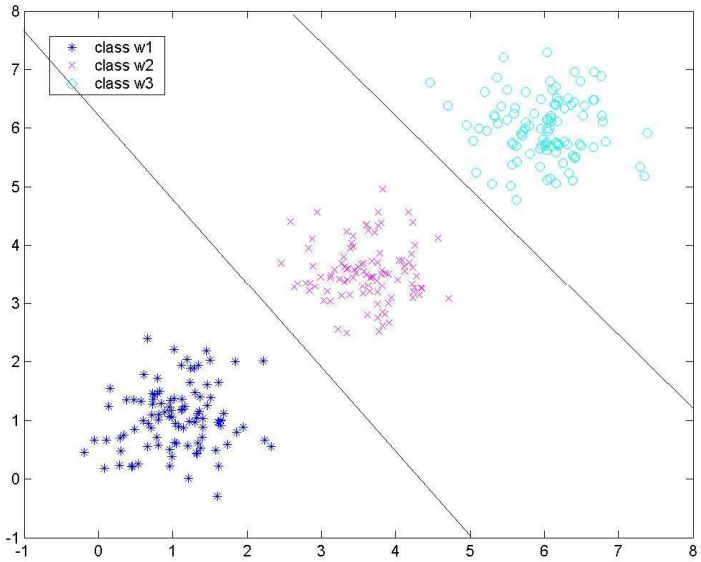
Perceptron

- 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_0 a w_n.

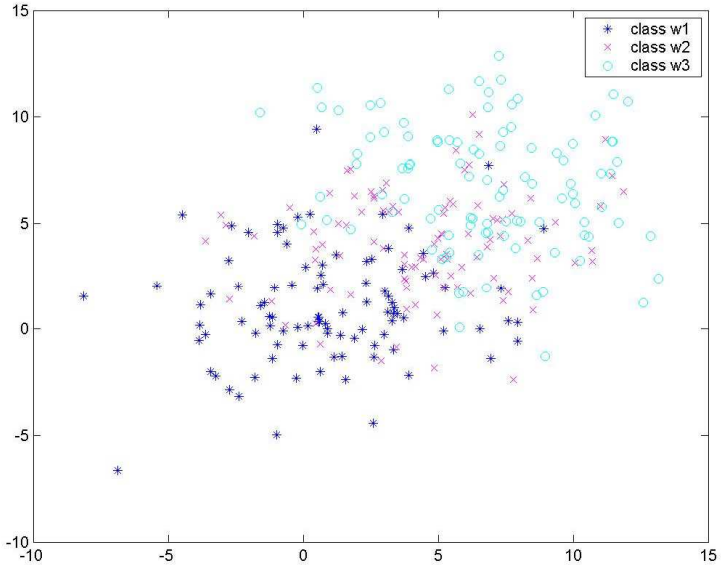
Superfícies de Decisão

- 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

Superfícies de Decisão

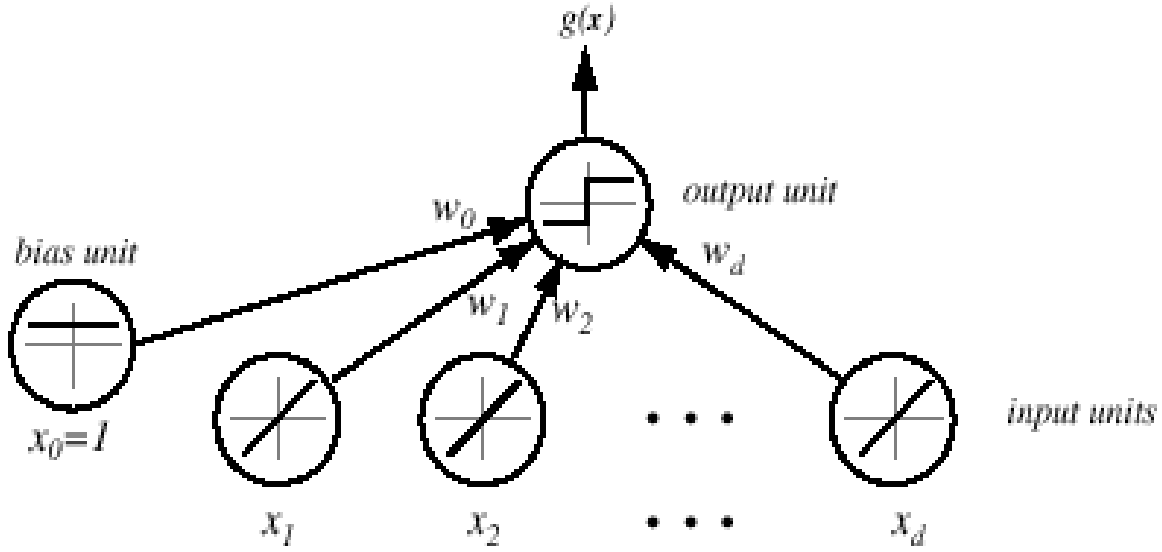


FIGURE 5.1. A simple linear classifier having d input units, each corresponding to the values of the components of an input vector. Each input feature value x_i is multiplied by its corresponding weight w_i; the effective input at the output unit is the sum all these products, $\sum w_i x_i$. We show in each unit its effective input-output function. Thus each of the d input units is linear, emitting exactly the value of its corresponding feature value. The single bias unit unit always emits the constant value 1.0. The single output unit emits a $+1$ if $\mathbf{w}^t\mathbf{x} + w_0 > 0$ or a -1 otherwise. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Superfícies de Decisão

