O Método Backpropagation

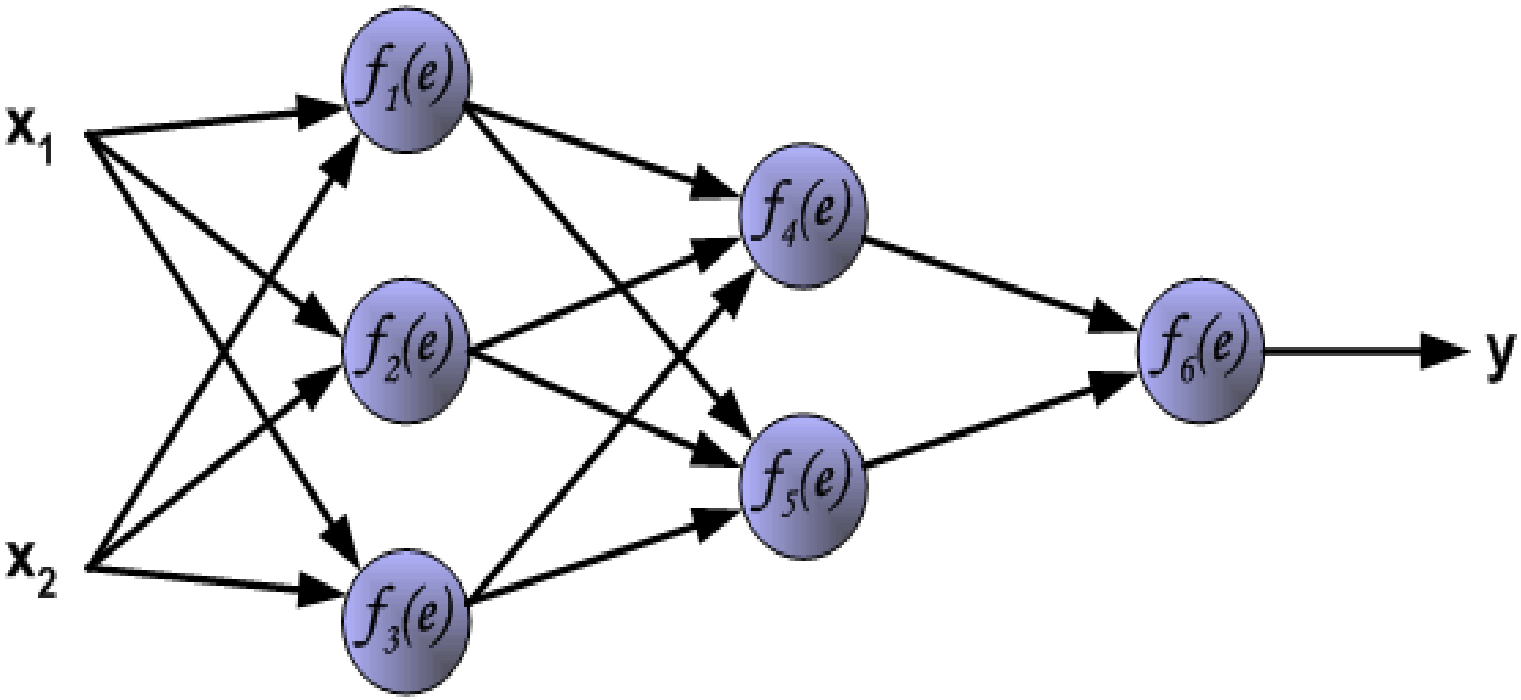
Marcelo Thielo

Baseado em material de

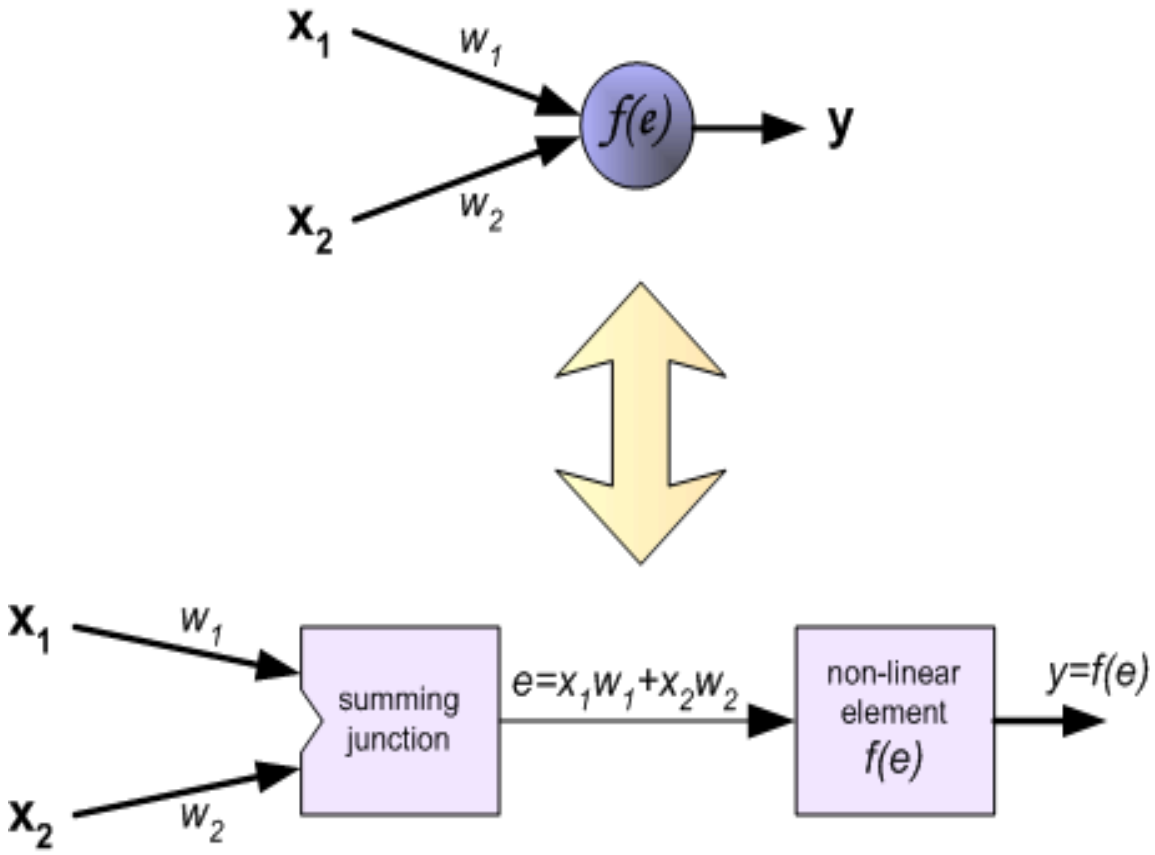
Mariusz Bernacki

Przemyslaw Wlodarczyk

Multi-Layer Perceptron

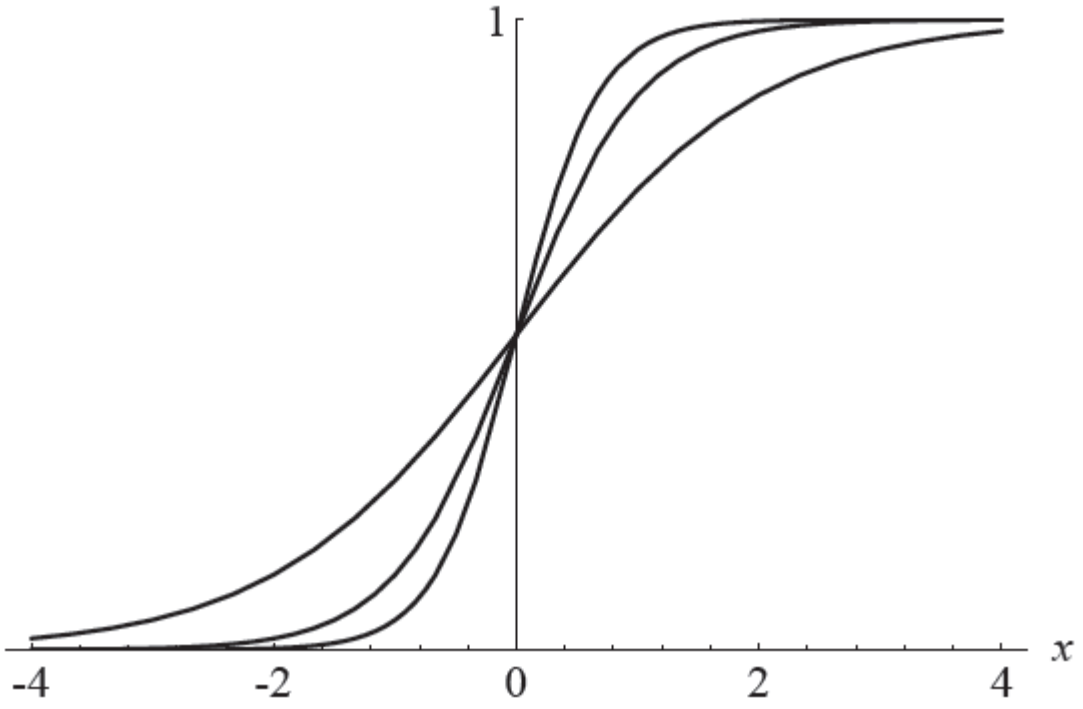


Célula=Perceptron



Função sigmoidal

$$f(x) = \frac{1}{1+e^{-cx}}$$



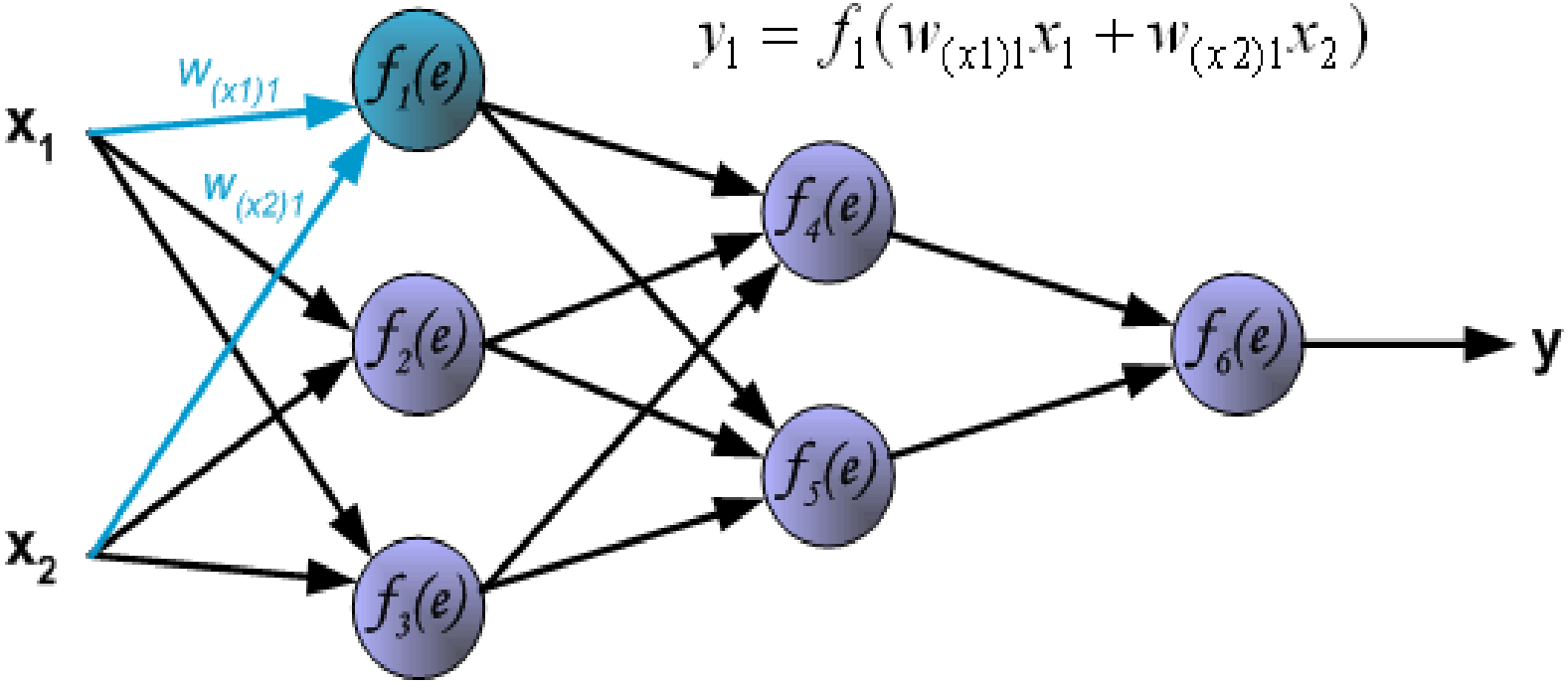
