

REDES NEURONAIS

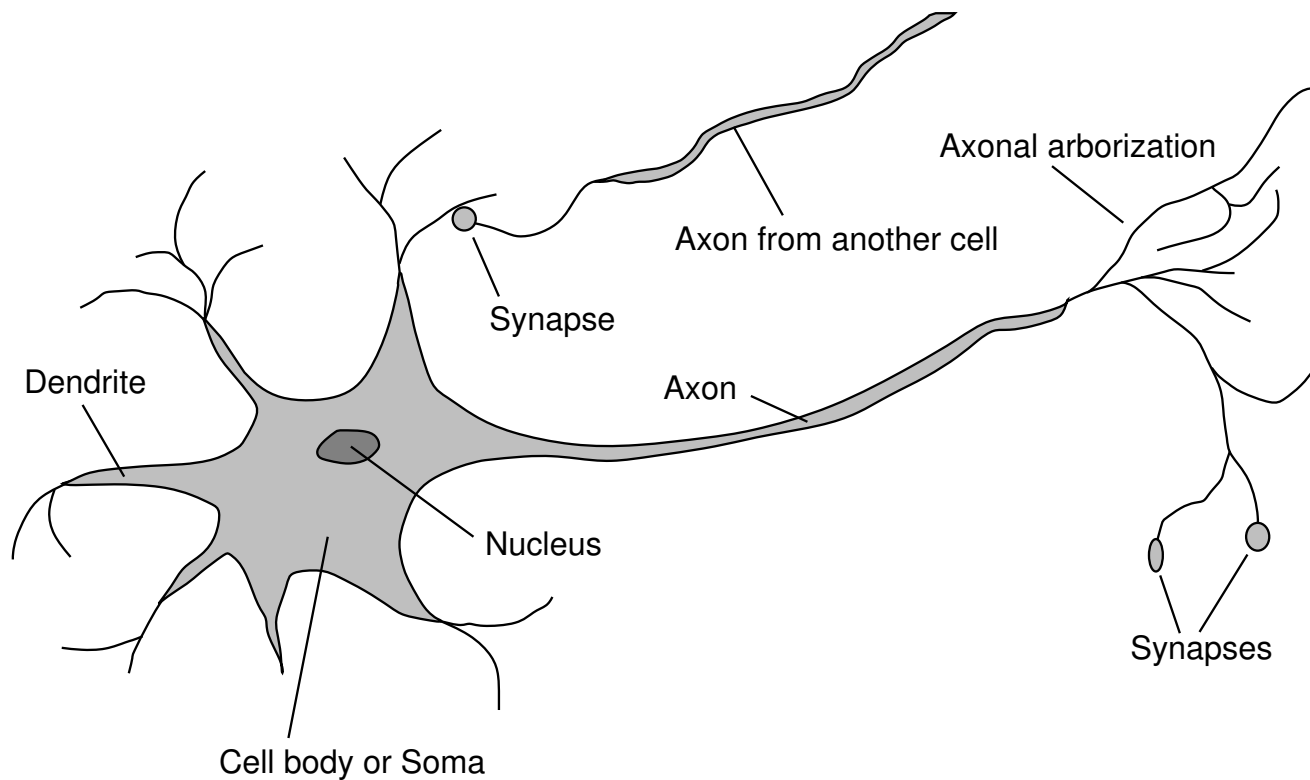
ANO LECTIVO 2013/2014 – SECÇÃO 18.7

Resumo

- ◇ Cérebro
- ◇ Redes Neurais
- ◇ Perceptrões
- ◇ Redes neuronais multicamada
- ◇ Aplicações de redes neuronais

Cérebro

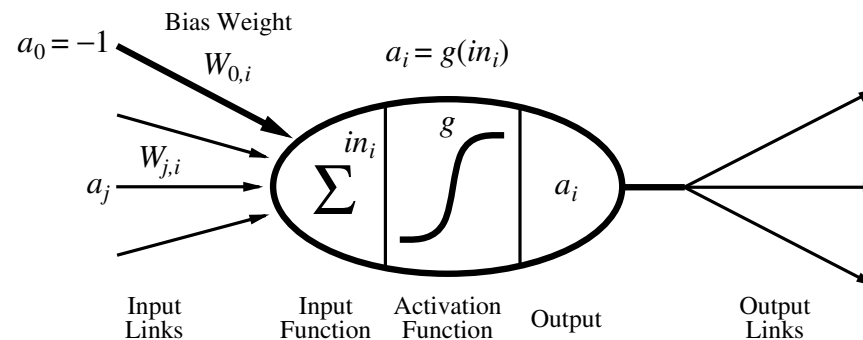
10^{11} neurónios de > 20 tipos, 10^{14} sinapses, 1ms–10ms tempo de ciclo
Sinais têm ruído “spike trains” de potenciais eléctricos



Unidade de McCulloch–Pitts

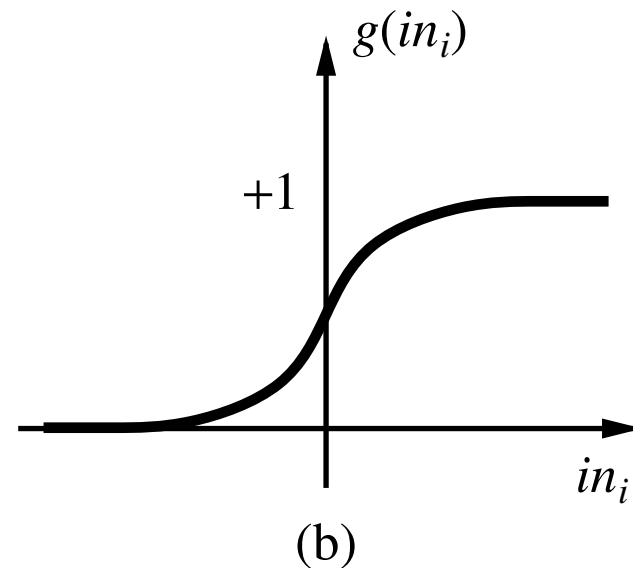
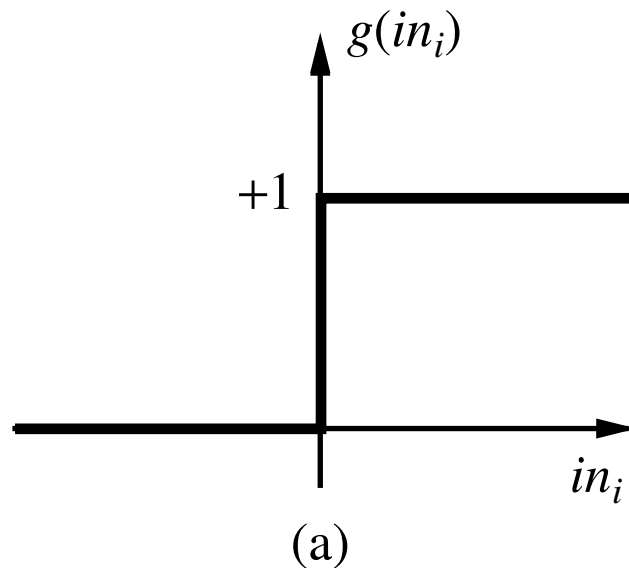
Saída é uma função linear “esmagada” das entradas:

$$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



Uma simplificação rude dos neurónios reais, mas o seu objectivo é obter conhecimento sobre o que conseguem fazer as redes de unidades simples

Funções de activação



(a) é a **função degrau** ou **função limiar**. Neste caso designamos a unidade por perceptrão.

(b) é a função **sigmóide** $1/(1 + e^{-x})$. Unidade designada por perceptrão sigmóide.