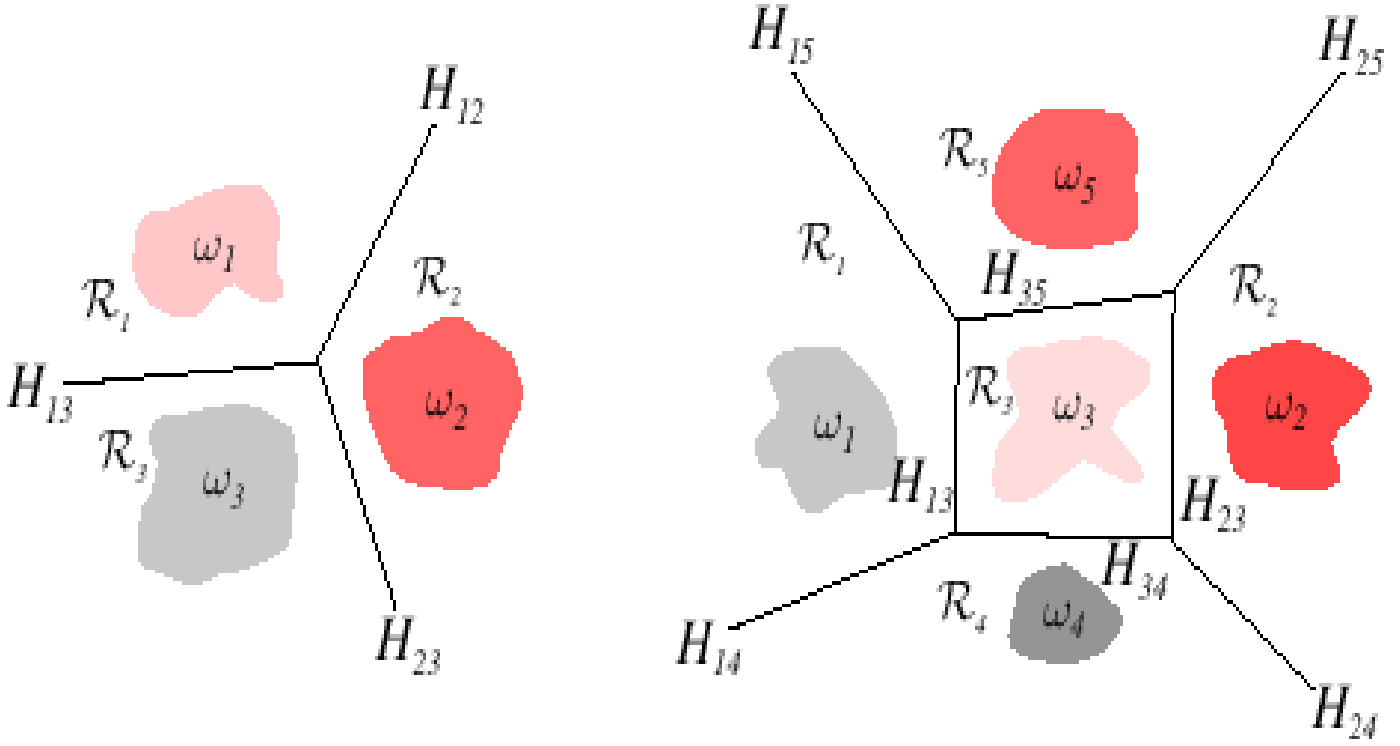


FIGURE 5.4. Decision boundaries produced by a linear machine for a three-class problem and a five-class problem. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.

Regra de Treinamento Perceptron

- Como aprender os pesos para um perceptron?
  - Problema: determinar um vetor de pesos que faça o perceptron produzir a saída correta (−1 ou +1) para cada um dos exemplos de treinamento.
  - Solução: Começar com um vetor de pesos aleatórios e aplicar iterativamente a regra *perceptron* para cada exemplo de treinamento, modificando os pesos cada vez que ele classificar um exemplo erroneamente.
    - Este processo é repetido várias vezes até que o *perceptron* classifique todos os exemplos de treinamento corretamente.

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 (e.g. 0.1) chamada de *taxa de aprendizagem*.

Regra de Treinamento Perceptron

- Se o exemplo de treinamento é classificado corretamente:

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

- 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:
  - Os dados de treinamento são linearmente separáveis;
  - η é suficientemente pequeno.

- Porém: falha em convergir se os dados forem *linearmente não–separáveis*.

Alternativa: 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.
- Idéia chave: usar a descida do gradiente para procurar no espaço de hipóteses o melhor vetor de pesos.
- Considerando uma unidade linear, isto é, um perceptron sem limiar:

Descida do Gradiente

- Considere uma unidade linear simples, onde:

$$o = w_0 + w_1 x_1 + \ldots + w_n x_n$$

- Especificando uma medida para o erro de treinamento de uma hipótese (vetor de pesos) relativamente aos exemplos de treinamento:

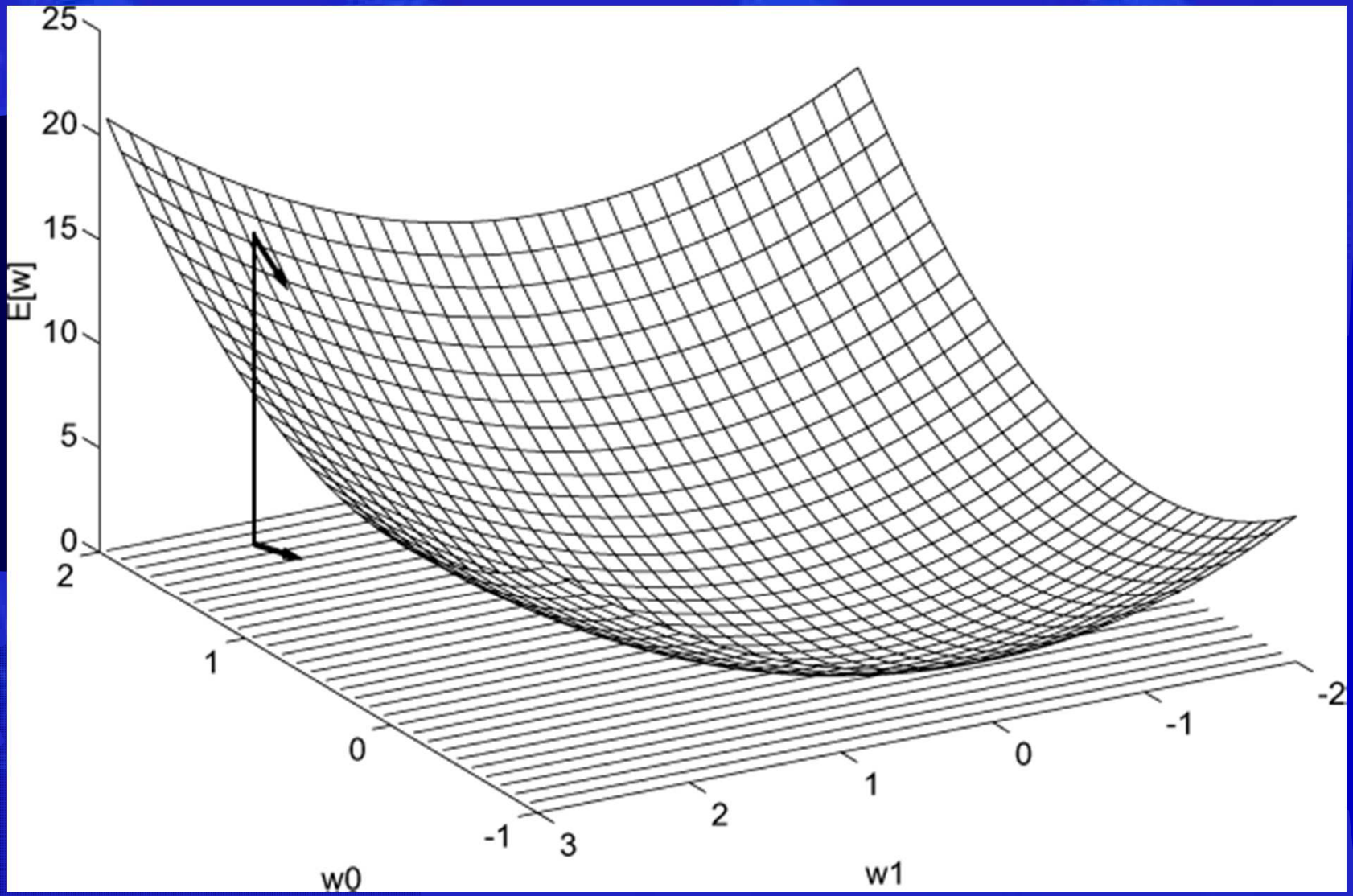
$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

  - D é o conjunto de exemplos de treinamento.
  - t_d é o valor alvo para o exemplo de treinamento d.
  - o_d é a saída da unidade linear para o exemplo d.
  - $E(w)$ é a metade do quadrado da diferença entre saída alvo e unidade linear de saída somada sobre todos os exemplos de treinamento.

Descida do Gradiente

- O algoritmo de descida do gradiente pode entendido através da visualização do espaço de hipóteses.
- A descida do gradiente determina um vetor de pesos que minimiza *E*, começando com um vetor inicial de pesos arbitrário e modificando–o repetidamente em pequenos passos.
- A cada passo, o vetor de pesos é alterado na direção que produz a maior queda ao longo da superfície de erro.
- Este processo continua até atingir um erro mínimo global.