Três sigmóides (para c = 1, c = 2 e c = 3)

Derivando a *f*

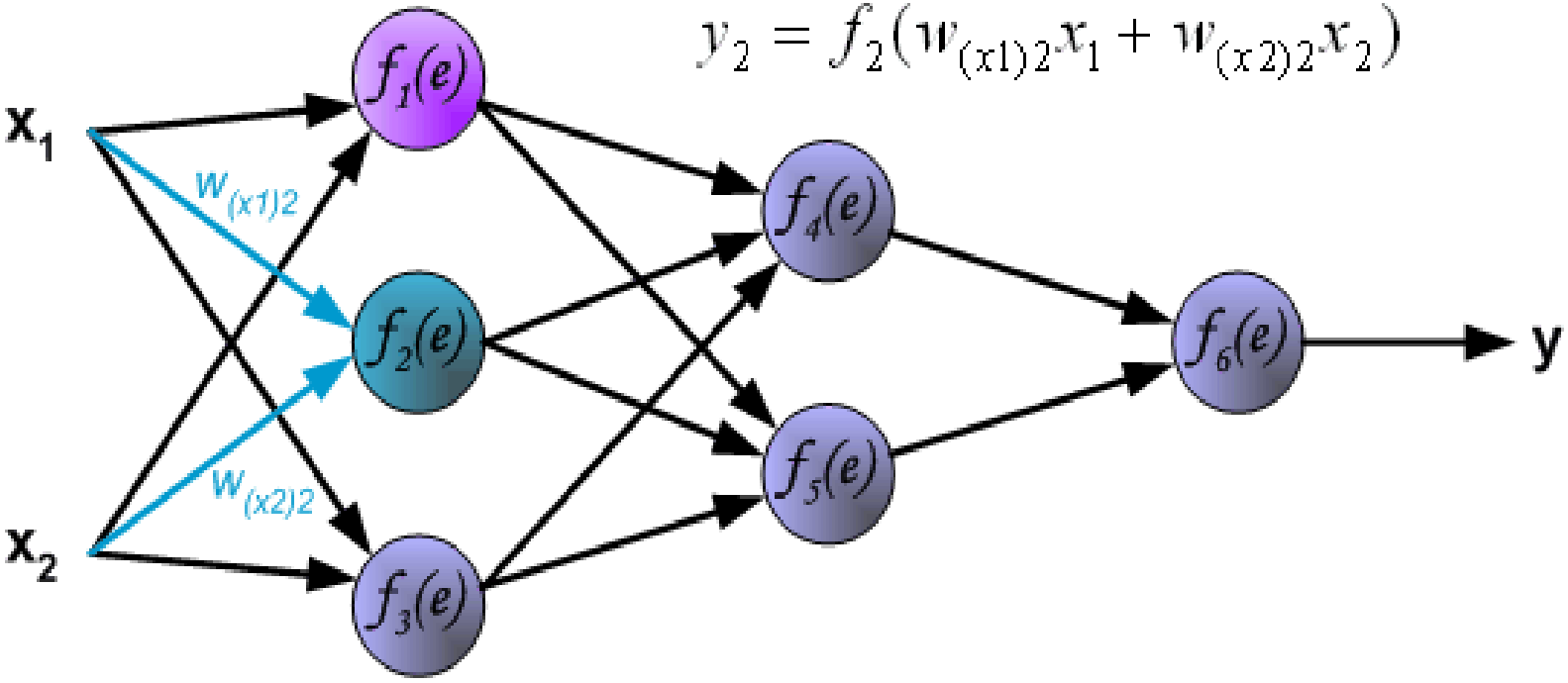
$$\frac{d}{dx} f(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x)(1-f(x))$$

$$S(x) = 2f(x) - 1 = \frac{1-e^{-x}}{1+e^{-x}}$$

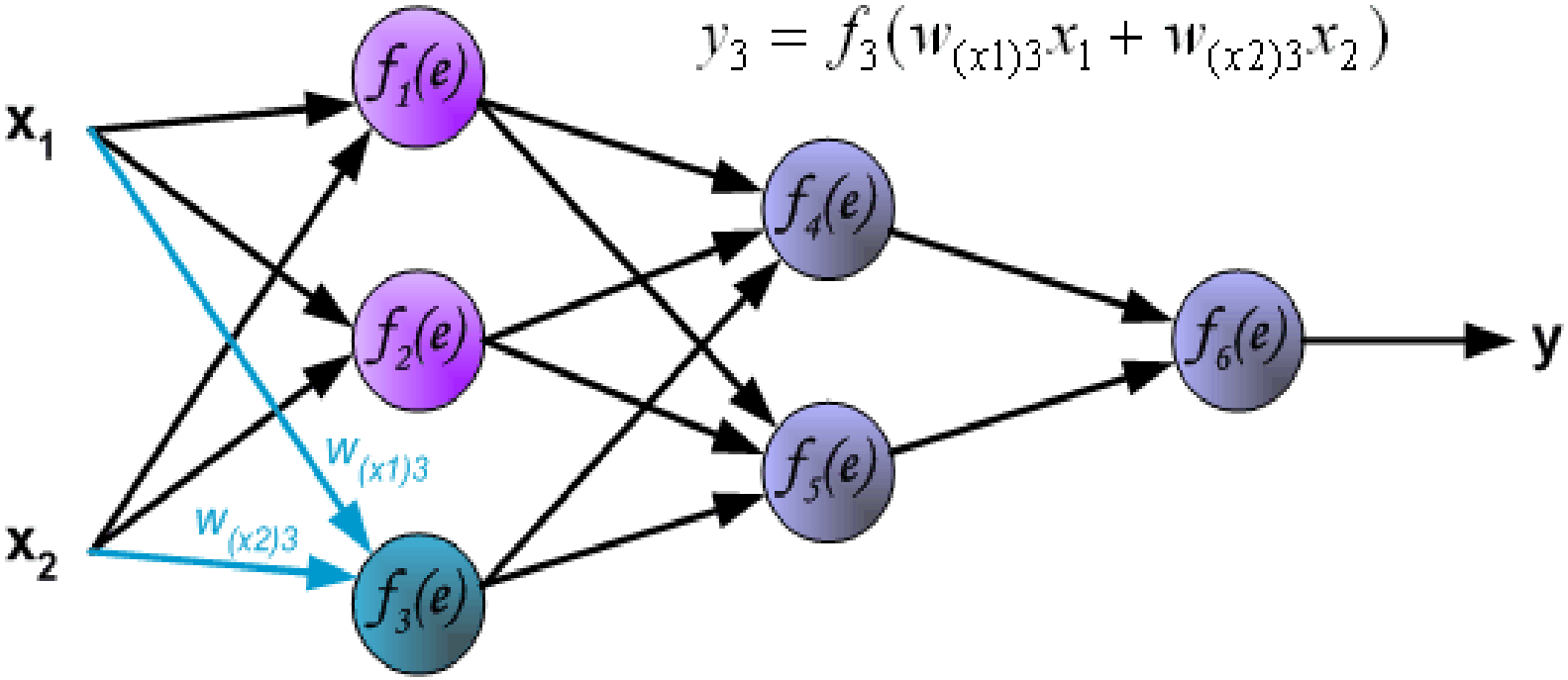
Primeira etapa: apresentação das entradas e cômputo das saídas da camada de entrada