Alteração da polarização (bias weight) $W_{0,i}$ desloca a localização do limiar

Funções de activação

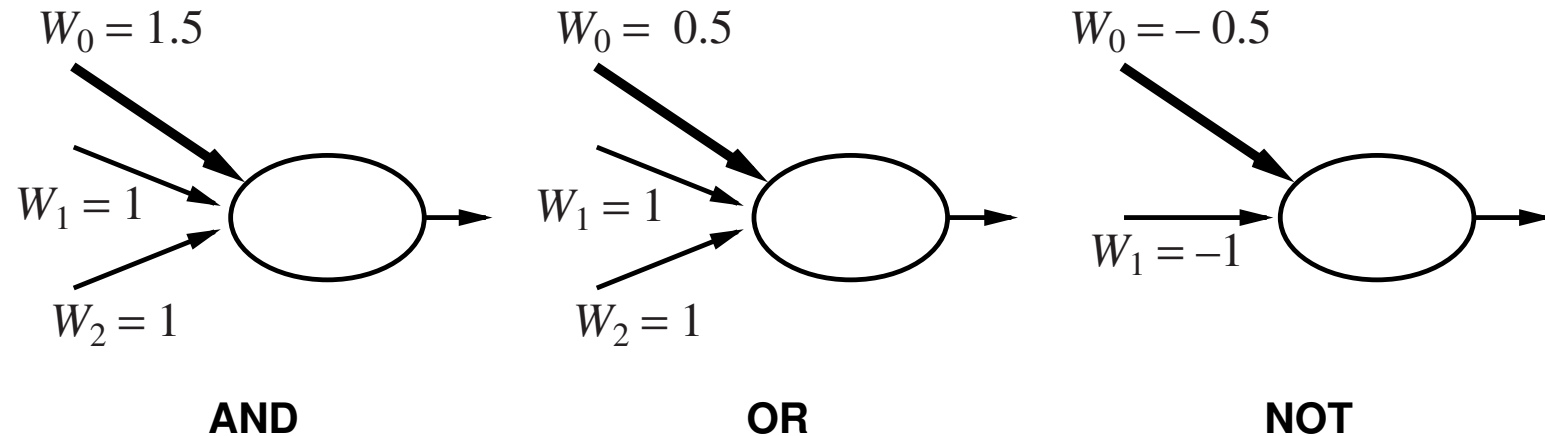
É importante que a função seja não-linear, caso contrário a saída reduz-se a uma combinação linear das entradas!

Os algoritmos de aprendizagem que estudaremos assumem que a função é diferenciável.

Outros exemplos:

- Tangente Hiperbólica (contra-domínio $[-1, 1]$)
- Seno (contra-domínio $[-1, 1]$)

Implementação das funções lógicas



McCulloch e Pitts: qualquer função Booleana pode ser implementada combinando as construções anteriores.

Topologia das redes

Redes alimentadas para a frente:

- rede monocamada de neurónios/perceptrões
- rede multicamada de neurónios/perceptrões

Redes alimentadas para a frente implementam funções, não têm estado interno

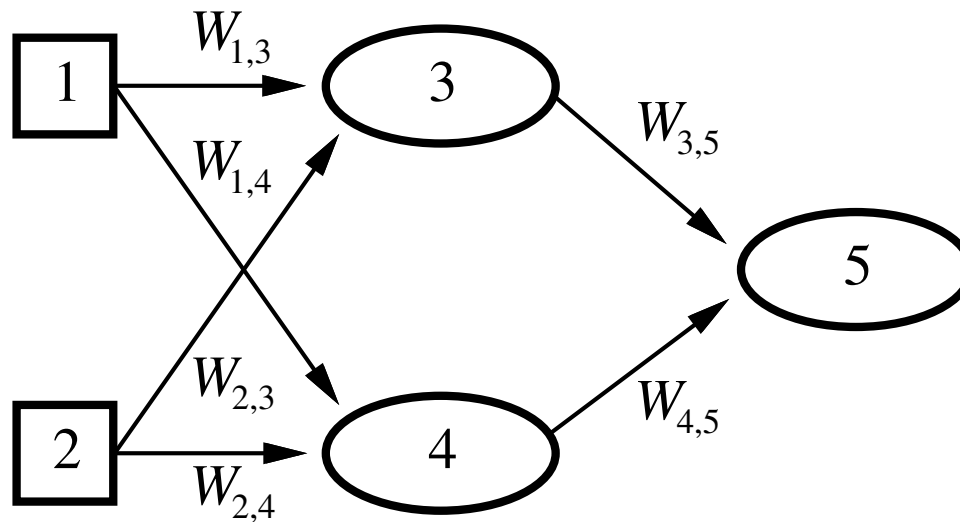
Redes recorrentes:

- Redes de Hopfield têm pesos simétricos ($W_{i,j} = W_{j,i}$)
 $g(x) = \text{sign}(x)$, $a_i = \pm 1$; **memórias holográficas associativas**
- Máquinas de Boltzmann utilizam funções de activação estocásticas,

Nota: As redes recorrentes têm ciclos dirigidos com atrasos (delays)

⇒ têm estado interno (como flip-flops), podem oscilar etc.

Exemplo de rede alimentada para a frente

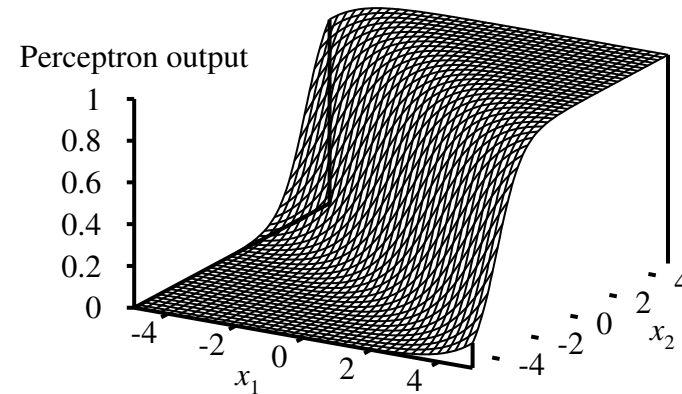
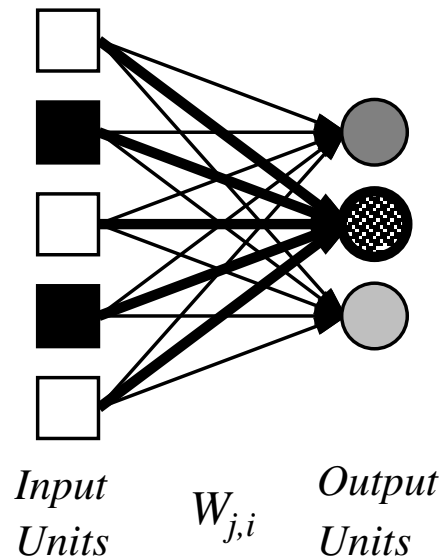


Rede alimentada para a frente = família parametrizada de funções não-lineares:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

Ajustando os pesos altera-se a função: efectuar aprendizagem desta maneira!

