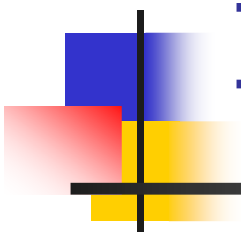
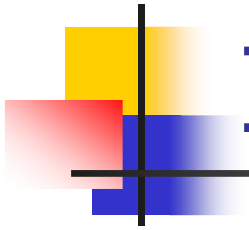


Inteligência Artificial

Prof. Tiago A. E. Ferreira

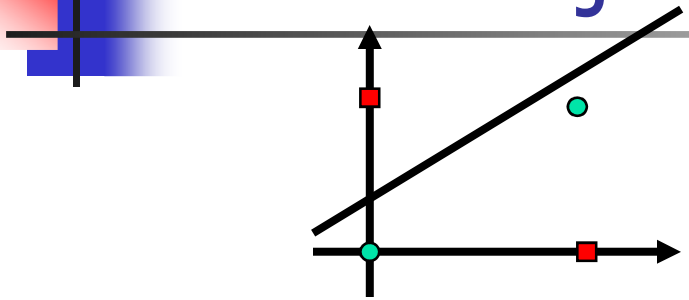
Aula 20 - Backpropagation

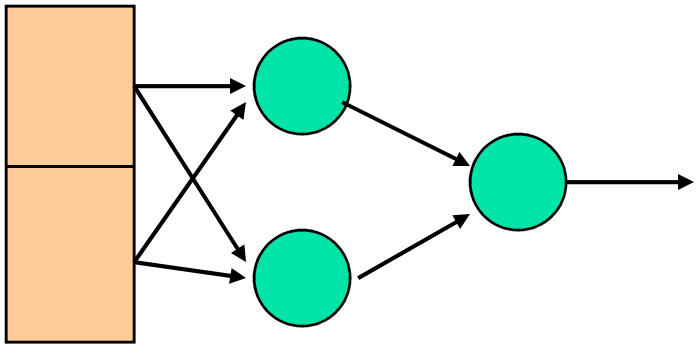


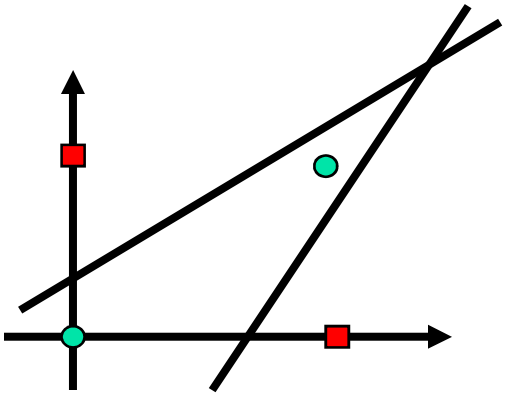
Introdução

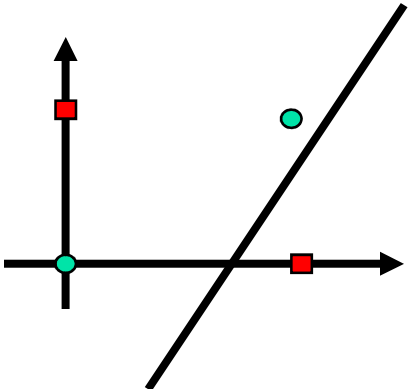
- Redes de uma camada resolvem apenas problemas linearmente separáveis
- Solução: utilizar mais de uma camada
  - Camada 1: uma rede Perceptron para cada grupo de entradas linearmente separáveis
  - Camada 2: uma rede combina as saídas das redes da primeira camada, produzindo a classificação final

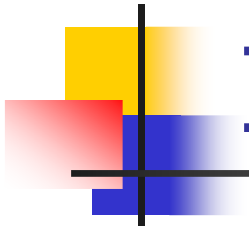
Introdução





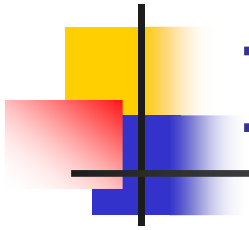






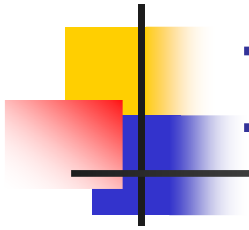
Introdução

- Treinamento da Rede
  - Treinar a rede independentemente
    - Saber como dividir o problema em sub-problemas
    - Nem sempre é possível
  - Treinar toda a rede
    - Qual o erro dos neurônios da camada intermediária?
    - Qual função de ativação?