Descida do Gradiente



Descida do Gradiente

- Assim, a regra para atualização dos pesos para o gradiente descendente é

$$\Delta w_i = \eta = \sum_{d \in D} (t_d - o_d) x_{id}$$

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

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

  - For each linear unit weight w_i, Do

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

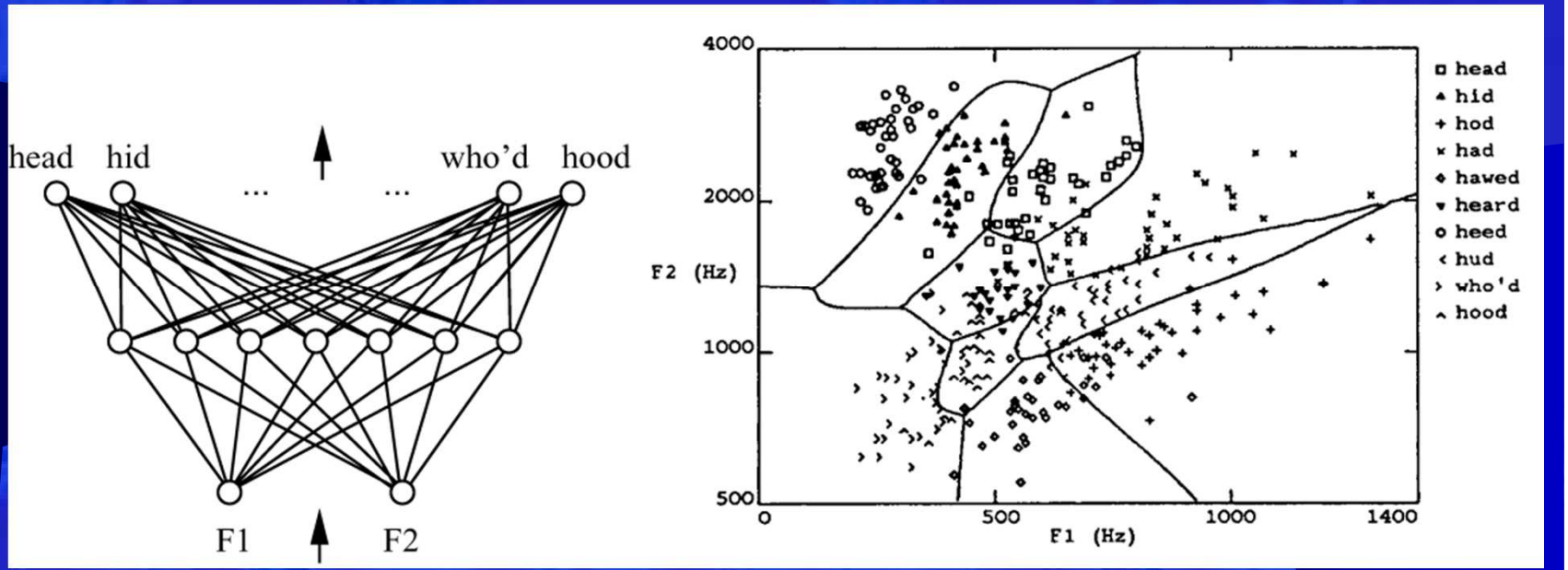
Resumo

- A regra de treinamento *perceptron* tem sucesso se:
  - Exemplos de treinamento são linearmente separáveis
  - Taxa de aprendizagem η for suficientemente pequena
- Regra de treinamento da unidade linear utiliza a *descida do gradiente*
  - Convergência garantida para a hipótese com erro quadrado mínimo
  - Dada uma taxa de aprendizagem η suficientemente pequena
  - Mesmo quando dados de treinamento contém ruído
  - Mesmo quando dados de treinamento não forem separáveis

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**.
- Rede multicamadas podem representar superfícies de decisão altamente não lineares.
- Por exemplo, uma típica rede multicamadas e sua superfície de decisão (Fig.).

Redes Multicamadas



- Exemplo: Distinção entre 10 vogais possíveis no reconhecimento da fala.