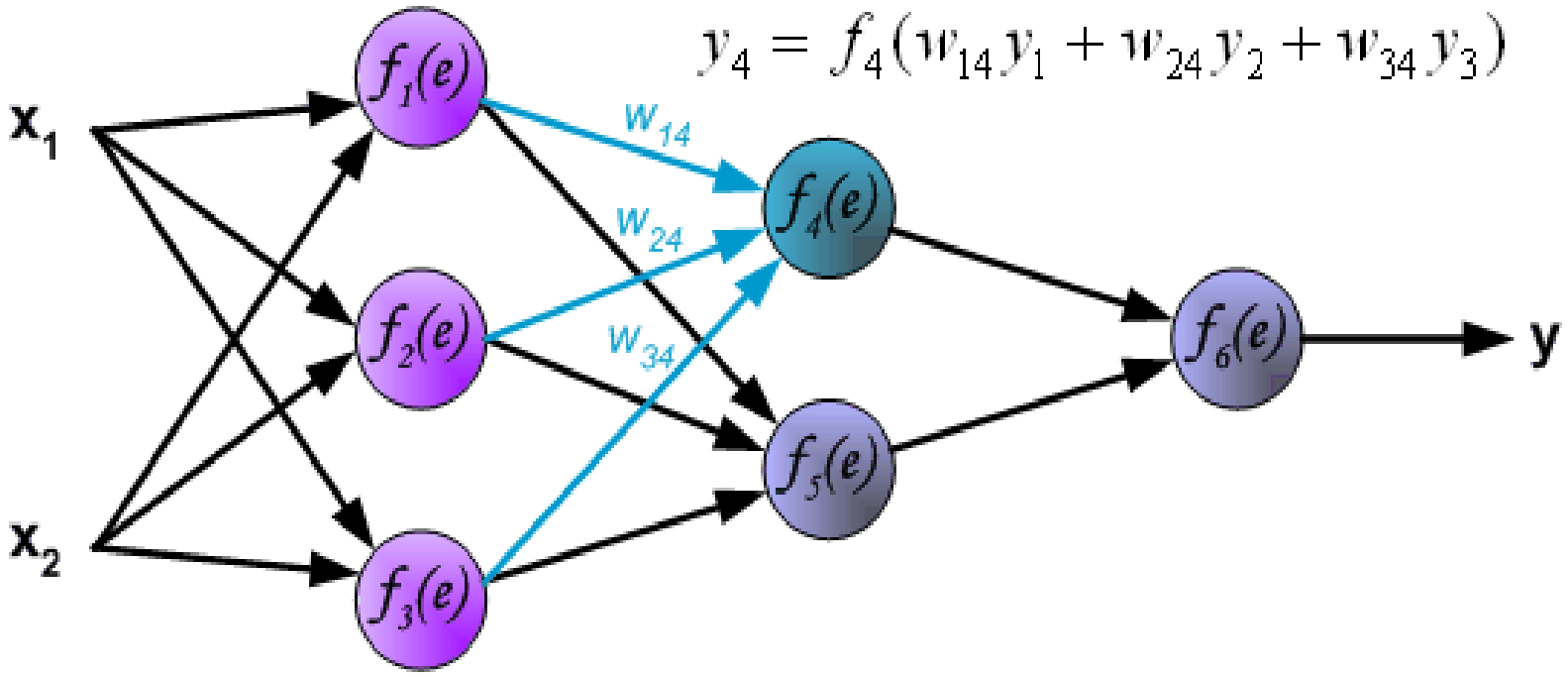
Primeira etapa: apresentação das entradas e cômputo das saídas da camada de entrada



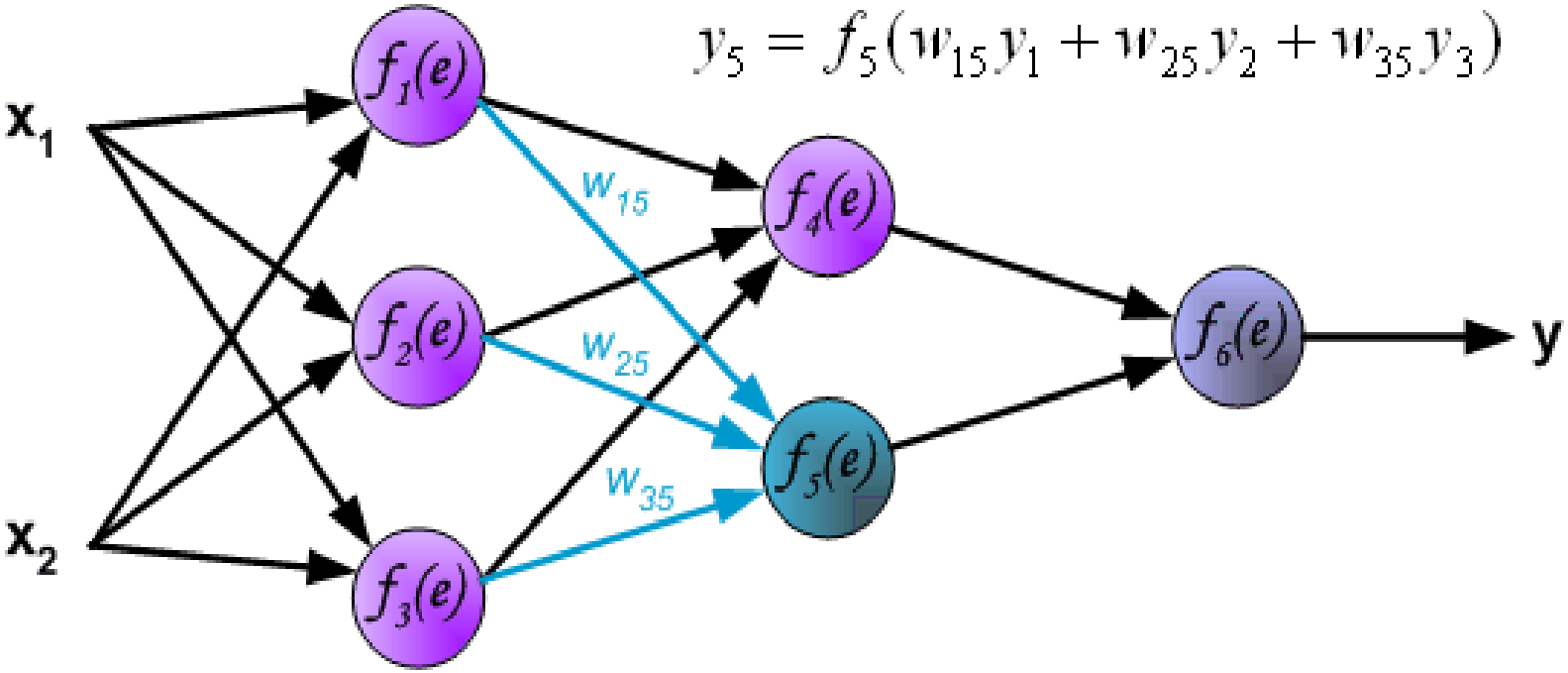
Primeira etapa: apresentação das entradas e cômputo das saídas da camada de entrada