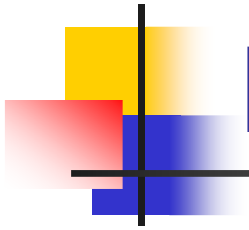
Introdução

- Função de ativação linear
  - Cada camada computa uma função linear
    - Composição de funções lineares é uma função linear
    - Sempre vai existir uma rede com uma camada equivalente a uma rede multicamadas com funções de ativação lineares



Introdução

- Função de ativação para redes multicamadas (MLP)
  - Não devem ser todas lineares
  - Exemplos de função não lineares
    - Função sigmóide logística
    - Função tangente hiperbólica
    - Etc...

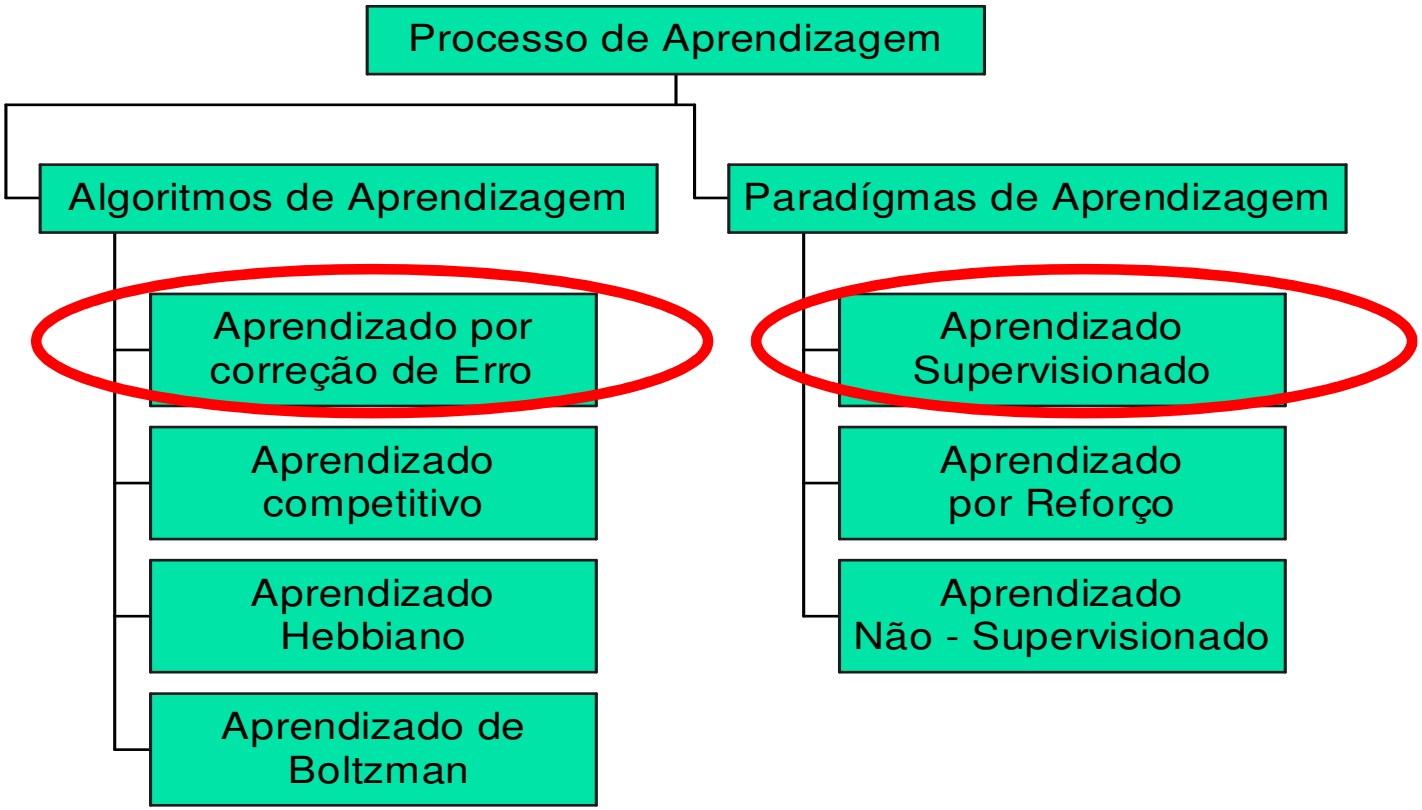


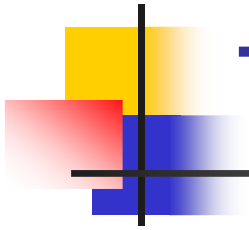
Rede Multi-Layer Perceptron

- A rede MLP é a arquitetura de RNA mais utilizada
- Possuem uma ou mais camadas intermediárias de nós
  - Geralmente utiliza função de ativação sigmóide logística

Aprendizagem em uma RNA

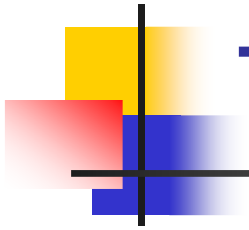