Redes Multicamadas

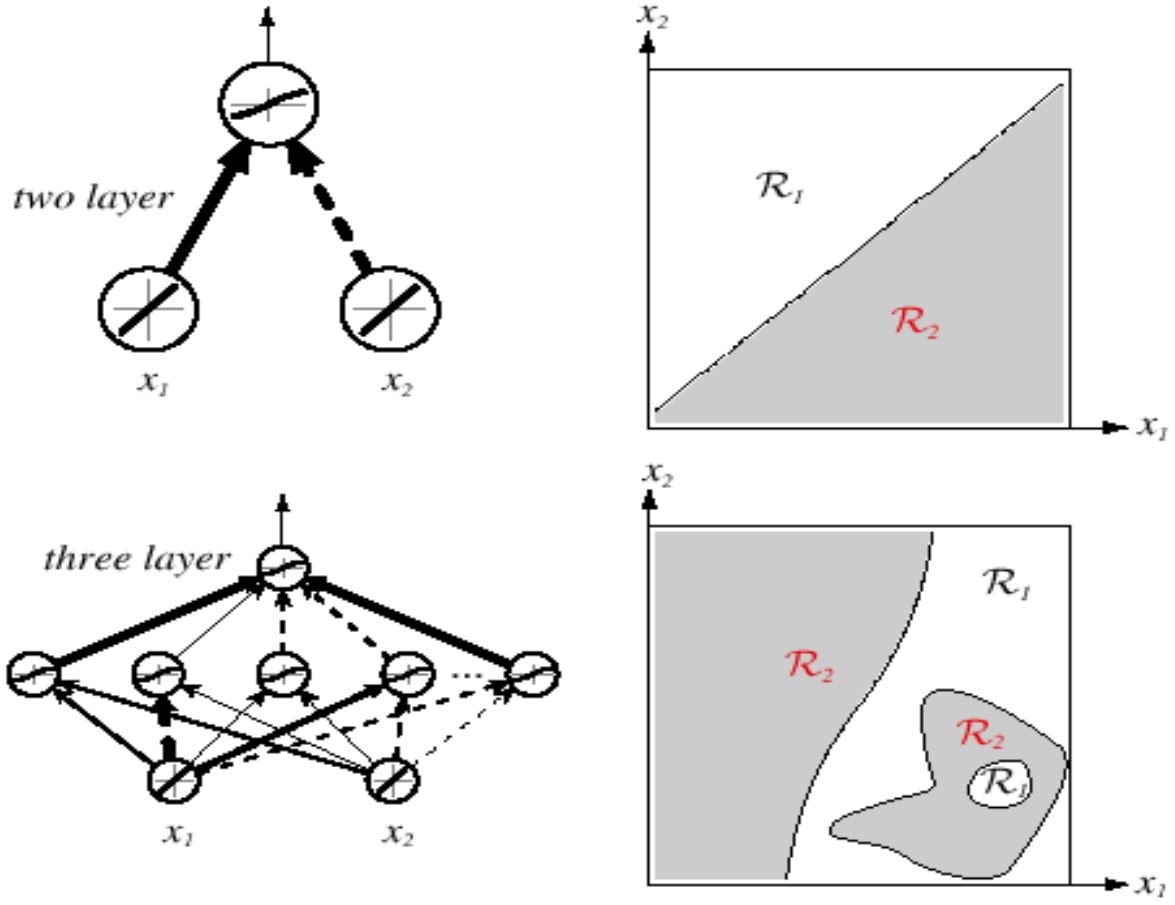
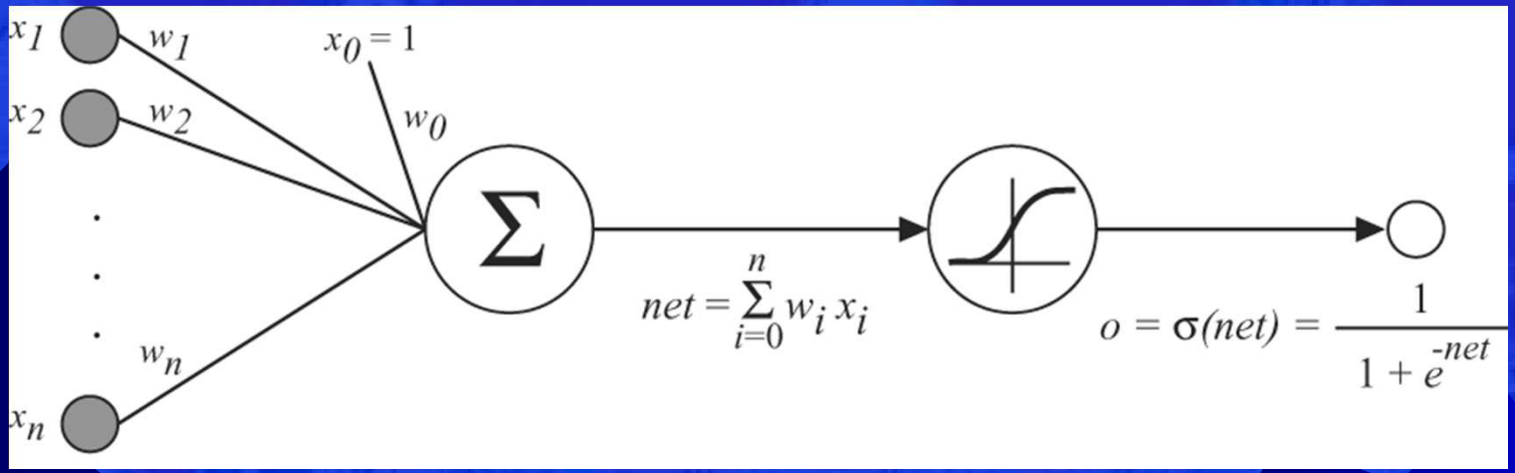


FIGURE 6.3. Whereas a two-layer network classifier can only implement a linear decision boundary, given an adequate number of hidden units, three-, four- and higher-layer networks can implement arbitrary decision boundaries. The decision regions need not be convex or simply connected. From: Richard O.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

Redes Multicamadas

- Que tipo de unidades devemos utilizar como base de uma rede multicamadas?
  - Lineares ?
  - Múltiplas camadas de unidades lineares cascateadas produzem somente funções lineares (Fig.).
- Redes capazes de representar funções altamente não lineares.
  - Unidade cuja saída seja uma função não linear de suas entradas → ***unidade sigmoidal***.

Unidade *Sigmoidal*



- $\sigma(x)$ é a função sigmoidal: $\frac{1}{1+e^{-x}}$
- Propriedade interessante: $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$
- Podemos derivar regras do gradiente descendente para treinar:
  - Uma unidade sigmoidal
  - Redes multicamadas de unidades sigmoidais → *backpropagation*

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*

- Como temos múltiplas unidades de saída, redefinimos E como sendo a soma dos erros sobre todas as unidades de saída da rede:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

  - *outputs* é o conjunto de unidades de saída na rede
  - t_{kd} valor alvo associado com a k–ésima unidade de saída e exemplo de treinamento d.
  - o_{kd} valor de saída associado com a k–ésima unidade de saída e exemplo de treinamento d.

Algoritmo *Backpropagation*

- Problema de aprendizagem do algoritmo *backpropagation:*
  - Busca em um amplo espaço de hipóteses definido por todos os valores de pesos possíveis para todas as unidades na rede.
  - Encontrar a hipótese, i.e. os pesos que minimizem o erro médio quadrático (E).

