Redes Perceptron de Uma ou Múltiplas Camadas

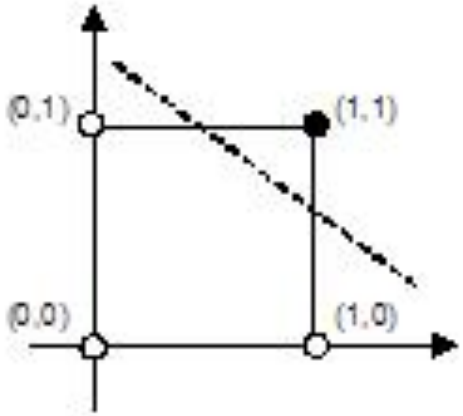
Computação Natural

Gisele L. Pappa

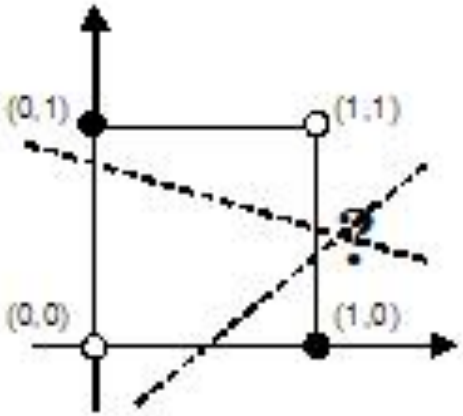
Perceptron de uma Camada

- Primeiro modelo para aprendizagem supervisionada
- Padrões linearmente separáveis

Inputs		Output
x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1



Inputs		Output
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0



Aprendizagem

- Seja $w(t)$ um peso sináptico de um dado neurônio, no instante de tempo t. O ajuste $\Delta w(t)$ é aplicado ao peso sináptico $w(t)$ no instante t, gerando o valor corrigido $w(t+1)$, na forma:

$$w(t+1) = w(t) + \Delta w(t)$$

- Várias maneira de obter $\Delta w(t)$:
  - regra de Hebb, regra Delta, algoritmo de *backpropagation*, estratégias de competição, máquina de Boltzmann

Aprendizado no perceptron

- Pode seguir 2 abordagens:
  - Regra do Perceptron
  - Regra do Adeline *(Adaptive Linear Neuron)*

Intuição da Regra de Aprendizado

- Dada uma instância (x,y)
  - Se o erro é positivo (saída real é maior que desejada)
    - Quero aumentar $w_k x_k$
  - Se o erro é negativo (saída real é menor que desejada)
    - Quero diminuir $w_k x_k$
- Se não exite erro, não muda pesos

Regra do Perceptron

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha\, e_i\, \mathrm{x} \quad \text{(peso)}$$

$$\mathbf{b}(t+1) = \mathbf{b}(t) + \alpha\, e_i \quad \textit{(bias)}$$

α é a taxa de aprendizagem

e_i é o erro entre a saída encontrada e a desejada

- No perceptron de uma camada, a taxa de aprendizagem têm pouco impacto e pode ser igual a 1

Treinamento Perceptron

Inicializa rede com pesos arbitrários [-0.5,0.5]

Repita (até convergência dos pesos)

Para cada instância (x,y)

Para cada peso w_k

$$\mathbf{w_k}(t+1) = \mathbf{w_k}(t) + \alpha\, e_i\, x_k$$

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	–1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	–1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	–1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Regra do Adeline

- Define uma função de erro
- Escolhe os pesos para minimizar essa função de erro
  - Através do método da descida do gradiente

Descida do gradiente

- Algoritmo de otimização – pode ser usado para encontrar o mínimo local de uma função
- Dada uma função diferenciável definida na vizinhança de um ponto *a*, sabe-se que essa função cai mais rápido se ela for de *a* em direção a negativa do gradiente

Descida do gradiente

- Gradiente: direção para onde a "função cresce"

$$= \nabla f(\mathbf{w})$$

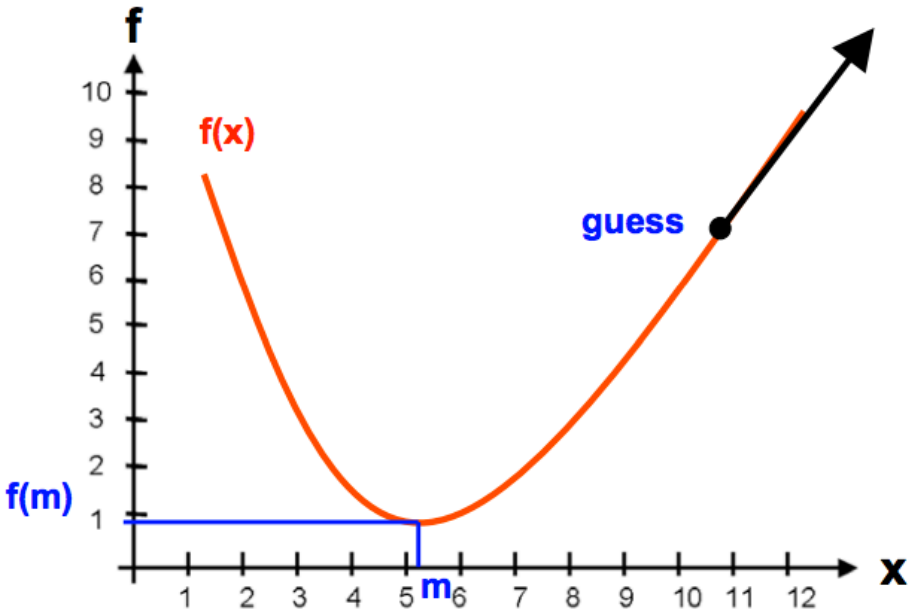
$$= \left[\frac{\partial f}{\partial w_0}(\mathbf{w}), \ldots, \frac{\partial f}{\partial w_n}(\mathbf{w}) \right]$$

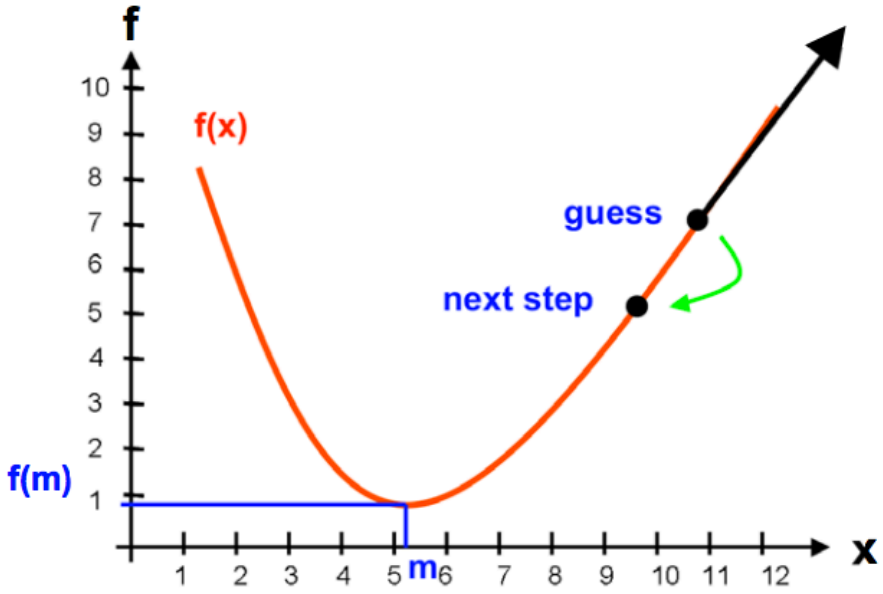
- Gradiente negativo: para onde a função decresce
- Regra de atualização:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla f(\mathbf{w})$$