Aprendizagem





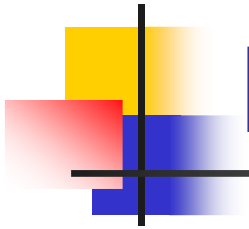
Treinamento de redes MLP

- Grande variedade de Algoritmos
  - Geralmente supervisionados
  - Estáticos
    - Não alteram a estrutura da rede
    - Backpropagation, Função de Base Redial (RBF)
  - Construtivos
    - Alteram estrutura da rede
    - Upstar, Cascade Correlation



Treinamento de Redes MLP

- Treinamento estático
  - MLP's com formatos e tamanhos diferentes podem utilizar mesma regra de aprendizado
    - Topologias diferentes podem resolver o mesmo problema
  - Regra mais utilizada: Backpropagation

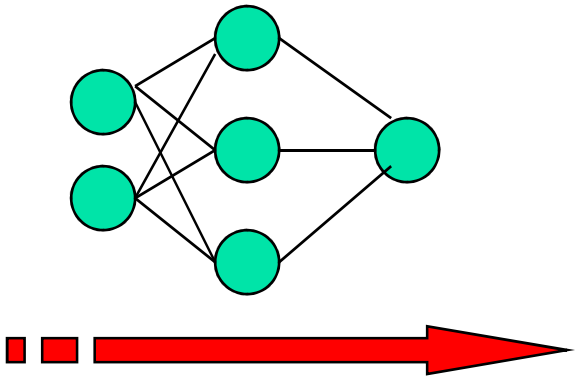


Backpropagation

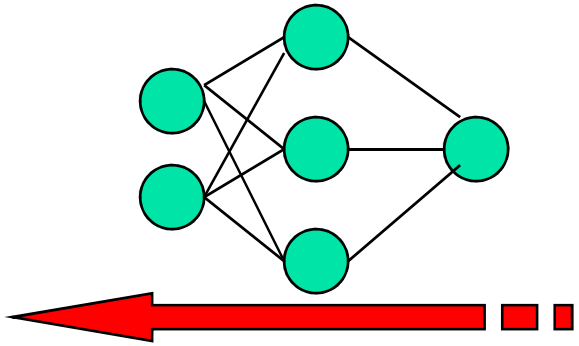
- Rede é treinada com pares entrada-saída
- Cada entrada de treinamento está associada a uma saída desejada
- Treinamento é feito em duas etapas, cada uma percorrendo a rede em um sentido