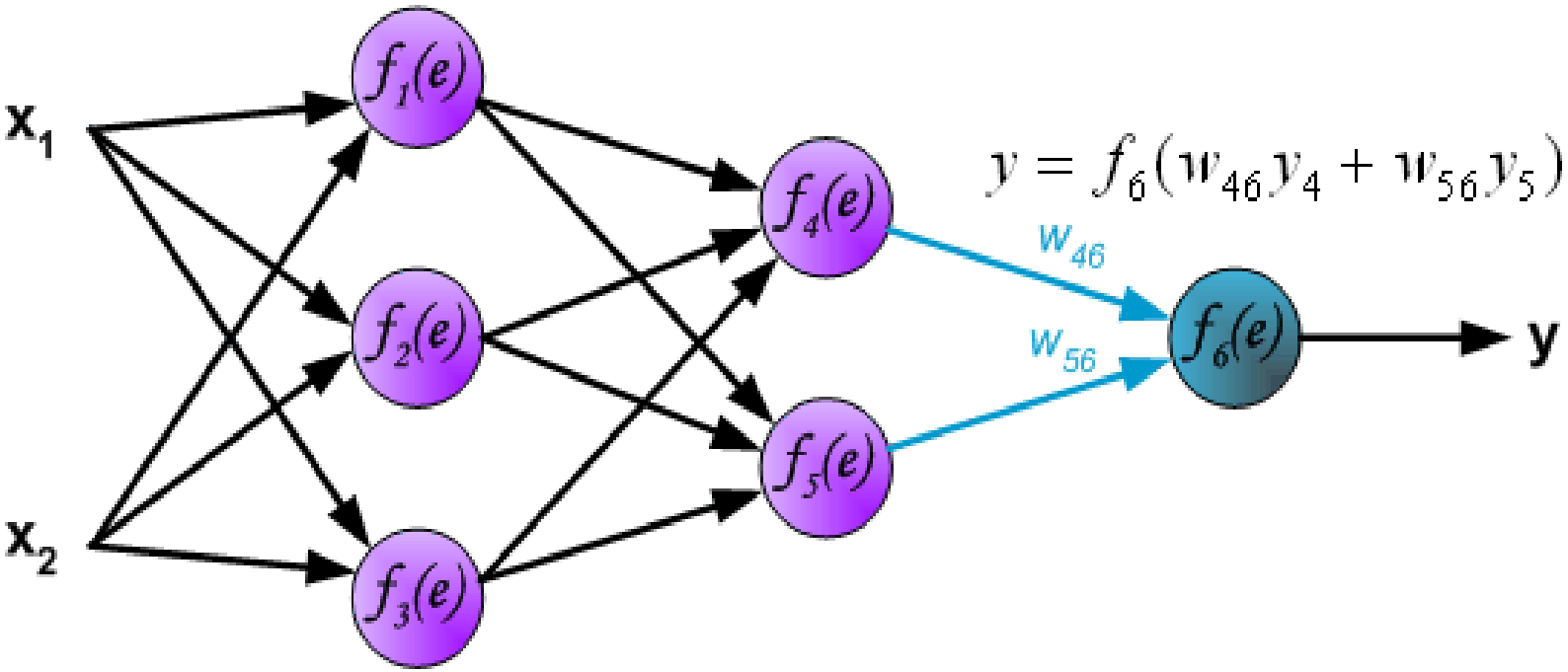
Primeira etapa: propagação dos sinais e cômputo das saídas da camada escondida



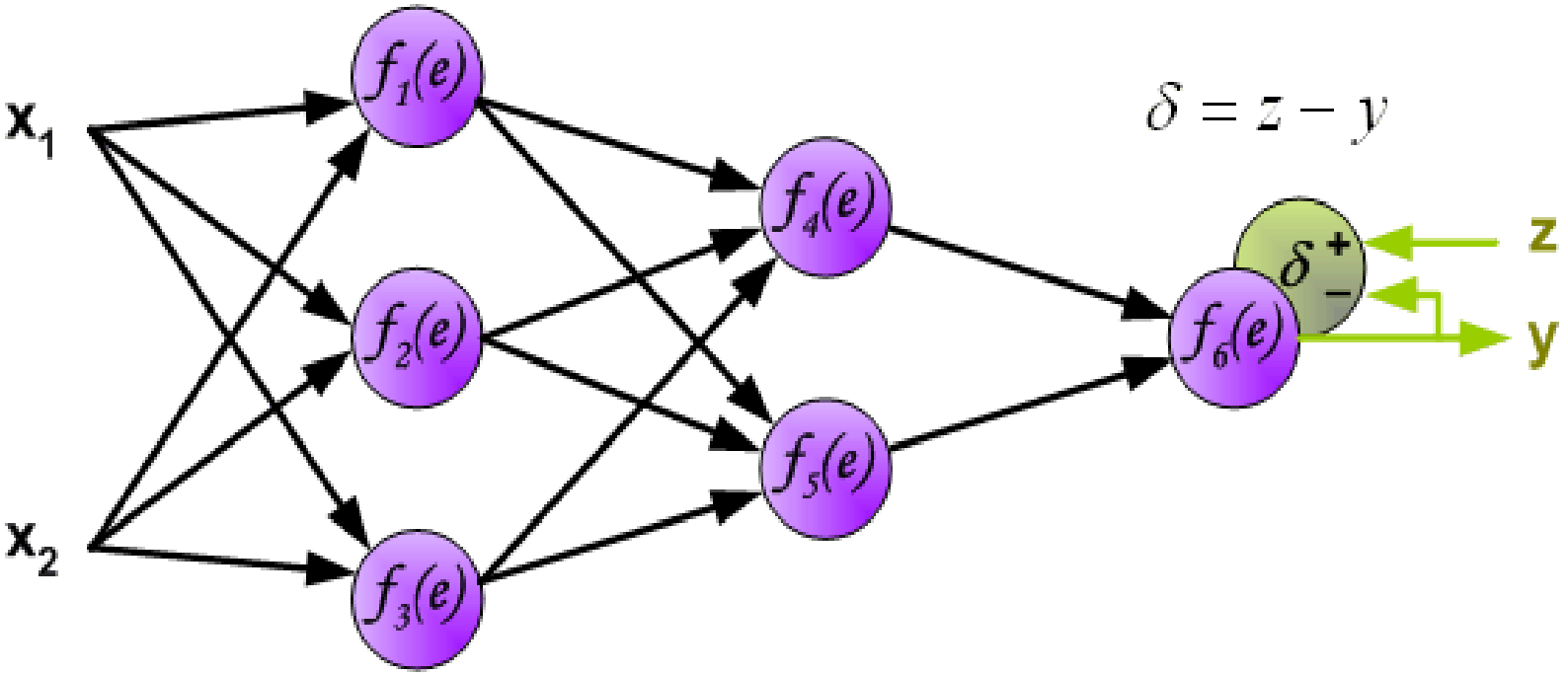
Primeira etapa: propagação dos sinais e cômputo das saídas da camada escondida



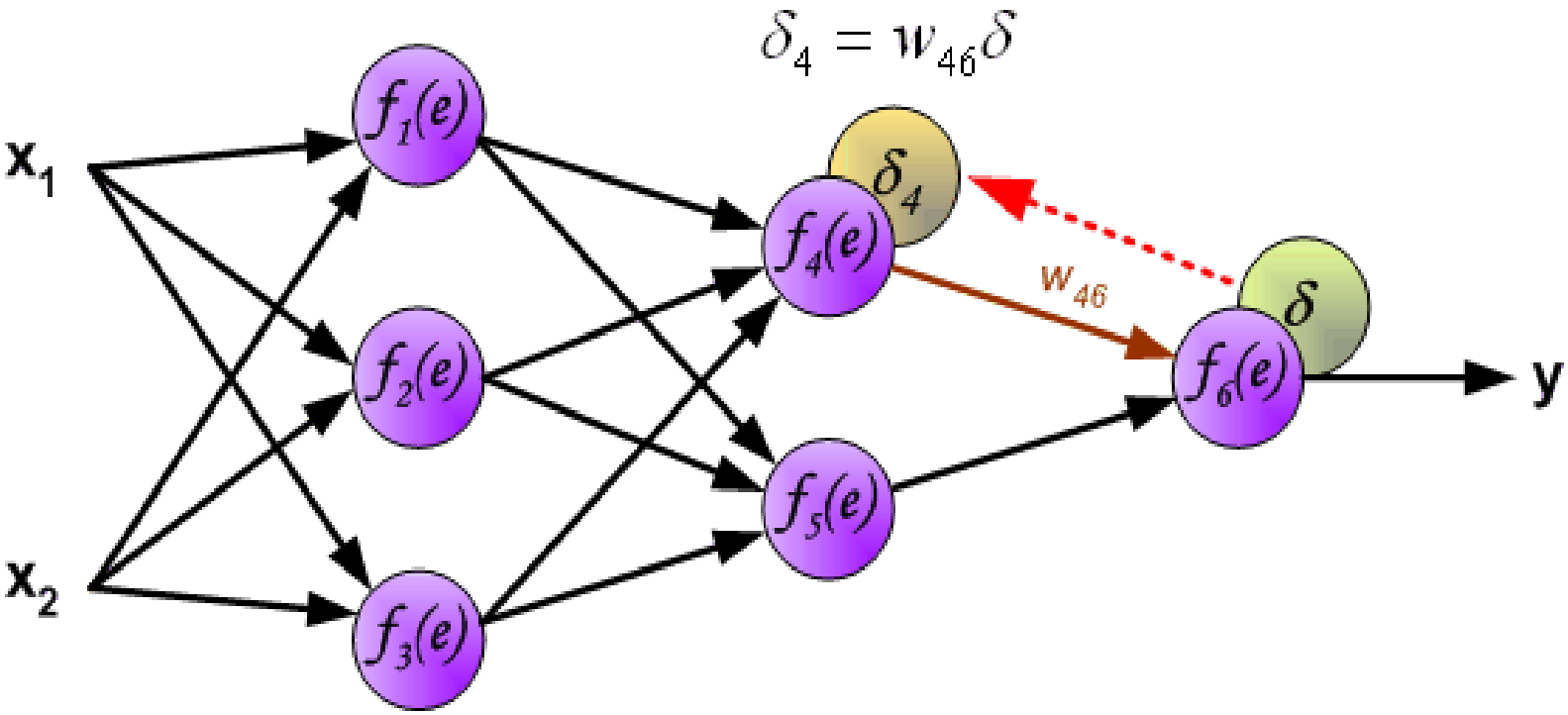
Primeira etapa: propagação dos sinais e cômputo da camada de saída