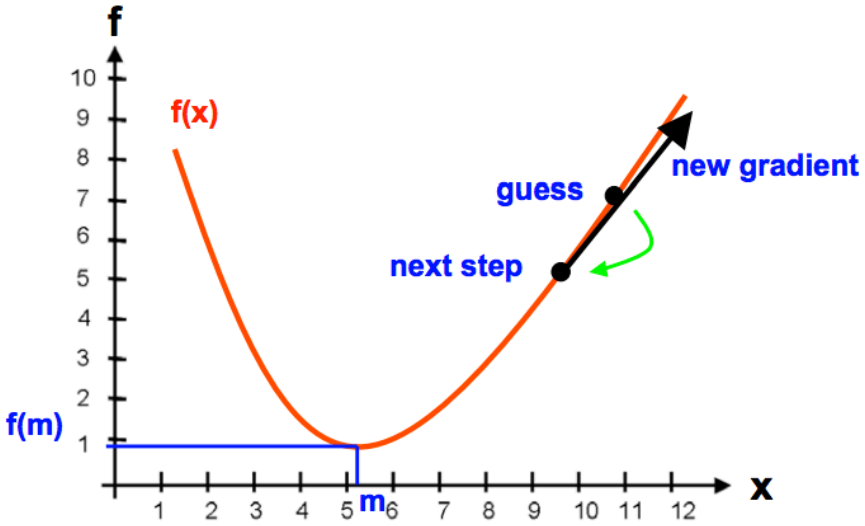
$$w_j \leftarrow w_j - \alpha \frac{\partial f}{\partial w_j}$$

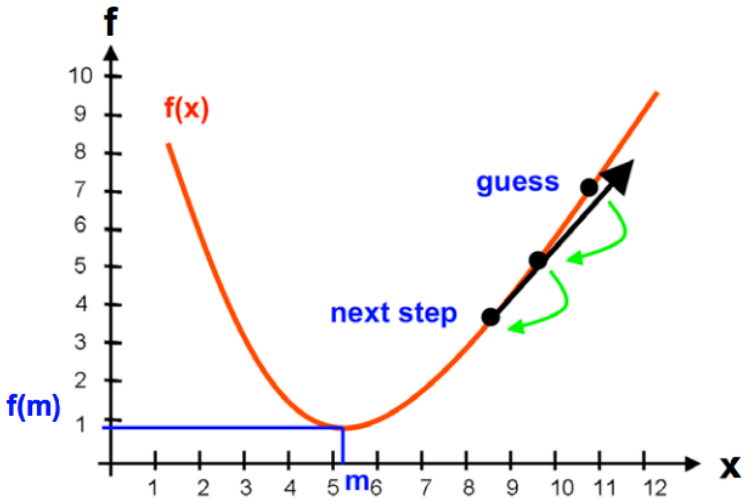
Descida do Gradiente

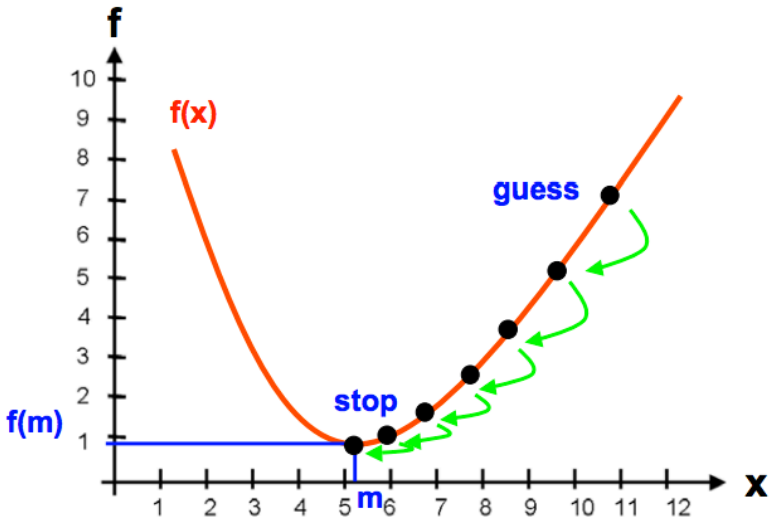




Descida do Gradiente







Escolha da função de erro

- Primeira escolha:
  - Soma dos quadrados dos erros
  - Não diferenciável
- Opção: usar a entrada para avaliar indiretamente a saída – entrada deve estar próxima da saída real

$$E(w) = \frac{1}{2}\sum_{d}(y^d - in^d)^2$$

d é o número de exemplos no treinamento
in é a entrada do neurônio

$$= \frac{1}{2}\sum_{d}\left(y^d - \sum_{j=0}^{n} w_j x_j^d\right)^2$$

Perceptron vs Adaline

- Perceptron:
  - Garante convergência quando os pontos são linearmente separáveis (vantagem)
  - Não garante convergência para mínimo local quando pontos não são linearmente separáveis (desvantagem)
  - Pontos classificados corretamente não influenciam no treino (vantagem)
  - Pontos classificados incorretamente têm a mesma influência no treino (desvantagem)

Perceptron vs Adaline

- Adeline:
  - Não garante convergência quando os pontos são linearmente separáveis (desvantagem)
  - Garante convergência para mínimo local quando pontos não são linearmente separáveis (vantagem)
  - Pontos classificados corretamente influenciam no treino (desvantagem).
  - Pontos classificados incorretamente não têm a mesma influência no treino (vantagem)

Treinamento do Perceptron

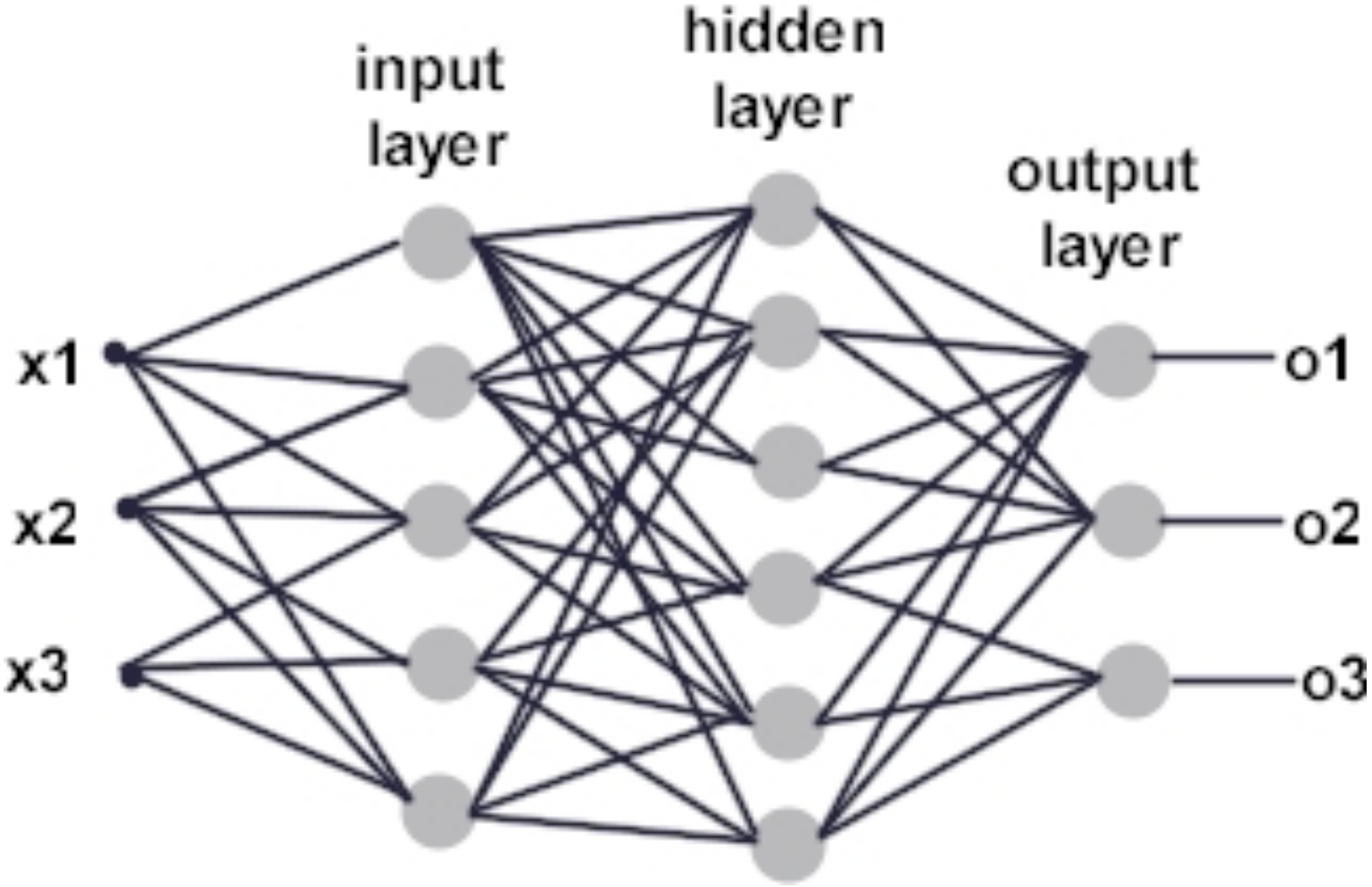
- Diferentes conjuntos iniciais de pesos para o perceptron podem levar a diferentes superfícies de decisão.
  - Na verdade, o problema de ajuste supervisionado de pesos pode ser visto como um processo de busca por um conjunto de pesos que otimizam uma determinada superfície de erro.
  - Uma escolha inadequada da condição inicial da rede pode levar o algoritmo a uma convergência para ótimos locais desta superfície de erro.

