

ICC204 - Aprendizagem de Máquina e Mineração de Dados

# Redes Neurais Artificiais



Prof. Rafael Giusti  
[rgiusti@icomp.ufam.edu.br](mailto:rgiusti@icomp.ufam.edu.br)

# Leitura recomendada

- Tom M. Mitchell. *Machine Learning*
  - Capítulo 4: *Artificial Neural Networks*  
(Leitura fortemente recomendada)
- Consulte também os livros
  - Alpaydin. *Introduction to Machine Learning*
  - Bishop. *Pattern Recognition and Machine Learning*  
(disponível no ColabWeb)

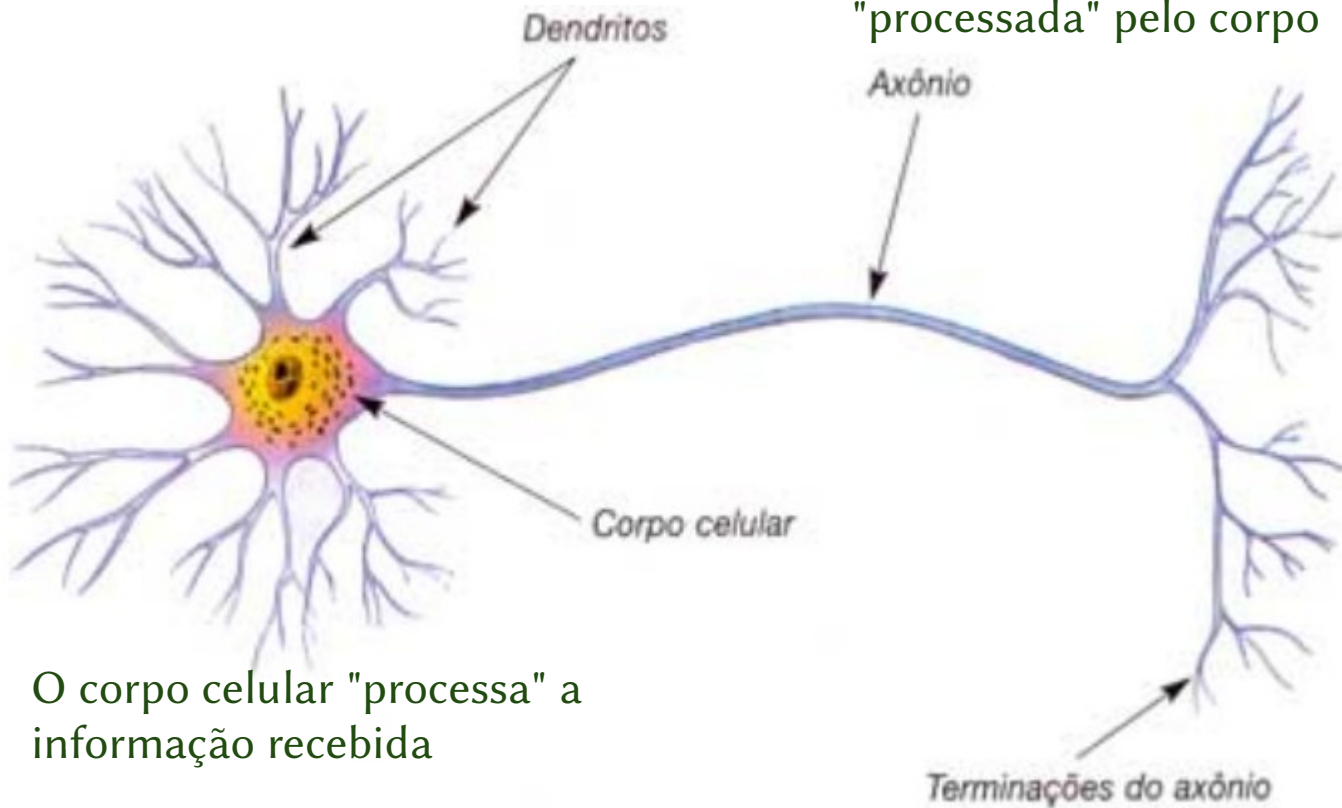
# Perceptron

- O **perceptron** é um modelo de neurônio artificial
  - Trata-se de um dos modelos mais antigos de inteligência artificial
  - Criado por volta de 1960
  - É baseado em um modelo do neurônio real

# Neurônio vs. perceptron

Os dendritos recebem informações de outros neurônios

O axônio propaga a informação "processada" pelo corpo

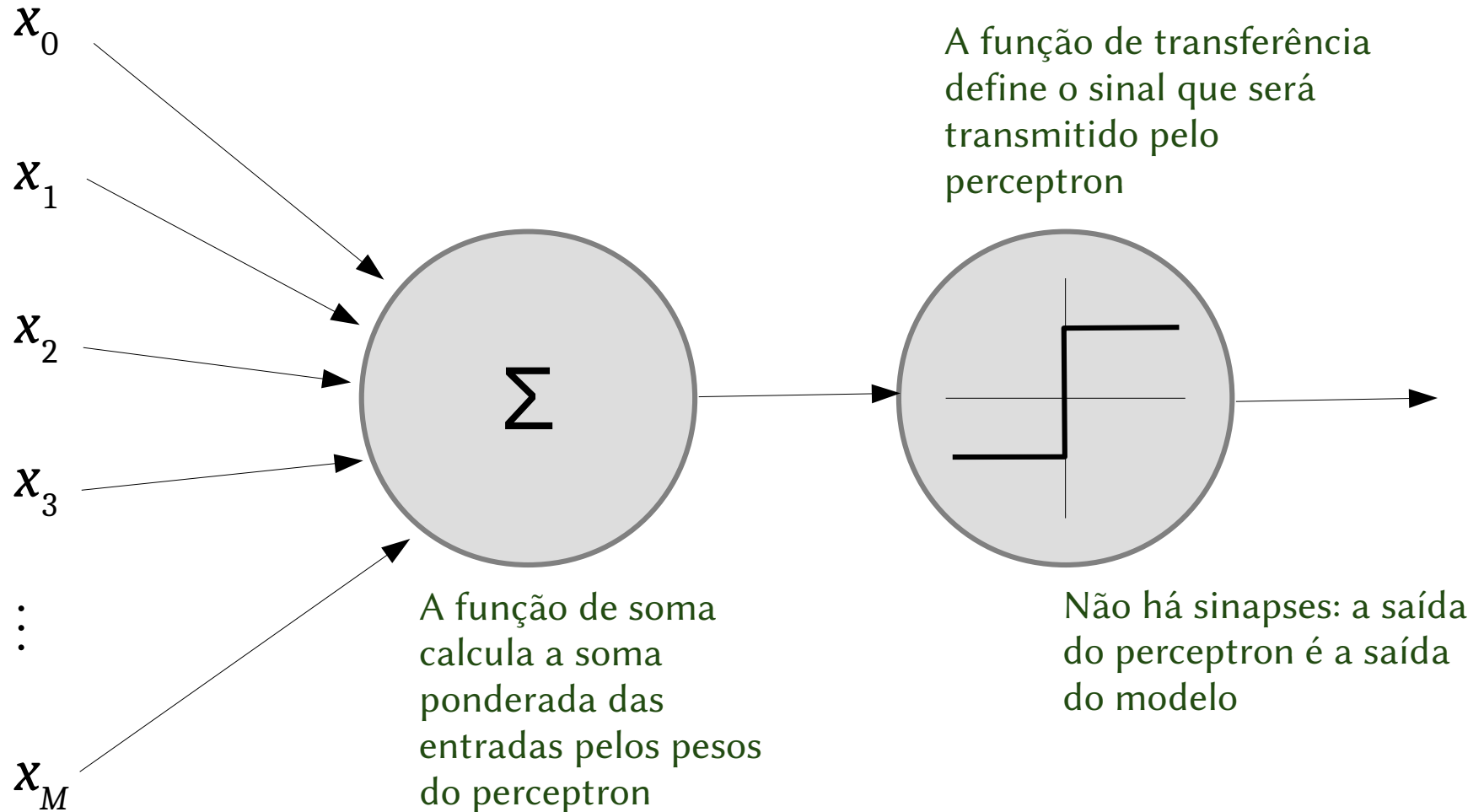


O corpo celular "processa" a informação recebida

As sinapses são conexões com outros neurônios que propagam o sinal do neurônio por meio de neurotransmissores

# Neurônio vs. perceptron

As entradas do perceptron simulam o sinal de recebido de outros neurônios



# Perceptron

- A **função de soma** simula o "processamento" do neurônio com base nas entradas recebidas
  - O perceptron possui  $M$  pesos
  - Os pesos  $w_1, w_2, \dots, w_M$  são associados ao vetor de características do exemplo  $\mathbf{x} = (x_1, x_2, \dots, x_M)$
  - O peso  $w_0$  é um fator de **viés** do perceptron
    - Não confundir com o viés do indutor!