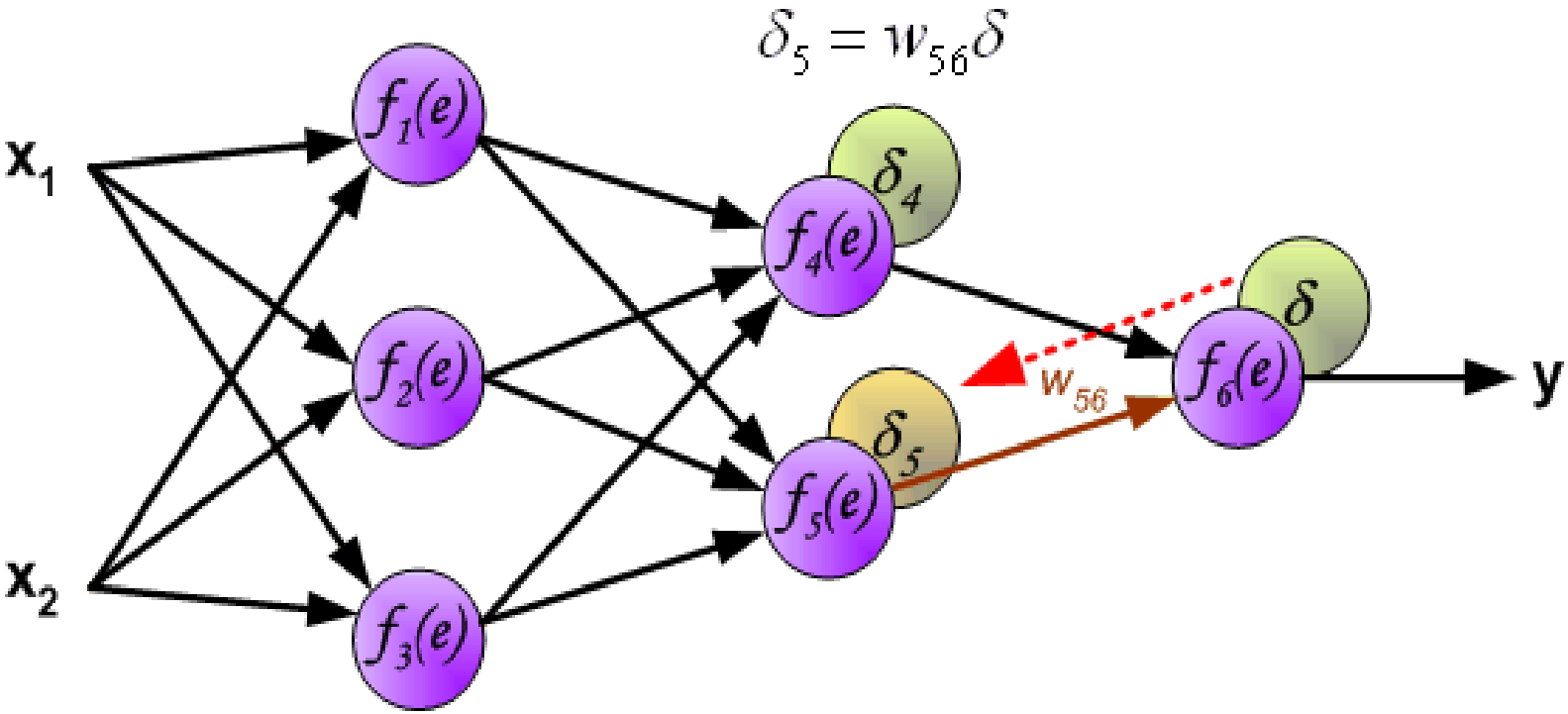
Primeira etapa: propagação dos sinais e cômputo do erro (desejado-obtido)



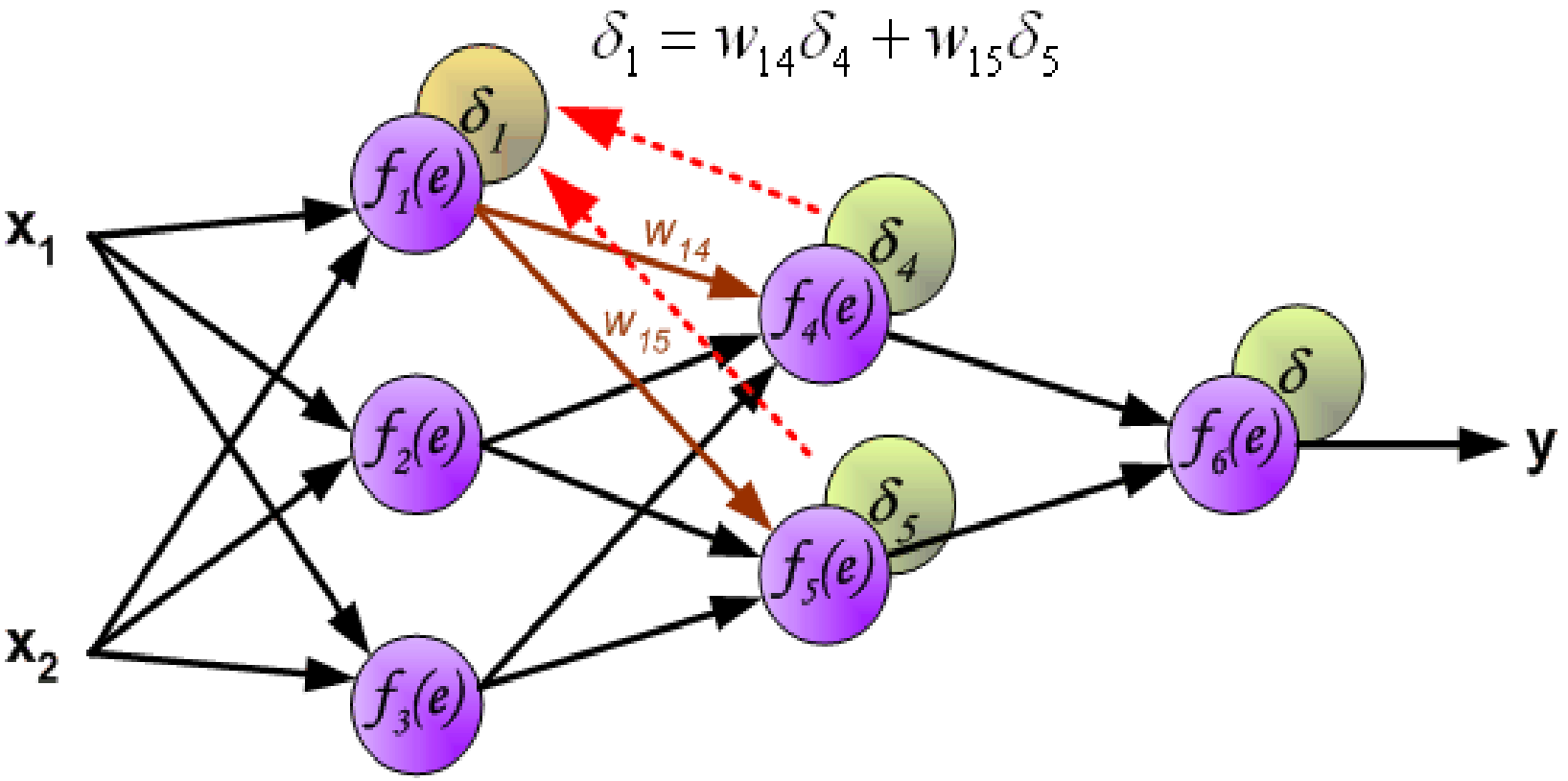
Segunda etapa: retro-propagação do erro para a camada escondida