Treinamento do Perceptron

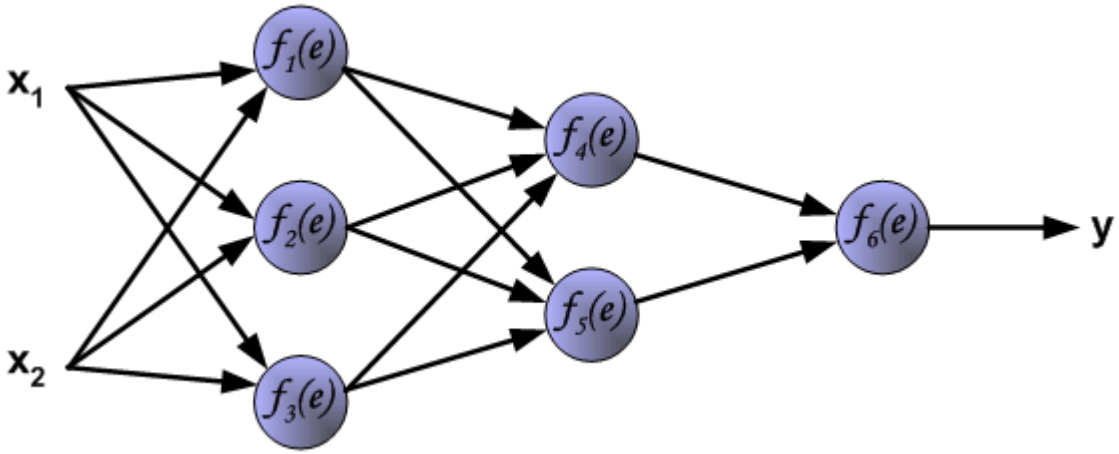
- **Parâmetros de treinamento**
  - Taxa de aprendizagem α
- **Treinamento versus aplicação da rede**
  - Diferenciar entre o processo de treinamento e aplicação da rede.
  - O treinamento da rede corresponde ao processo de ajuste de pesos.
  - Após treinada, verificar a qualidade do aprendizado para verificar sua *capacidade de generalização*.

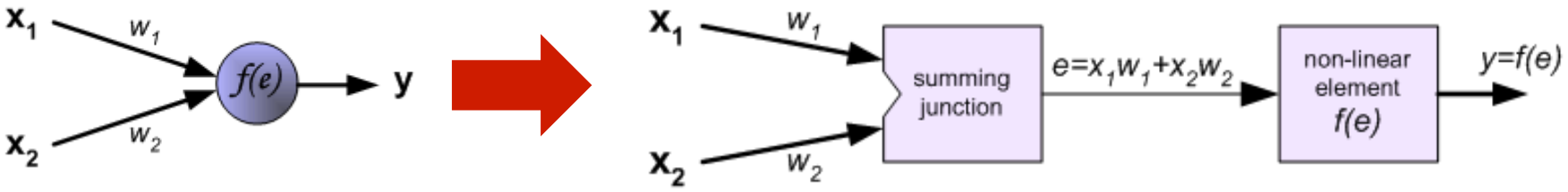
Multi-layer Perceptron (MLP)

Multi-layer Perceptron (MLP)



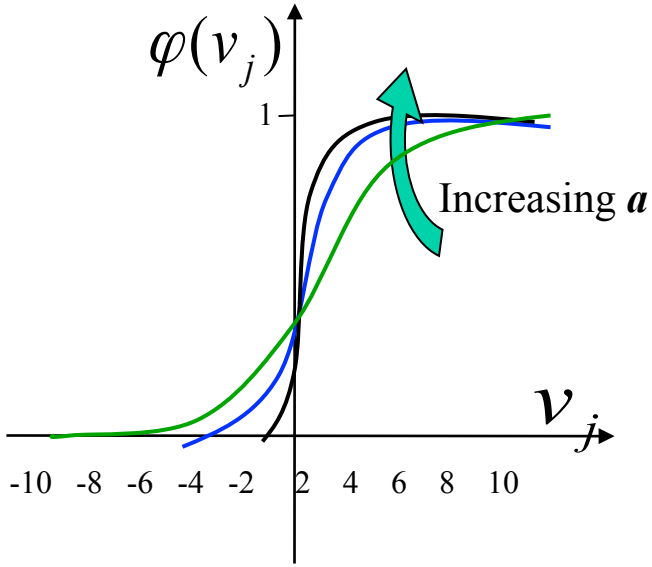
Modelo de Neurônio





Modelo de Neurônio

- Sigmoide



$$\varphi(\mathrm{v_j}) = \frac{1}{1+e^{-av_j}}$$

$$\mathrm{v_j} = \sum_{i=0,\ldots,m} w_{\mathrm{ji}} y_{\mathrm{i}}$$

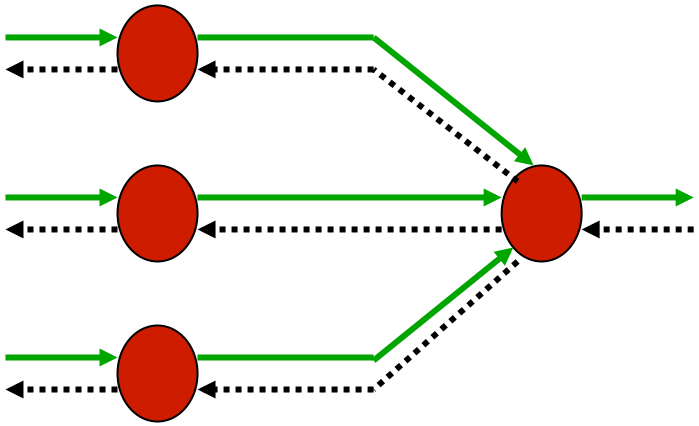
- Função de ativação mais comum
- Diferenciável