# Perceptron

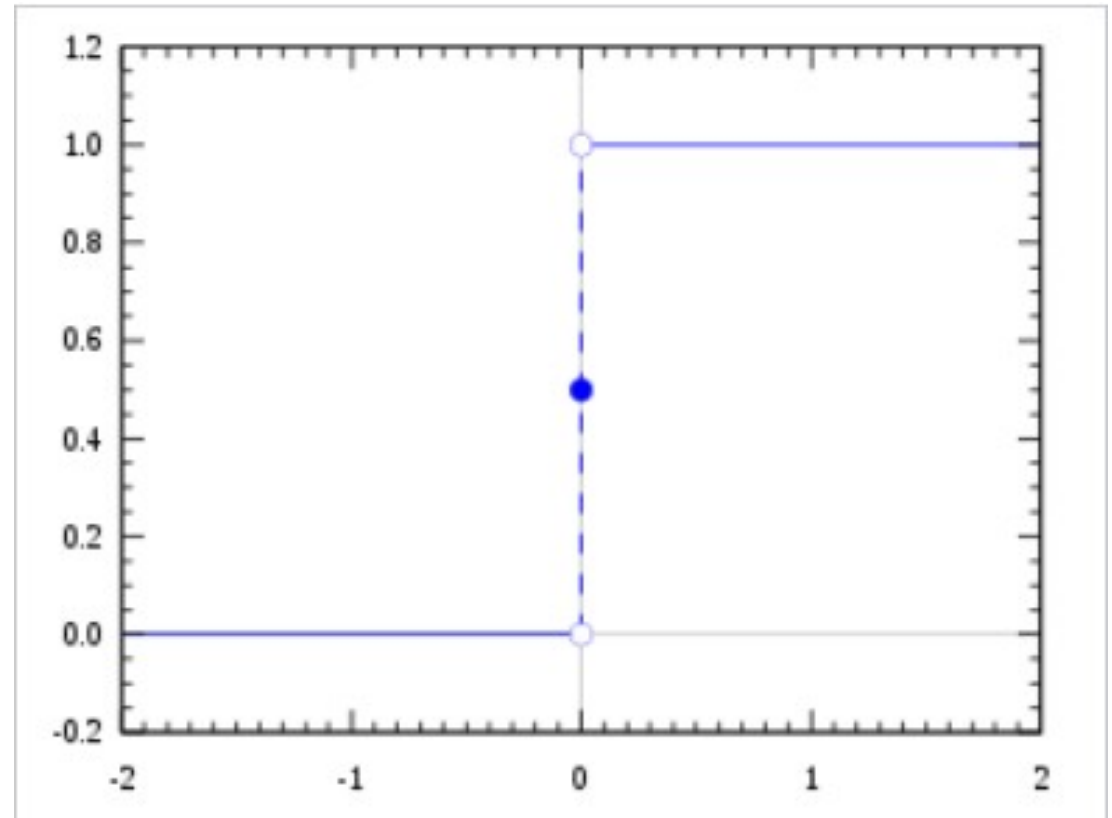
- Normalmente, definimos os **vetores aumentados** de pesos e de coeficientes
  - $\mathbf{x} = (1, x_1, x_2, \dots, x_M)$
  - $\mathbf{w} = (w_0, w_1, w_2, \dots, w_M)$
- Assim, a função de soma é a combinação linear

$$\hat{y}(\mathbf{x}, \mathbf{w}) = \mathbf{xw}^T$$

# Perceptron

- A **função de ativação** ou **função de transferência** determina a saída do perceptron
  - Exemplo: **função degrau**

$$f(x) = \begin{cases} 1, & \text{se } x \geq 0,5 \\ 0, & \text{c.c.} \end{cases}$$





# Perceptron

- Linguagem de descrição do modelo
  - O espaço de hipóteses é o conjunto

$$H = \{ \mathbf{w} \mid \mathbf{w} \in \mathbb{R}^{M+1} \}$$

- Os parâmetros do modelo são os coeficientes  $w_1, w_2, \dots, w_M$  associados aos atributos e o coeficiente  $w_0$ , denominado viés (ou *intercept*)
  - É um modelo de separação linear para problemas de **classificação binária**

# Treinamento do perceptron

- O perceptron pode ser treinado pela **regra do delta** ou pelo **método do gradiente**
  - Regra do delta
    - Os pesos são iniciados com valores aleatórios de baixa magnitude
    - A cada passo, para um exemplo cujo valor real é  $t$  e a saída do perceptron é  $o$ , os pesos são atualizados
      - $w_i \leftarrow w_i + \Delta w_i$
      - $\Delta w_i = \eta(t - o)x_i$

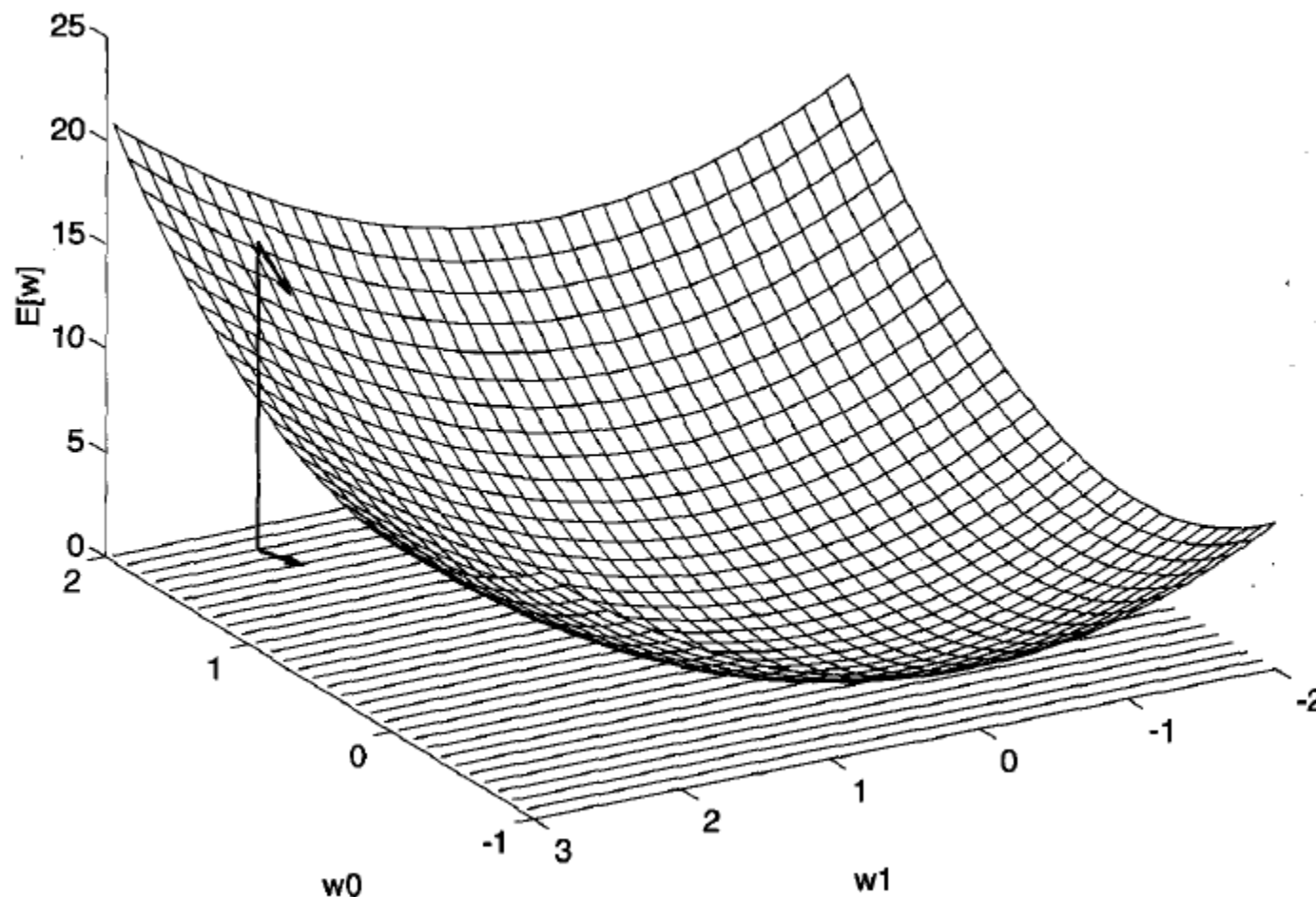
# Treinamento do perceptron

- A regra do delta é capaz de encontrar um conjunto de pesos adequado para problemas linearmente separáveis
- Para problemas que não são linearmente separáveis, podemos empregar o método do gradiente sobre a função de perda da função de soma