Rede monocamada de perceptrões



As unidades operam separadamente—não existem pesos partilhados

Ajustando os pesos altera-se a localização, orientação e inclinação do declive

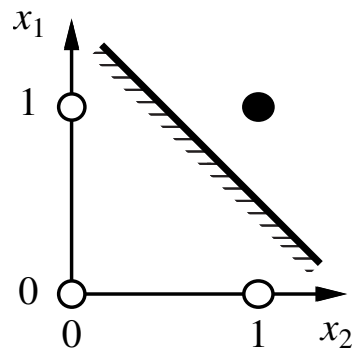
Expressividade dos perceptrões

Considere-se o perceptrão que usa a função de activação $g = \text{degrau}$ (Rosenblatt, 1957, 1960)

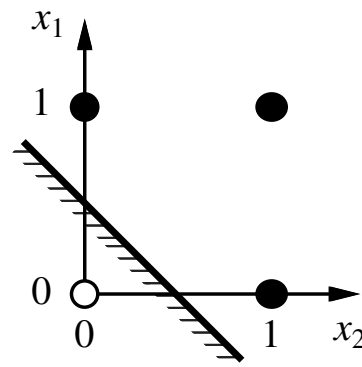
Pode representar AND, OR, NOT, maioria, etc., mas **não** o XOR

Representa um **separador linear** no espaço de entradas:

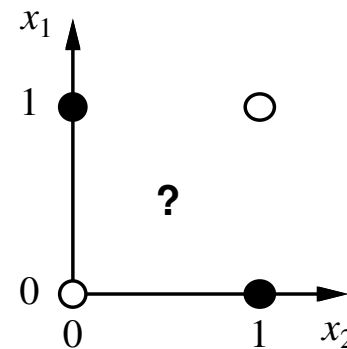
$$\sum_j W_j x_j > 0 \quad \text{or} \quad \mathbf{W} \cdot \mathbf{x} > 0$$



(a) x_1 **and** x_2



(b) x_1 **or** x_2



(c) x_1 **xor** x_2

Minsky & Papert (1969) “furaram” o balão das redes neuronais

Regra Delta - Widrow e Hoff

Aprender por ajustamento dos pesos de forma a reduzir o **erro** no conjunto de exemplos de treino. Tratamos primeiro o caso de função de activação linear:

O **erro quadrático** para um exemplo com entrada $\mathbf{x}$ e valor correcto y é

$$E = Err^2 \equiv (y - h_{\mathbf{W}}(\mathbf{x}))^2 ,$$

Recorrer ao método do gradiente descendente para minimizar E :

$$\begin{aligned} \frac{\partial E}{\partial W_j} &= 2Err \times \frac{\partial Err}{\partial W_j} = 2Err \times \frac{\partial}{\partial W_j} \left(y - \sum_{j=0}^n W_j x_j \right) \\ &= -2Err \times x_j \end{aligned}$$

Regra simples para actualização dos pesos (factor 2 incluído em η):

$$W_j \leftarrow W_j + \eta \times Err \times x_j \quad (\eta \text{ é ritmo de aprendizagem})$$

Também utilizada por Rosenblatt com função de activação limiar.

Neurónios (regra delta generalizada)

Aprender por ajustamento dos pesos de forma a reduzir o **erro** no conjunto de exemplos de treino. Caso de funções de activação diferenciáveis:

Recorrer novamente ao método do gradiente descendente para minimizar E :

$$\begin{aligned}\frac{\partial E}{\partial W_j} &= 2Err \times \frac{\partial Err}{\partial W_j} = 2Err \times \frac{\partial}{\partial W_j} \left(y - g(\sum_{j=0}^n W_j x_j) \right) \\ &= -2Err \times g'(in) \times x_j\end{aligned}$$

Regra simples para actualização dos pesos:

$$W_j \leftarrow W_j + \eta \times Err \times g'(in) \times x_j$$

E.g., erro +ve $\Rightarrow$ aumenta saída da rede

$\Rightarrow$ aumento de pesos com entradas com +ve , diminuir em entradas -ve

Nota: $sig'(x) = sig(x) \times (1 - sig(x))$

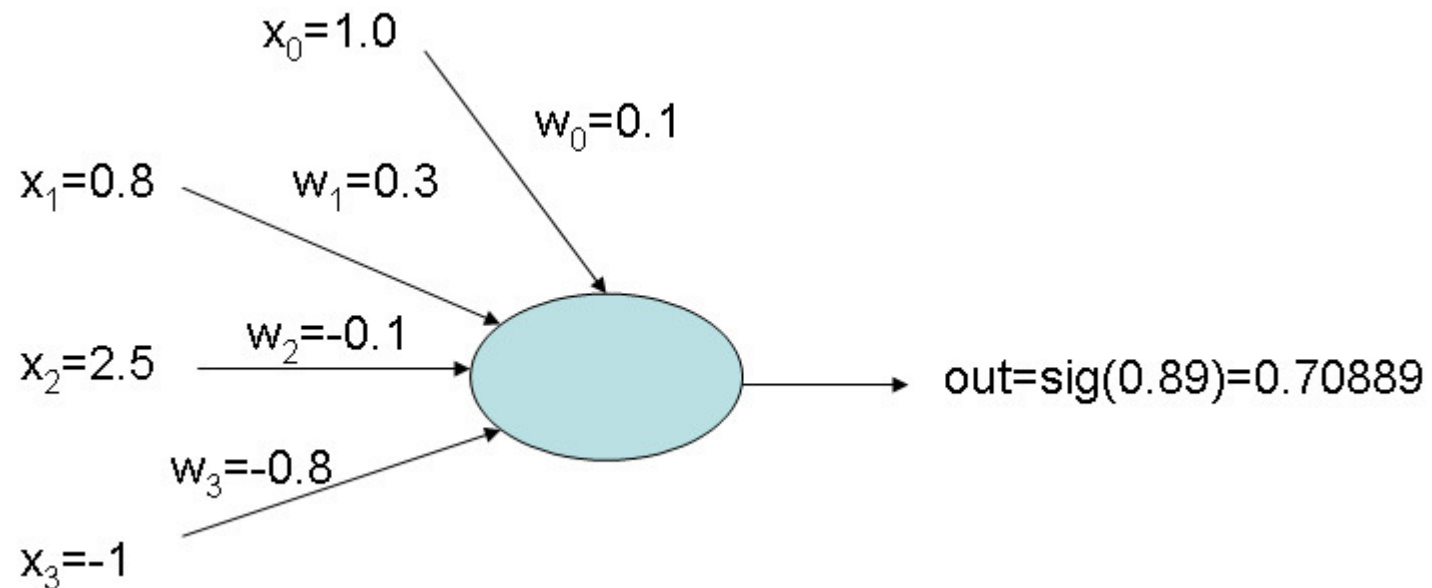
Algoritmo de Aprendizagem

```
function PERCEPTRON-LEARNING(network, examples,  $\eta$ ) returns a  
perceptron hypothesis  
  inputs: network, um perceptrão com pesos  $W_j$ ,  $J = 0, \dots, n$  e função de activação  $g$   
           examples, um conjunto de exemplos, com entrada  $\mathbf{x} = x_1, \dots, x_n$  e saída  $y$   
            $\eta$ , o ritmo de aprendizagem  
  
  repeat  
    for each  $e$  in examples do  
      /* Calcular o valor de saída para este exemplo */  
       $in \leftarrow \sum_{j=0}^n W_j x_j[e]$   
       $out \leftarrow g(in)$   
      /* Calcular o erro */  
       $Err \leftarrow y[e] - out$   
      /* Actualizar os pesos das entradas */  
      for each entrada  $j$  do perceptrão do  
         $W_j \leftarrow W_j + \eta \times Err \times g'(in) \times x_j[e]$   
      end  
    end  
  until se tenha atingindo um critério de paragem
```

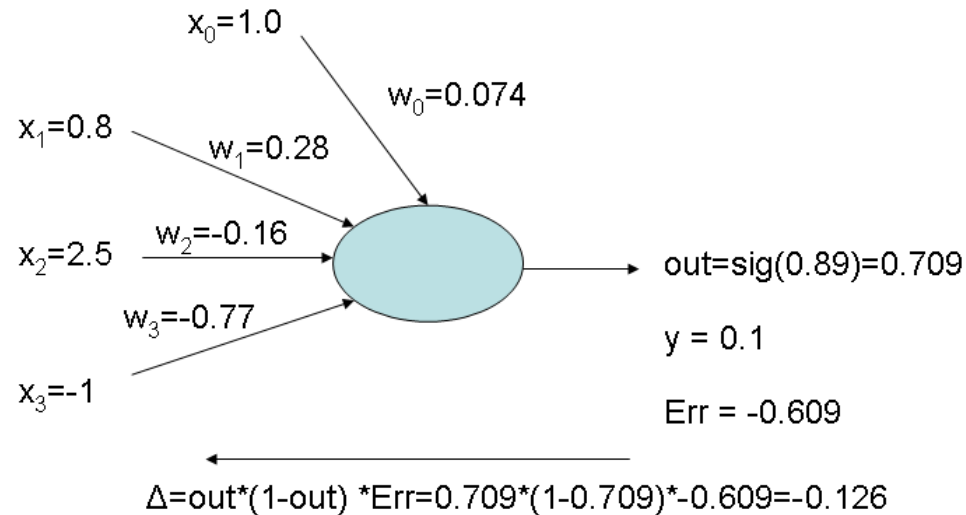
Notas:

Com função de activação sigmóide a expressão de actualização dos pesos é: $W_j \leftarrow W_j + \eta \times Err \times out \times (1 - out) \times x_j[e]$. Com função limiar a expressão de actualização de pesos é $W_j \leftarrow W_j + \eta \times Err \times x_j[e]$

Exemplo de aplicação



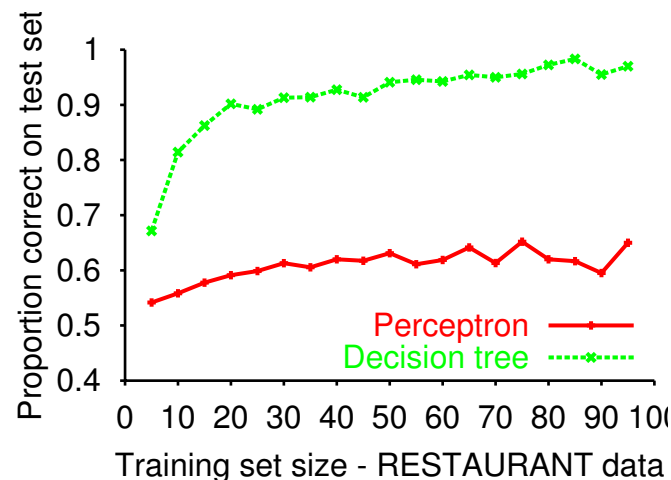
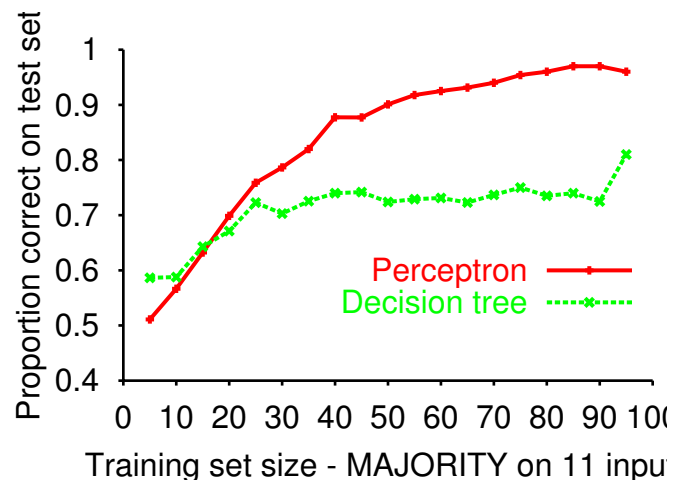
Exemplo de aplicação



Peso Inicial		Entrada	Actualização(ΔW_i)	Peso Final
W_0	0.1	$x_0 = 1.0$	$\eta * \Delta * x_0 = 0.2 * -0.126 * 1.0 = -0.025$	0.074
W_1	0.3	$x_1 = 0.8$	$\eta * \Delta * x_1 = 0.2 * -0.126 * 0.8 = -0.02$	0.28
W_2	-0.1	$x_2 = 2.5$	$\eta * \Delta * x_2 = 0.2 * -0.126 * 2.5 = -0.06$	-0.16
W_3	-0.8	$x_3 = -1.0$	$\eta * \Delta * x_3 = 0.2 * -0.126 * -1.0 = 0.03$	-0.77

Aprendizagem do perceptrão

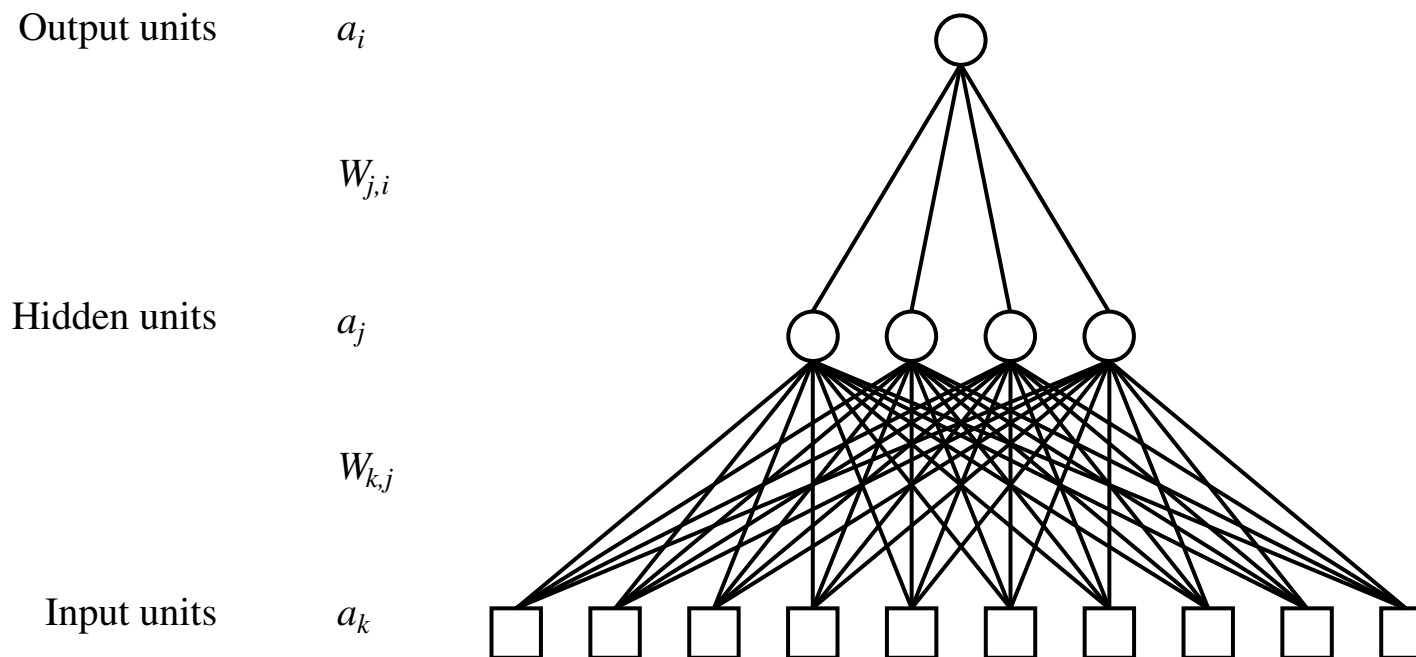
A regra de aprendizagem do perceptrão converge para uma função consistente **para qualquer conjunto de dados linearmente separável**



- ◇ Perceptrão aprende função de maioria facilmente, indução de árvore de decisão é inútil
- ◇ Indução de árvore de decisão aprende função do restaurante, perceptrão não pode representá-la.