Algoritmo *back-propagation*

- Algoritmo mais utilizado
- Com uma camada escondida e uma função sigmoide, é capaz de aproximar qualquer função contínua $f : [-1,+1]^k \rightarrow [0,1]$
- Porém, o número de neurônios na camada escondida pode ser muito grande

Algoritmo de Aprendizagem

- *Back-propagation* (generalização do LMS*)*



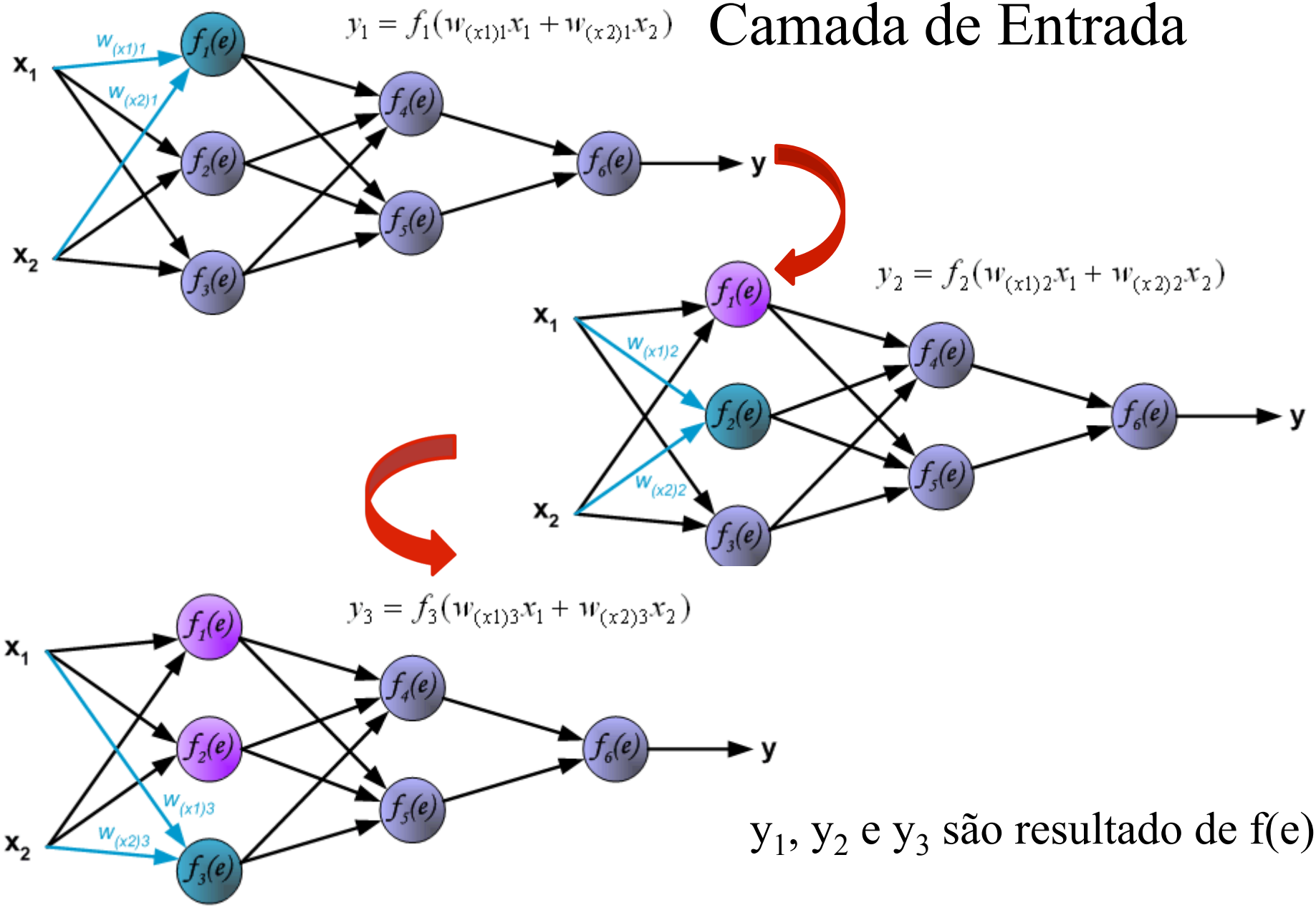
Os sinais das funções são enviados em uma direção

Os sinais de erro são enviados na direção contrária

- Ajusta o peso da rede visando minimizar o erro médio quadrado.

Propagação para frente

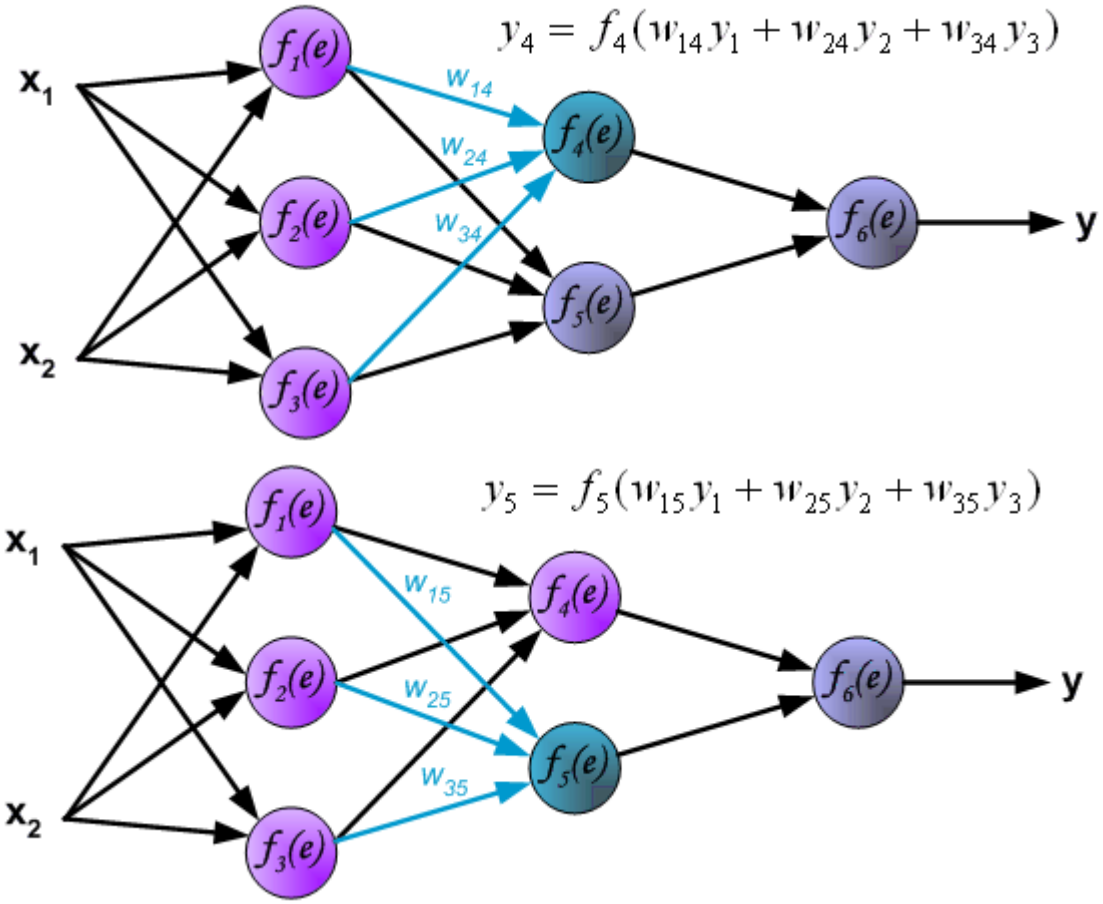
Camada de Entrada



y_1, y_2 e y_3 são resultado de f(e)