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

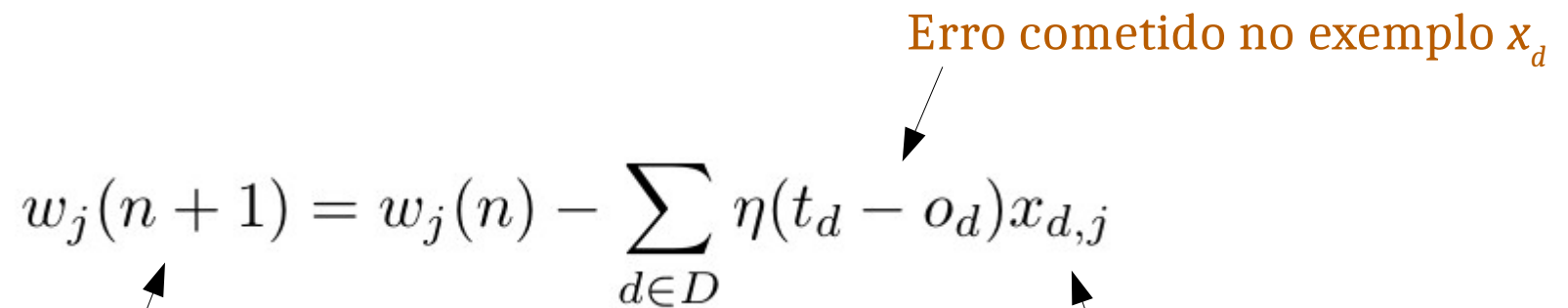
# Treinamento do perceptron

- Essa função de perda define uma superfície convexa que possui um único mínimo global



# Treinamento do perceptron

- Método do gradiente pra o perceptron
  - Atualização dos pesos no passo  $n$

$$w_j(n+1) = w_j(n) - \sum_{d \in D} \eta(t_d - o_d)x_{d,j}$$
The diagram shows the weight update equation with three arrows pointing to its components: one from the text 'Peso w\_j na próxima iteração' to w\_j(n+1), one from 'Erro cometido no exemplo x\_d' to the term (t\_d - o\_d), and one from 'j-ésimo coeficiente do vetor de características do exemplo x\_d' to x\_{d,j}.

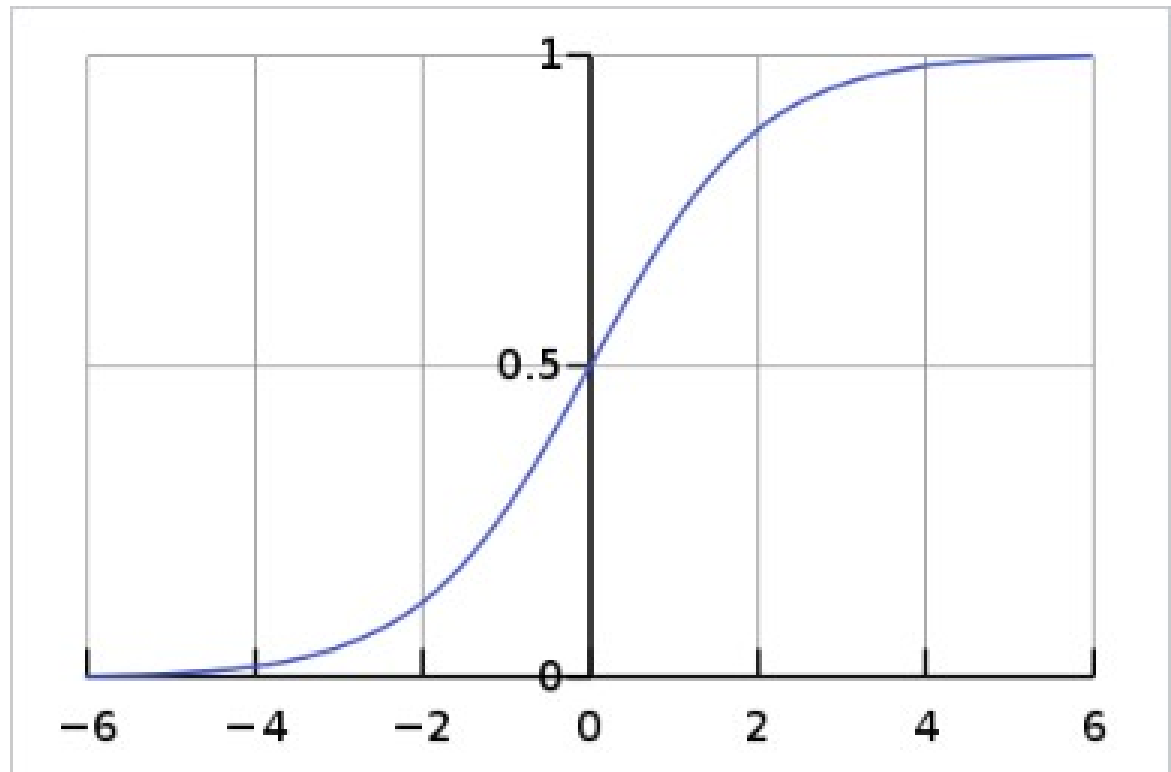
Peso  $w_j$  na próxima iteração

$j$ -ésimo coeficiente do vetor de características do exemplo  $x_d$

# Perceptron com função sigmoide

- A função de ativação não precisa ser linear
- Exemplo: **função sigmoide**

$$s(x) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$



# Perceptron com função sigmoide

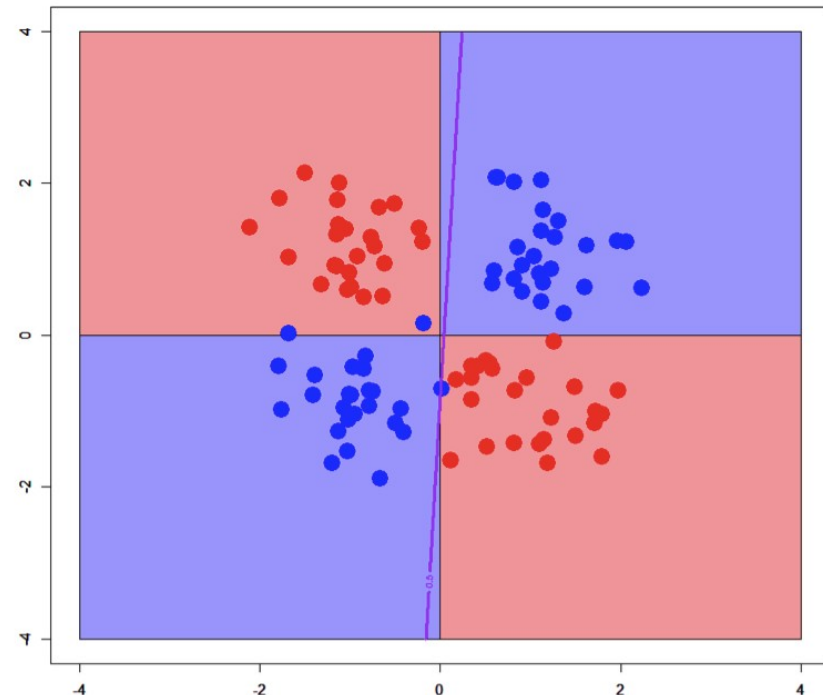
- A função sigmoide é um caso particular de **função logística**
- Modelos de classificação baseados na função logística são conhecidos como **regressores logísticos**
- Em particular, o perceptron que utiliza função logística é também denominado **logistic unit** ou **logit**

# Limitações do perceptron

- O perceptron é um separador linear, mesmo quando utilizado como regressor logístico ou outra função de ativação não linear
- Famoso problema **XOR**
  - Demonstração no Google Playground

[playground.tensorflow.org/](https://playground.tensorflow.org/)

Minsky e Pappert, 1969: Perceptrons: o problema do XOR





# Redes neurais

- Porém, o perceptron é capaz de representar adequadamente as funções booleanas AND, OR, NAND e NOR

X	Y	AND	OR	NAND	NOR
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	1	0	0

# Redes neurais

- Logit para função AND (booleana)
  - $w_0 = -0,8$
  - $w_1 = 0,5$
  - $w_2 = 0,5$

Entrada	Função soma	Sigmoide	Saída
(0, 0)	-0,8	0,31	0
(0, 1)	-0,3	0,43	0
(1, 0)	-0,3	0,43	0
(1, 1)	0,2	0,55	1

# Redes neurais

- Logit para função OR (booleana)

- $w_0 = -0,3$

- $w_1 = 0,5$

- $w_2 = 0,5$

Entrada	Função soma	Sigmoide	Saída
(0, 0)	-0,3	0,42	0
(0, 1)	0,2	0,55	1
(1, 0)	0,2	0,55	1
(1, 1)	0,7	0,67	1

