APRENDER A PARTIR DE EXEMPLOS

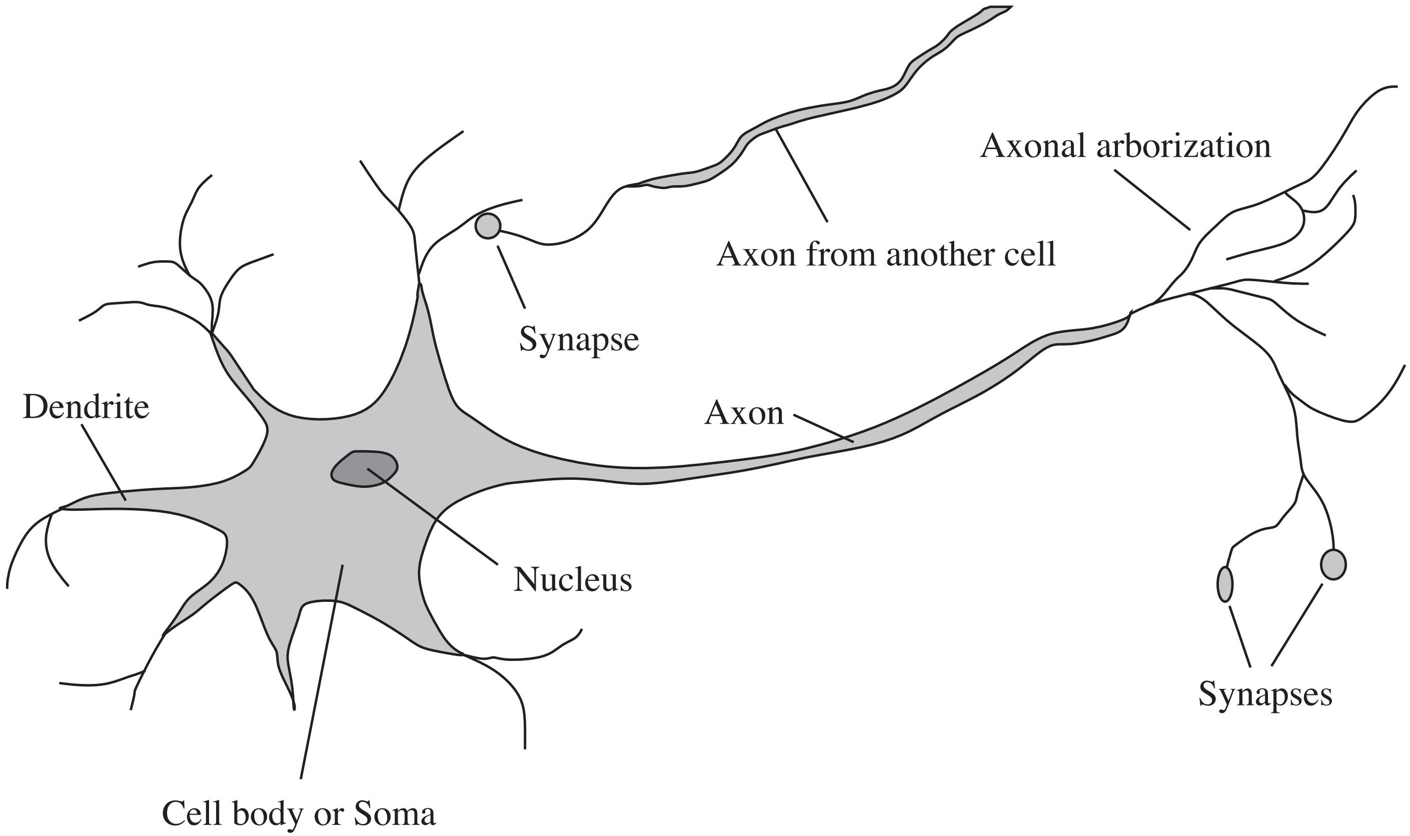
Capítulo 18, secção 7

Resumo

- Cérebro
- Redes Neuronais
- Perceptrões
- Redes neuronais multicamada
- Aplicações de redes neuronais

Cérebro

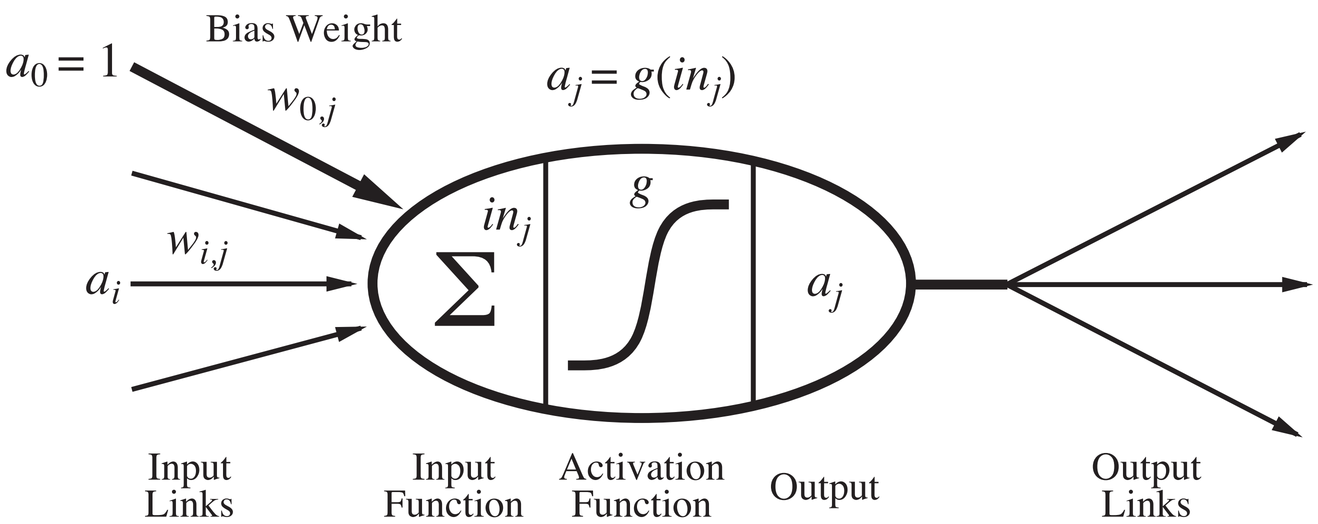
- 10^{11} neurónios de > 20 tipos, 10^{14} sinapes, 1-10ms tempo de ciclo
- Sinais têm ruído “spike trains” de potenciais eléctricos



Unidade de McCulloch-Pitts

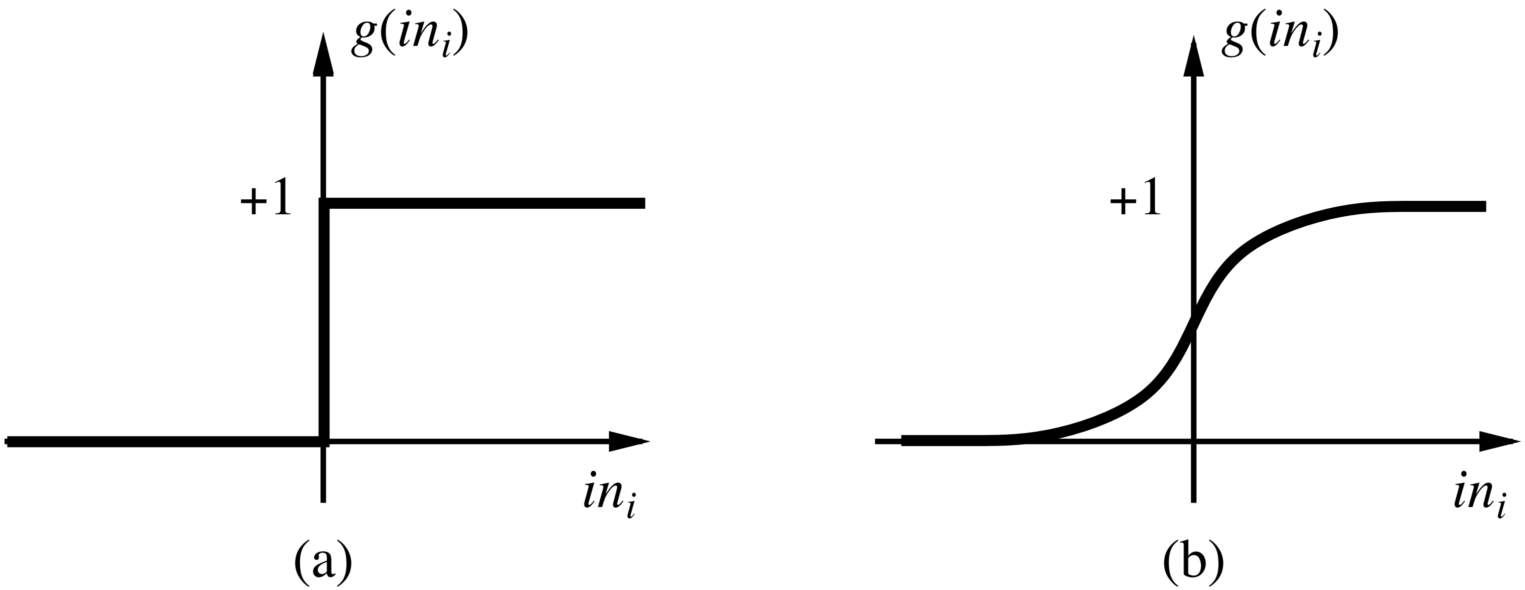
- Saída é uma função linear “esmagada” das entradas:

$$a_j \leftarrow g\left(in_j\right) = g\left(\sum_i w_{i,j} a_i\right)$$



- 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

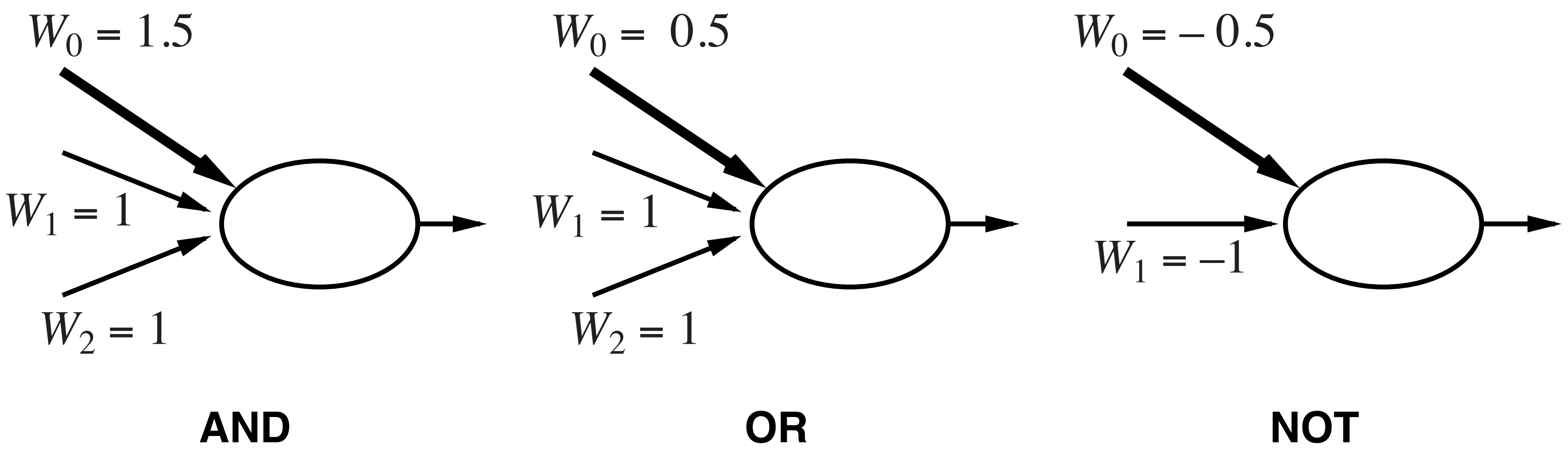


- (a) é a função degrau ou função limiar. Neste caso designamos a unidade por perceptrão.
- (b) é a função sigmóide $1/(1 + e^{-x})$. Unidade designada por perceptrão sigmóide .
- Alteração da polarização (bias weight) $W_{0,i}$ desloca a localização do limiar

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 de funções lógicas

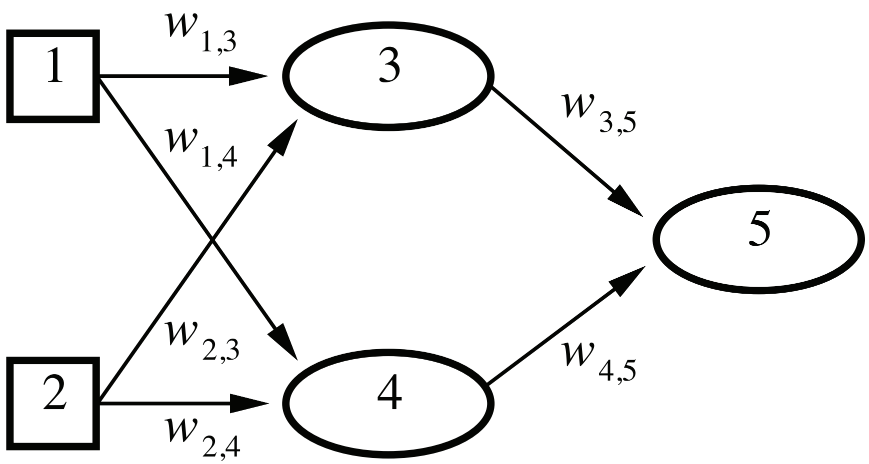


- McCulloch e Pitts: qualquer função Booleana pode ser implementada combinando as construções anteriores.

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
  - Maquinas de Boltzmann utilizam funções de activação estocásticas,
- Nota: As redes recorrentes têm ciclos dirigidos com atrasos (delays)
  - têm estado interno (como fli-flops), podem oscilar etc.

