MULTI-LAYER PERCEPTRON

- Redes de apenas uma camada só representam funções linearmente separáveis
- Redes de múltiplas camadas solucionam essa restrição
- O desenvolvimento do algoritmo Back-Propagation foi um dos motivos para o ressurgimento da área de redes neurais

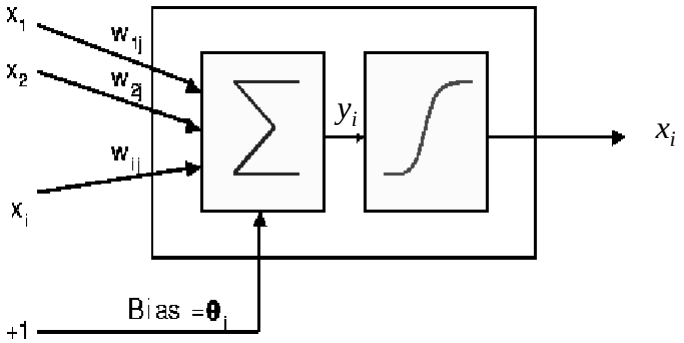
MULTI-LAYER PERCEPTRON

- O grande desafio foi achar um algoritmo de aprendizado para atualizar dos pesos das camadas intermediarias
- Idéia Central

  os erros dos elementos processadores da camada de saída (conhecidos pelo treinamento supervisionado) são **retro-propagados** para as camadas intermediarias

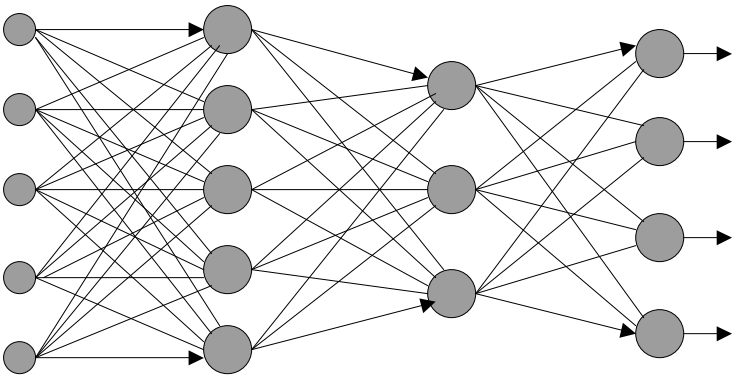
MODELO BÁSICO DO NEURÔNIO ARTIFICIAL

Padrão de entrada vetor X



MULTI-LAYER PERCEPTRON

Rede de 3 camadas 5/5/3/4



Entrada | Camada Escondida | Camada Escondida | Saída

CARACTERISTICAS BASICAS

- Regra de propagação $y_j = \sum_i x_i w_{ij} + \theta_i$
- Função de ativação: Função não-linear diferenciável em todos os pontos;
- Topologia: Múltiplas camadas;
- Algoritmo de Aprendizado: Supervisionado;
- Valor de Entrada/Saída: Binários e/ou Contínuos.

PROCESSO DE APRENDIZADO

• Processo de minimização do erro quadrático pelo método do *Gradiente Descendente*

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

• Cada peso sináptico *i* do elemento processador *j* é atualizado proporcionalmente ao *negativo da derivada parcial do erro* deste processador com relação ao peso.

PROCESSO DE APRENDIZADO

Onde o erro quadrático do processador *j* é definido como:

$$E_j = \frac{1}{2}\left(t_j - x_j\right)^2$$

- t_j – valor desejado de saída para o processador *j* da camada de saída
- x_j – estado de ativação do processador *j* da camada de saída

PROCESSO DE APRENDIZADO

Na verdade, deve-se minimizar o erro de *todos os processadores* da camada de saída, para padrões *p*

$$E_p = \frac{1}{2}\sum_{j=1}^{N}\left(t_j - x_j\right)^2$$

- t_j – valor desejado de saída do padrão *p* para o *processador j* da camada de saída
- x_j – estado de ativação do processador *j* da camada de saída quando apresentado o padrão *p*