# Redes neurais

- Logit para função NAND (booleana)

- $w_0 = 4$

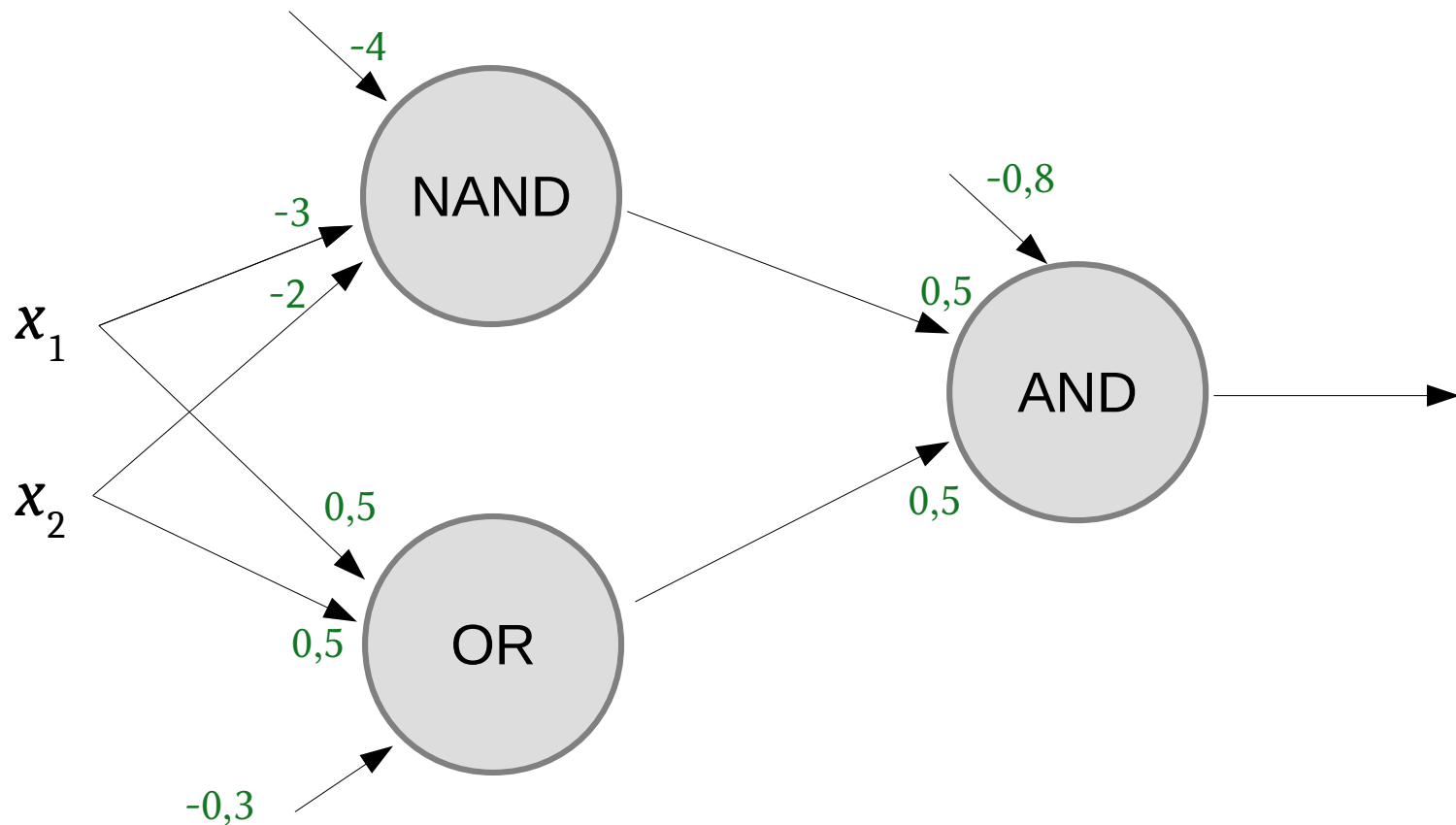
- $w_1 = -3$

- $w_2 = -2$

Entrada	Função soma	Sigmoide	Saída
(0, 0)	4	0,98	1
(0, 1)	2	0,88	1
(1, 0)	1	0,73	1
(1, 1)	-1	0,27	0

# Redes neurais

- Aproximação do XOR com três perceptrons em duas camadas (para função booleana,  $f: \{0,1\} \rightarrow \{0,1\}$ )



# Redes neurais

- A dificuldade está em como encontrar o conjunto de pesos que produz a saída desejada para a rede
  - O perceptron "caiu" em 1969
  - Apenas em 1986 um trabalho por Rumelhart, Hinton e Williams, publicado na revista Nature, tornou popular um algoritmo para encontrar pesos de redes neurais com múltiplas camadas
    - Algoritmo de **propagação retrógrada** ou ***backpropagation***

# Backpropagation

- O treinamento da rede é feito em duas etapas
  - Na primeira, ocorre um *feed-forward*, em que as unidades da rede processam seus sinais de entrada sequencialmente
  - A saída da rede é comparada com a saída esperada e o erro é calculado
    - Na saída
    - Nas camadas ocultas, considerando a proporção com que cada unidade contribuiu para o erro

# Backpropagation

- Através do método do gradiente descendente, os erros de cada unidade são utilizados para atualizar os pesos
  - Essa etapa é chamada **backward** e dá o nome ao algoritmo, pois o erro é propagado "para trás"
  - Para que seja possível calcular o gradiente, é necessário que a função seja diferenciável
    - Portanto a propagação retrógrada toma como base o neurônio sem a discretização da saída



# Backpropagation

- Algoritmo de Rumelhart, Hinton e Williams, Nature, 323, outubro de 1986

## Learning representations by back-propagating errors

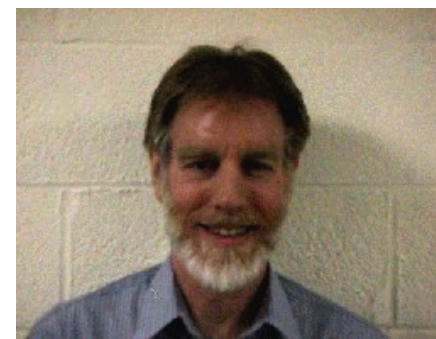
David E. Rumelhart\*, Geoffrey E. Hinton†  
& Ronald J. Williams\*

\* Institute for Cognitive Science, C-015, University of California,  
San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Philadelphia 15213, USA