Exemplo de rede alimentada para a frente

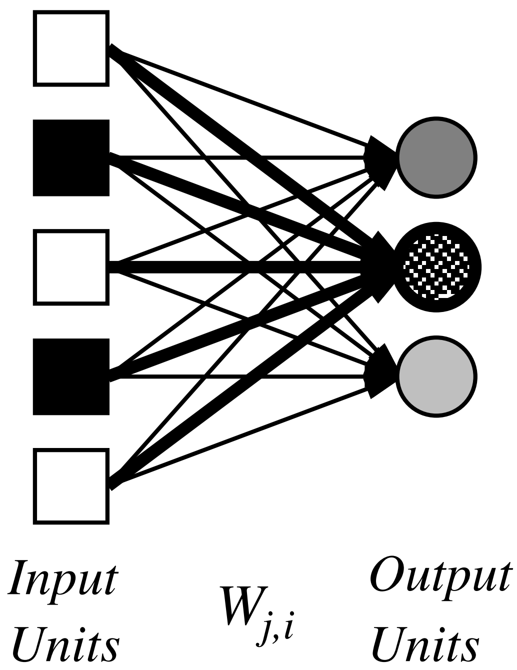


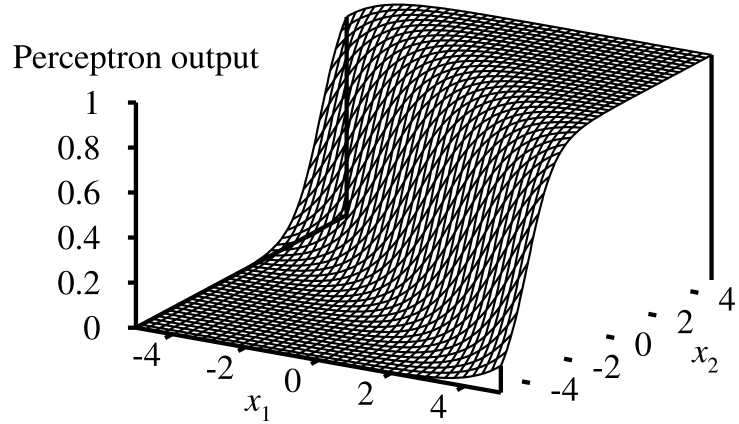
- Rede alimentada para a frente = família parametrizada de funções não-lineares:

$$a_5 = g\left(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4\right)$$
$$= g\left(w_{3,5} \cdot g\left(w_{1,3} \cdot a_1 + w_{2,3} \cdot a_2\right) + w_{4,5} \cdot g\left(w_{1,4} \cdot a_1 + w_{2,4} \cdot a_2\right)\right)$$

- 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

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:
- Erro quadrático para um exemplo com entrada **x** e valor correto y é:

$$Loss(\mathbf{w}) \equiv Err^2 \equiv \left(y - h_{\mathbf{w}}(\mathbf{x})\right)^2$$

- Recorremos ao método do gradiente descendente para minimizar a perda (erro quadrático):

$$\frac{\partial Loss(\mathbf{w})}{\partial w_i} = \frac{\partial Err^2}{\partial w_i} = 2Err \times \frac{\partial Err}{\partial w_i} = 2\left(y - h_{\mathbf{w}}(\mathbf{x})\right) \times \frac{\partial}{\partial w_i}\left(y - h_{\mathbf{w}}(\mathbf{x})\right)$$

$$= 2\left(y - h_{\mathbf{w}}(\mathbf{x})\right) \times \frac{\partial}{\partial w_i}\left(y - \sum_i w_i x_i\right) = -2Err \times x_i$$

- Regras simples para actualização dos pesos

$$w_i \leftarrow w_i + \alpha \times Err \times x_i \qquad \equiv \qquad w_i \leftarrow w_i + \alpha\left(y - h_{\mathbf{w}}(\mathbf{x})\right) \times x_i$$

Neurónios – Regra Delta generalizada

- Aprender por ajustamento dos pesos para o caso de função de activação diferenciável (g). O erro quadrático para um exemplo com entrada **x** e valor correto y é:

$$Loss(\mathbf{w}) \equiv Err^2 \equiv \left(y - g\left(\mathbf{w}\cdot\mathbf{x}\right)\right)^2$$

- Recorrer ao método do gradiente descendente para minimizar Err^2:

$$\frac{\partial Loss(\mathbf{w})}{\partial w_i} = \frac{\partial Err^2}{\partial w_i} = 2Err \times \frac{\partial Err}{\partial w_i} = 2\left(y - g\left(\mathbf{w}\cdot\mathbf{x}\right)\right) \times \frac{\partial}{\partial w_i}\left(y - g\left(\mathbf{w}\cdot\mathbf{x}\right)\right)$$

$$= -2\left(y - g\left(\mathbf{w}\cdot\mathbf{x}\right)\right) \times g'\left(\mathbf{w}\cdot\mathbf{x}\right) \times \frac{\partial}{\partial w_i}\mathbf{w}\cdot\mathbf{x} = -2\left(y - g\left(\mathbf{w}\cdot\mathbf{x}\right)\right) \times g'\left(\mathbf{w}\cdot\mathbf{x}\right) \times x_i$$

- Regra simples para atualização dos pesos

$$w_i \leftarrow w_i + \alpha\left(y - g\left(\mathbf{w}\cdot\mathbf{x}\right)\right) \times g'\left(\mathbf{w}\cdot\mathbf{x}\right) \times x_i$$

Neurónios – Regra Delta generalizada

- Regra simples para atualização dos pesos

$$w_i \leftarrow w_i + \alpha\left(y - g\left(\mathbf{w}\cdot\mathbf{x}\right)\right)\times g'\left(\mathbf{w}\cdot\mathbf{x}\right)\times x_i$$

- No caso da função sigmóide

$$g\left(\mathbf{w}\cdot\mathbf{x}\right) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$

a regra para atualização dos pesos é:

$$w_i \leftarrow w_i + \alpha\left(y - g\left(\mathbf{w}\cdot\mathbf{x}\right)\right)\times g\left(\mathbf{w}\cdot\mathbf{x}\right)\left(1 - g\left(\mathbf{w}\cdot\mathbf{x}\right)\right)\times x_i$$

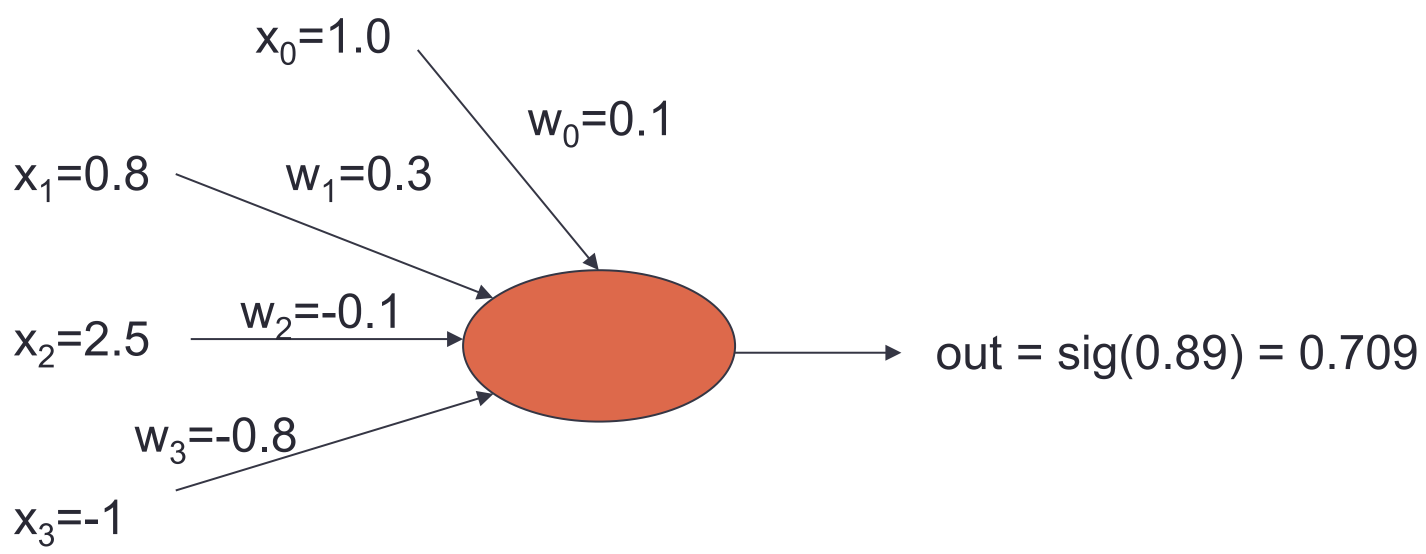
Algoritmo de Aprendizagem

function PERCEPTRON-LEARNING(*network*, *examples*, α) **returns** a perceptron hypothesis

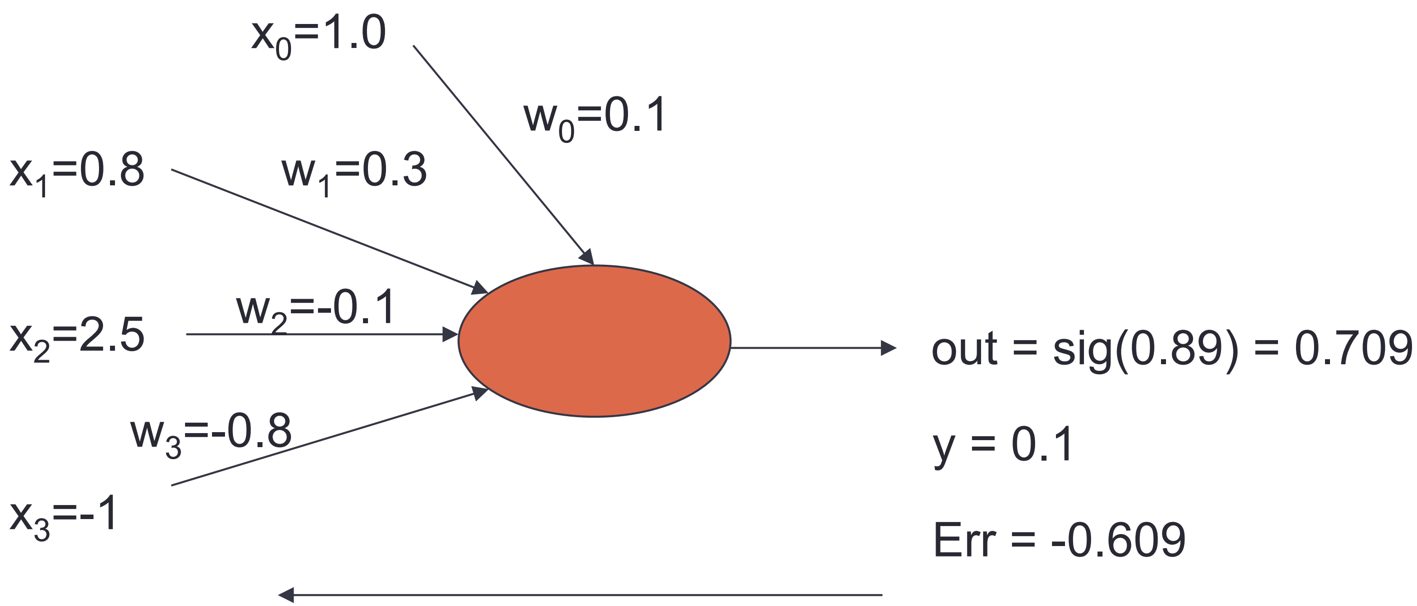
inputs: *network*, um perceptrão com pesos w_i, $i = 0, \ldots, n$ e função de activação g
examples, um conjunto de exemplos, com entrada $\mathbf{x} = x_1, \ldots, x_n$ e saída y
α, o ritmo de aprendizagem

repeat
  for each e **in** *examples* **do**
    /* *Calcular o valor de saída para este exemplo* */
      $in \leftarrow \sum_{i=0}^{n} \; w_i \; x_i[e]$
      $out \leftarrow g(in)$
    /* *Calcular o erro* */
      $Err \leftarrow y[e] \; - \; out$
    /* *Actualizar os pesos das entradas* */
      for each entrada i do perceptrão **do**
        $w_i \leftarrow w_i \; + \; \alpha \; \times \; Err \; \times \; g'(in) \; \times \; x_i[e]$
    /* *Com sigmoide:* $g'(in) = out \; \times \; (1 - out)$; *Com limiar:* $g'(in) = 1$ */
      end
  end
until se tenha atingindo um critério de paragem

Exemplo de aplicação



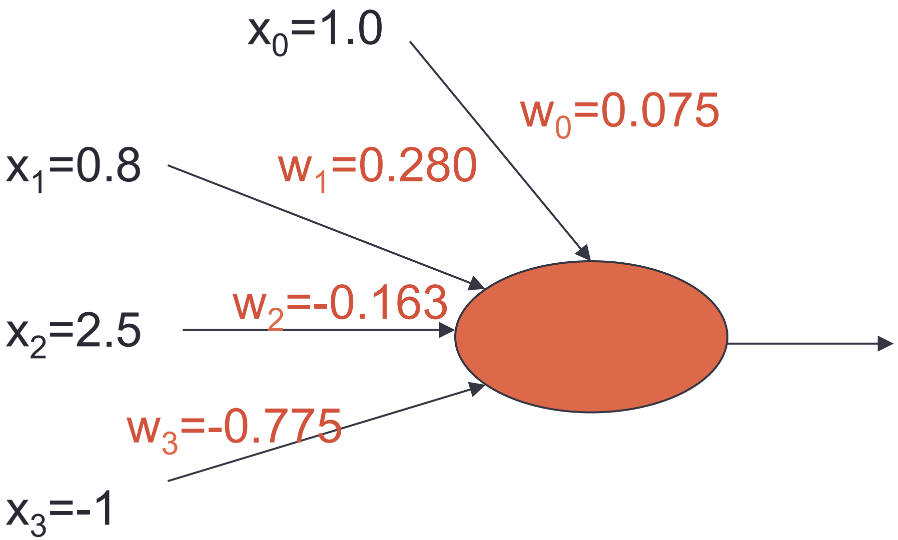
Exemplo de aplicação



$\Delta = out \cdot (1 - out) \cdot Err = 0.709 \cdot (1 - 0.709) \cdot -0.609 = -0.126$