Segunda etapa: retro-propagação do erro para a camada escondida

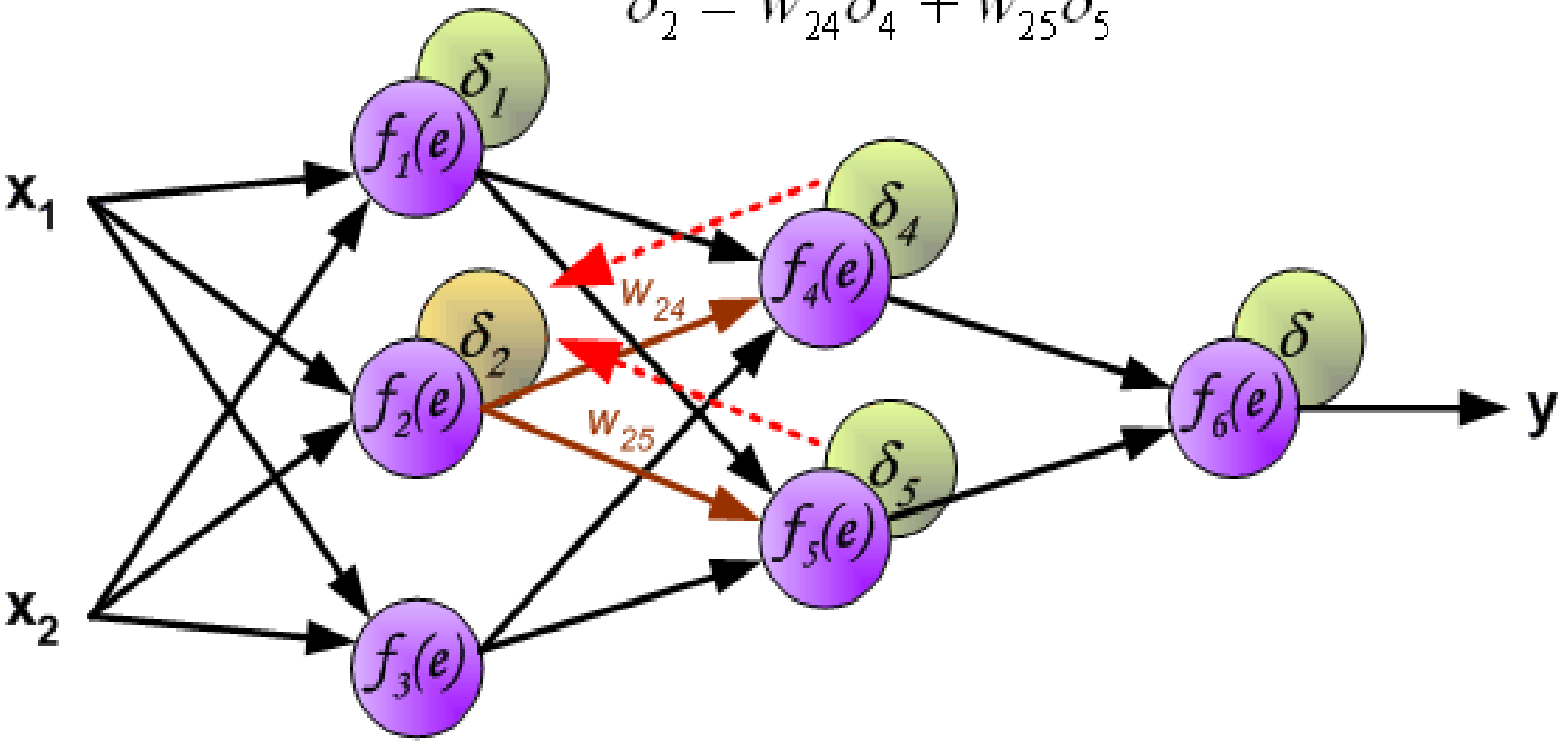


Segunda etapa: retro-propagação do erro para a camada de entrada

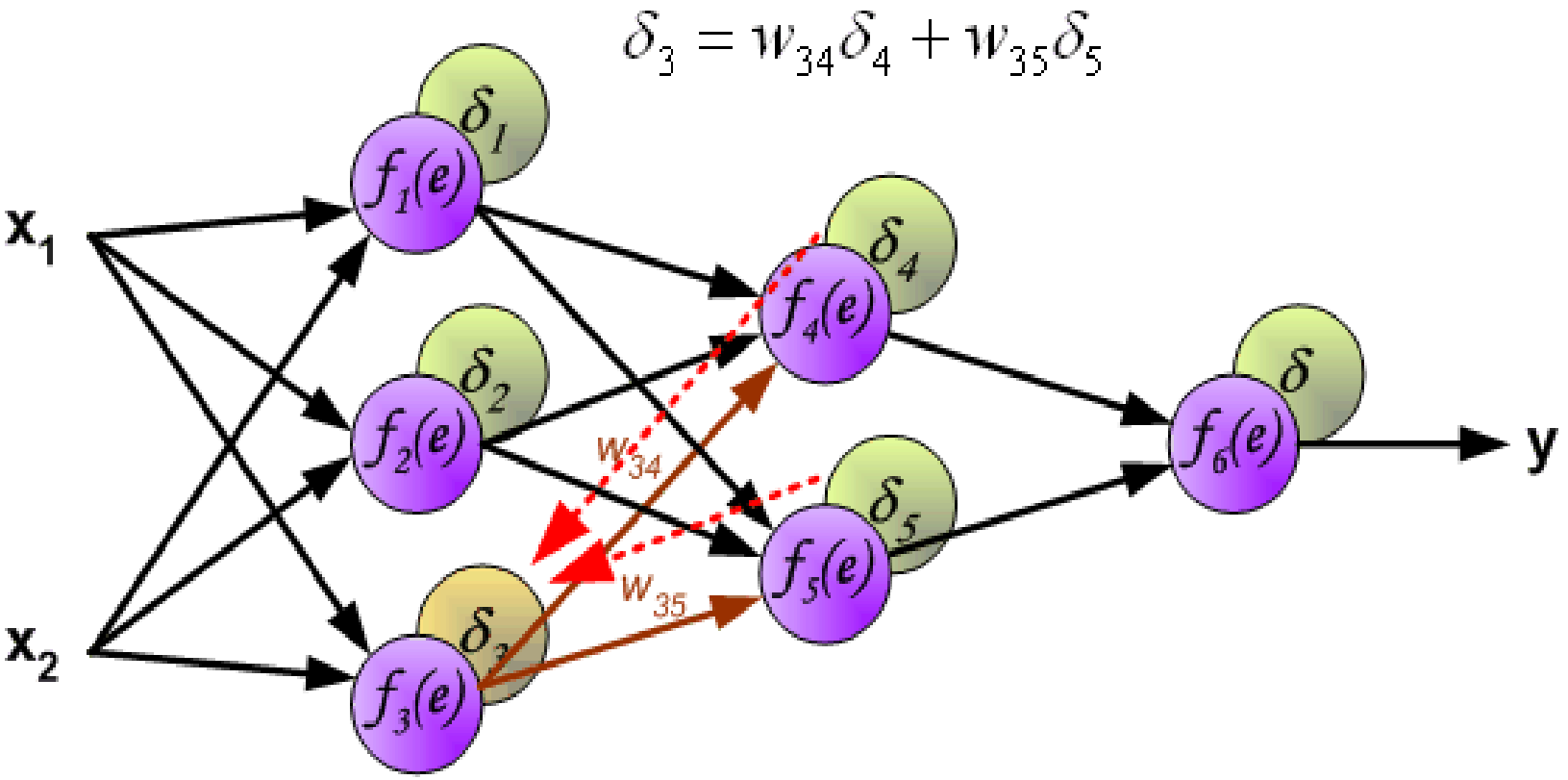


Segunda etapa: retro-propagação do erro para a camada de entrada

$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$



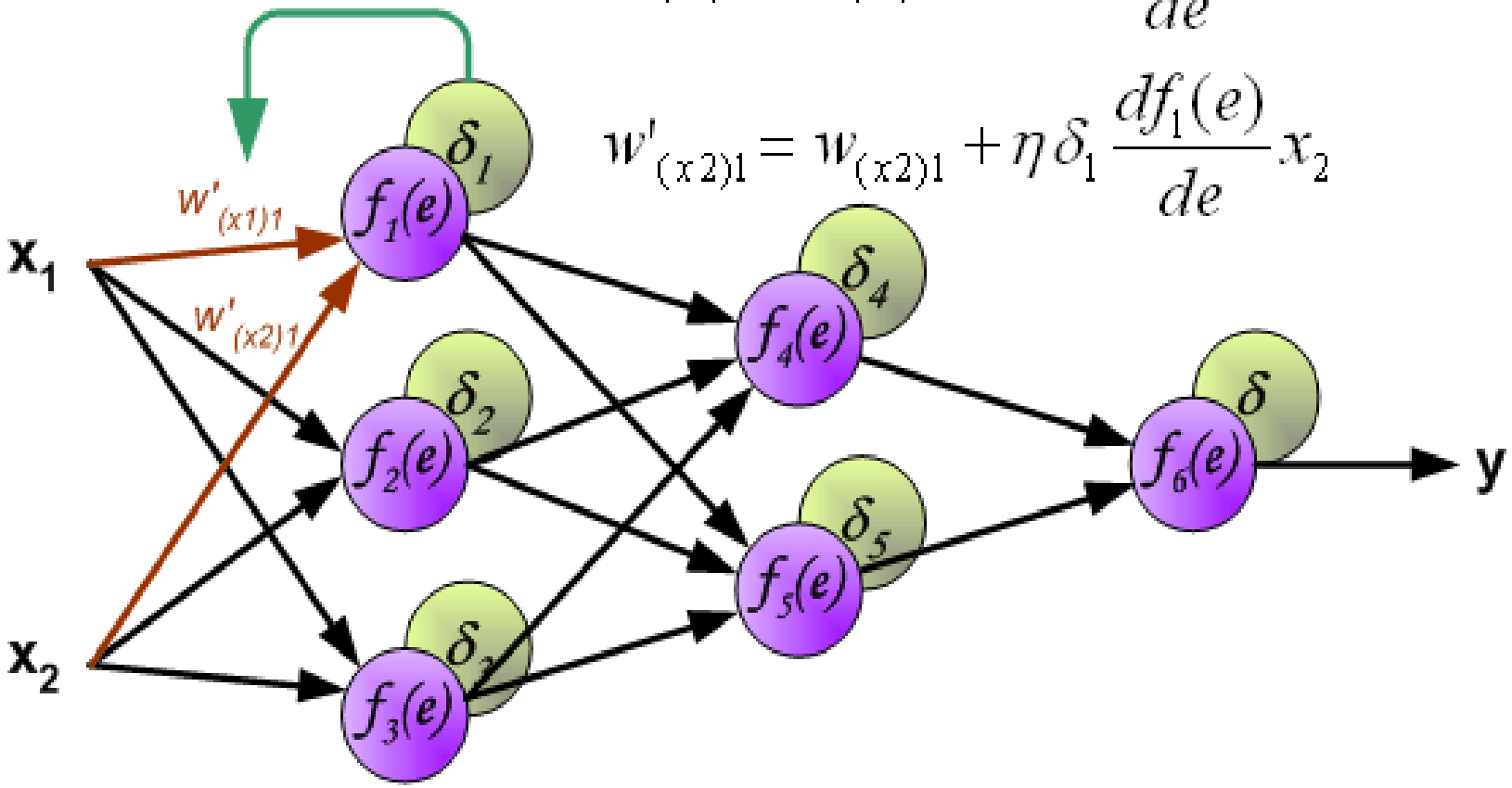
Segunda etapa: retro-propagação do erro para a camada de entrada