PROCESSO DE APRENDIZADO

Calcula Δw_{ij}

$$\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}} = -\eta \frac{\partial E_p}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}}$$

$$e_j = -\frac{\partial E_p}{\partial y_j} \qquad y_j = \sum x_i w_{ij} + \theta_j$$

$$\Delta w_{ij} = \eta e_j x_i$$

PROCESSO DE APRENDIZADO

Calcula e_j

Depende da camada à qual o processador *j* pertence

$$e_j = -\frac{\partial E_p}{\partial y_j} = -\frac{\partial E_p}{\partial x_j} \frac{\partial x_j}{\partial y_j}$$

$$x_j = F(y_j)$$

$$F'(y_j)$$

Se $j \in$ Camada de Saída

Se $j \notin$ Camada de Saída

PROCESSO DE APRENDIZADO

Calcula e_j, $j \in$ Camada de Saída

$$e_j = -\frac{\partial E_p}{\partial y_j} = -\frac{\partial E_p}{\partial x_j}\frac{\partial x_j}{\partial y_j}$$

$$E_p = \frac{1}{2}\sum_j (t_j - x_j)^2 \qquad x_j = F(y_j)$$

$$2 \times \frac{1}{2} \times (t_j - x_j)(-1) \qquad F'(y_j)$$

$$e_j = -\left[-(t_j - x_j)\right] \times F'(y_j) = (t_j - x_j)F'(y_j)$$

PROCESSO DE APRENDIZADO

Calcula e_j, $j \in$ Camada Escondida

$$e_j = -\frac{\partial E_p}{\partial y_j} = -\frac{\partial E_p}{\partial x_j}\frac{\partial x_j}{\partial y_j}$$

$$E_p = ? \qquad x_j = F(y_j)$$

$$F'(y_j)$$

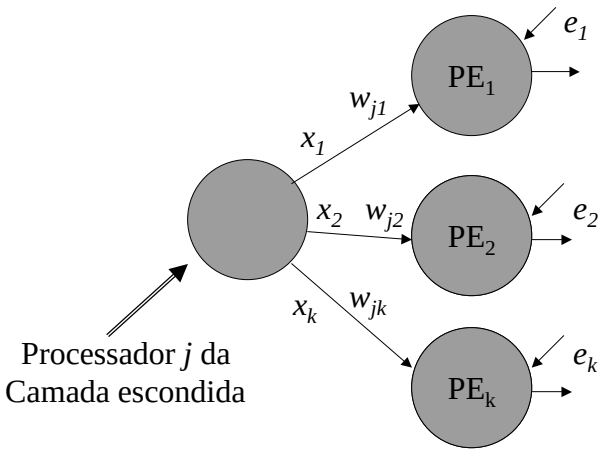
PROCESSO DE APRENDIZADO

Calcula e_j, $j \in$ Camada Escondida

- Pelo aprendizado supervisionado, só se conhece o erro na camada de saída;
- Erro na saída é função do potencial interno do processador (y_k)
- O y_k depende dos estados de ativação dos processadores da camada anterior (x_j) e dos pesos das conexões (w_{ij});
- Portanto, x_j de uma camada escondida afeta, em maior ou menor grau, o erro de todos os processadores da camada subsequente.

PROCESSO DE APRENDIZADO

Calcula de e_j, $j \in$ Camada Escondida



$$e_j = -\frac{\partial E_p}{\partial y_j} = -\frac{\partial E_p}{\partial x_j}\frac{\partial x_j}{\partial y_j} = -\frac{\partial}{\partial x_j}\left(\frac{1}{2}\sum_k d_k^2\right)F'(y_j)$$

$$= -F'(y_j)\sum_k\left(\frac{1}{2}\left(\frac{\partial}{\partial x_k}d_k^2\right)\frac{\partial x_k}{\partial x_j}\right)$$

$$= F'(y_j)\sum_k\left(d_k\frac{\partial x_k}{\partial x_j}\right)$$

$$= F'(y_j)\sum_k\left(d_k\frac{\partial x_k}{\partial y_k}\frac{\partial y_k}{\partial x_j}\right)$$