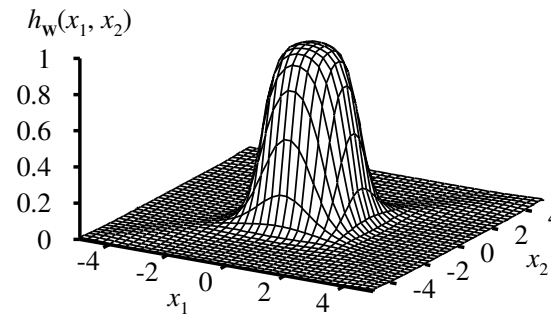
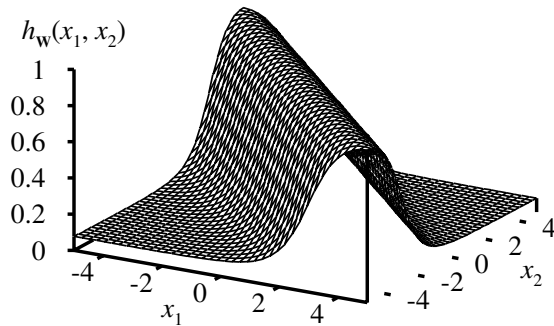
Redes multicamada

Normalmente as camadas são totalmente ligadas;
número de **unidades escondidas** tipicamente escolhido manualmente



Expressividade de redes multicamada

Todas as funções contínuas com 1 camada escondida, todas as funções com 2 camadas escondidas



- ◇ Combinar duas funções limiar opostas para construir uma crista
- ◇ Combinar duas cristas perpendiculares para construir um morro
- ◇ Juntar morros de vários tamanhos e localizações para obter qualquer superfície.
A prova requer um número exponencial de unidades escondidas.

Aprendizagem por retropropagação

Camada de saída: o mesmo para o perceptrão monocamada,

$$W_{j,i} \leftarrow W_{j,i} + \eta \times a_j \times \Delta_i$$

em que $\Delta_i = Err_i \times g'(in_i)$

Camada escondida: **retropropaga** o erro da camada de saída:

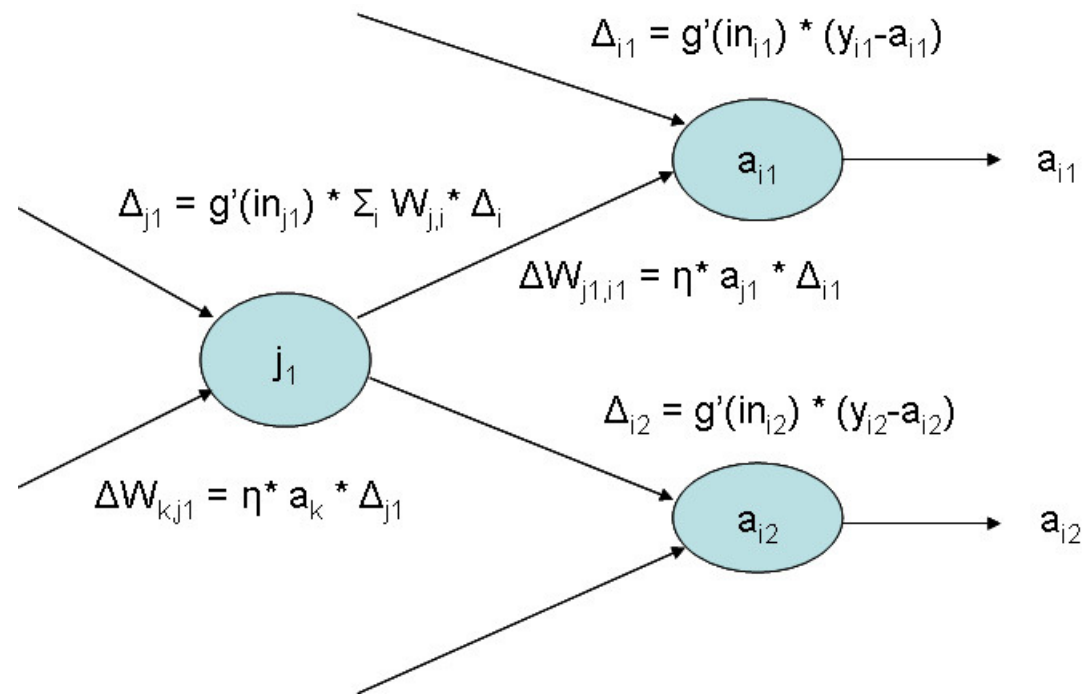
$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i .$$

Regra de actualização para os pesos da camada escondida:

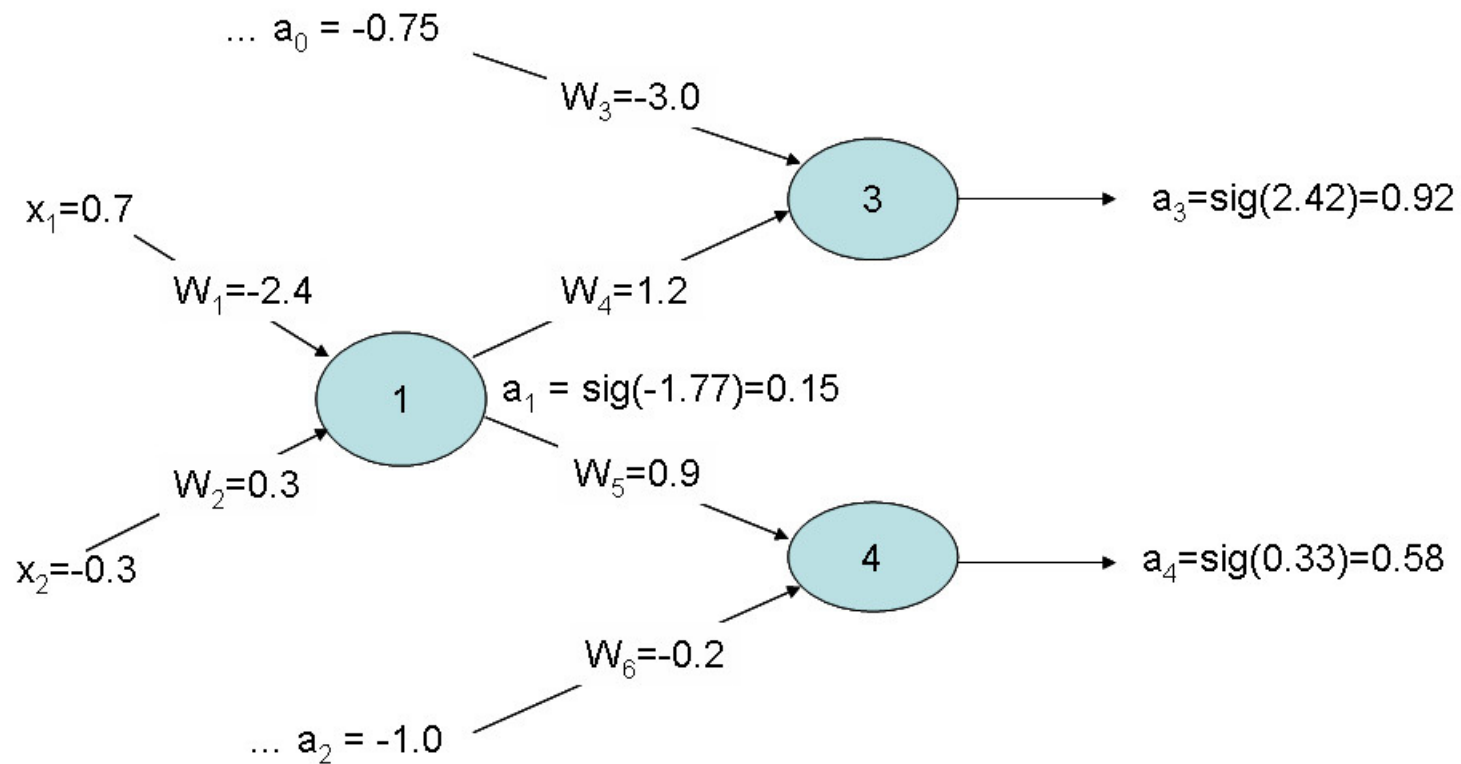
$$W_{k,j} \leftarrow W_{k,j} + \eta \times a_k \times \Delta_j .$$