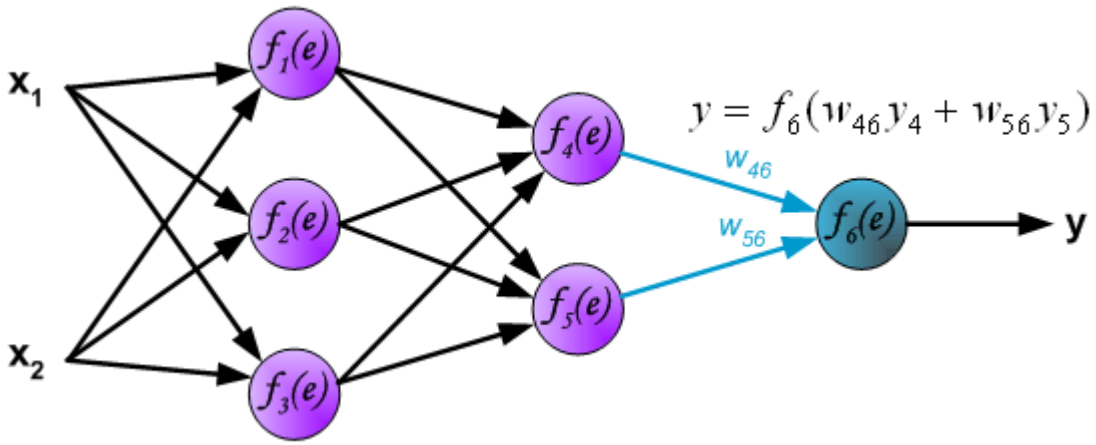
Propagação para frente

Camada Escondida



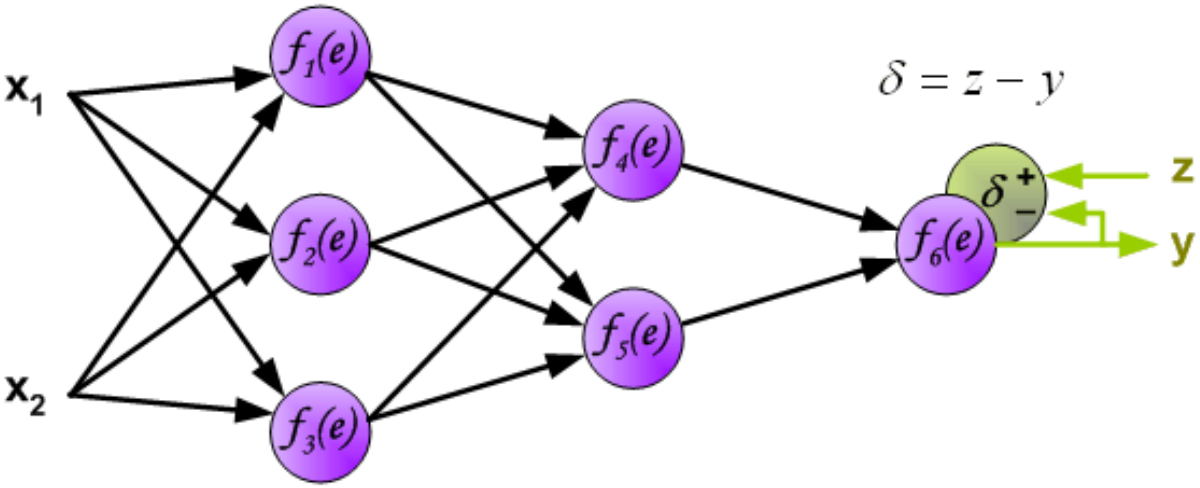
Propagação para frente

Camada de Saída



Propagação para frente

- z é a saída esperada



Aprendizado

- Perceptron
  - Erros ajustados de acordo com a saída real e a esperada
  - Para a camada de saída, sabe-se o erro esperado
  - Mas e para as camadas escondidas?
  - Ajusta os pesos para reduzir esse erro

- Como: descida do gradiente

Usando o Gradiente

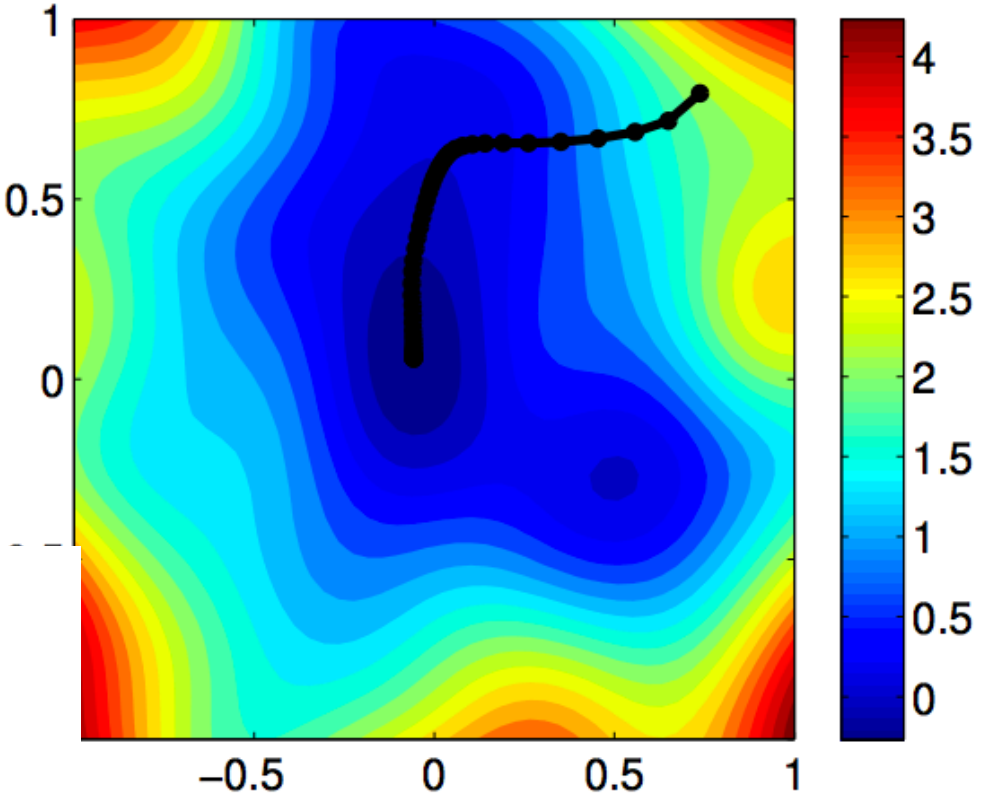
$$\nabla_{\boldsymbol{w}} f(\boldsymbol{w}) = \left[\frac{\partial}{\partial w_1} f(\boldsymbol{w}), \frac{\partial}{\partial w_2} f(\boldsymbol{w}), \cdots, \frac{\partial}{\partial w_K} f(\boldsymbol{w})\right]^{\mathsf{T}}$$

- Mover os pesos em direção a descida mais rápida

- Regra de atualização:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla \mathrm{f}(\mathbf{w})$$

$$\mathrm{w_j} \leftarrow \mathrm{w_j} - \alpha \frac{\partial \mathrm{f}}{\partial \mathrm{w_j}}$$



Erro Médio Quadrado

- Sinal de erro de um neurônio *j* na camada de saída na iteração *n* *(i.e. apresentação do n-ésimo exemplo)*