Peso Inicial	Entrada	Actualização (ΔW_i)	Peso Final
$W_0 = 0.1$	$x_0 = 1.0$	$\alpha \cdot \Delta \cdot x_0 = 0.2 \cdot -0.126 \cdot 1.0 = -0.025$	0.075
$W_1 = 0.3$	$x_1 = 0.8$	$\alpha \cdot \Delta \cdot x_1 = 0.2 \cdot -0.126 \cdot 0.8 = -0.020$	0.280
$W_2 = -0.1$	$x_2 = 2.5$	$\alpha \cdot \Delta \cdot x_2 = 0.2 \cdot -0.126 \cdot 2.5 = -0.063$	-0.163
$W_3 = -0.8$	$x_3 = -1.0$	$\alpha \cdot \Delta \cdot x_3 = 0.2 \cdot -0.126 \cdot -1.0 = 0.025$	-0.775

Exemplo de aplicação



out = sig(0.667) = 0.661

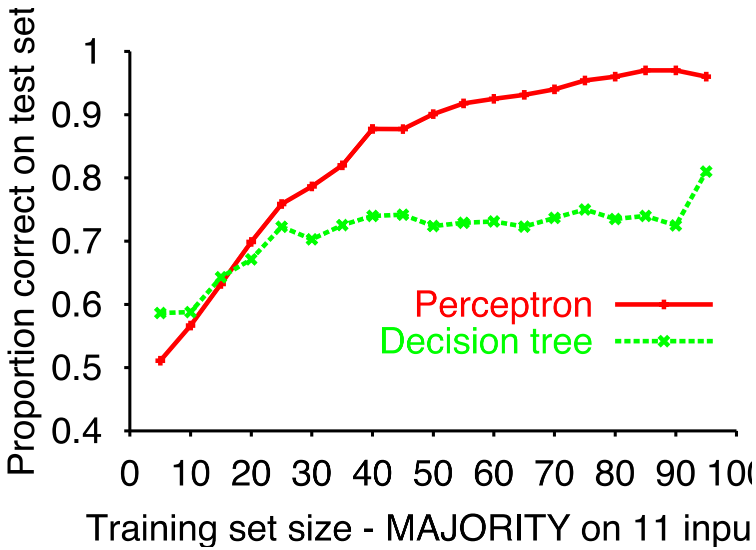
y = 0.1

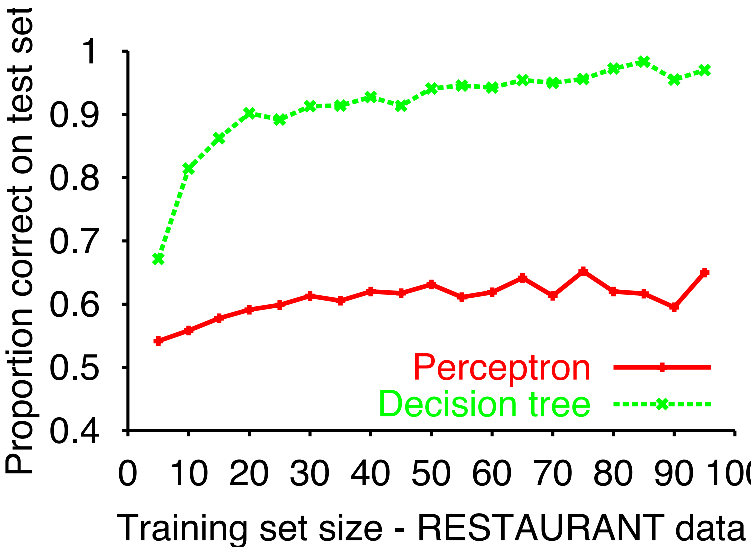
Err = -0.561

- O erro decresceu de -0.609 para -0.561

Aprendizagem do perceptrão

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



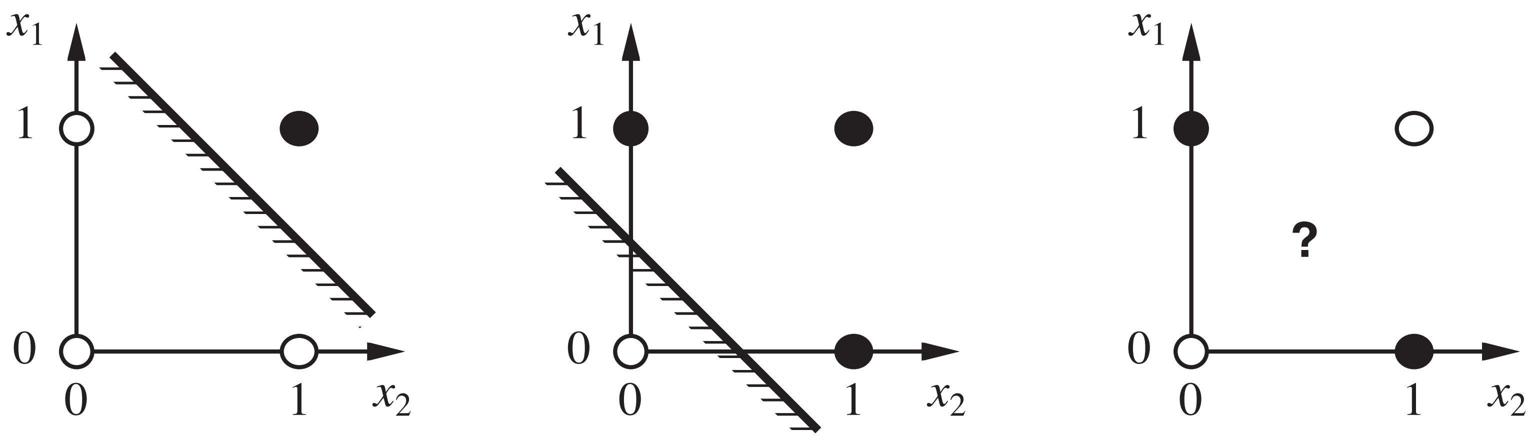


- Perceptrão aprende função de maioria facilmente, indução de árvore de decisão é inútil
- Indução de árvore de decisão aprende função do restaurante, perceptrão não pode representá-la.

Expressividade dos perceptrões

- Considere-se o perceptrão que usa a função de activação g = degrau (Rosenblatt, 1957, 1960)
- Pode representar AND, OR, NOT, maioria, etc., mas não o XOR
- Representa um separador linear no espaço de entradas:

$$\sum_j w_j x_j > 0 \qquad \text{ou} \qquad \mathbf{w} \cdot \mathbf{x} > 0$$

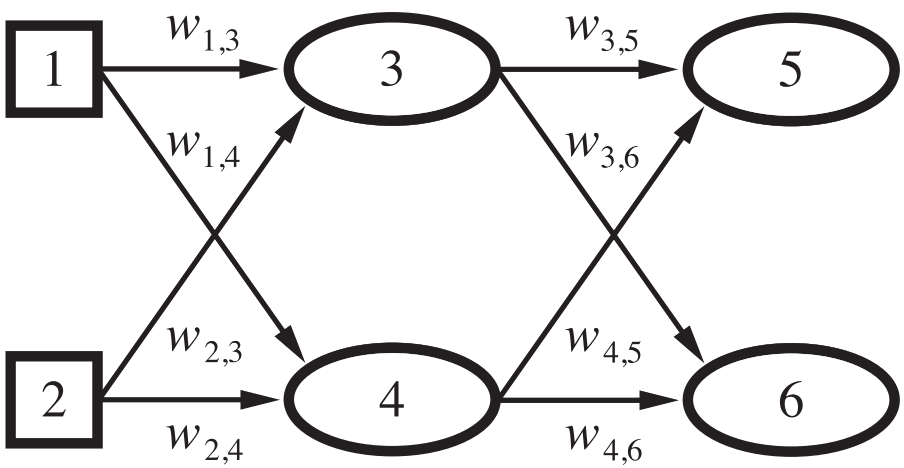


(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

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

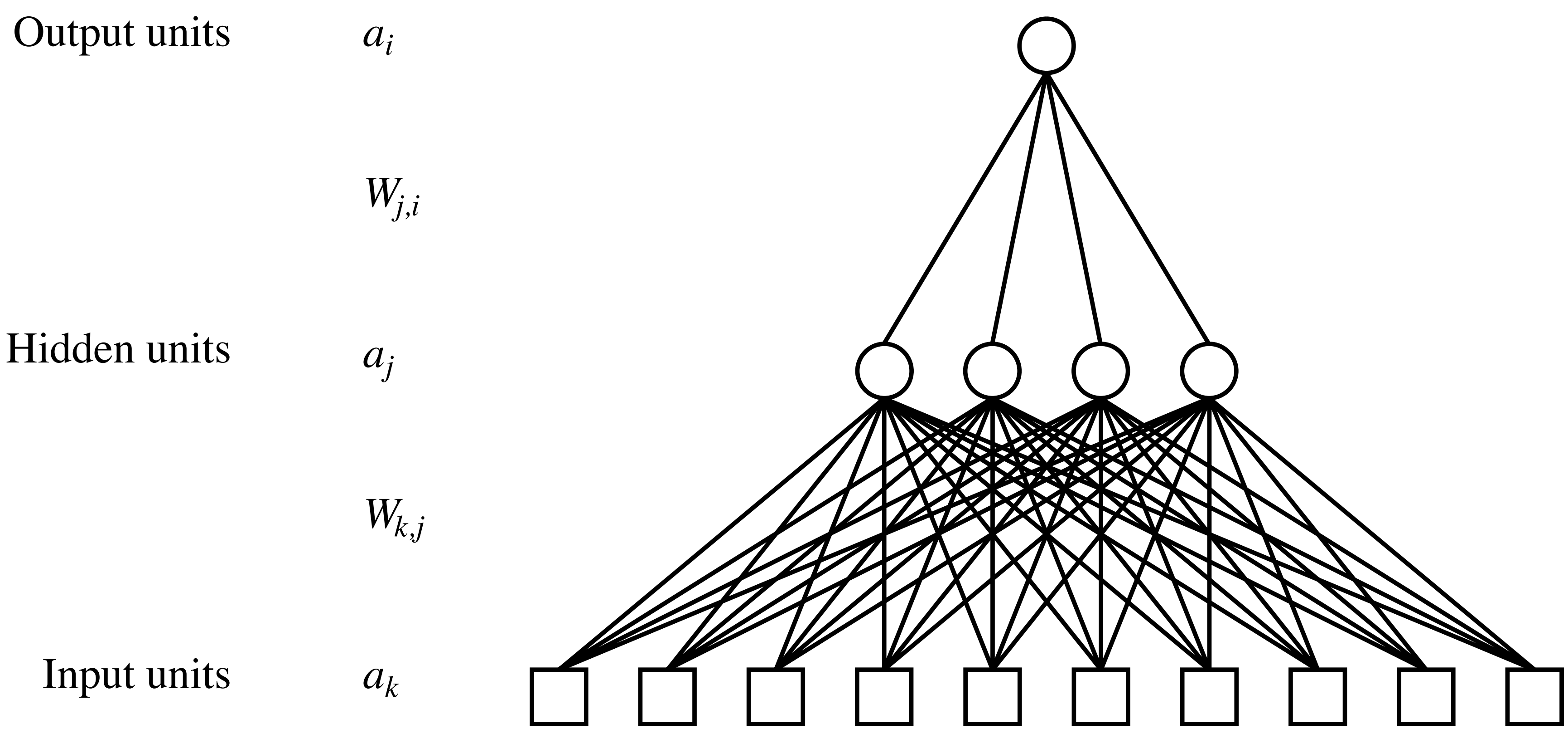
Redes multicamada

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



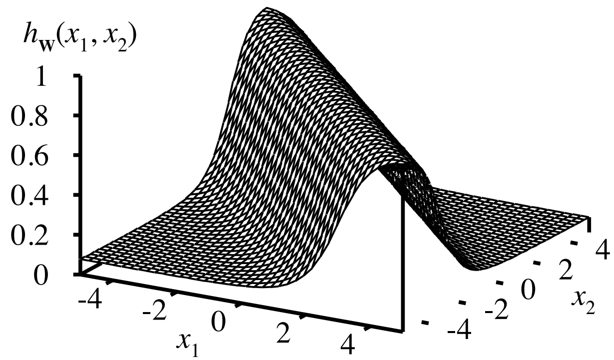
Redes multicamada

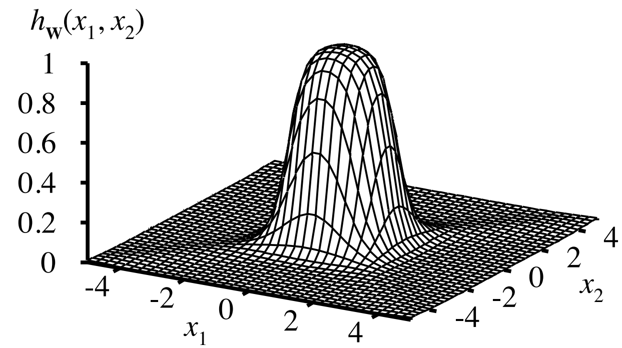
- Normalmente as camadas são totalmente ligadas;
- número de unidades escondidas tipicamente escolhido manualmente