$$= F'(y_j)\sum_k\left(d_k F'(y_k) w_{jk}\right)$$

$$= F'(y_j)\sum_k\left(e_k w_{jk}\right)$$

Obs.: $d_k = t_k - x_k$

PROCESSO DE APRENDIZADO

• Em resumo, após o cálculo da derivada, tem-se

$$\Delta w_{ij} = \eta e_j x_i$$

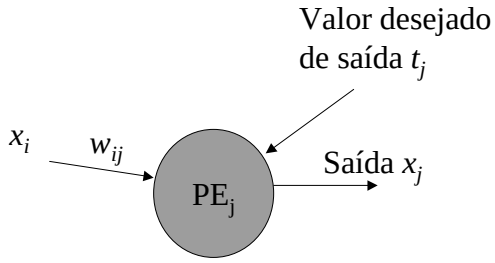
Onde

x_i – valor de entrada recebido pela conexão *i*

e_j – valor calculado do erro do processador *j*

PROCESSO DE APRENDIZADO

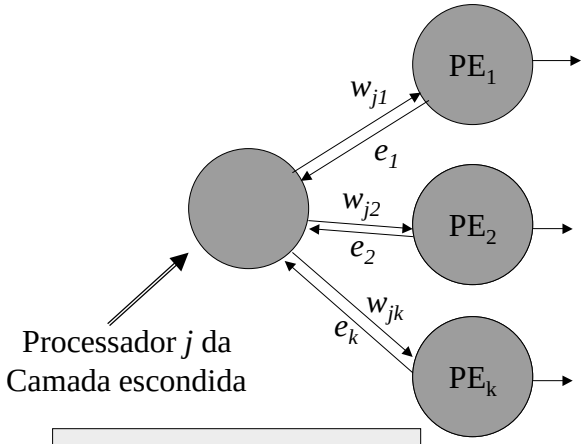
Processador *j* pertence à Camada de Saída:



$$e_j = (t_j - x_j) F'(y_j)$$

PROCESSO DE APRENDIZADO

Processador *j* pertence à Camada Escondida:



$$e_j = F'(y_j) \sum_k (e_k w_{jk})$$

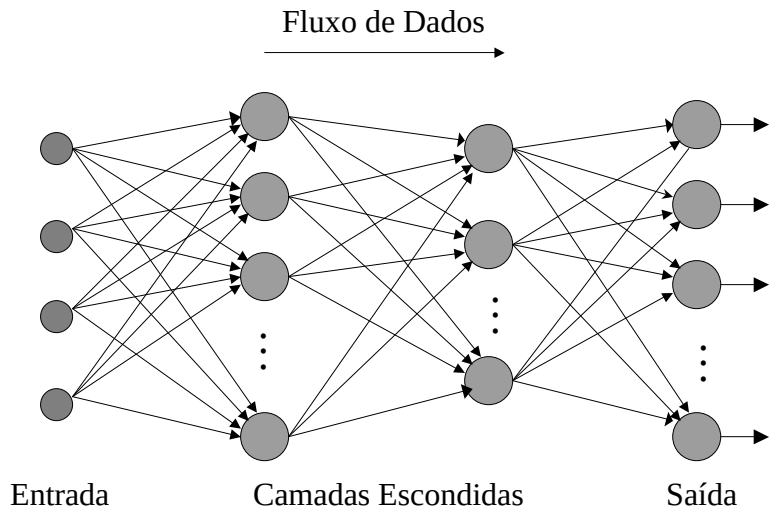
PROCESSO DE APRENDIZADO

O algoritmo **Back-Propagation** tem portanto *duas fases*, para cada padrão apresentado

- **Feedforward** - as *entradas* se propagam, pela rede, da camada de entrada para a camada de saída
- **Feedback** - os *erros* se propagam, na *direção contraria ao fluxo de dados*, indo da camada de saída até a primeira camada escondida