(A maioria dos neurocientistas nega a existência de retropropagação no cérebro)

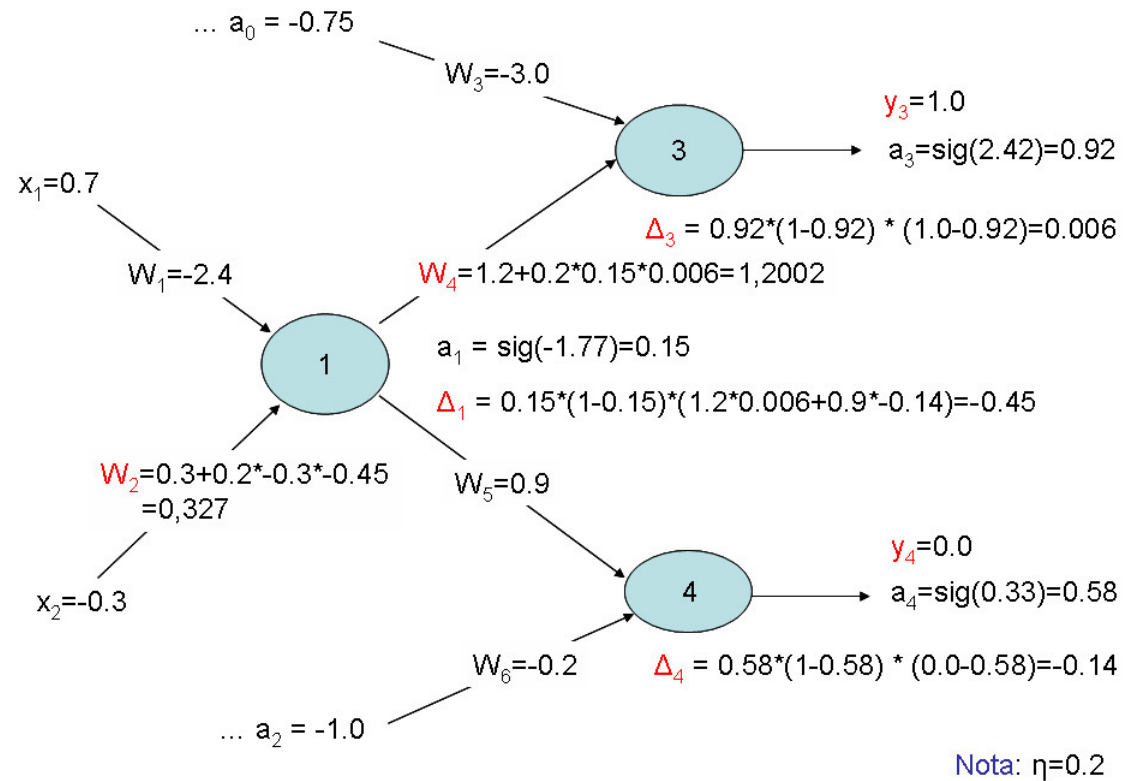
Esquema de funcionamento



Exemplo de aplicação (propagação)



Exemplo de aplicação (retropropagação)



Só mostrada actualização dos pesos W_2 e W_4 a partir dos termos de erro Δ_1 , Δ_3 e Δ_4 . Os outros pesos são alterados de maneira semelhante.

O algoritmo de retropropagação

```

function BACK-PROP-UPDATE(network, examples,  $\eta$ ) returns a
network with changed weights
  inputs: network, a multilayer network
           examples, a set of input/output pairs
            $\eta$ , the learning rate
  repeat
    for each e in examples do
      /* Compute the output for this example having inputs  $\mathbf{I}^e$  */
       $\mathbf{O}^e \leftarrow \text{RUN-NETWORK}(\textit{network}, \mathbf{I}^e)$ 
      /* Compute the error and  $\Delta$  for units in the output layer */
       $\text{Err}^e \leftarrow \mathbf{Y}^e - \mathbf{O}^e$       /*  $\mathbf{Y}^e$  are the expected output values for the example */
      /* Update the weights leading to the output layer */
       $W_{j,i} \leftarrow W_{j,i} + \eta \times a_j \times \text{Err}_i^e \times g'(in_i)$ 
      for each subsequent layer in network do
        /* Compute the error at each node */
         $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i} \Delta_i$ 
        /* Update the weights leading into the layer */
         $W_{k,j} \leftarrow W_{k,j} + \eta \times a_k \times \Delta_j$ 
      end
    end
  until network has converged
  return network

```

Derivação da regra de retropropagação

O erro quadrático num único exemplo¹ definido como

$$E = \sum_i (y_i - a_i)^2 ,$$

em que a soma inclui todos os nós da camada de saída

$$\begin{aligned} \frac{\partial E}{\partial W_{j,i}} &= -2(y_i - a_i) \frac{\partial a_i}{\partial W_{j,i}} = -2(y_i - a_i) \frac{\partial g(in_i)}{\partial W_{j,i}} \\ &= -2(y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{j,i}} = -2(y_i - a_i) g'(in_i) \frac{\partial}{\partial W_{j,i}} \left(\sum_j W_{j,i} a_j \right) \\ &= -2(y_i - a_i) g'(in_i) a_j = -2a_j \Delta_i \end{aligned}$$

¹O erro médio quadrático (MSE) é a média dos erros quadráticos de todos os exemplos. O RMSE (root mean square error) é a raiz quadrada do MSE.

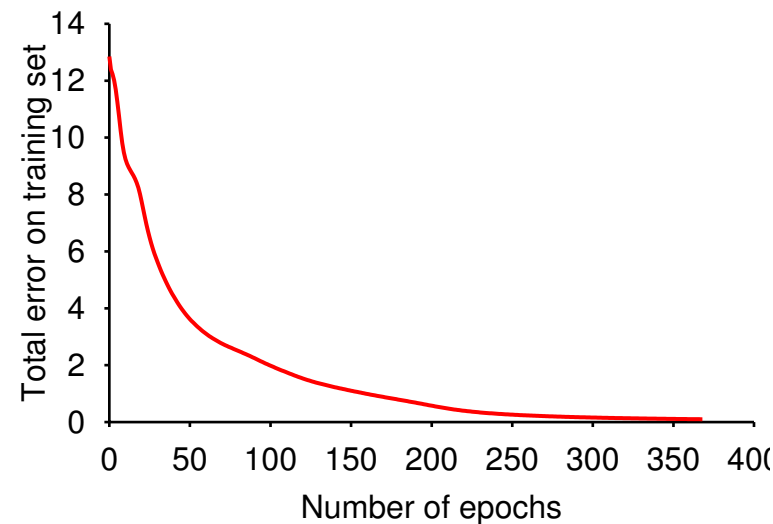
Derivação retropropagação outras camadas

$$\begin{aligned}\frac{\partial E}{\partial W_{k,j}} &= -2 \sum_i (y_i - a_i) \frac{\partial a_i}{\partial W_{k,j}} = -2 \sum_i (y_i - a_i) \frac{\partial g(in_i)}{\partial W_{k,j}} \\&= -2 \sum_i (y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{k,j}} = -2 \sum_i \Delta_i \frac{\partial}{\partial W_{k,j}} \left(\sum_j W_{j,i} a_j \right) \\&= -2 \sum_i \Delta_i W_{j,i} \frac{\partial a_j}{\partial W_{k,j}} = -2 \sum_i \Delta_i W_{j,i} \frac{\partial g(in_j)}{\partial W_{k,j}} \\&= -2 \sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial in_j}{\partial W_{k,j}} \\&= -2 \sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial}{\partial W_{k,j}} \left(\sum_k W_{k,j} a_k \right) \\&= -2 \sum_i \Delta_i W_{j,i} g'(in_j) a_k = -2 a_k \Delta_j\end{aligned}$$