Expressividade de redes multi-camada

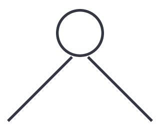
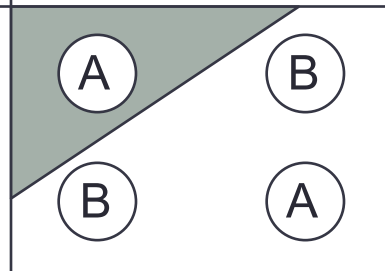
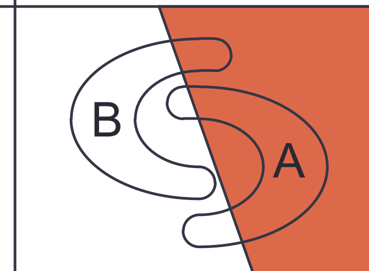
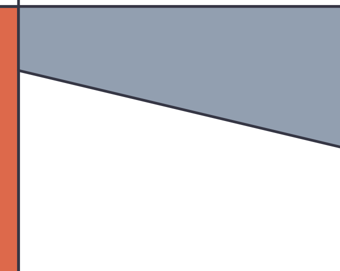
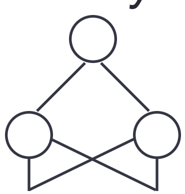
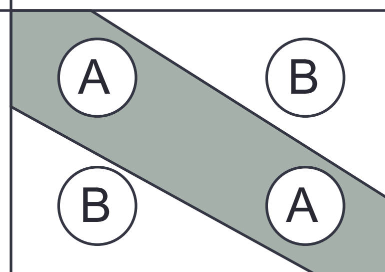
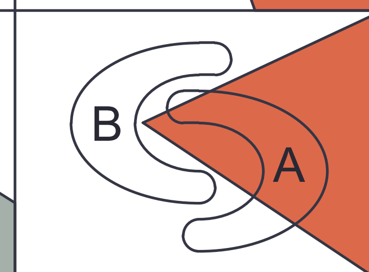
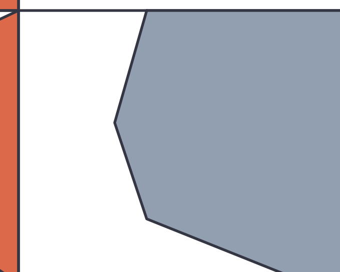
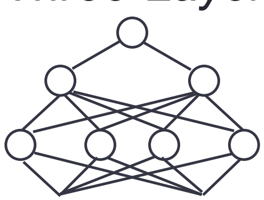
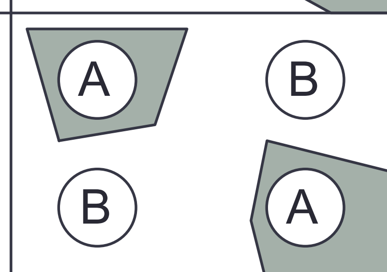
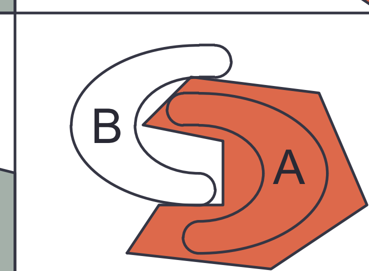
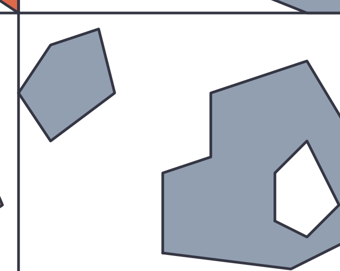
- Todas as funções contínuas podem ser representadas com 1 camada escondida
- Todas as funções podem ser representadas com 2 camadas escondidas





- 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 numero exponencial de unidades escondidas.

Expressividade de redes neuronais

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Aprendizagem por retropropagação

- Numa rede neuronal multi-camada, cada elemento da função vector $\mathbf{h}_w$ de saída depende de todos os pesos w.
- Assumindo uma função de erro aditiva, podemos decompor a contribuição de cada elemento do output para o erro.

$$\frac{\partial}{\partial w} Loss(\mathbf{w}) = \frac{\partial}{\partial w}\left(y - \mathbf{h}_{\mathbf{w}}(\mathbf{x})\right)^2 = \frac{\partial}{\partial w}\sum_k \left(y_k - a_k\right)^2 = \sum_k \frac{\partial}{\partial w}\left(y_k - a_k\right)^2$$

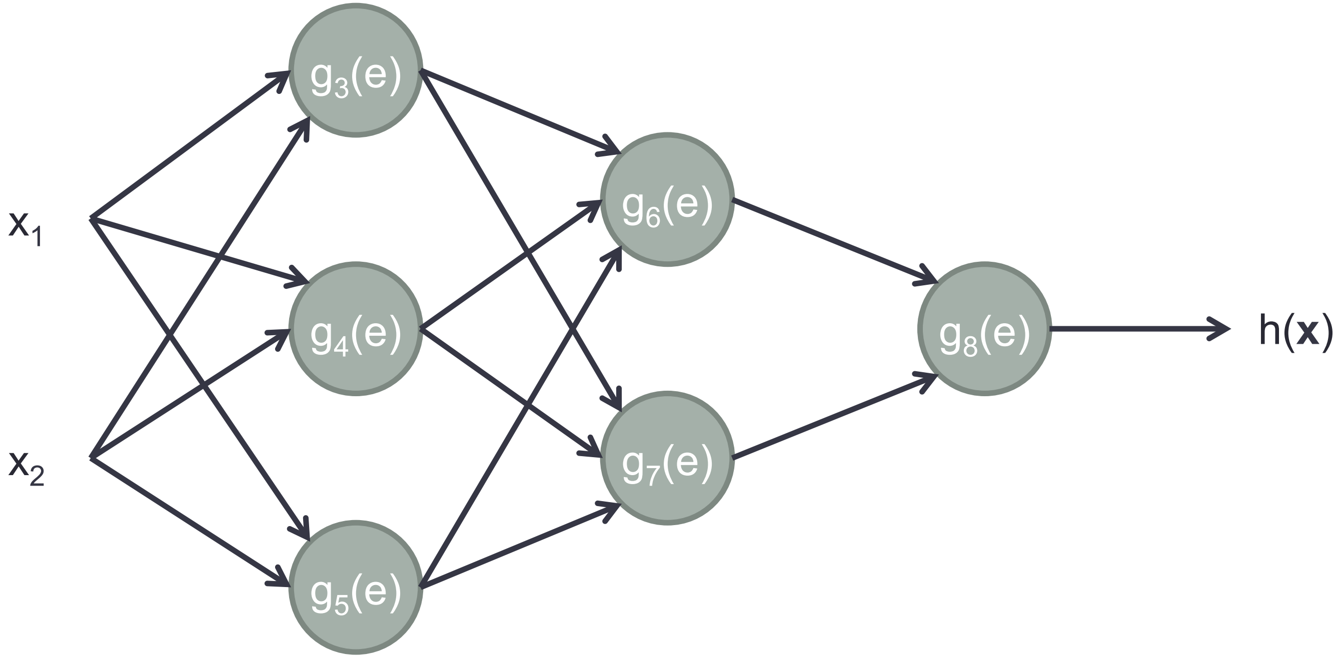
  - Onde k varia pelos nós de saída.
- A complicação maior reside na adição de camadas escondidas. Como determinar o erro se não sabemos qual o valor correto?
- A resposta está na retro-propagação do erro da saída para as camadas intermédias.

Aprendizagem por retropropagação

- Camada de saída: idêntico ao perceptrão monocamada
  - Erro: $\Delta_k = \left(y_k - a_k\right) \times g'(in_k)$
  - Regra de actualização: $w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k$

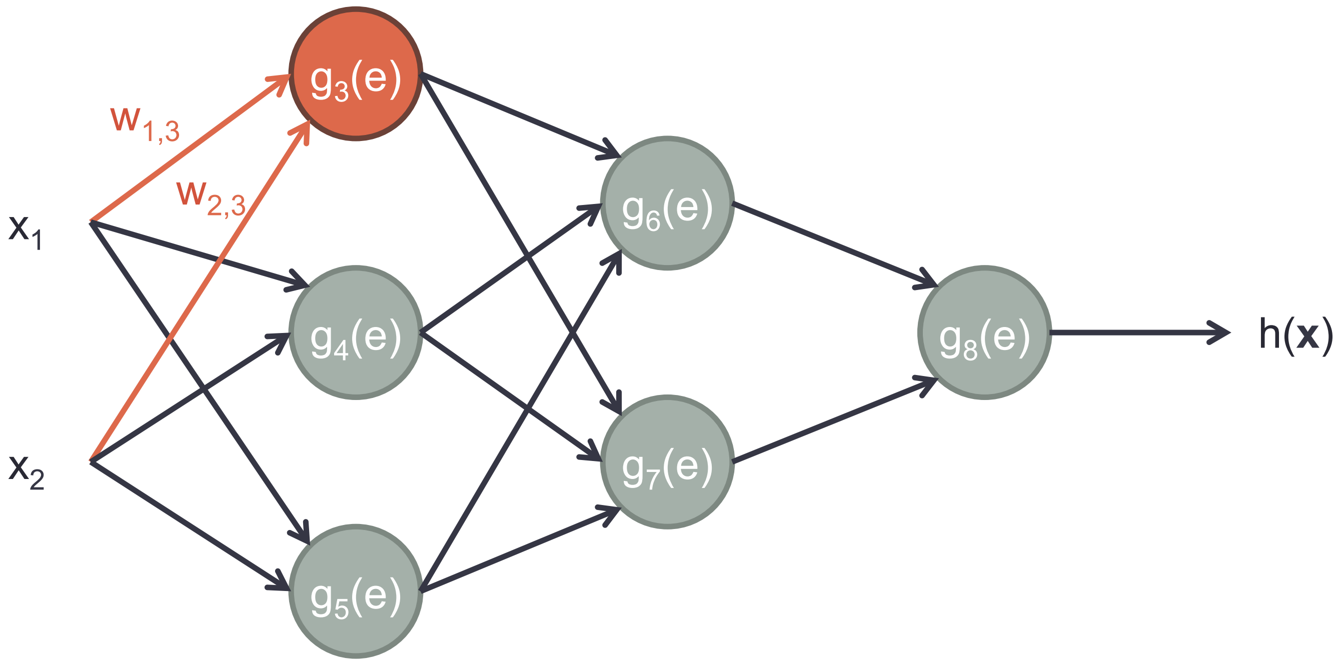
- Camada escondida: o erro é retropropagado para as camadas escondidas
  - Erro: $\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k$
  - Regra de actualização: $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta_j$