- Erro na iteração *n*:

$$e_j(n) = d_j(n) - y_j(n)$$

- Erro quadrático médio:

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

C: conjunto de neurônios na camada de saída

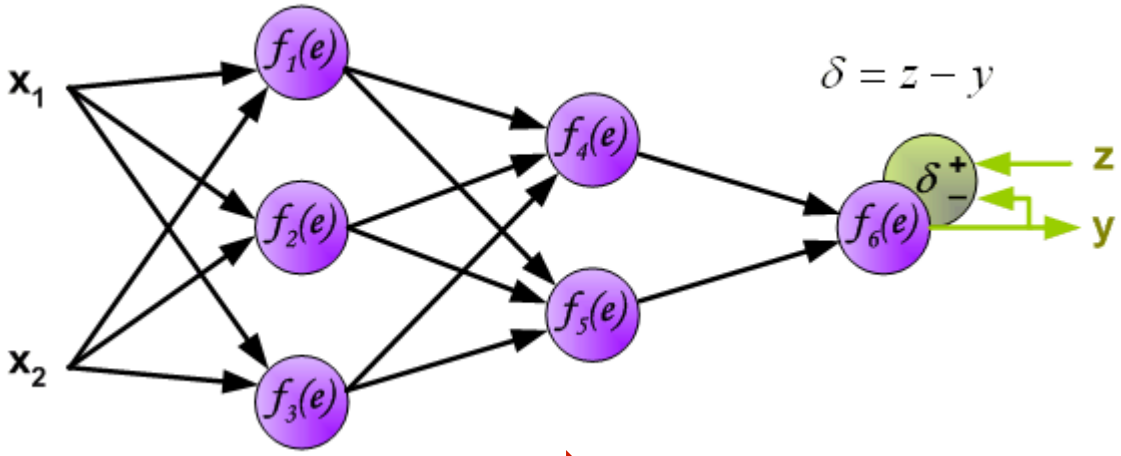
- Métrica de performance de aprendizagem:

$$E_{AV} = \frac{1}{N} \sum_{n=1}^{N} E(n)$$

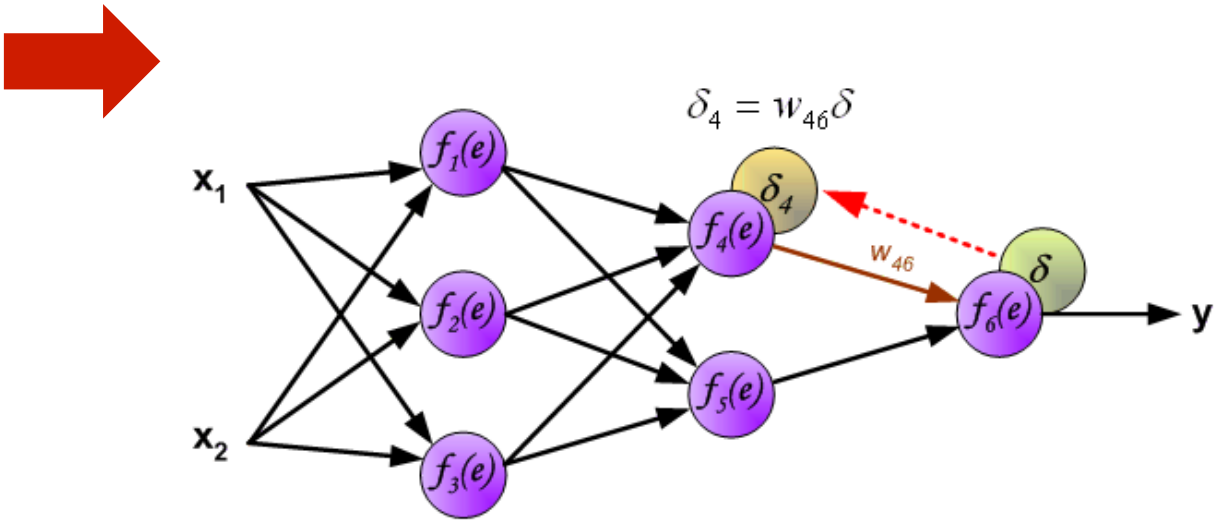
N: tamanho do conjunto de treinamento

- **Objetivo do Algoritmo de Aprendizagem: Atualizar os pesos da rede de forma a diminuir E_{AV}**