Terceira etapa: atualização dos pesos

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

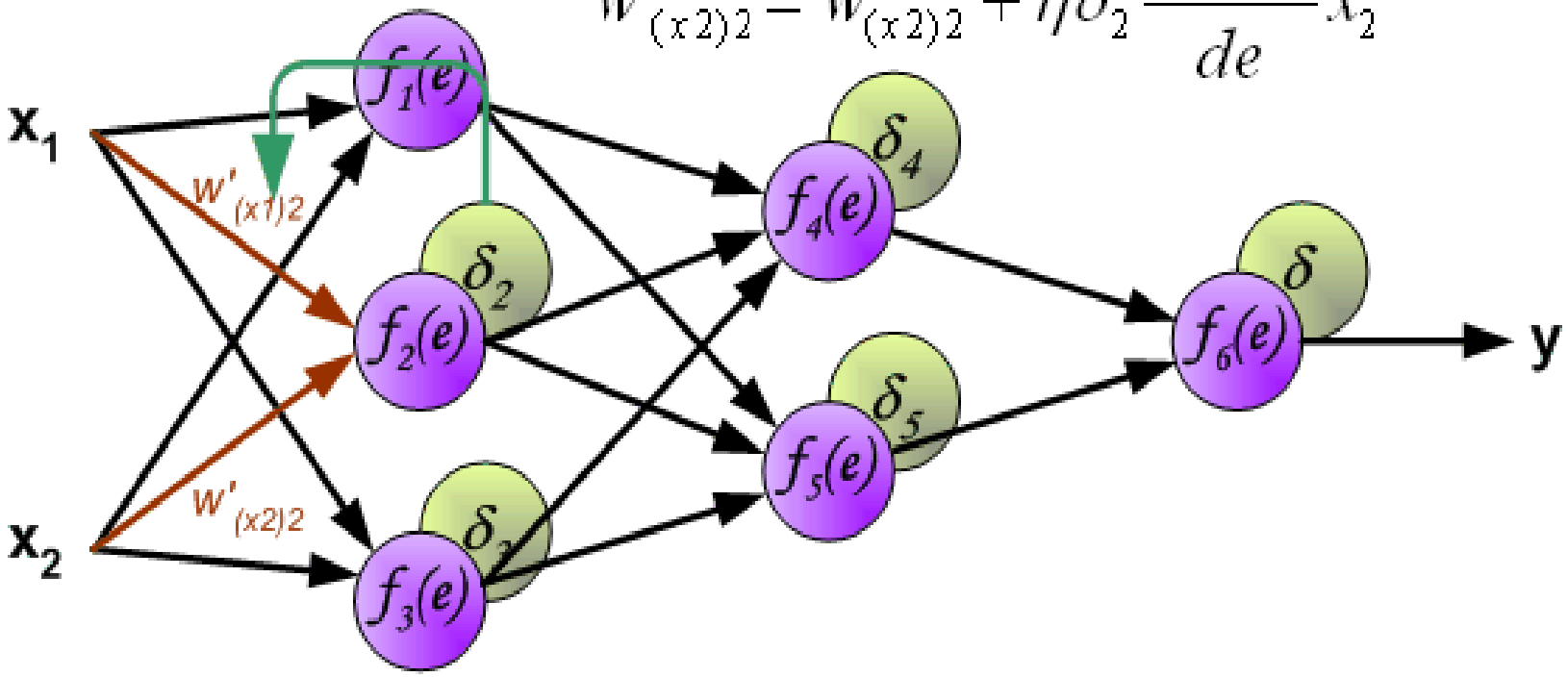
$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



Terceira etapa: atualização dos pesos

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

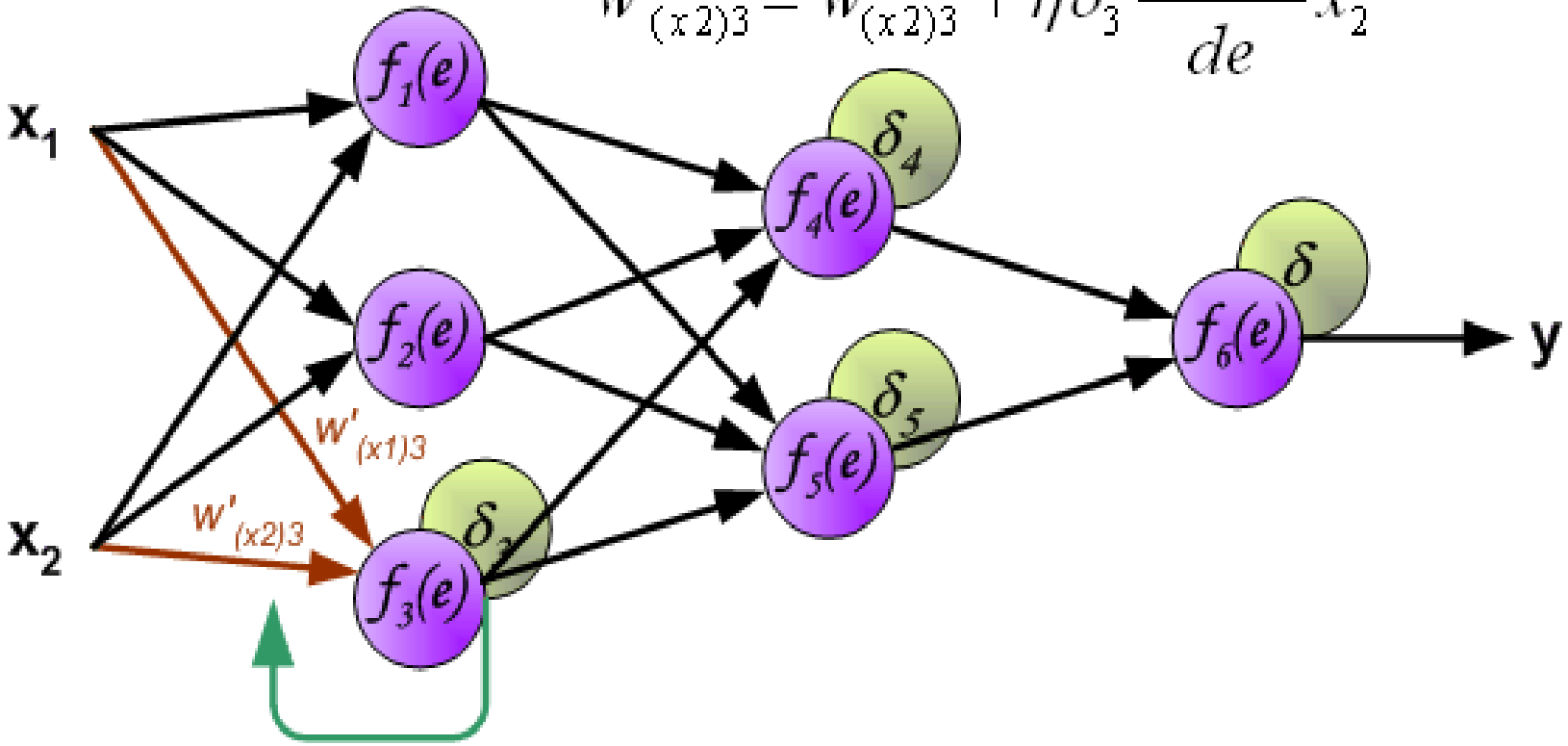
$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