Exemplo de aplicação



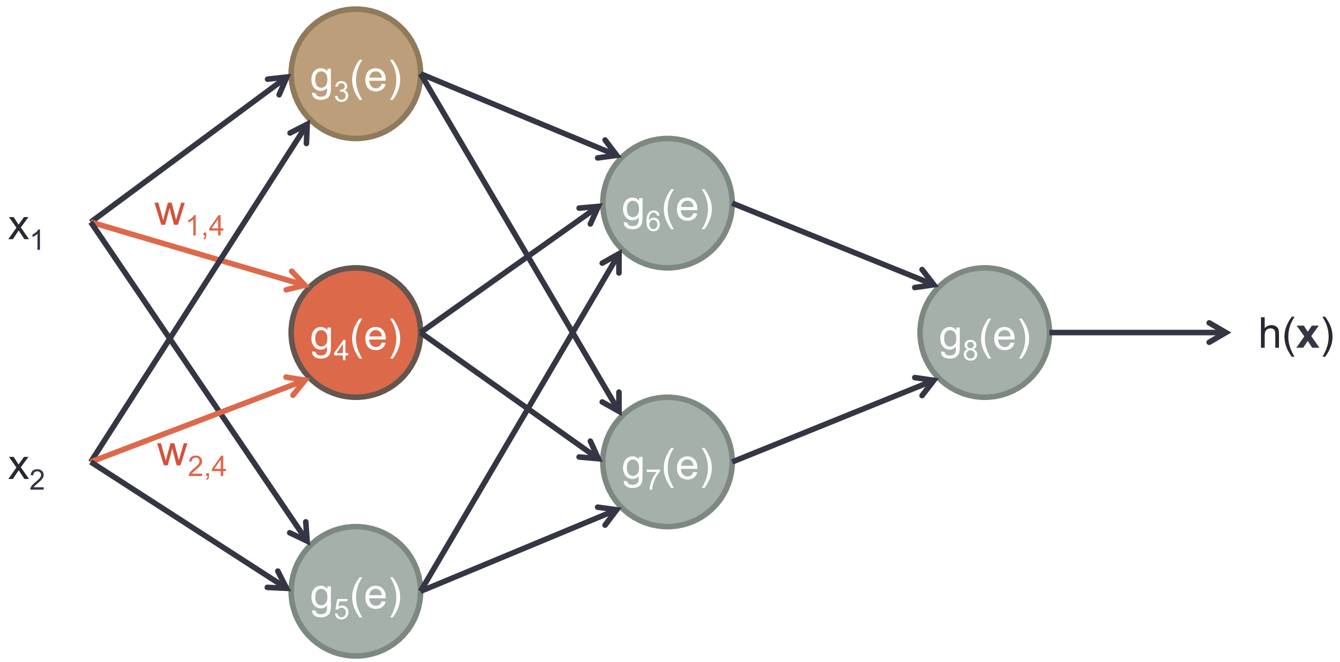
Exemplo de aplicação

$$a_3 = g_3\left(w_{1,3} \cdot x_1 + w_{2,3} \cdot x_2\right)$$



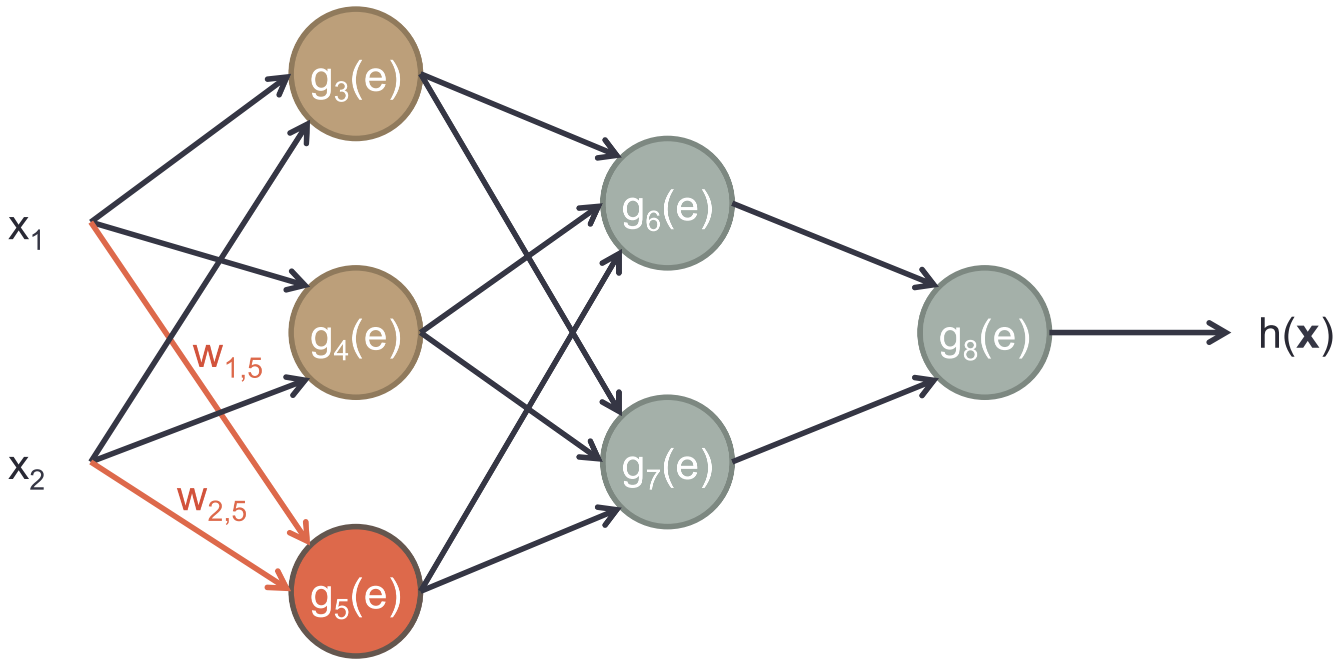
Exemplo de aplicação

$$a_4 = g_4\left(w_{1,4} \cdot x_1 + w_{2,4} \cdot x_2\right)$$



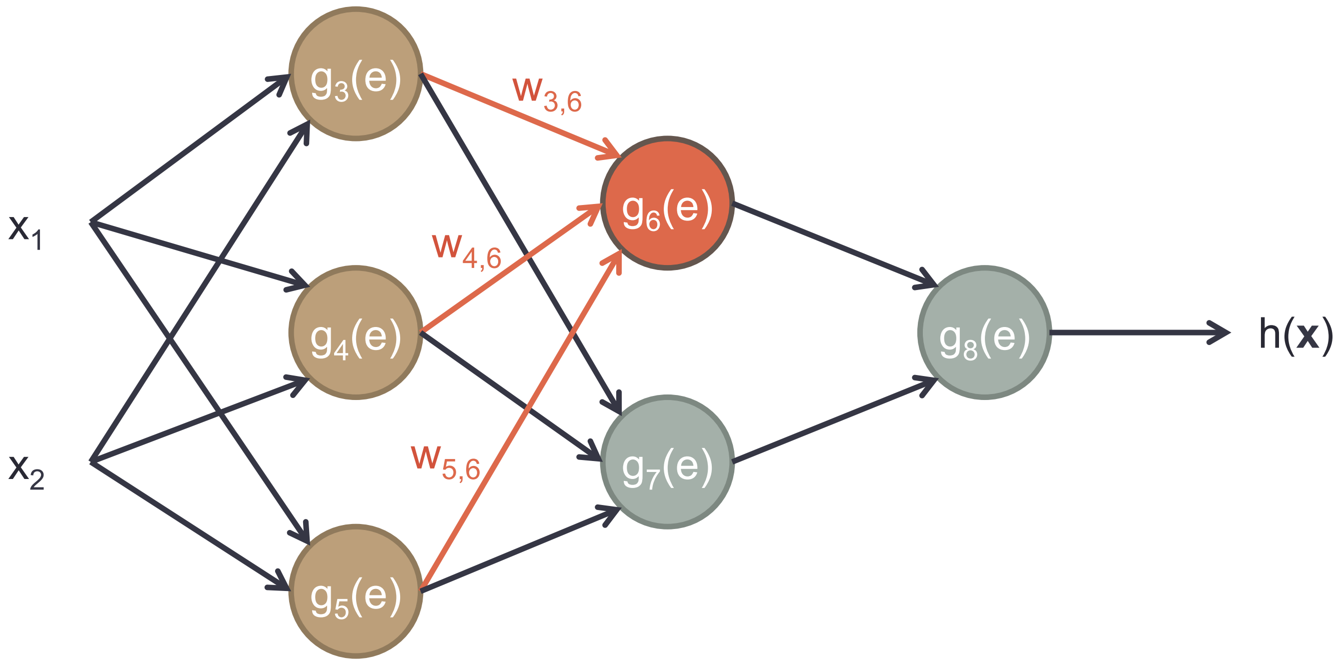
Exemplo de aplicação

$$a_5 = g_5\left(w_{1,5} \cdot x_1 + w_{2,5} \cdot x_2\right)$$



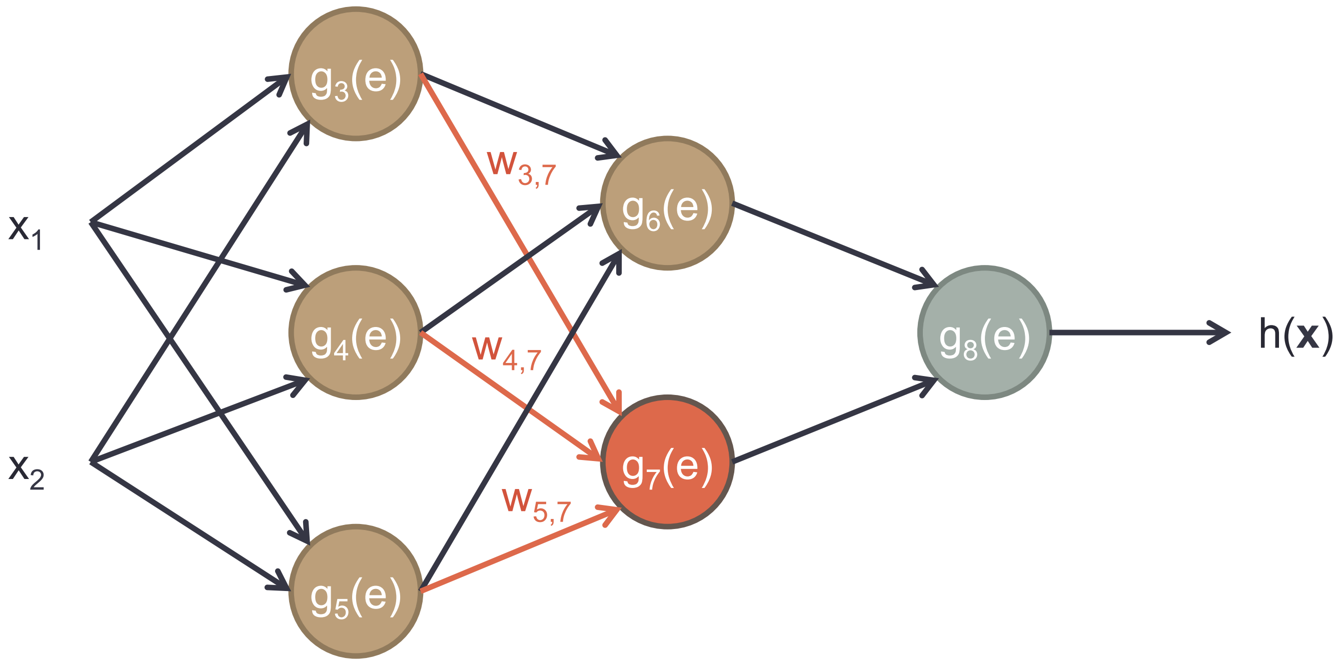
Exemplo de aplicação

$$a_6 = g_6\left(w_{3,6} \cdot a_3 + w_{4,6} \cdot a_4 + w_{5,6} \cdot a_5\right)$$



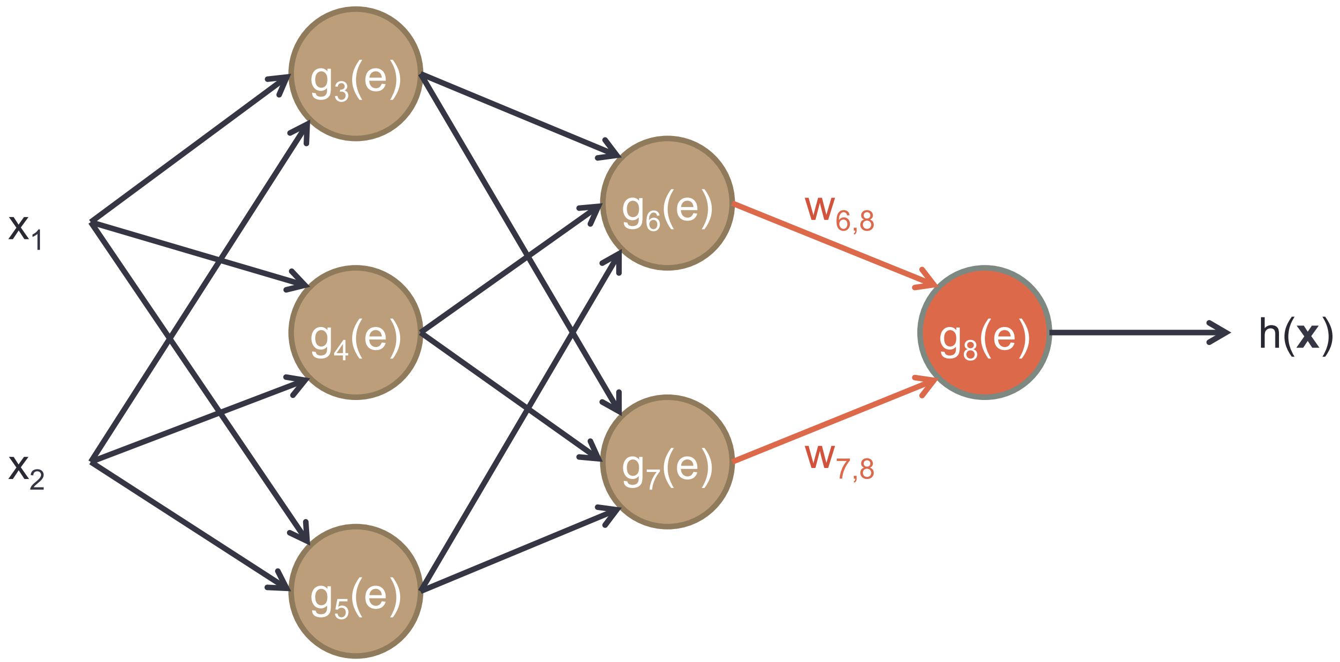
Exemplo de aplicação

$$a_7 = g_7\left(w_{3,7} \cdot a_3 + w_{4,7} \cdot a_4 + w_{5,7} \cdot a_5\right)$$