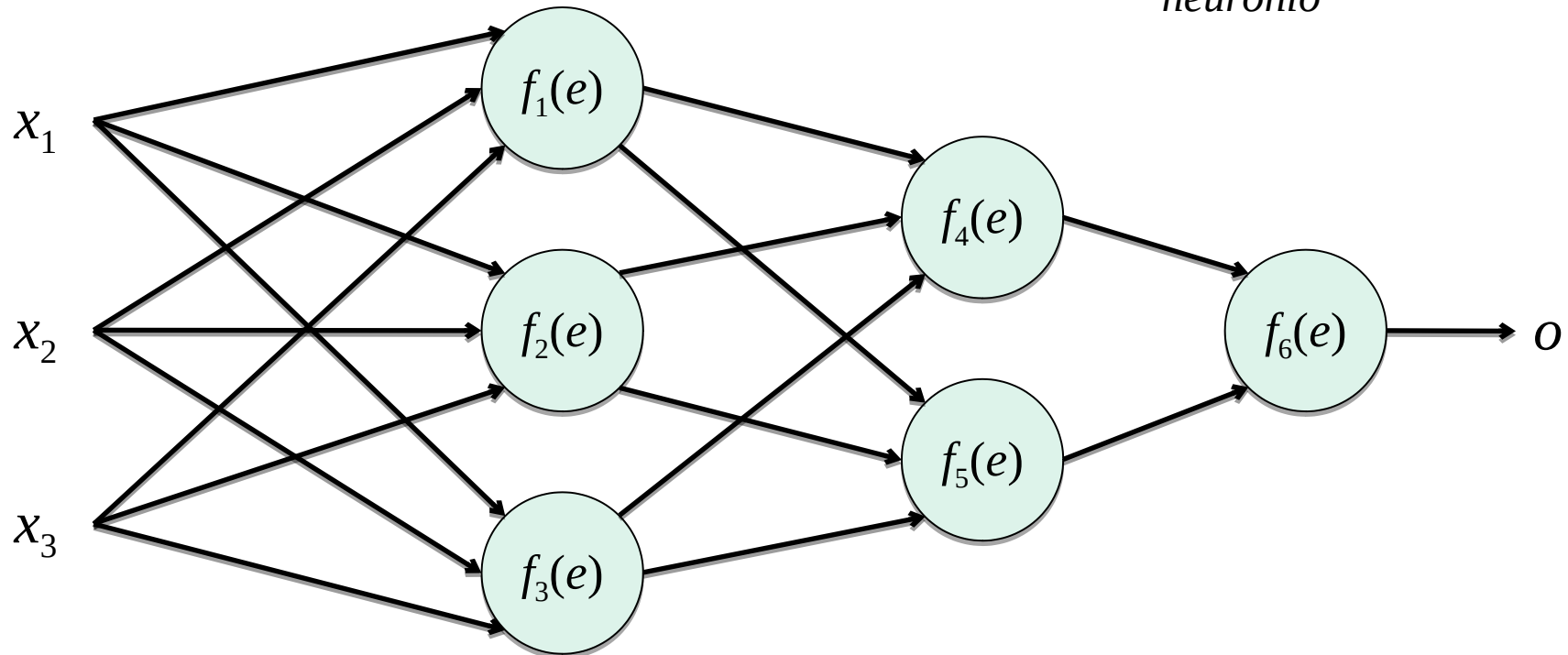
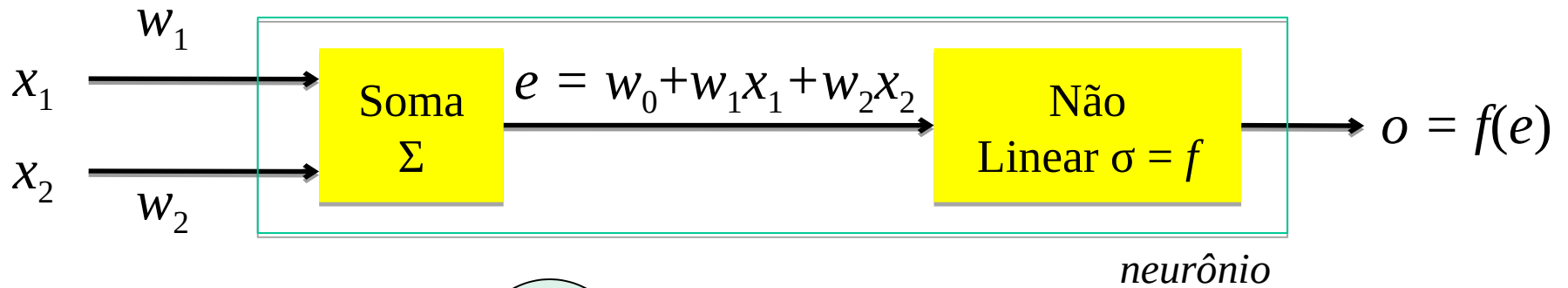
---

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure<sup>1</sup>.



# O algoritmo

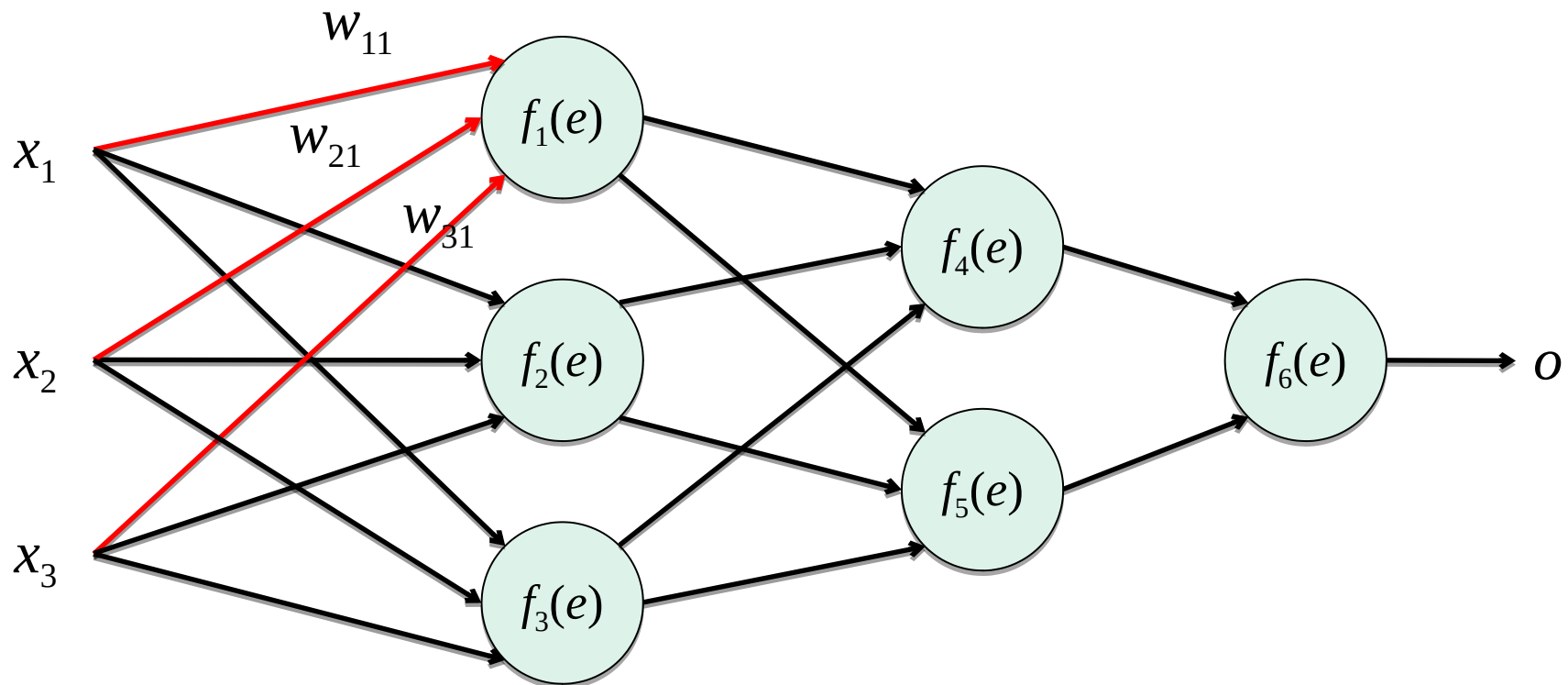
Uma rede neural com 2 camadas escondidas:



# O algoritmo

Uma rede neural com 2 camadas escondidas:

$$y_1 = f_1(w_{01} + w_{11}x_1 + w_{21}x_2 + w_{31}x_3)$$

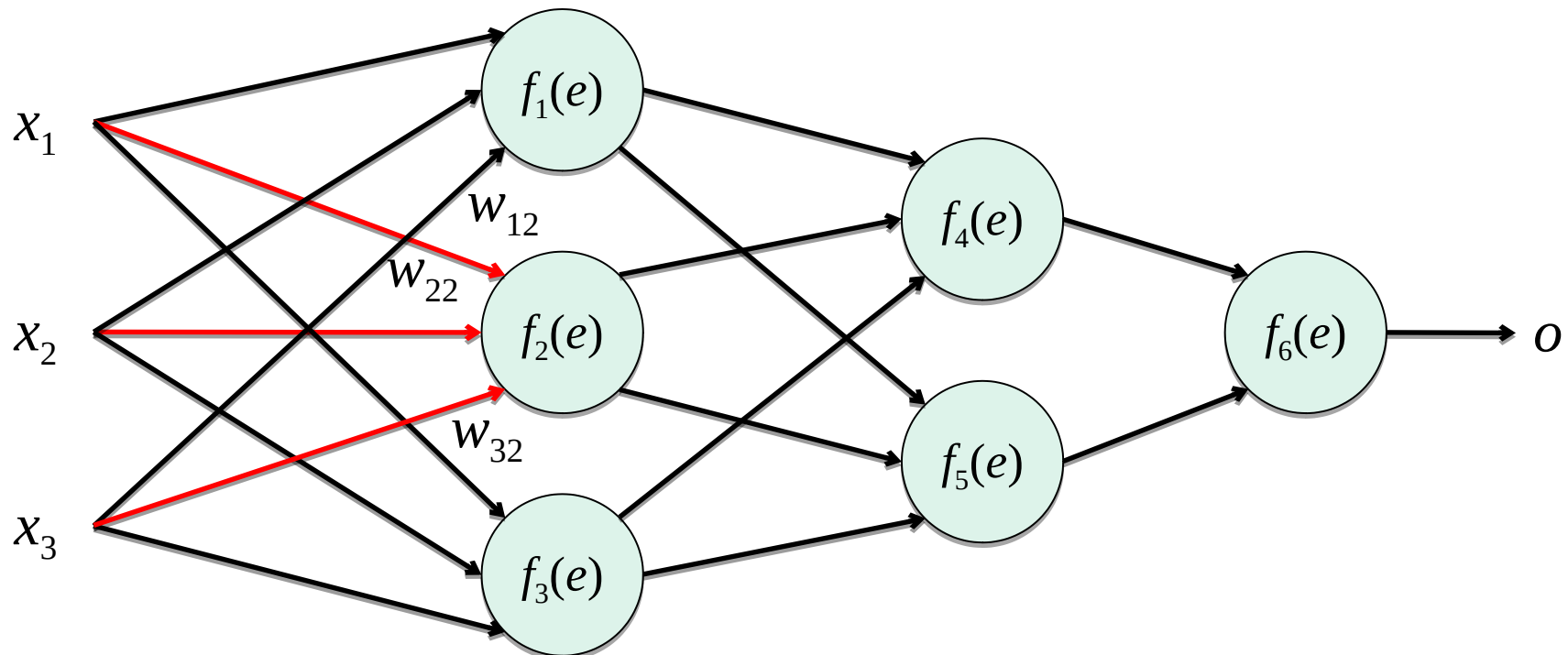


# O algoritmo

Uma rede neural com 2 camadas escondidas:

$$y_1 = f_1(w_{01} + w_{11}x_1 + w_{21}x_2 + w_{31}x_3)$$

$$y_2 = f_2(w_{02} + w_{12}x_1 + w_{22}x_2 + w_{32}x_3)$$



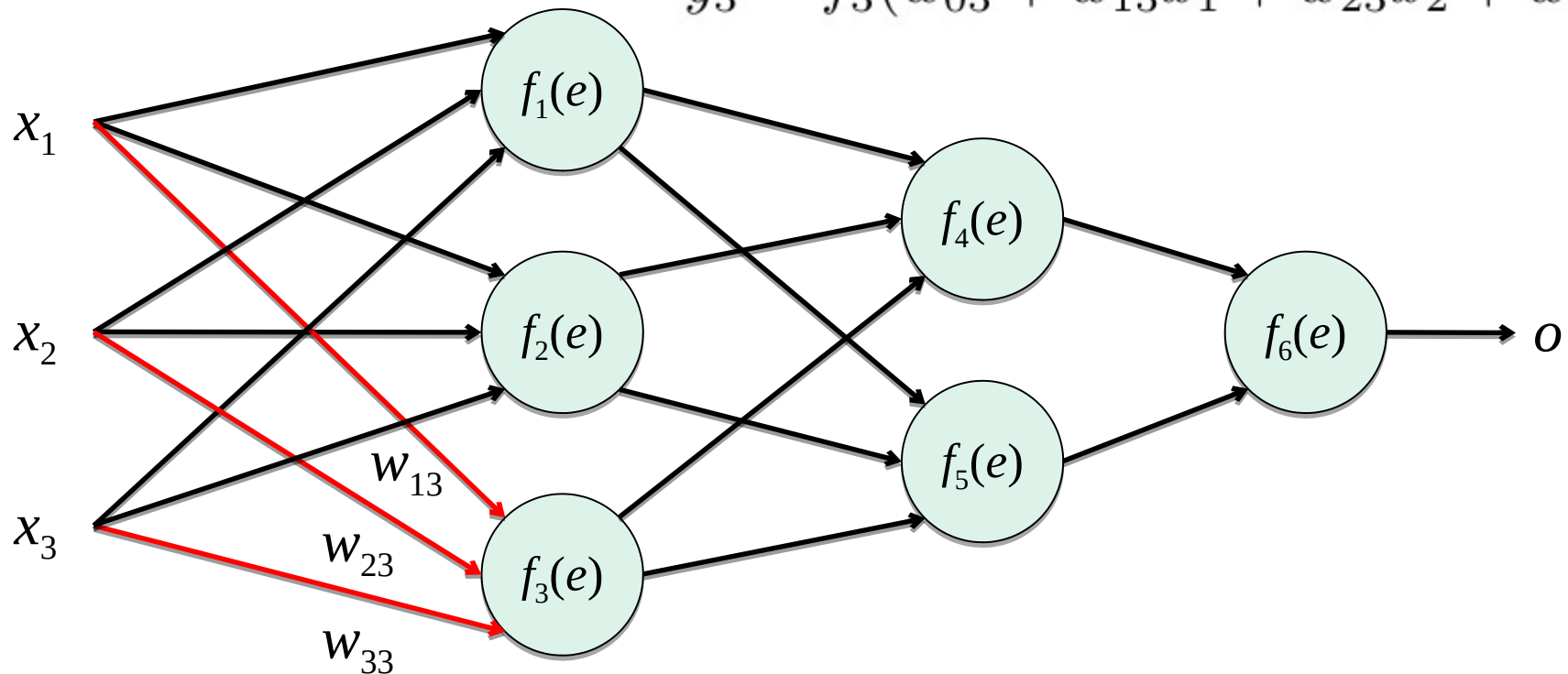
# O algoritmo

Uma rede neural com 2 camadas escondidas:

$$y_1 = f_1(w_{01} + w_{11}x_1 + w_{21}x_2 + w_{31}x_3)$$

$$y_2 = f_2(w_{02} + w_{12}x_1 + w_{22}x_2 + w_{32}x_3)$$

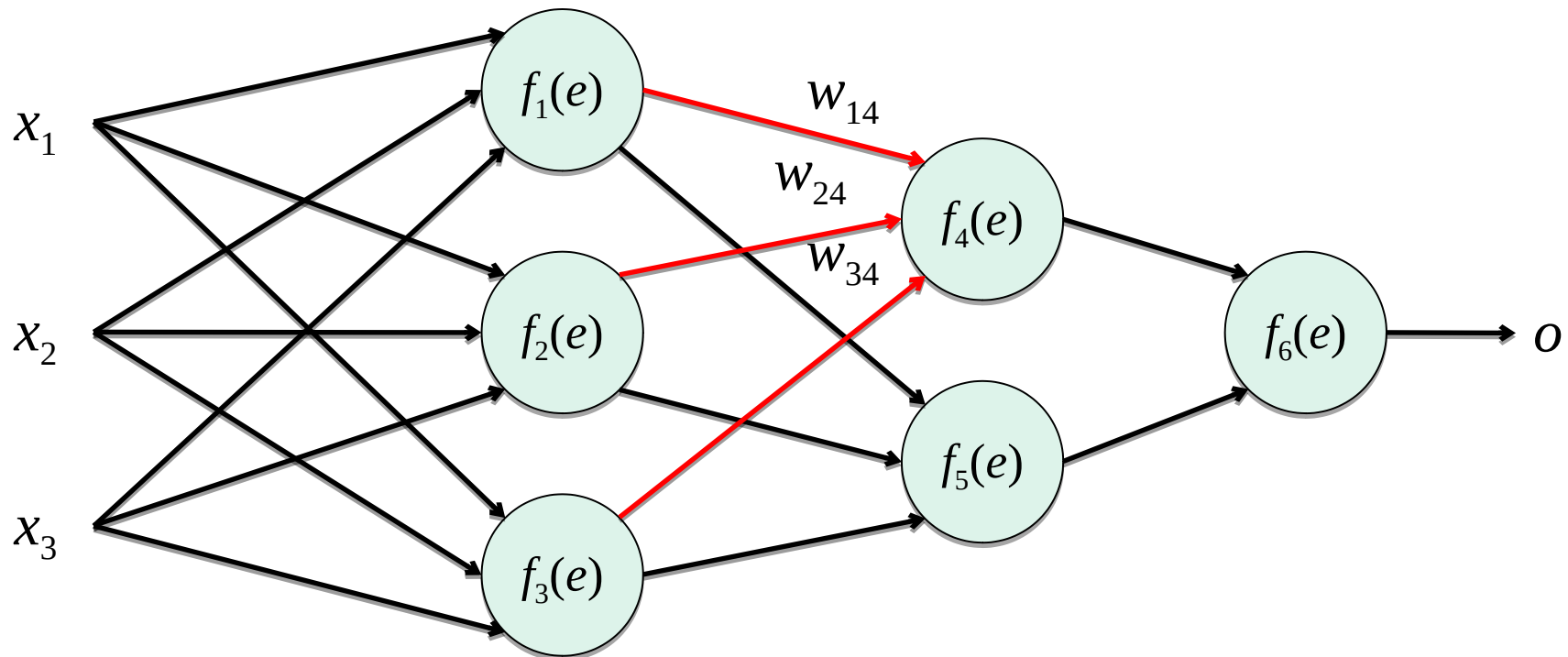
$$y_3 = f_3(w_{03} + w_{13}x_1 + w_{23}x_2 + w_{33}x_3)$$