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*

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, *where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.*

η *is the learning rate (e.g., .05).* n_{in} *is the number of network inputs,* n_{hidden} *the number of units in the hidden layer, and* n_{out} *the number of output units.*

The input from unit i into unit j is denoted x_{ji}, *and the weight from unit i to unit j is denoted* w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji} \tag{T4.5}$$

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
- Facilmente generalizada para grafos diretos arbitrários
- 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

Generalização e Sobreajuste

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

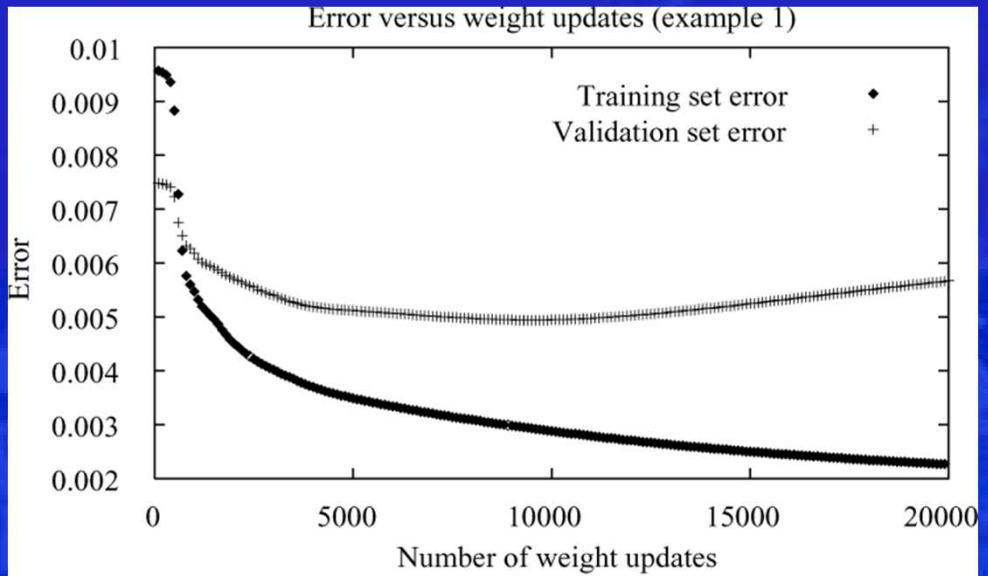
- Quando devemos parar o treinamento, i.e. parar de atualizar os pesos?
  - Escolha óbvia: continuar o treinamento até que o erro (E) seja menor do que um valor pré-estabelecido.
  - Porém, isto implica em sobreajuste (*overfitting*) !!!