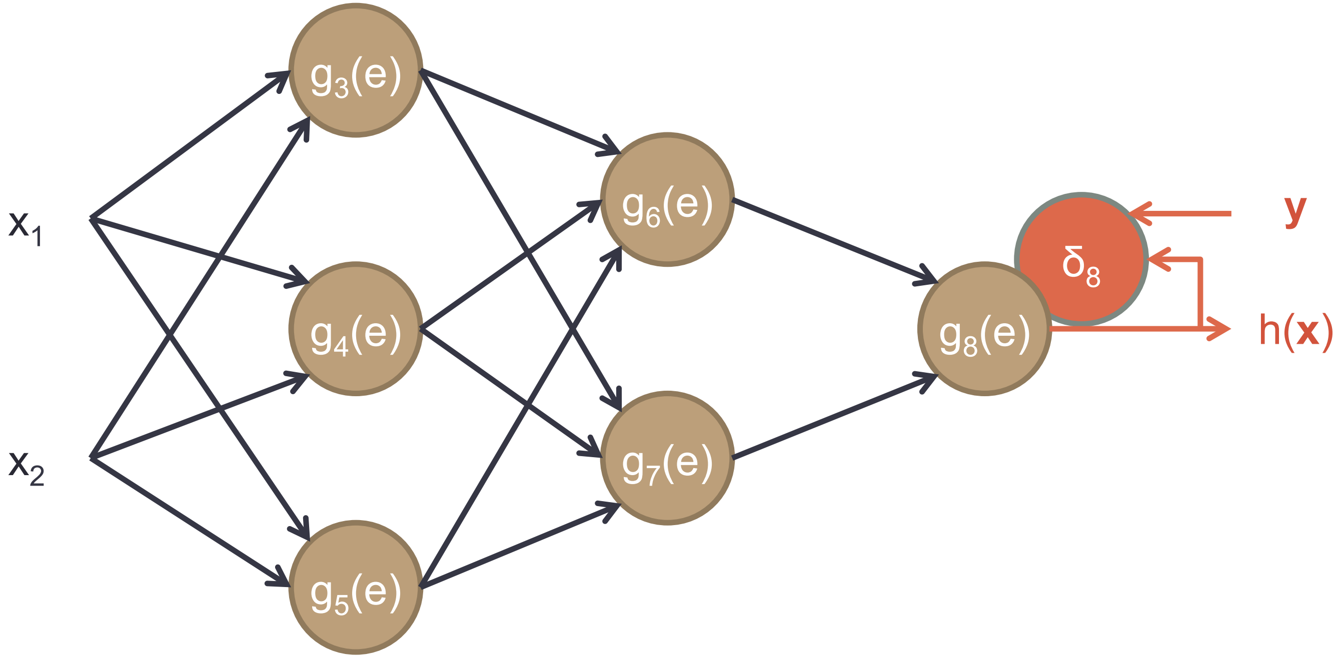
Exemplo de aplicação

$$a_8 = h(\mathbf{x}) = g_8\left(w_{6,8} \cdot a_6 + w_{7,8} \cdot a_8\right)$$



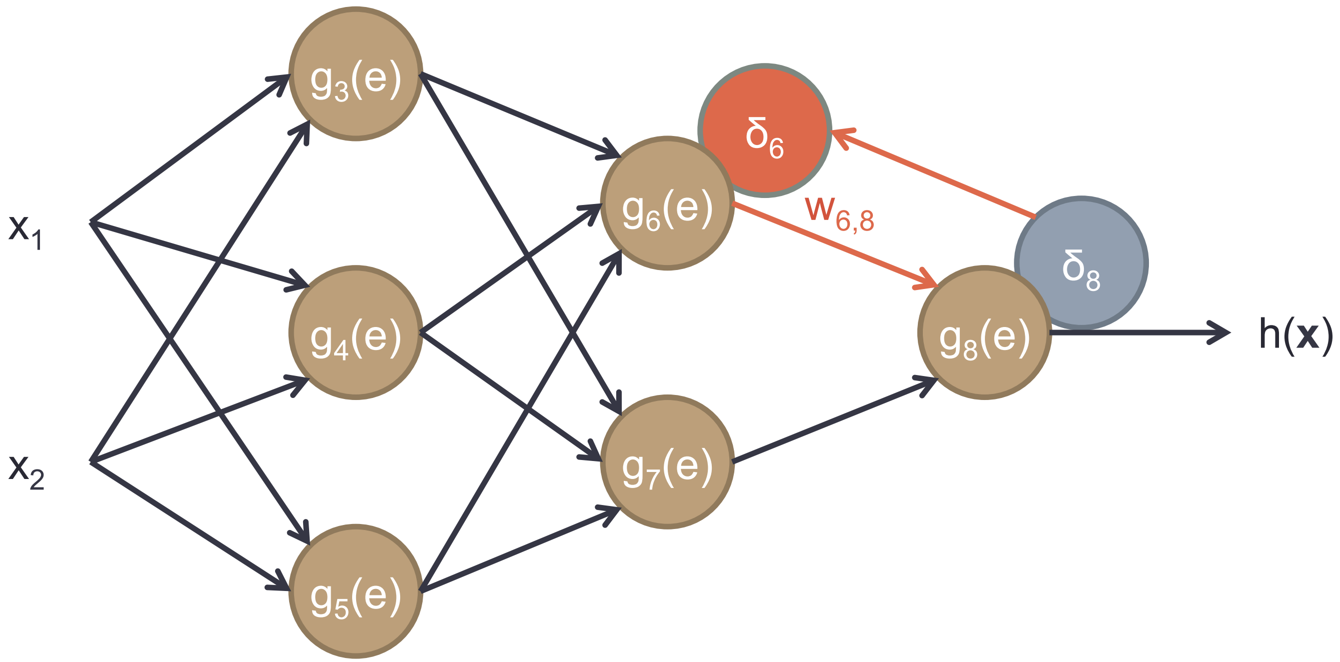
Exemplo de aplicação

$$\delta_8 = \left(y - h(\mathbf{x})\right) \times g_8'(w_{6,8} \cdot a_6 + w_{7,8} \cdot a_8)$$
$$= \left(y - h(\mathbf{x})\right) \times g_8'(in_8)$$



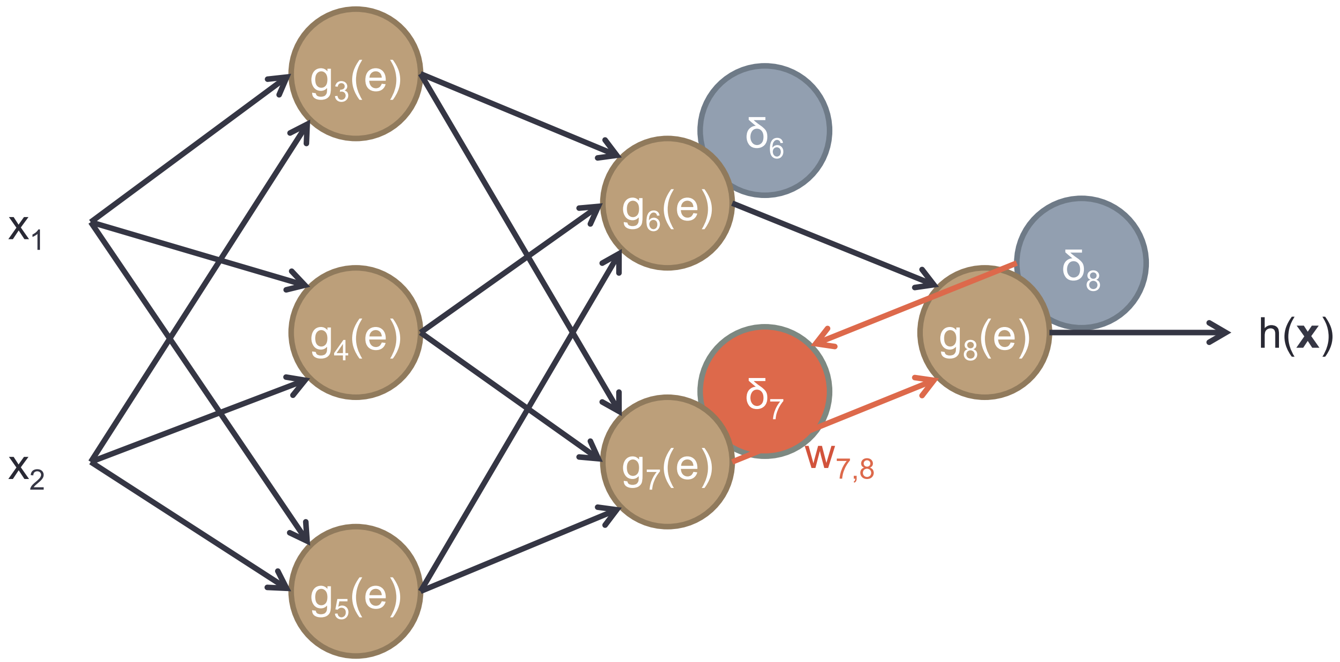
Exemplo de aplicação

$$\delta_6 = w_{6,8}\delta_8 \times g_6'(in_6)$$



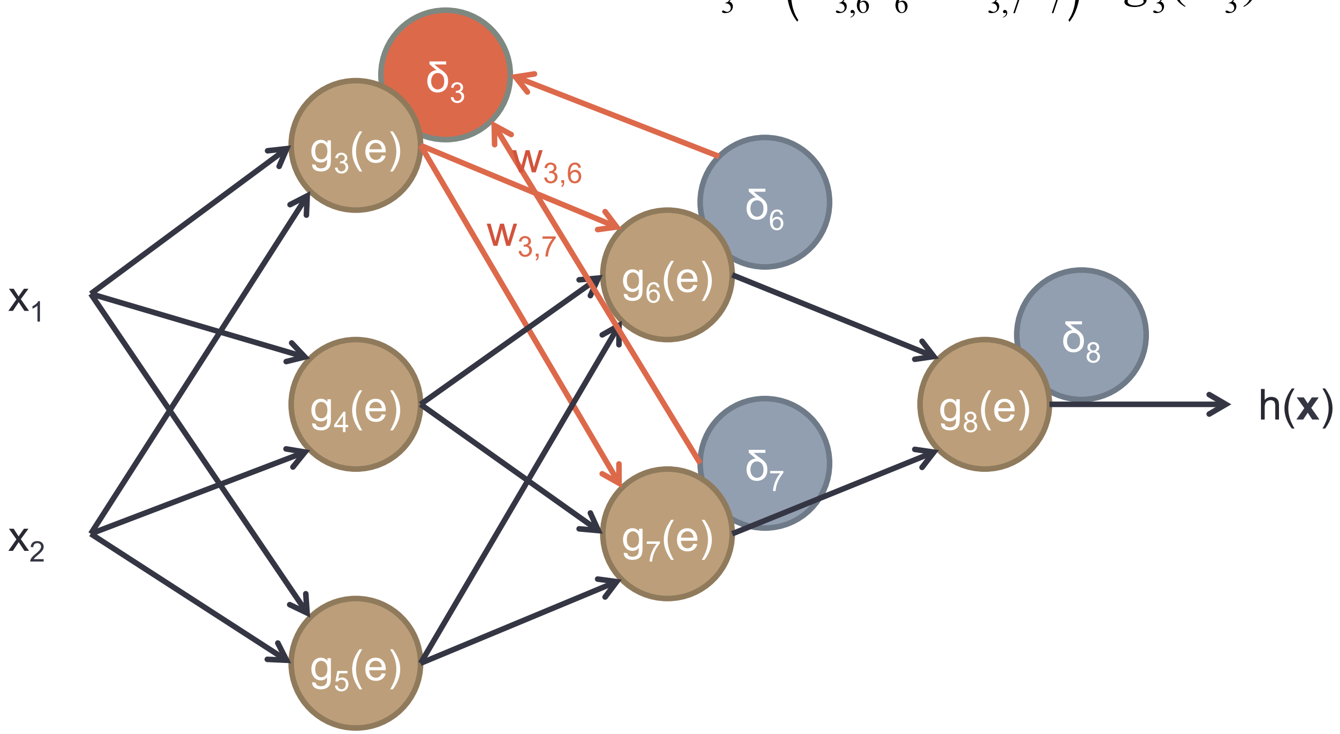
Exemplo de aplicação

$$\delta_7 = w_{7,8}\delta_8 \times g_7'(in_7)$$



Exemplo de aplicação

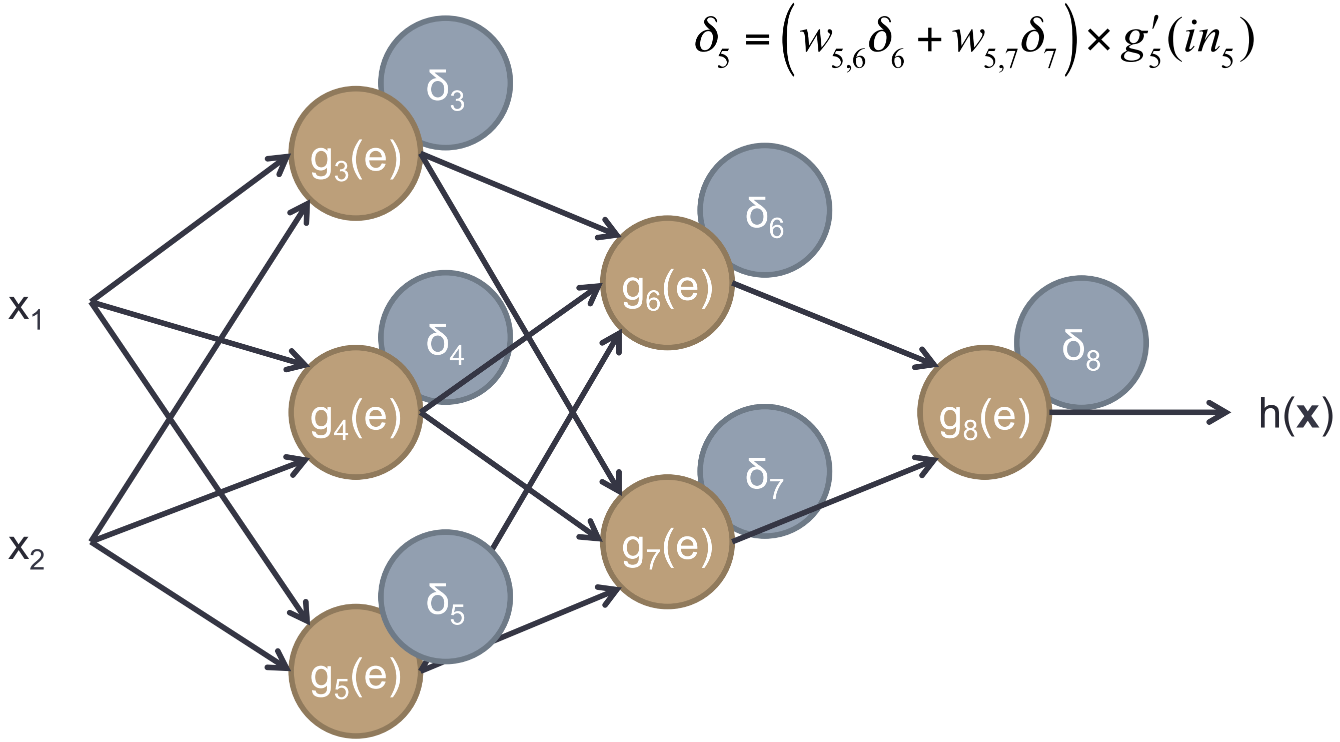
$$\delta_3 = \left(w_{3,6}\delta_6 + w_{3,7}\delta_7\right) \times g_3'(in_3)$$