# O algoritmo

Uma rede neural com 2 camadas escondidas:

$$y_4 = f_4(w_{04} + w_{14}y_1 + w_{24}y_2 + w_{34}y_3)$$

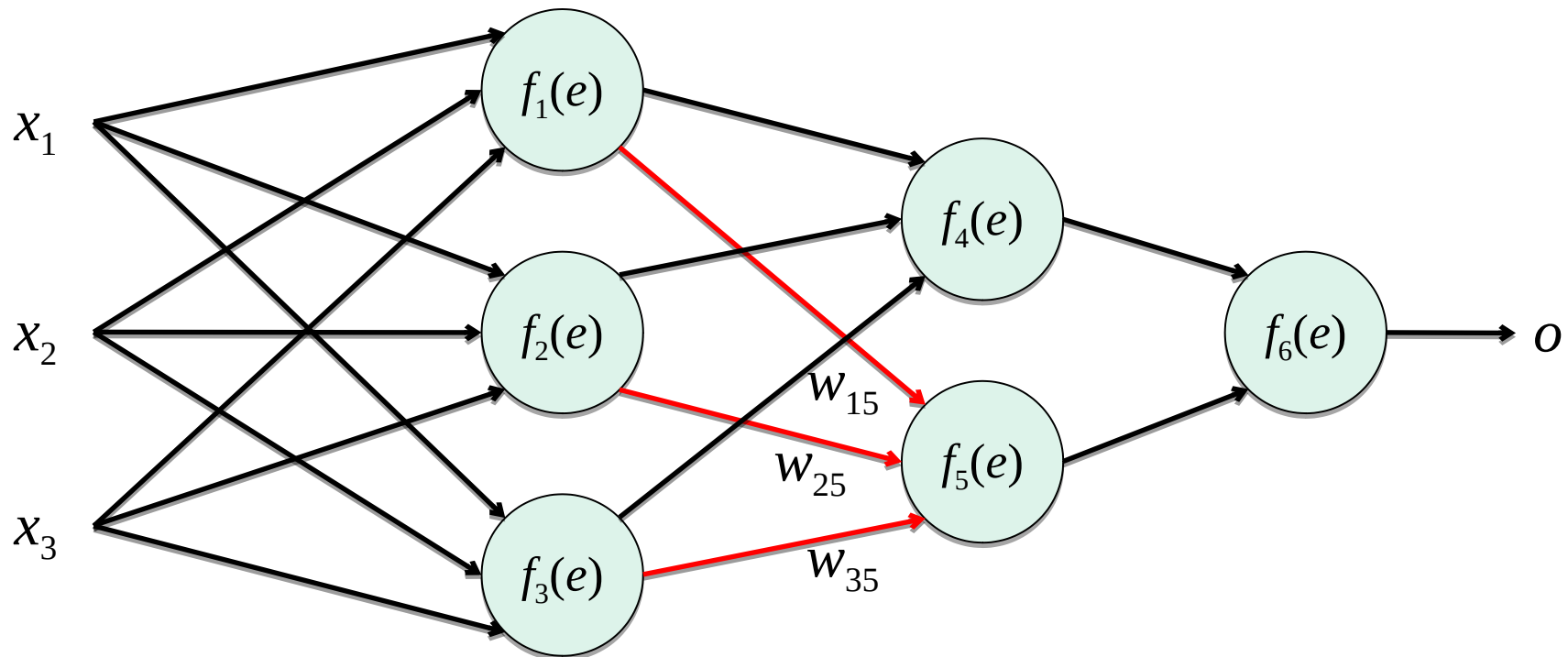


# O algoritmo

Uma rede neural com 2 camadas escondidas:

$$y_4 = f_4(w_{04} + w_{14}y_1 + w_{24}y_2 + w_{34}y_3)$$

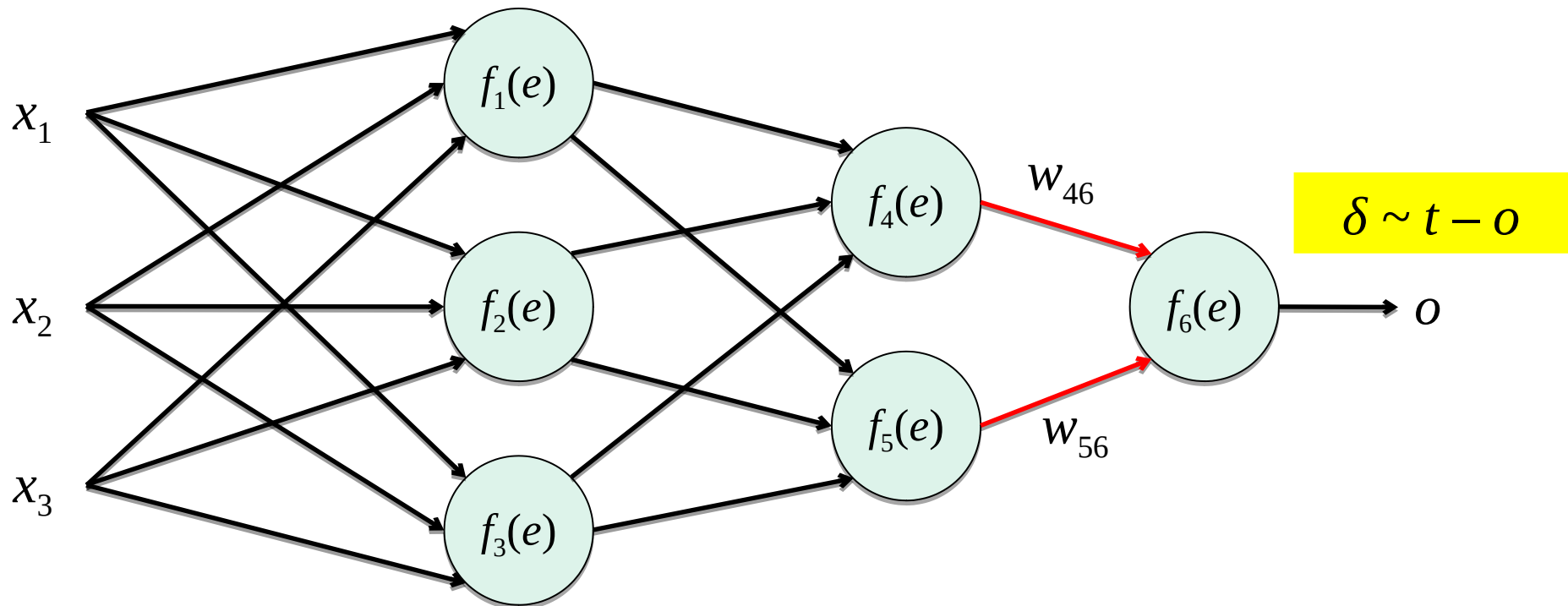
$$y_5 = f_5(w_{05} + w_{15}y_1 + w_{25}y_2 + w_{35}y_3)$$



# O algoritmo

Uma rede neural com 2 camadas escondidas:

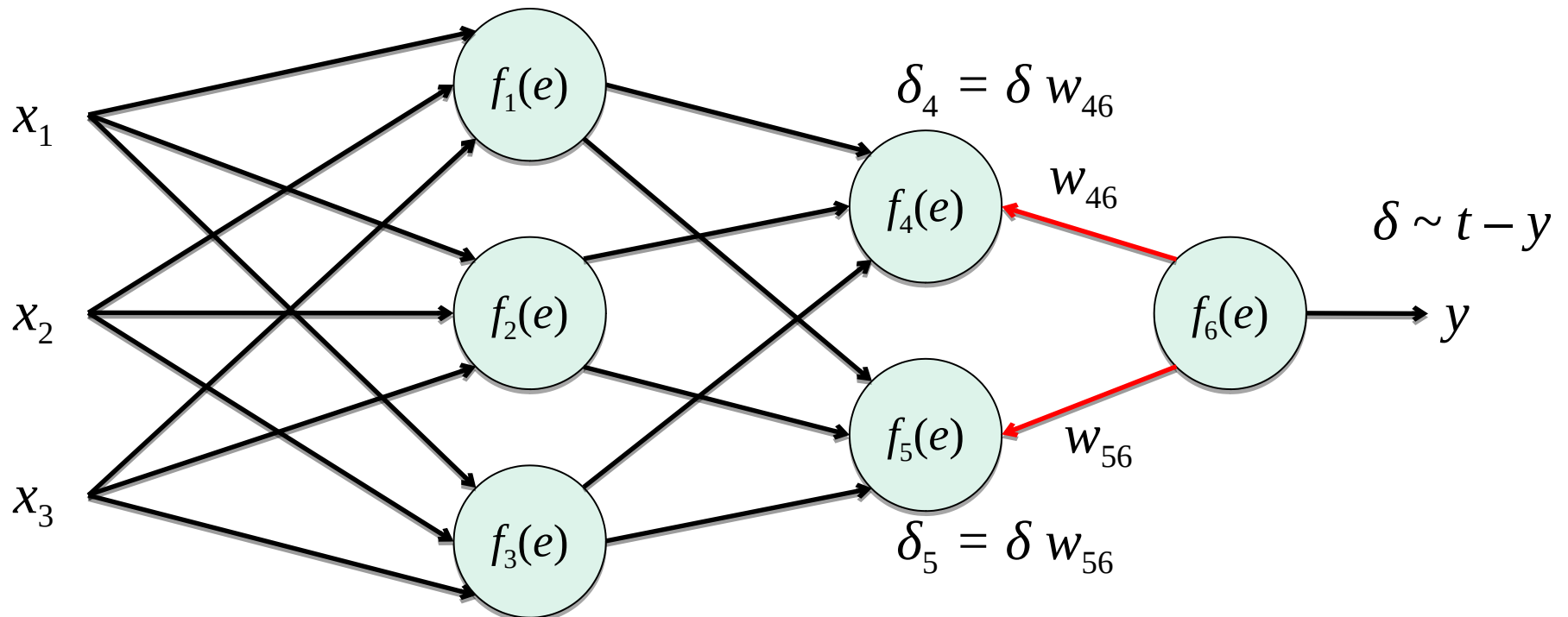
$$o = y_6 = f_6(w_{06} + w_{16}y_4 + w_{26}y_5)$$