Fases do treinamento

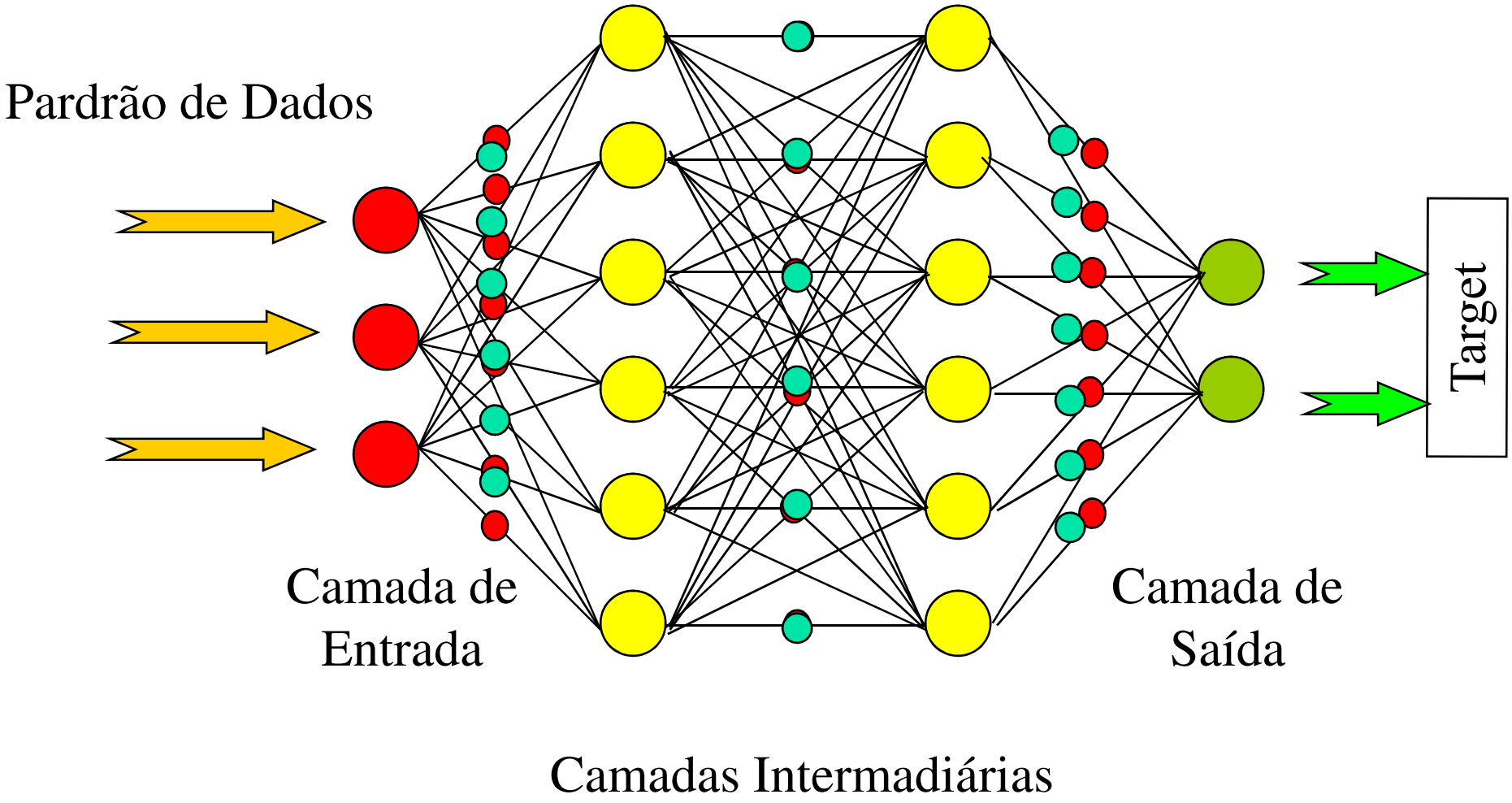
- Fase forward(sinal)

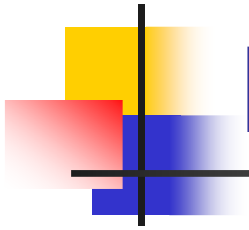


- Fase backward (erro)