Exemplo de aplicação

$$\delta_4 = \left(w_{4,6}\delta_6 + w_{4,7}\delta_7\right) \times g_4'(in_4)$$

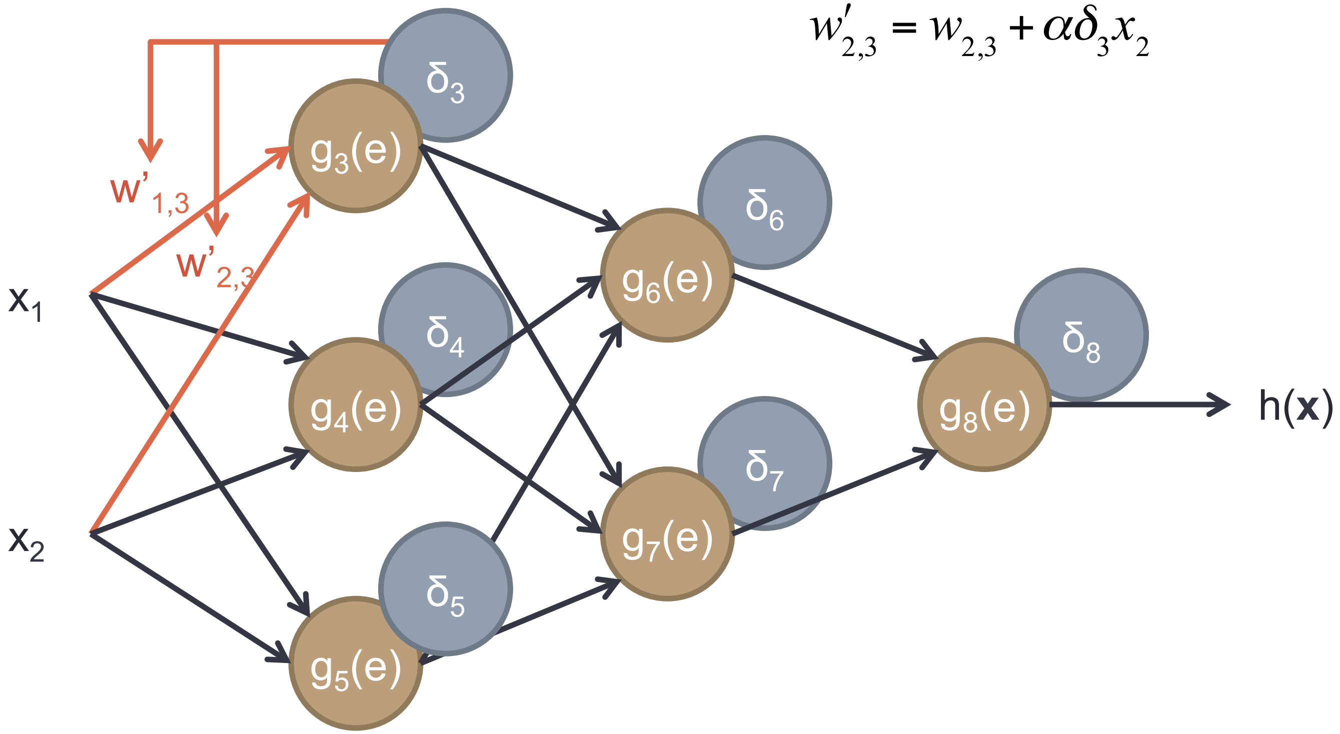
$$\delta_5 = \left(w_{5,6}\delta_6 + w_{5,7}\delta_7\right) \times g_5'(in_5)$$



Exemplo de aplicação

$$w'_{1,3} = w_{1,3} + \alpha\delta_3 x_1$$

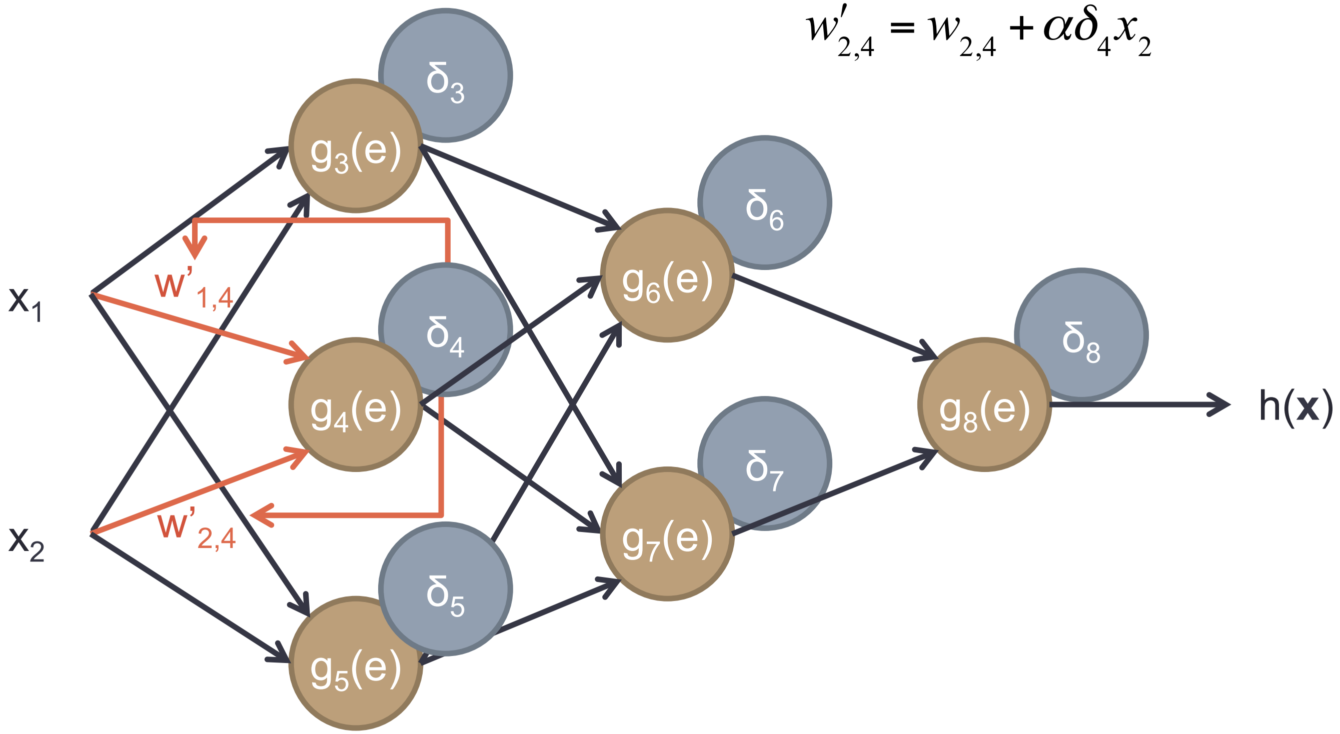
$$w'_{2,3} = w_{2,3} + \alpha\delta_3 x_2$$



Exemplo de aplicação

$$w'_{1,4} = w_{1,4} + \alpha \delta_4 x_1$$

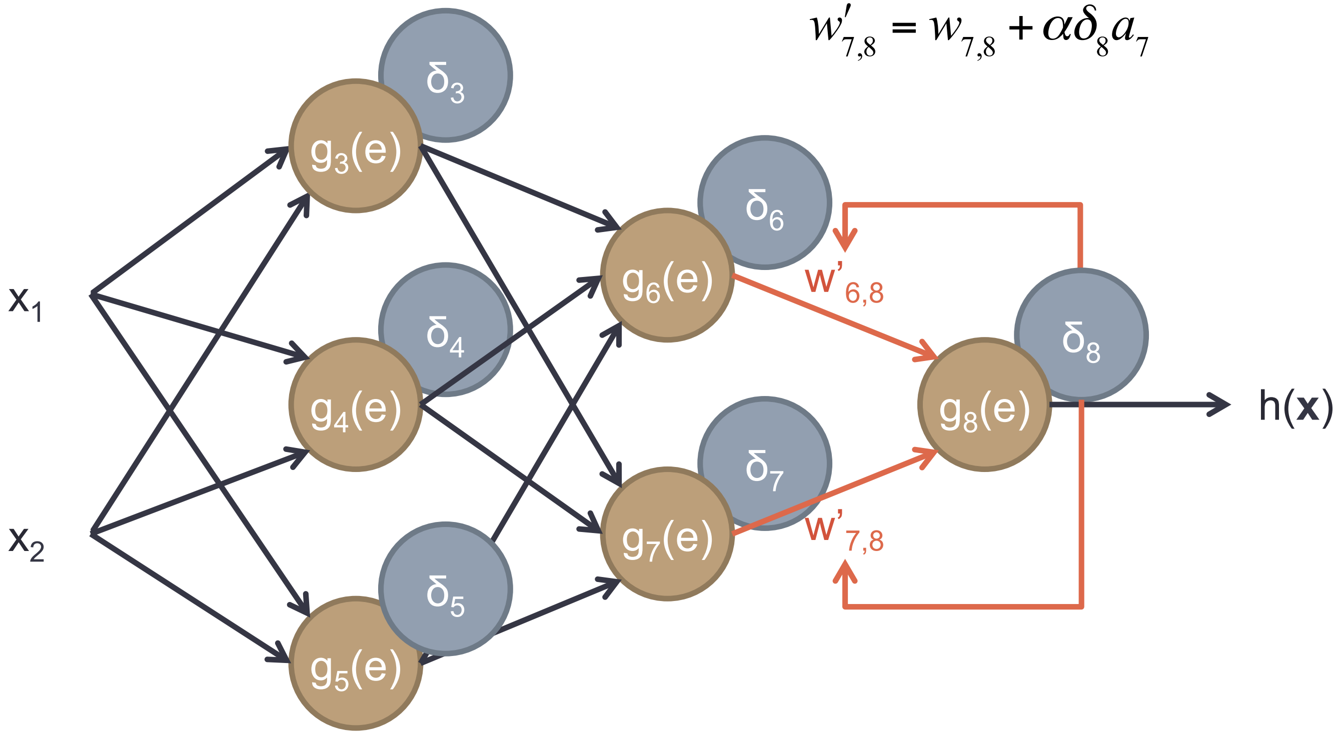
$$w'_{2,4} = w_{2,4} + \alpha \delta_4 x_2$$



Exemplo de aplicação

$$w'_{6,8} = w_{6,8} + \alpha\delta_8 a_6$$

$$w'_{7,8} = w_{7,8} + \alpha\delta_8 a_7$$



Algoritmo de retropropagação

```
function BACK-PROP-UPDATE(network, examples, α) returns a network
with modified weights
   inputs: network, a multilayer network
           examples, a set of input/output pairs
           α, the learning rate

   repeat
      for each e in examples do
         /* Compute the output for this example */
            O ← RUN-NETWORK(network, I^e)
         /* Compute the error and Δ for units in the output layer */
            Err^e ← T^e − O
         /* Update the weights leading to the output layer */
            W_{j,i} ← W_{j,i} + α × a_j × Err_i^e × g'(in_i)
         for each subsequent layer in network do
            /* Compute the error at each node */
               Δ_j ← g'(in_j) Σ_i W_{j,i} Δ_i
            /* Update the weights leading into the layer */
               W_{k,j} ← W_{k,j} + α × I_k × Δ_j
         end
      end
   until network has converged
   return network
```

Derivação da regra de retropropagação (camada de saída)

- Temos que calcular o gradiente para $Loss_k = \sum_k \left(y_k - a_k\right)^2$ na saída k. O gradiente deste erro será zero excepto para os pesos $w_{j,k}$ que ligam à unidade k. Para esses, temos:

$$\frac{\partial Loss_k}{\partial w_{j,k}} = -2\left(y_k - a_k\right)\frac{\partial a_k}{\partial w_{j,k}} = -2\left(y_k - a_k\right)\frac{\partial g(in_k)}{\partial w_{j,k}}$$

$$= -2\left(y_k - a_k\right)g'(in_k)\frac{\partial in_k}{\partial w_{j,k}} = -2\left(y_k - a_k\right)g'(in_k)\frac{\partial}{\partial w_{j,k}}\left(\sum_j w_{j,k}a_j\right)$$

$$= -2\left(y_k - a_k\right)g'(in_k)a_j = -2a_j\Delta_k$$

Derivação da regra de retropropagação (outras camadas)

$$\frac{\partial Loss_k}{\partial w_{i,j}} = -2\left(y_k - a_k\right)\frac{\partial a_k}{\partial w_{i,j}} = -2\left(y_k - a_k\right)\frac{\partial g(in_k)}{\partial w_{i,j}}$$

$$= -2\left(y_k - a_k\right)g'(in_k)\frac{\partial in_k}{\partial w_{i,j}} = -2\Delta_k \frac{\partial}{\partial w_{i,j}}\left(\sum_j w_{j,k}a_j\right)$$

$$= -2\Delta_k w_{j,k}\frac{\partial a_j}{\partial w_{i,j}} = -2\Delta_k w_{j,k}\frac{\partial g(in_j)}{\partial w_{i,j}}$$