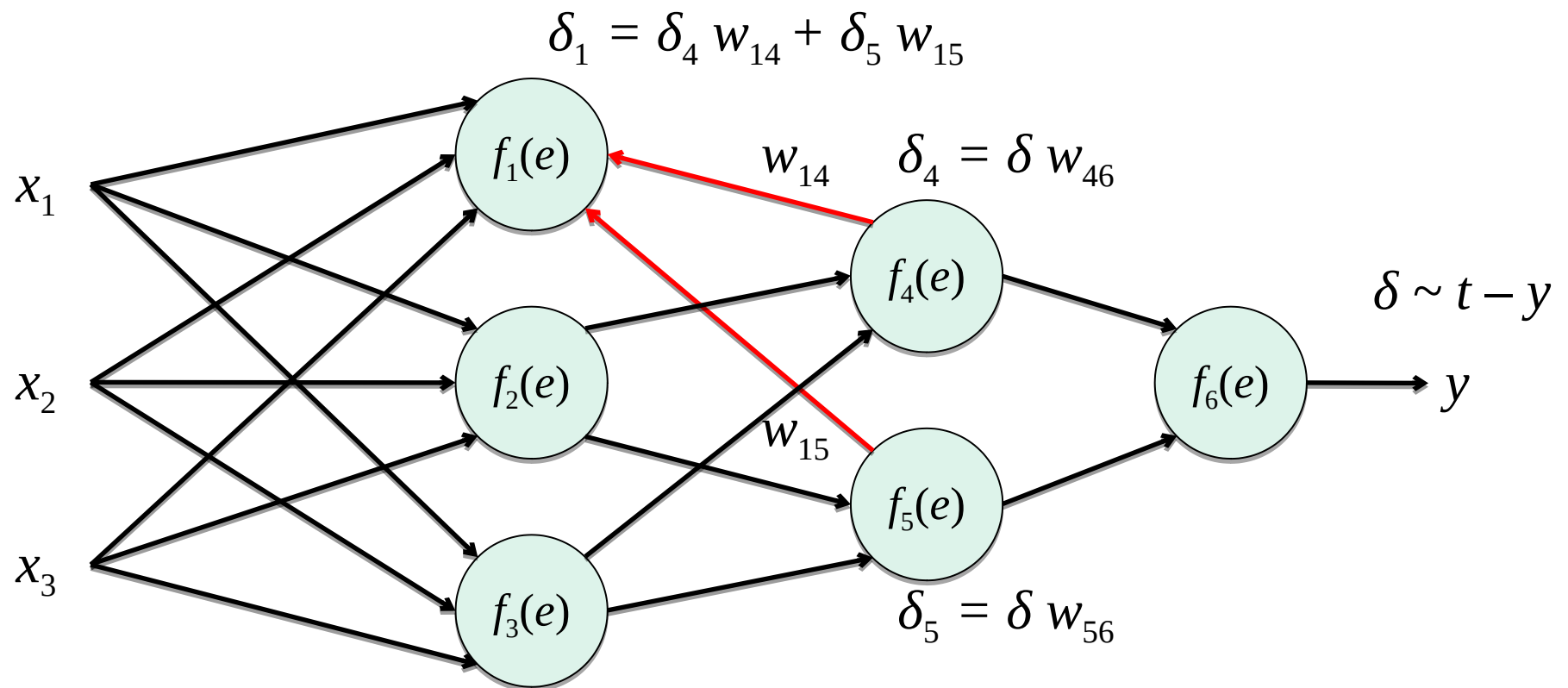
# O algoritmo

Uma rede neural com 2 camadas escondidas:



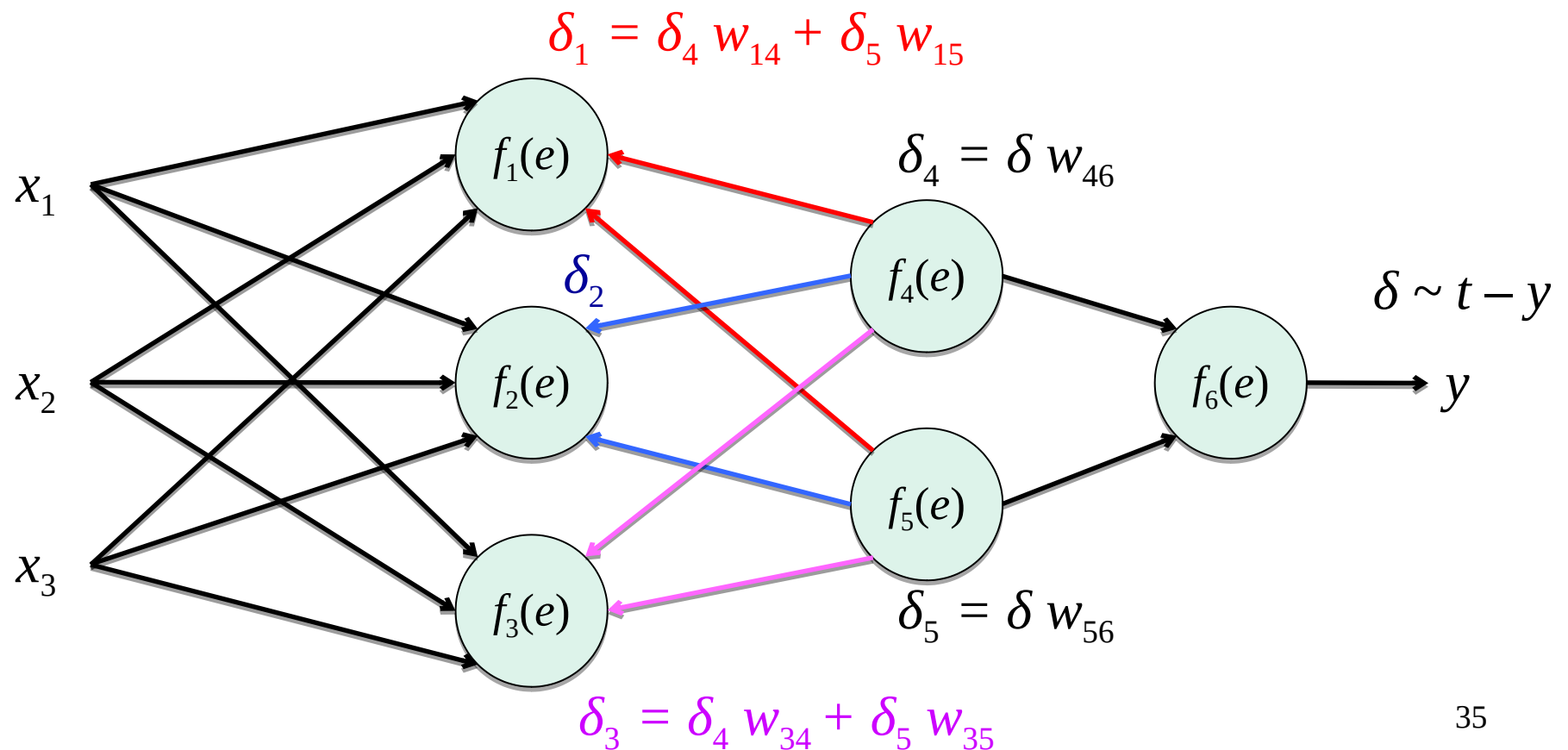
# O algoritmo

Uma rede neural com 2 camadas escondidas:



# O algoritmo

Uma rede neural com 2 camadas escondidas:



# O algoritmo

Uma rede neural com 2 camadas escondidas:

Como alterar os pesos?