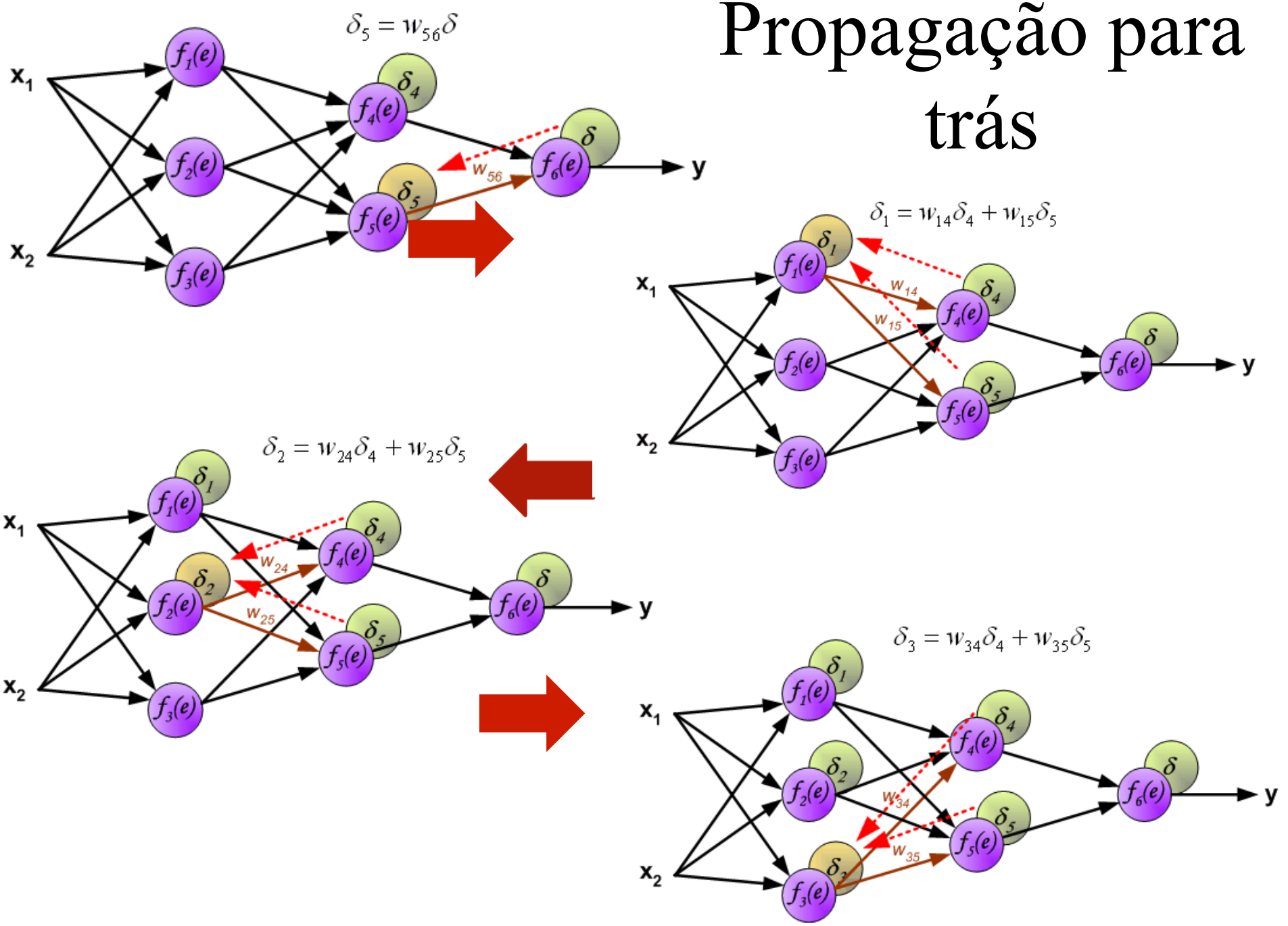
Propagação para trás



- z é a saída esperada



Propagação para trás



Atualização dos Pesos

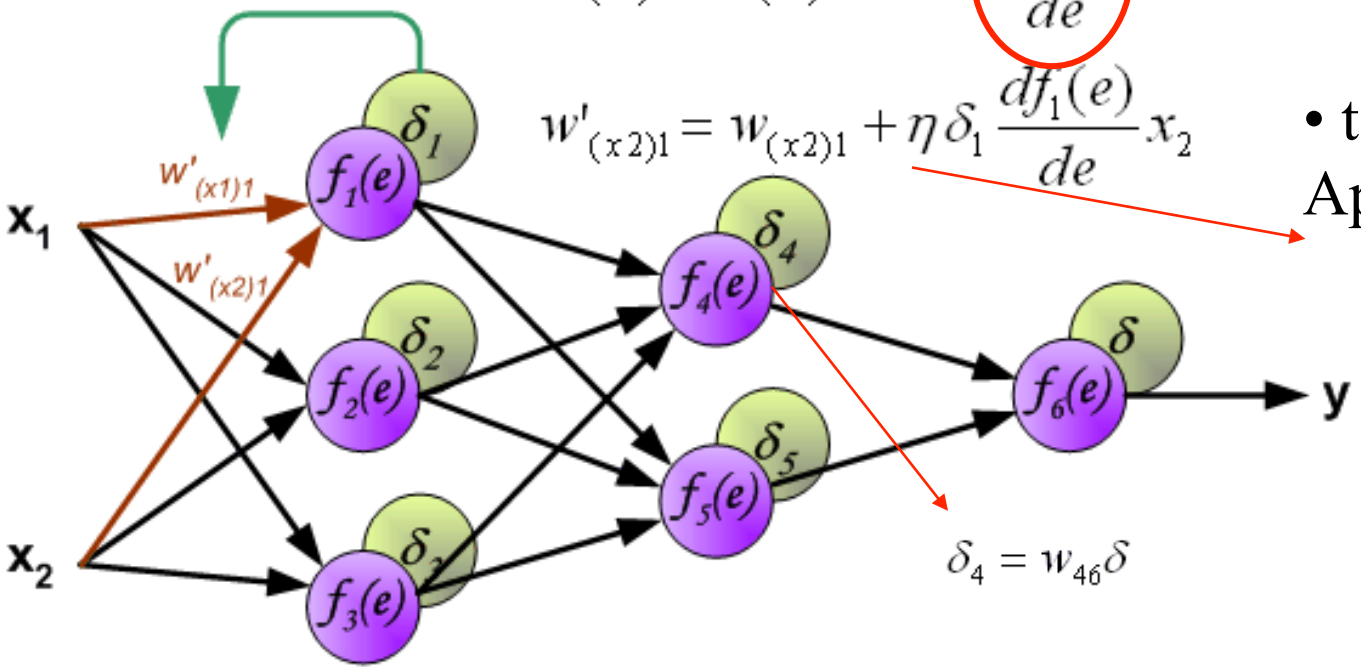
$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$

- Derivada da função de ativação do neurônio

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

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

- taxa de Aprendizagem (α)

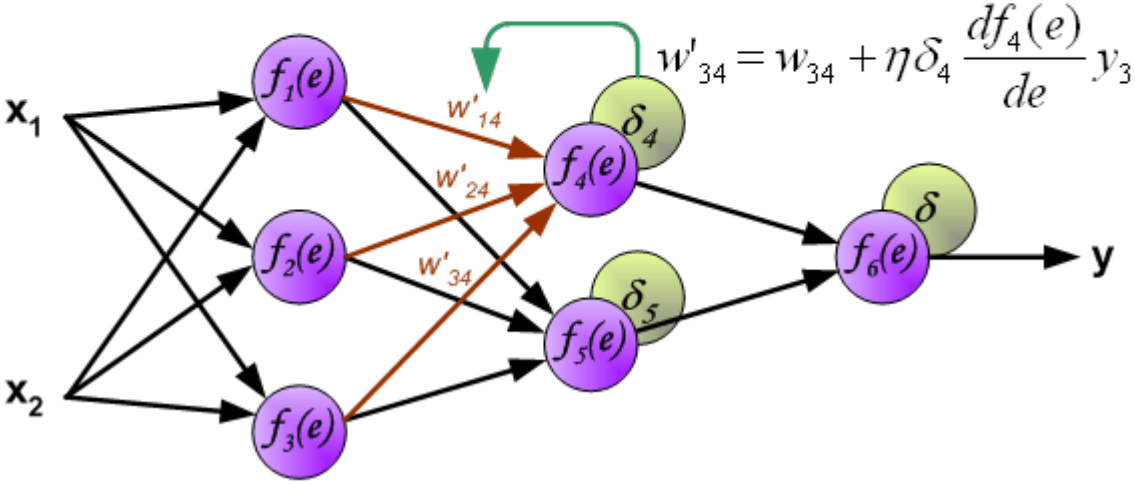


$$\delta_4 = w_{46}\delta$$

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

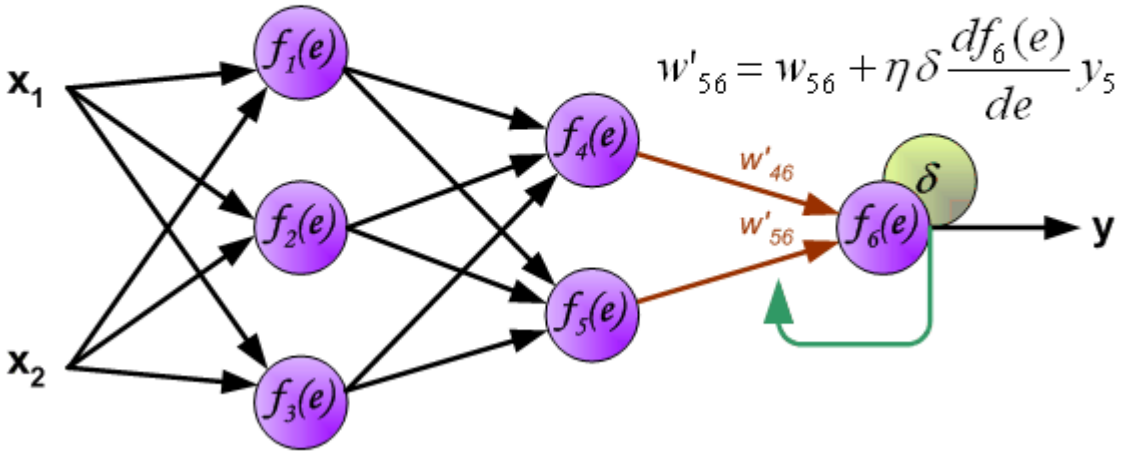
$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$