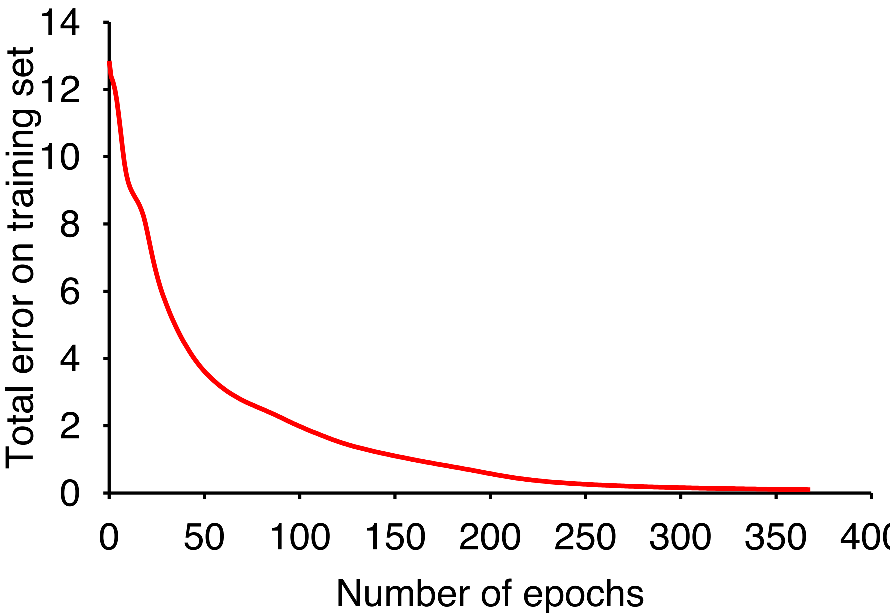
$$= -2\Delta_k w_{j,k} g'(in_j)\frac{\partial in_j}{\partial w_{i,j}}$$

$$= -2\Delta_k w_{j,k} g'(in_j)\frac{\partial}{\partial w_{i,j}}\left(\sum_i w_{i,j}a_i\right)$$

$$= -2\Delta_k w_{j,k} g'(in_j)a_i = -2a_i\Delta_j$$

Aprendizagem com retropropagação

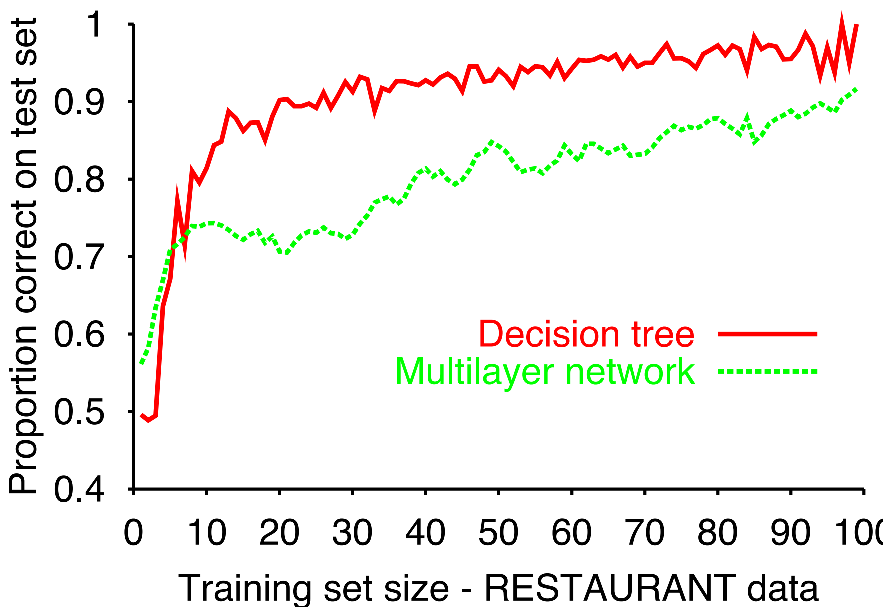
- Em cada época, somar atualizações de gradiente para todos os exemplos e aplicar
- Curva de treino para 100 exemplos do restaurante: encontra ajustamento perfeito



- Problemas típicos: convergência lenta, mínimos locais

Aprendizagem com retropropagação

- Curva de aprendizagem para rede multicamada com 4 unidades escondidas:

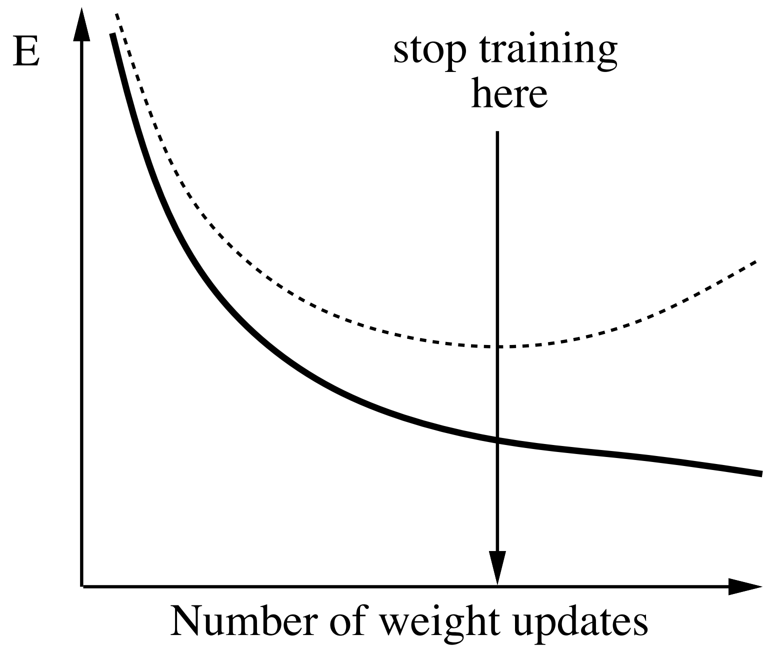


- Redes multicamada são adequadas para tarefas complexas de reconhecimento de padrões, mas o resultado pode não ser facilmente compreendido

Utilização prática do algoritmo

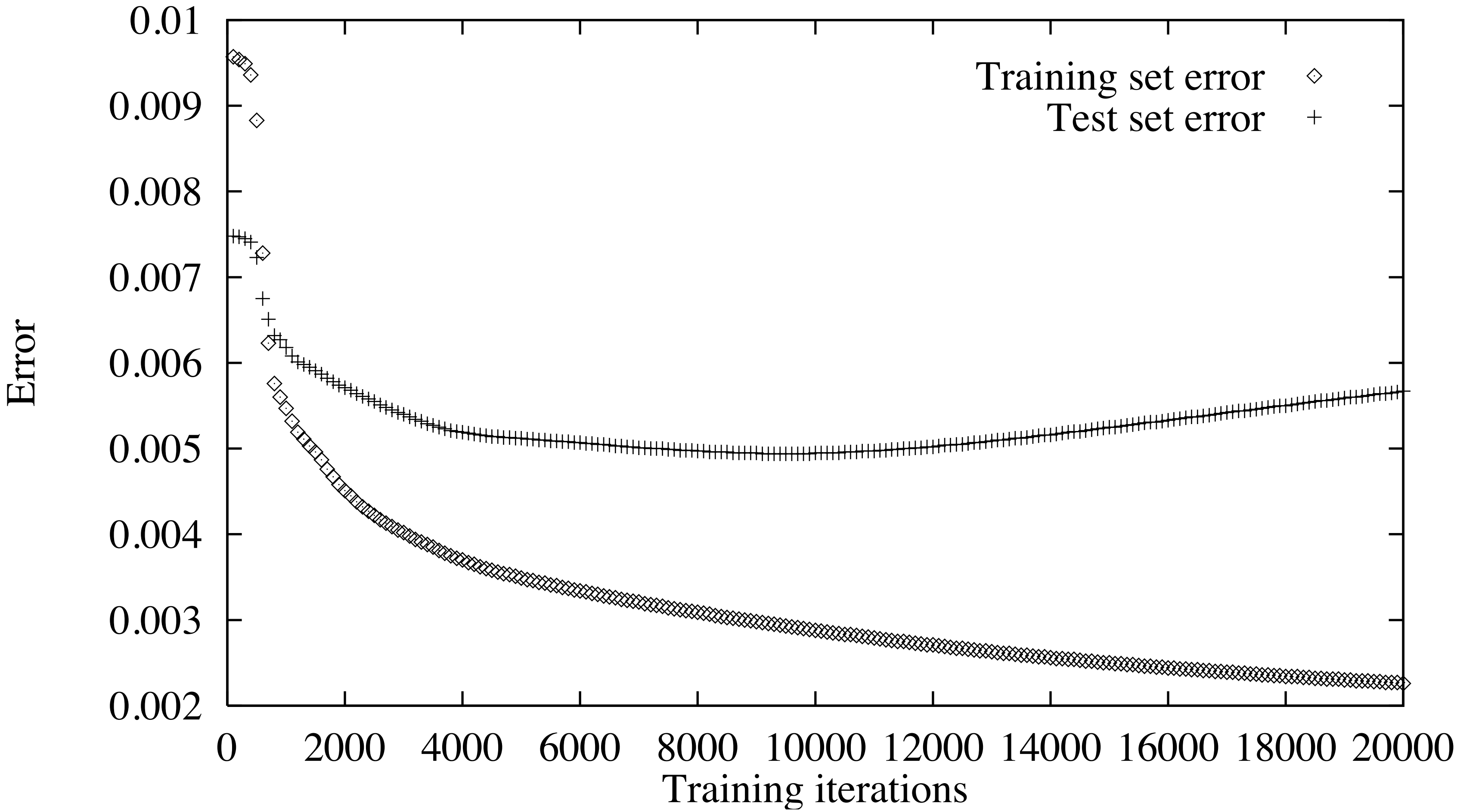
- Para o caso da função sigmóide é habitual inicializarem-se os pesos com valores aleatórios no intervalo [-0.5, 0.5] ou [-1, 1].
- O prolongamento da aprendizagem pode levar a problemas de sobreajustamento, ou seja, a rede classifica bem o conjunto de treino mas mal o conjunto de validação.
- O numero de exemplos de treino também e difícil de determinar, mas pode-se seguir uma das seguintes regras práticas:
  - O numero de exemplos deve ser 5 a 10 vezes maior do que o número de pesos.
  - 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

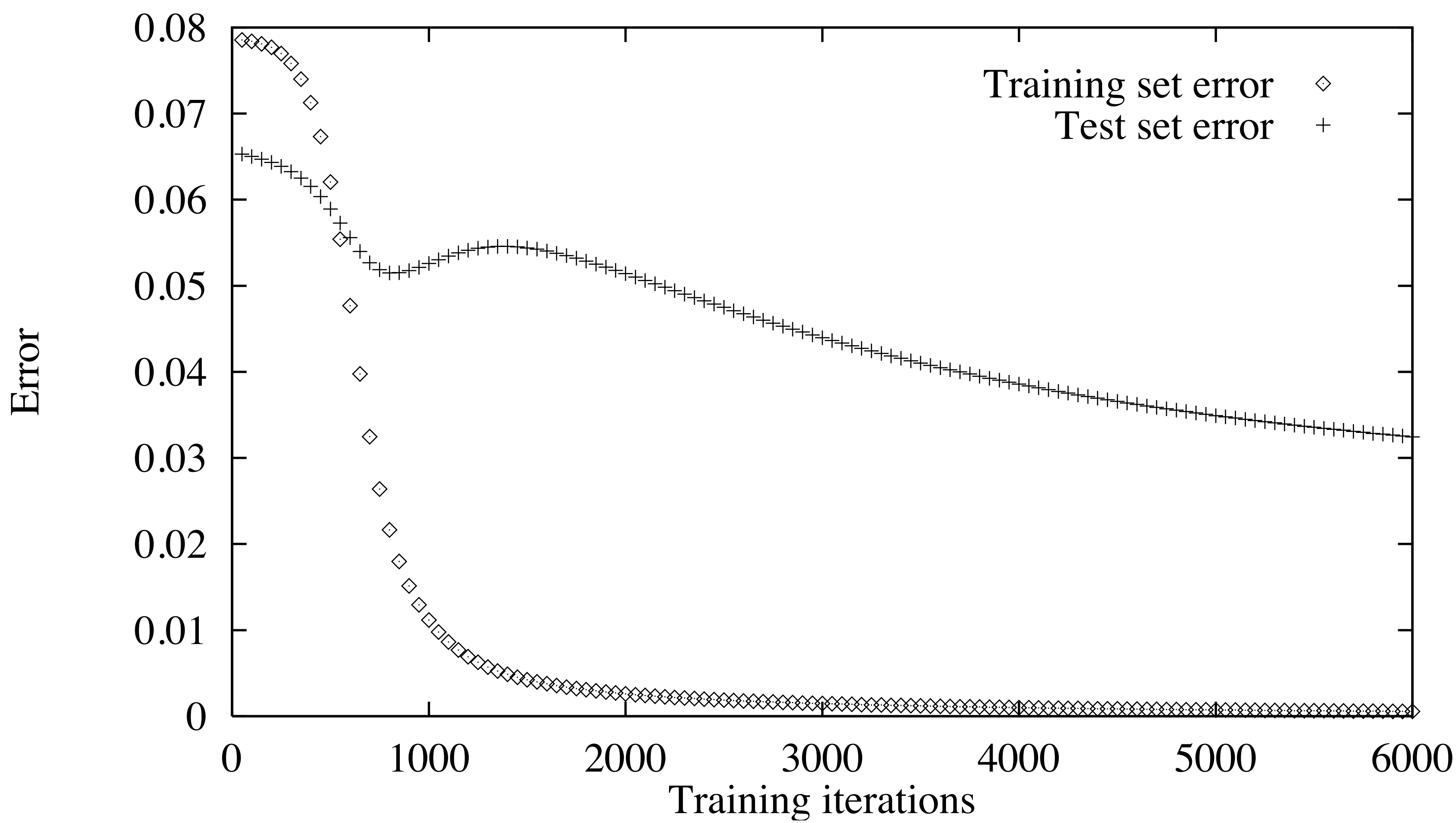


- Os exemplos devem ser divididos em conjunto de treino e conjunto de validação.
  - Deve haver um conjunto de teste que não é utilizado na aprendizagem.
- Utiliza-se o conjunto de treino no algoritmo de aprendizagem, parando-se quando se minimiza o erro no conjunto de validação.

Sobreajustamento (quando parar?)



Sobreajustamento (quando parar?)



Problemas de convergência

- Pode haver grandes oscilações no erro. Uma das formas para resolver consiste em alterar a regra para utilizar momento - μ
- As variações dos pesos da iteração t para t + 1 são dadas por:

$$\Delta w_{i,j}(t+1) = \mu \Delta w_{i,j}(t) + \alpha \times a_i \times \Delta_j$$

- Paralelo com bola a descer superfcie:
  - Ritmo de aprendizagem: velocidade (valores tpicos entre 0.1 e 0.9).
  - Momento: inercia - mantem direção do movimento anterior.
- Acelerar convergência atraves 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 (Support vector machines; algoritmos de visão) ≈ 0.6% erro

- Humanos = 0.2% erro

Sumário

- A maioria dos cérebros tem muitos neurónios; cada neurónio ≈ 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.