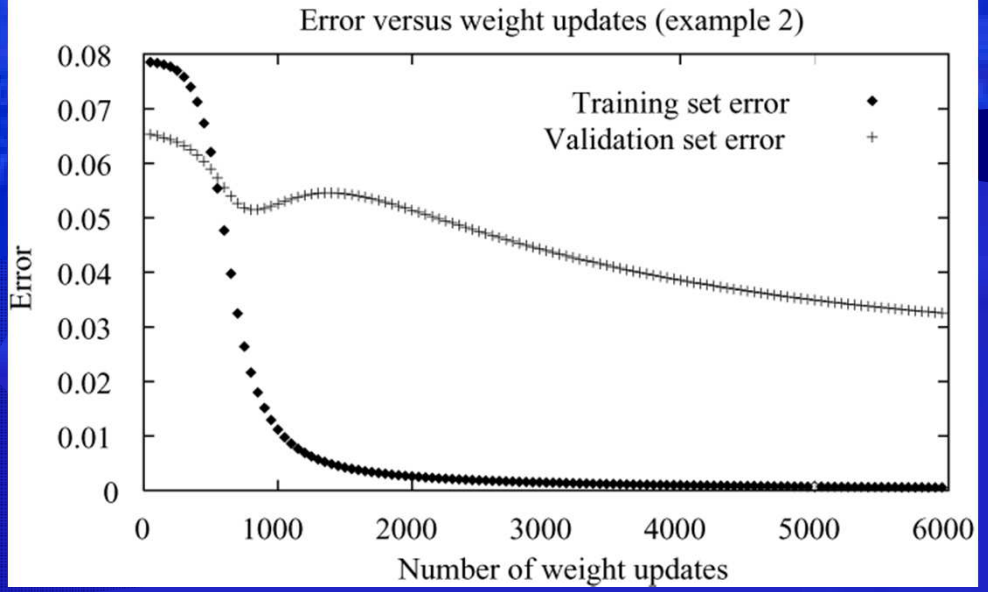
Generalização e Sobreajuste

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

- A figura 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

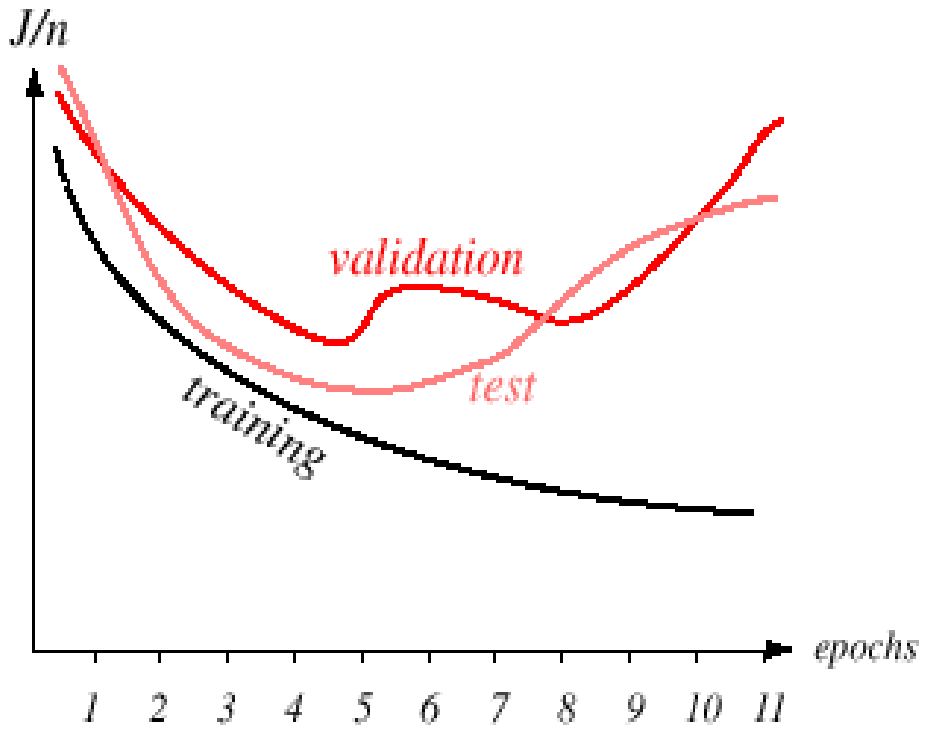


FIGURE 6.6. A learning curve shows the criterion function as a function of the amount of training, typically indicated by the number of epochs or presentations of the full training set. We plot the average error per pattern, that is, $1/n \sum_{p=1}^{n} J_p$. The validation error and the test or generalization 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*. Copyright © 2001 by John Wiley & Sons, Inc.

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.

Reconhecimento de Faces RNAs

- Tarefa de Aprendizagem: classificar de imagens de faces de várias pessoas em várias poses.
  - 20 pessoas diferentes
  - 32 imagens por pessoa
  - Variações na expressão: alegre, triste, braba, neutra, etc...
  - No total: 624 imagens 120 x 128 em níveis de cinza.
- Função Alvo: uma variedade de funções tais como identificar uma pessoa, direção em que ela está olhando, o sexo, usando óculos ou não, etc . . .

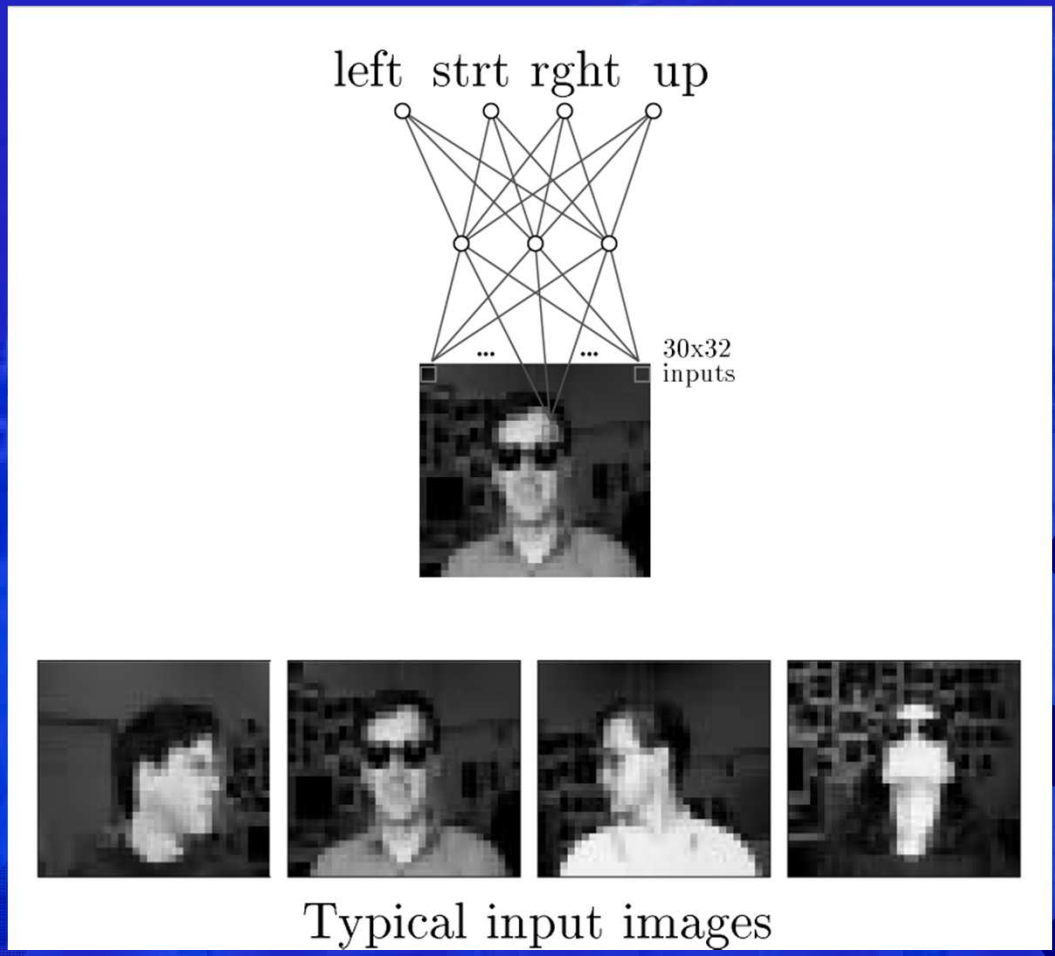
Reconhecimento de Faces RNAs

- Tarefa Particular: aprender a direção em que a pessoa está olhando (left, right, straight ahead, upward)
- Resultados: treinamento em 260 imagens. 90% de precisão em um conjunto de testes.
- Codificação da Entrada: imagens de 30 x 32 pixels. Uma redução na resolução das imagens originais.
- Codificação da Saída: uma única unidade de saída com valores 0.2, 0.4, 0.6 e 0.8 para codificar cada uma das possíveis saídas. Não !!!! Ao invés disso, 4 unidades de saída (0,0,0,1) ...(1,0,0,0)