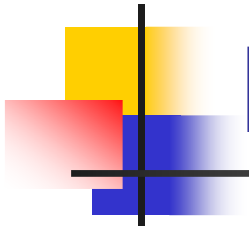
MLP com Backpropagation





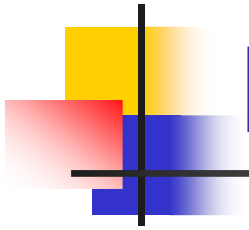
Fase Forward

- Entrada é apresentada à primeira camada da rede
  - Após os neurônios da camada **i** calcularem seus sinais de saída, os neurônios da camada **i+1** calculam seus sinais de saída
- Saídas produzidas pelos neurônios da última camada são comparadas com as saídas desejadas
- Erro para cada neurônio da camada de saída é calculado



Fase Backward

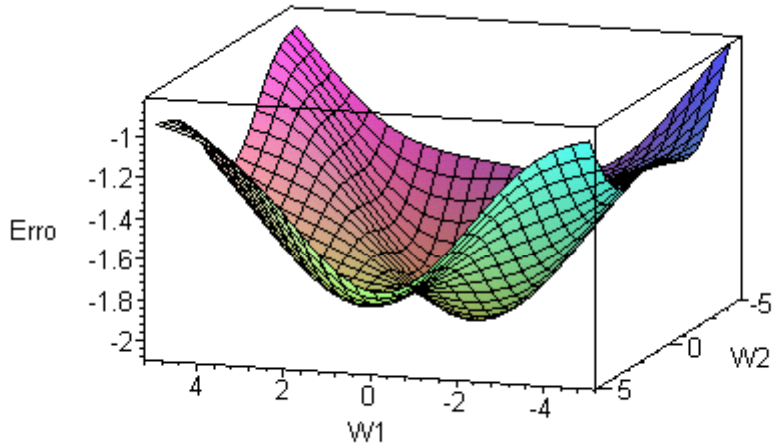
- A partir da última camada
  - O nó ajusta seu peso de modo a reduzir o seu erro
  - Nós das camadas anteriores tem seu erro definidos por:
    - Erros dos nós da camada seguinte conectados a ele ponderados pelos pesos das conexões entre eles

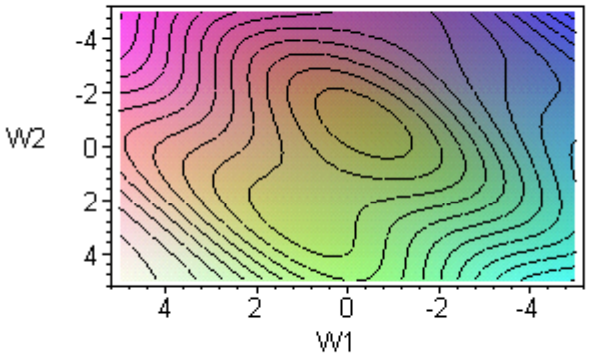


Backpropagation

- Processamento
  - Forward
    - Testa um determinado padrão
  - Backward
    - Treinamento – Ajusta os pesos da rede

Aprendizado por Correção de Erro





$$Ep = \frac{1}{2}\sum_{j}\left(t_{pj} - o_{pj}\right)^2$$

$E_p \equiv$ Erro para o padrão p

$o_{pj} \equiv$ Saída atual do nodo j para o padrão p

$t_{pj} \equiv$ Saída desejada (target)

Minimização do Erro

Vetor que aponta na direção de decrescimo da função erro:

$$-\nabla E_p = -\sum \frac{\partial E_p}{\partial w_{ij}} \hat{w}_{ij}$$

Pela regra da Cadeia:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ij}}$$

Desenvolvendo

$$\frac{\partial\, net_{pj}}{\partial\, w_{ij}} = \frac{\partial}{\partial\, w_{ij}} \sum_k w_{kj}\, o_{pj} = o_{pj}$$

$$\frac{\partial\, E_p}{\partial\, net_{pj}} = \frac{\partial\, E_p}{\partial\, o_{pj}} \frac{\partial\, o_{pj}}{\partial\, net_{pj}}$$

Desenvolvendo

$$\frac{\partial o_{pj}}{\partial net_{pj}} = \frac{\partial}{\partial net_{pj}}\left(f_j\left(net_{pj}\right)\right) = f_j'\left(net_{pj}\right)$$

O outro termo depende de qual camada estamos: intermediária ou final!