Terceira etapa: atualização dos pesos

$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$

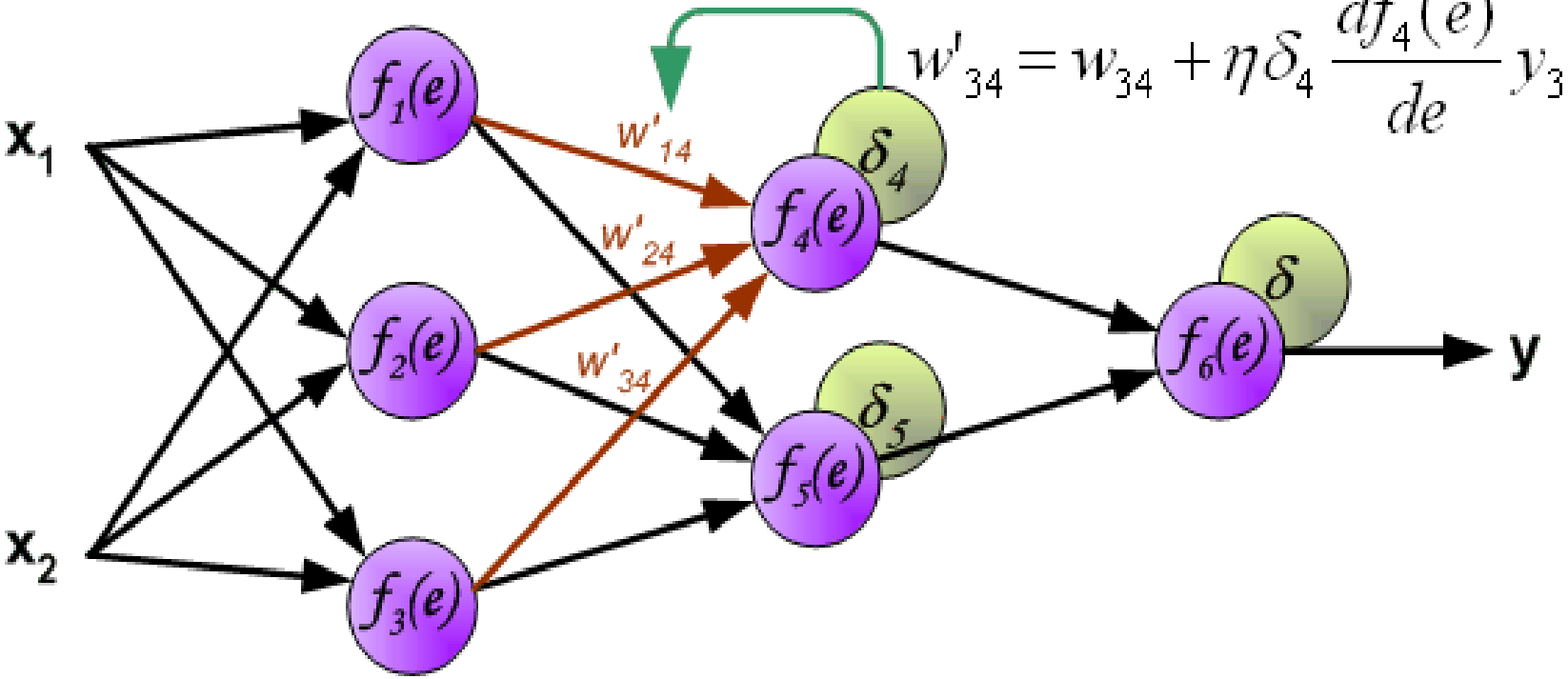


Terceira etapa: atualização dos pesos

$$w'_{14} = w_{14} + \eta \delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta \delta_4 \frac{df_4(e)}{de} y_3$$

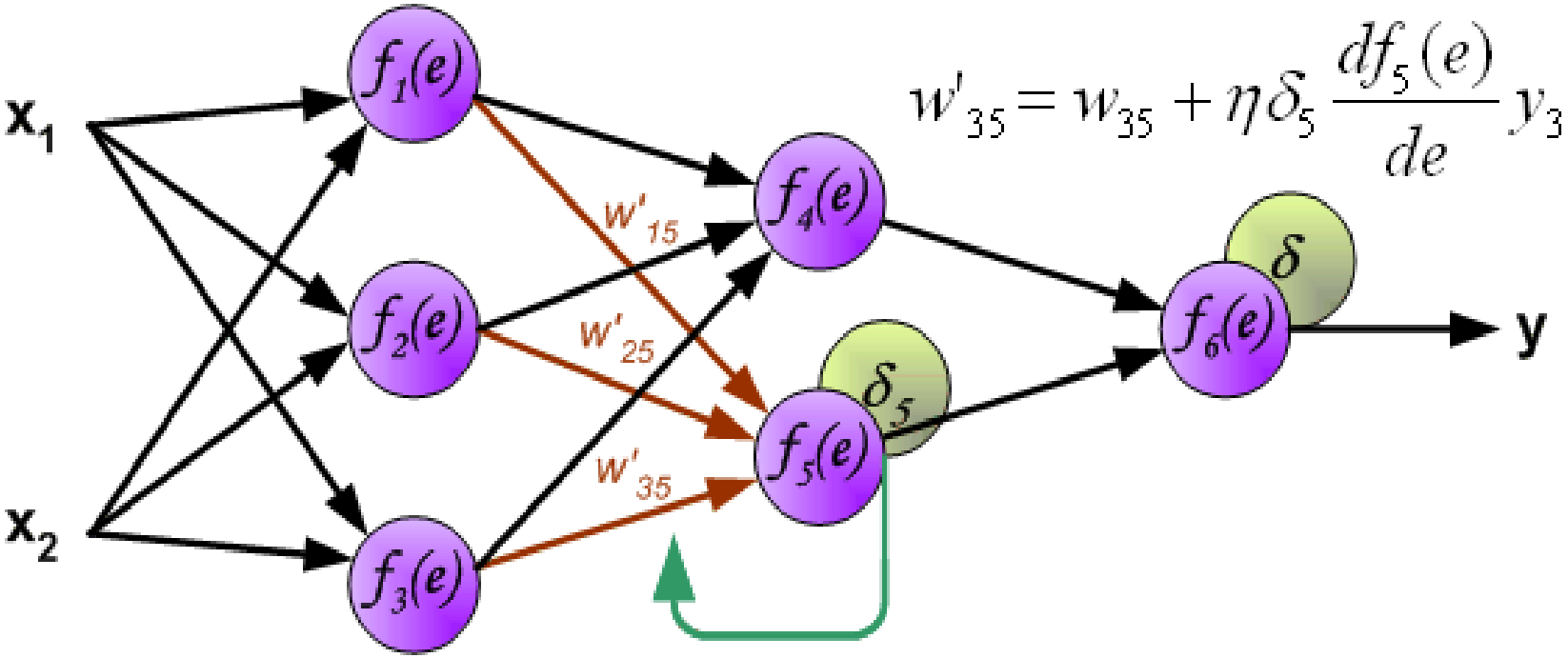


Terceira etapa: atualização dos pesos

$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta \delta_5 \frac{df_5(e)}{de} y_2$$

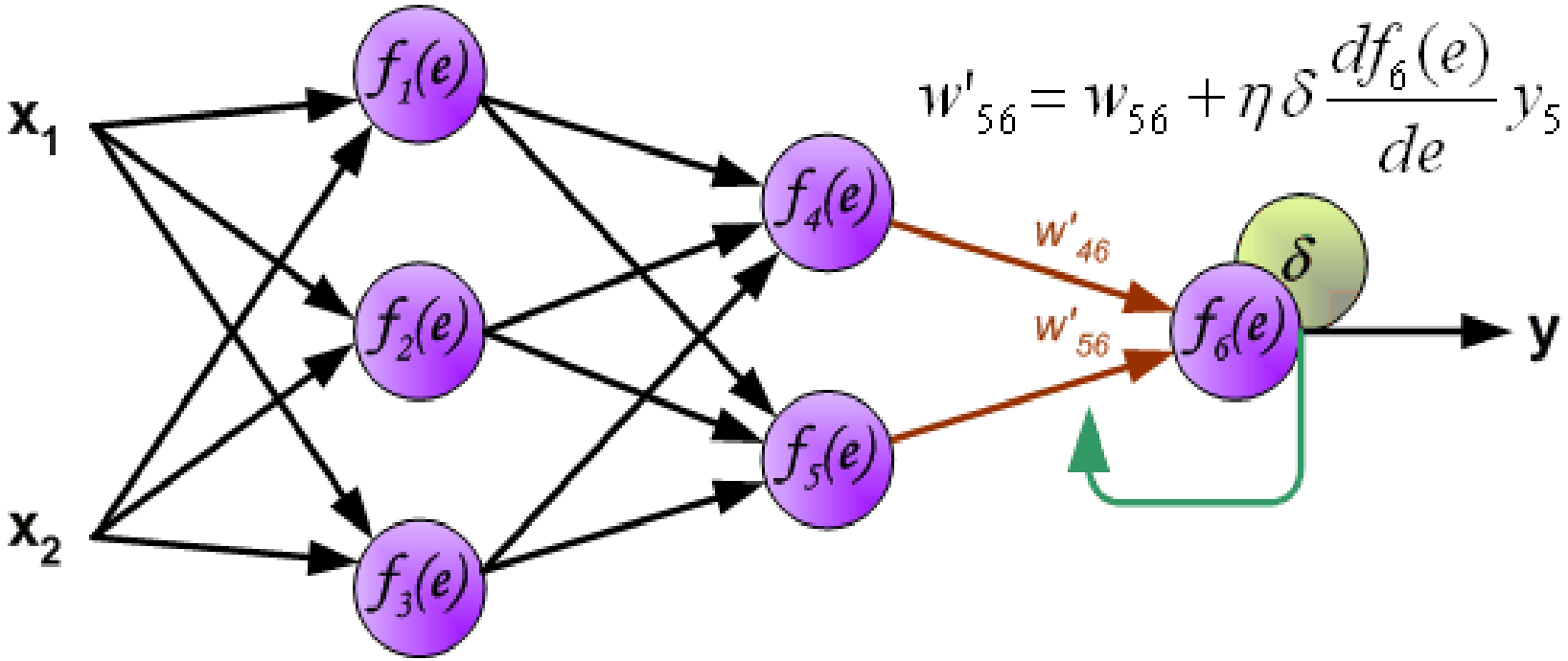
$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$



Terceira etapa: atualização dos pesos

$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

$$w'_{56} = w_{56} + \eta \delta \frac{df_6(e)}{de} y_5$$



Tarefa:

- Utilizando o código postado no moodle*, modifique os arquivos de treinamento para obter uma rede capaz de reproduzir o comportamento de uma porta lógica XOR (ou exclusivo).

- *Adaptado de C++ Neural Networks and Fuzzy Logic – Rao & Rao