Aprendizagem com Retropropagação (cont)

Em cada *época*, somar actualizações de gradiente para todos os exemplos e aplicar

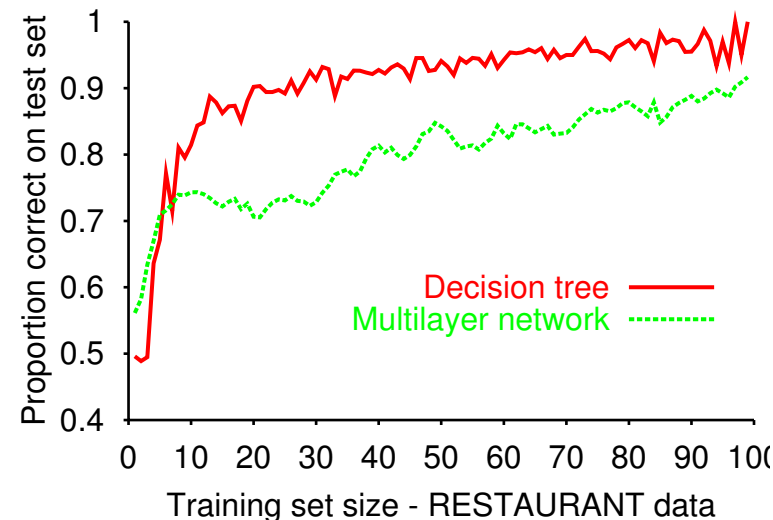
Curva de treino para 100 exemplos do restaurante: encontra ajustamente perfeito



Problemas típicos: convergência lenta, mínimos locais

Aprendizagem com Retropropagação (cont)

Curva de aprendizagem para rede multicamada com 4 unidades escondidas:

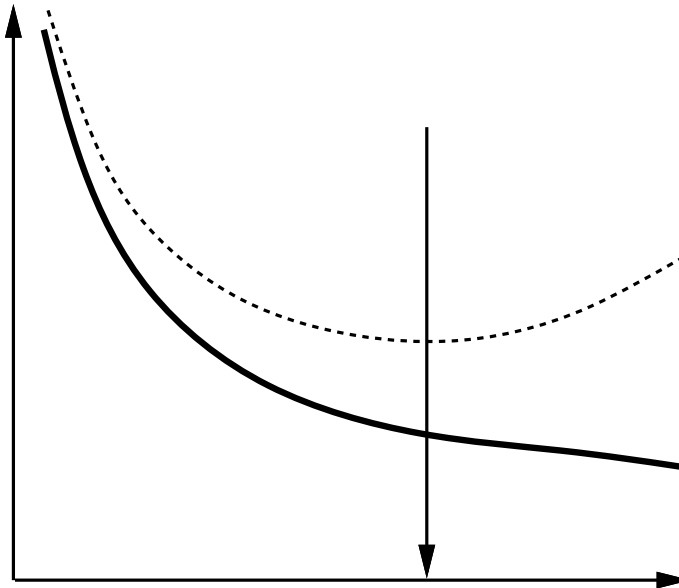


Redes multicamada são adequadas para tarefas complexas de reconhecimento de padrões, mas o resultado pode não ser facilmente compreendido

Utilização prática do algoritmo

- ◇ Para o caso da função sigmóide é habitual inicializarem-se os pesos com valores aleatórios no intervalo $[-0.5, 0.5]$ ou $[-1, 1]$.
- ◇ O prolongamento da aprendizagem pode levar a problemas de sobreajustamento, ou seja, a rede classifica bem o conjunto de treino mas mal o conjunto de validação.
- ◇ O número de exemplos de treino também é difícil de determinar, mas pode-se seguir uma das seguintes regras práticas:
 - O número de exemplos deve ser 5 a 10 vezes mais que o número de pesos.
 - Treino da rede para classificar com precisão $1 - (e/2)$ exemplos de treino. Para se obter precisão $1 - e$ no conjunto de validação serão necessários tantos exemplos de treino quanto o número de pesos na rede divididos por e .
- ◇ Unidades escondidas a menos resulta na impossibilidade de aprendizagem da função; unidades escondidas a mais pode redundar em sobreajustamento.