- Camada de Saída:

$$\frac{\partial E_p}{\partial o_{pj}} = \frac{\partial}{\partial o_{pj}}\left(\frac{1}{2}\sum_j \left(t_{pj} - o_{pj}\right)^2\right) = -\left(t_{pj} - o_{pj}\right)$$

Desenvolvendo

Sendo assim $$\frac{\partial E_p}{\partial net_{pj}} = -f'_j(net_{pj})(t_{pj} - o_{pj}) = -\delta_{pj}$$

- Camadas intermediárias:

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \frac{\partial net_{pk}}{\partial o_{pj}}$$

$$= \sum_k -\delta_{pk} w_{kj}$$

Desenvolvendo

Gradiente do Erro:

$$\frac{\partial E_p}{\partial w_{ij}} = -\delta_{pj} o_{pi}$$

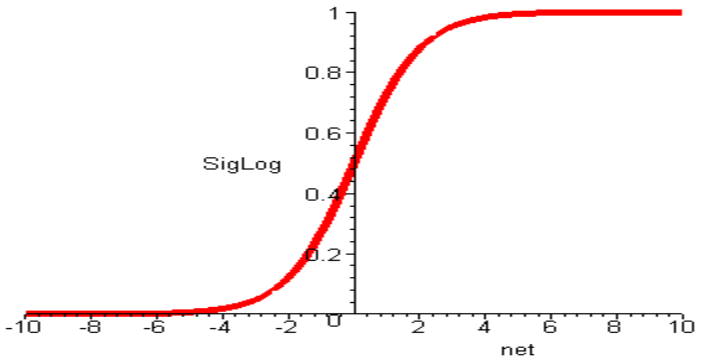
Onde:

$$\delta_{pj} = \begin{cases} f_j'(net_{pj})(t_{pj} - o_{pj})o_{pi}, & \text{Se camada de Saída} \\ f_j'(net_{pj})\left(\sum_k \delta_{pk} w_{jk}\right), & \text{Se camada intemediária} \end{cases}$$

Função de Ativação

Sigmoide Logistica:

$$f(net) = \frac{1}{1 + e^{-\lambda net}}$$



Deriovando:

$$f'(net) = \lambda f(net)(1 - f(net))$$
$$= \lambda o_{pj}(1 - o_{pj})$$

Atualização dos Pesos

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}$$

Ajuste para o mínimo de Erro:

(η - Taxa de Aprendizagem)

$$\Delta_p w_{ij} \propto -\frac{\partial E_p}{\partial w_{ij}}$$

$$\Delta_p w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}} = \eta \delta_{pj} o_{pi}$$

Backpropagation - Resumo

- Função de saída
  - Função de identidade $y_i = a_i$
- Treinamento
  - Supervisionado
  - Ajuste de Pesos
    - $\Delta w_{ij} = \eta x_i y_i(1-y_i)\delta_i$
      - $\delta_j = (d_i - y_i)$ (se última camada)
      - $\delta_j = \Sigma w_{jk}\delta_k$ (se camadas intermediárias)
  - Não há garantias de convergência para o treinamento

Treinamento - Algorítimo

1) Iniciar Todas as conexões com valores aleatórios e pequenos ([-0.5,0.5]
2) Repita

    Erro = 0

    Para cada para de treinamento (X, d)

        Para cada camda k:=1 to N

            Para cada Neurônio j:=1 To M

                Calcular Saída y_{jk}

    Se Erro > ε

    Então para cada camada k:=N to 1

        Para cada neurônio j:=1 to M

            Atualizar pesos

Até Erro < ε