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

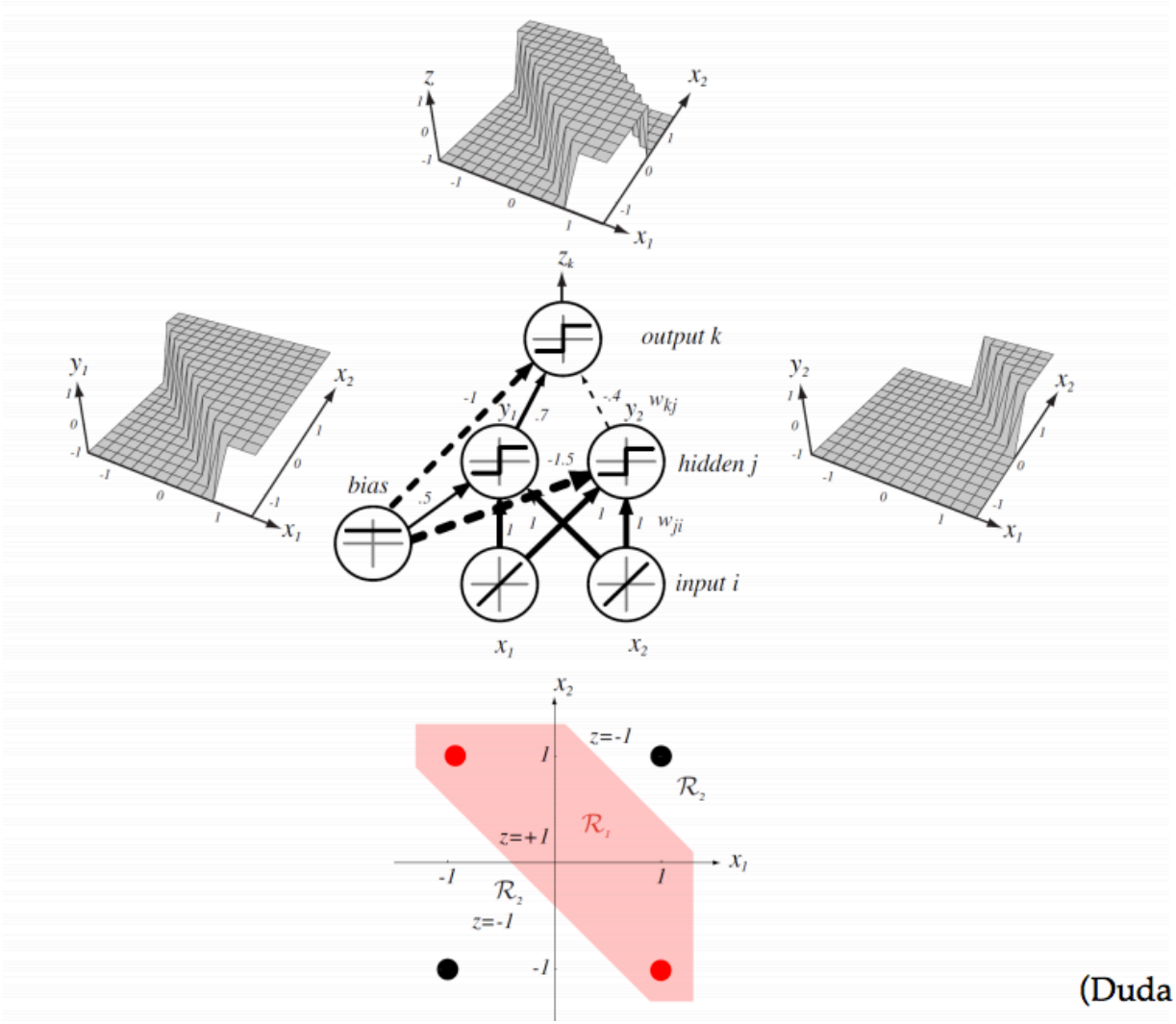


$$w'_{46} = w_{46} + \eta \delta \frac{df_6(e)}{de} y_4$$

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



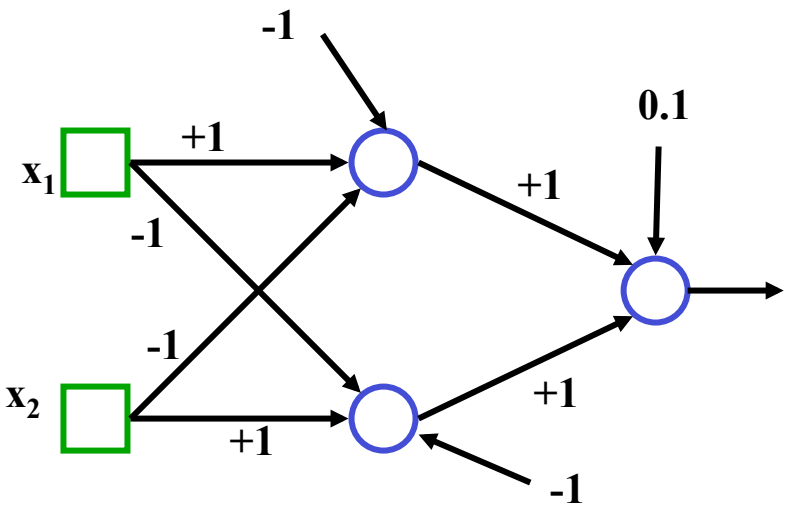
O MLP resolve o problema do XOR?

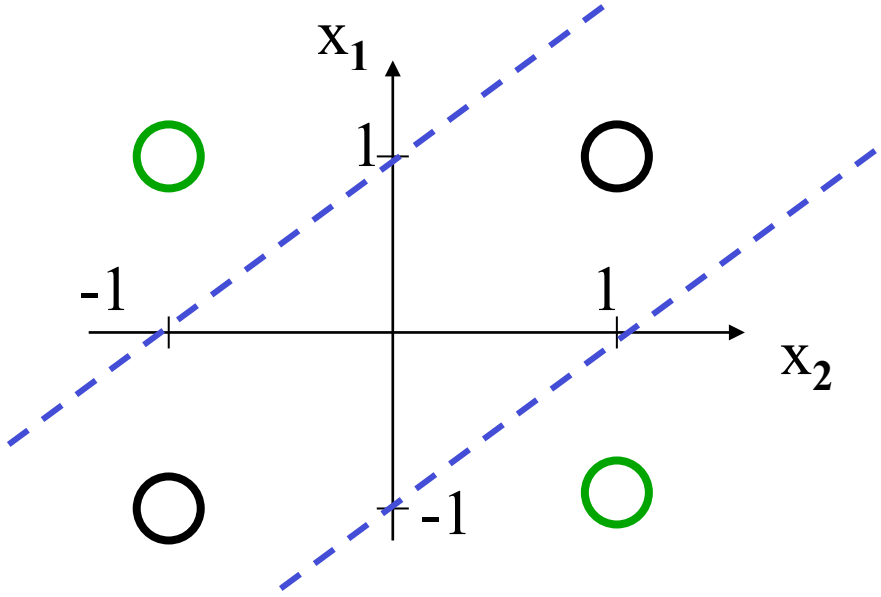


(Duda et al)

Solução para o problema do XOR

x_1	x_2	x_1 xor x_2
-1	-1	**-1**
-1	1	**1**
1	-1	**1**
1	1	**-1**





$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ -1 & \text{if } v \leq 0 \end{cases}$$

φ é uma sigmoide

Sumário do *Backpropagation*

- *Backpropagation*
  - Passo para frente
    - Pesos são fixos, e dados de entrada são fornecidos
    - Para cada nó, as saídas são calculadas
    - Para os nós da camada de saída, um erro e_j é calculado

      $e_j = d_j - y_j$ (saída desejada – saída esperada)
  - Passo para trás
    - Começando pelos nós de saída
    - Computa o erro local de cada neurônio recursivamente
    - A partir dos erros locais, calcula $\Delta w(t)$

Leitura Recomendada

- Neural and Adaptive Systems: Fundamentals Through Simulations, Jose C. Principe , Neil R. Euliano, W. Curt Lefebvre, John Wiley & Sons, Inc.

- http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html