Reconhecimento de Faces RNAs

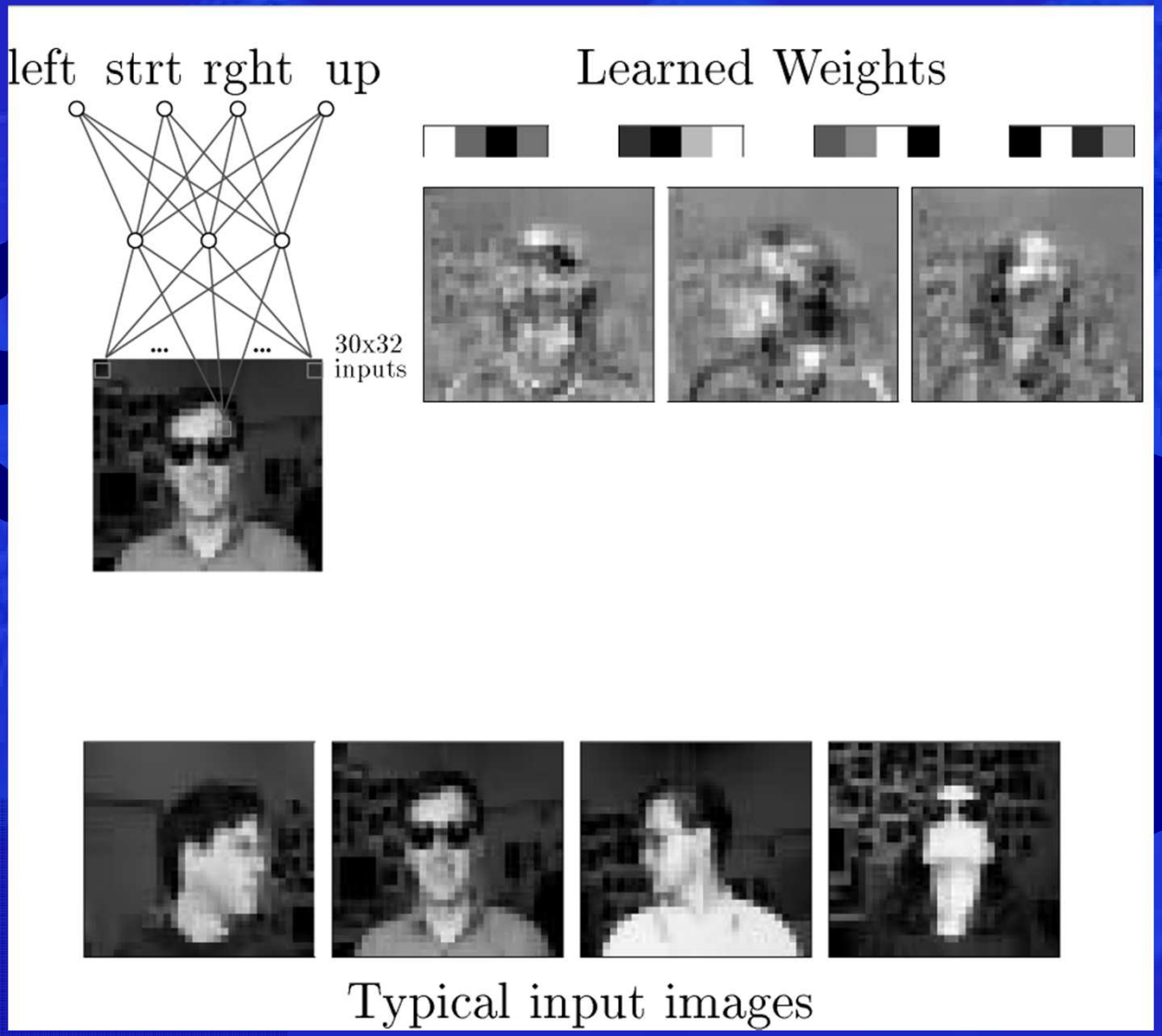
- Estrutura da rede: entradas e saídas são determinadas pelos dados. Na camada escondida:
  - 3 neurônios: 90% (5 minutos)
  - 30 neurônios: 92% (1 hora)

Reconhecimento de Faces RNAs



- Aprendendo corretamente posição da cabeça, reconhecendo 1 em 20 faces.

Reconhecimento de Faces RNAs



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 de hipóteses considerado pelo algoritmo *backpropagation* é o espaço de todas as funções que podem ser representadas pelos pesos.
- O *backpropagation* busca o espaço de hipóteses possíveis usando a descida do gradiente para reduzir iterativamente o erro em uma rede (ajustar aos dados de treinamento).

Resumo

- Sobreajuste resulta em redes que não generalizam bem. Métodos de validação cruzada para aliviar este problema (utilizar um conjunto de validação).
- *Backpropagation* é o algoritmo de aprendizagem mais comum, porém existem muitos outros . . .