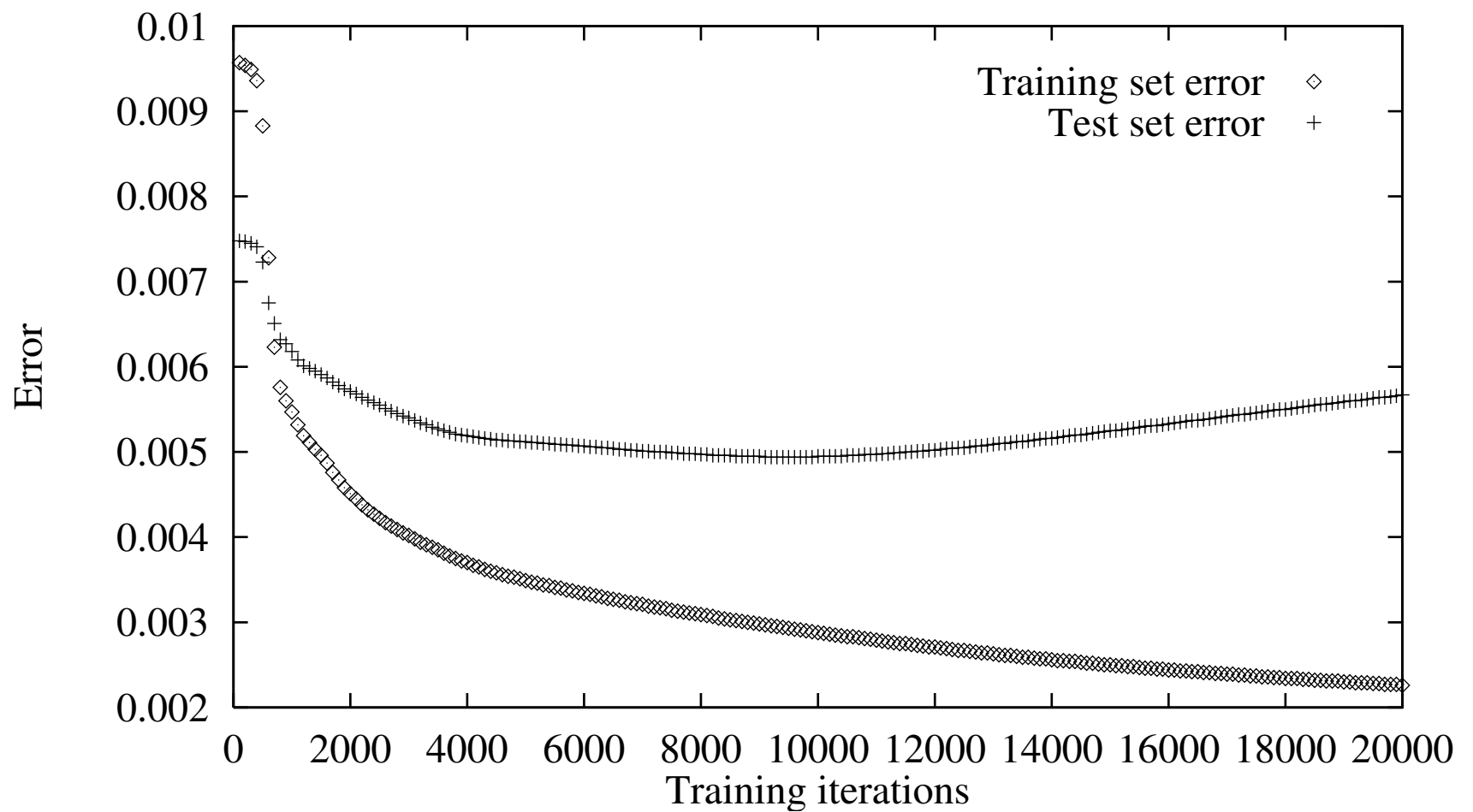
Sobreajustamento



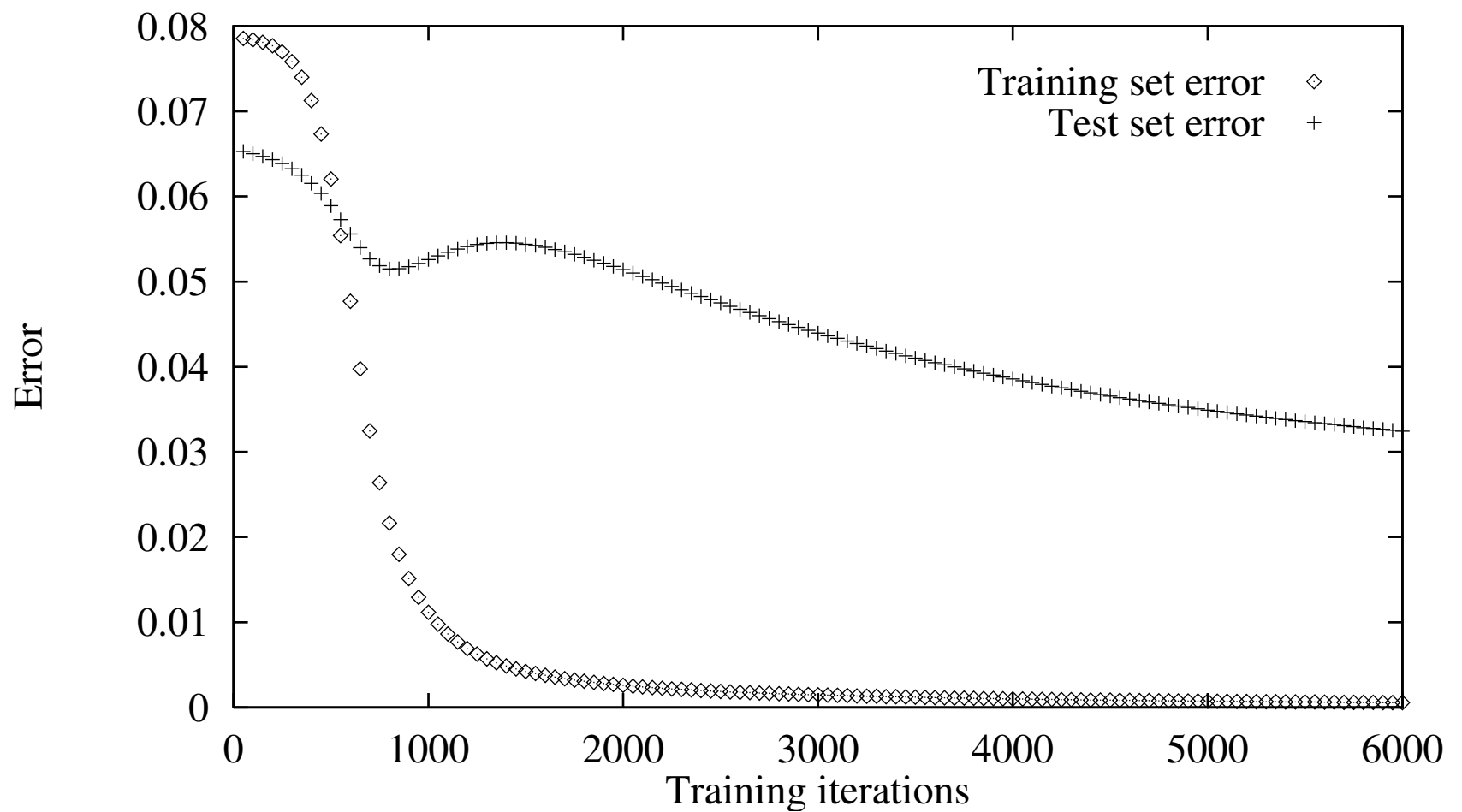
Os exemplos devem ser divididos em conjunto de treino e conjunto de validação. Deve haver um conjunto de teste que não é utilizado na aprendizagem.

Utiliza-se o conjunto de treino no algoritmo de aprendizagem, parando-se quando se minimiza o erro no conjunto de validação.

Sobreajustamento (quando parar?)



Sobreajustamento (quando parar?)



Problemas de convergência

Pode haver grandes oscilações no erro. Uma das formas consiste em alterar a regra para utilizar **momento** - μ .

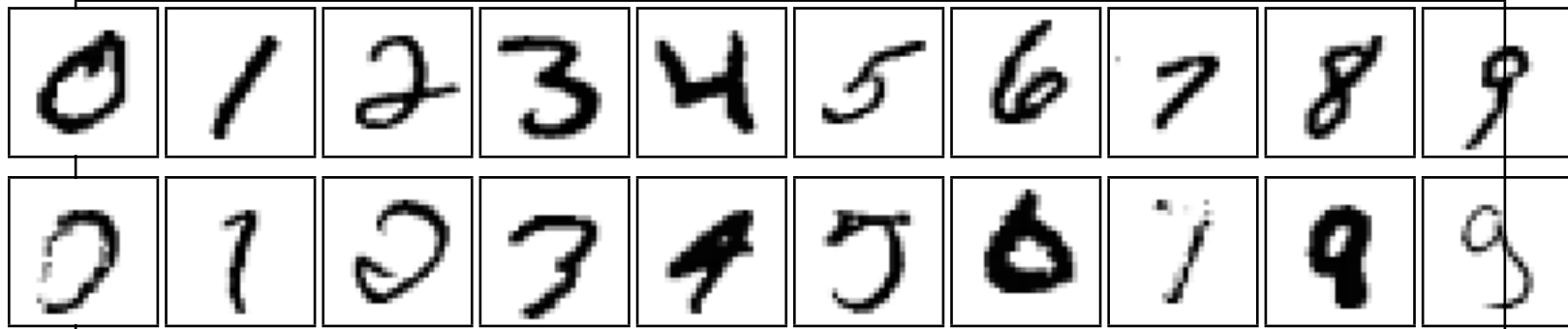
As variações dos pesos da iteração t para $t + 1$ são dadas por:

$$\begin{aligned}\Delta W_{j,i}(t+1) &= \eta \times a_j \times \Delta_i + \mu \Delta W_{j,i}(t) \\ \Delta W_{k,j}(t+1) &= \eta \times a_k \times \Delta_j + \mu \Delta W_{k,j}(t).\end{aligned}$$

Paralelo com bola a descer superfície:

- ◇ **Ritmo de aprendizagem**: velocidade (valores típicos entre 0.1 e 0.9).
- ◇ **Momento**: inércia - mantém direcção do movimento anterior.
- ◇ Acelerar convergência através de treino em lote. As alterações nos pesos de todos os casos de treino são acumuladas e só no final propagadas após o processamento integral do conjunto de treino. Esta versão normalmente converge mais rapidamente, mas pode ficar preso mais facilmente em mínimos locais.

Reconhecimento de escrita



3-nearest-neighbor = 2.4% erro

400–300–10 unit MLP = 1.6% erro

LeNet: 768–192–30–10 unit MLP = 0.9% erro

Melhores (kernel machines, algoritmos de visão) $\approx$ 0.6% erro

Sumário

- ◇ A maioria dos cérebros tem muitos neurónios; cada neurónio $\approx$ unidade de limiar (?)
- ◇ Perceptrões (redes monocamada) são pouco expressivos
- ◇ Redes multicamada são suficientemente expressivas; podem ser treinadas pelo método de descida do gradiente, i.e., retropropagação do erro
- ◇ Sobreajustamento é um problema sério a evitar, devendo-se utilizar técnicas de validação cruzada.
- ◇ Muitas aplicações: fala, condução, reconhecimento escrita, detecção de fraudes, etc.