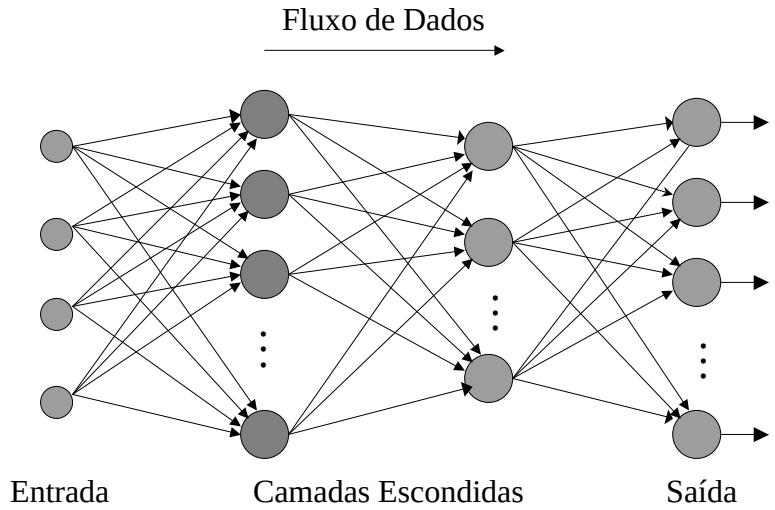
PROCESSO DE APRENDIZADO

Ilustração Fase 1: Feed-Forward



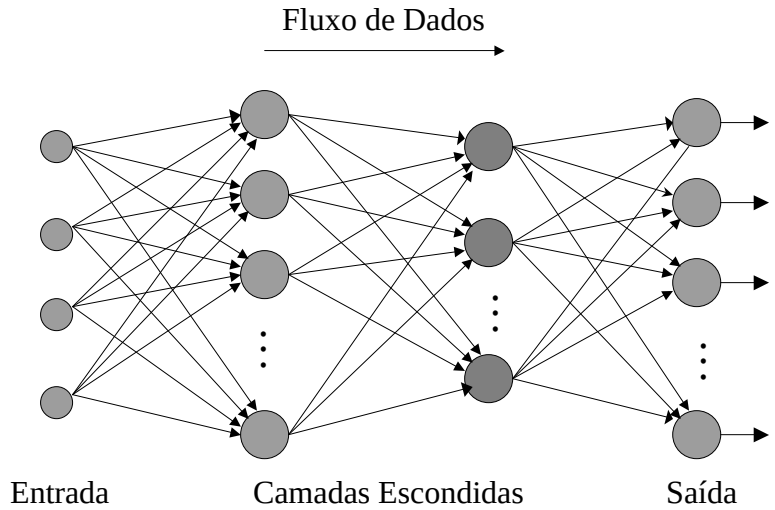
PROCESSO DE APRENDIZADO

Ilustração Fase 1: Feed-Forward



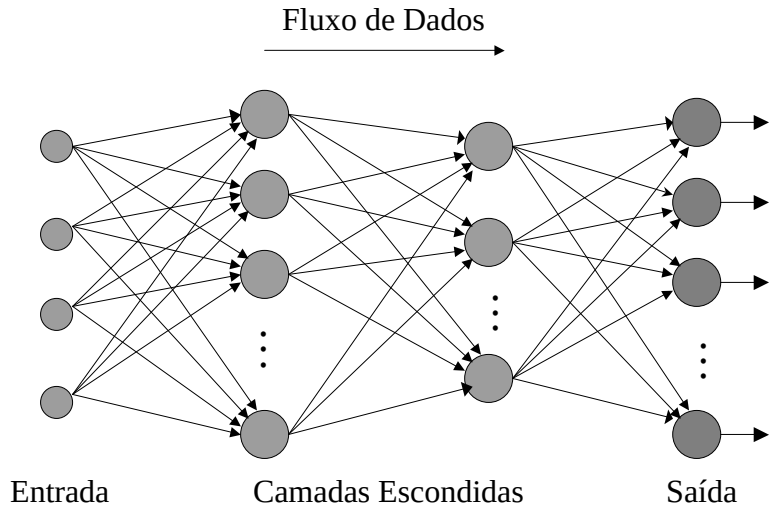
PROCESSO DE APRENDIZADO

Ilustração Fase 1: Feed-Forward



PROCESSO DE APRENDIZADO

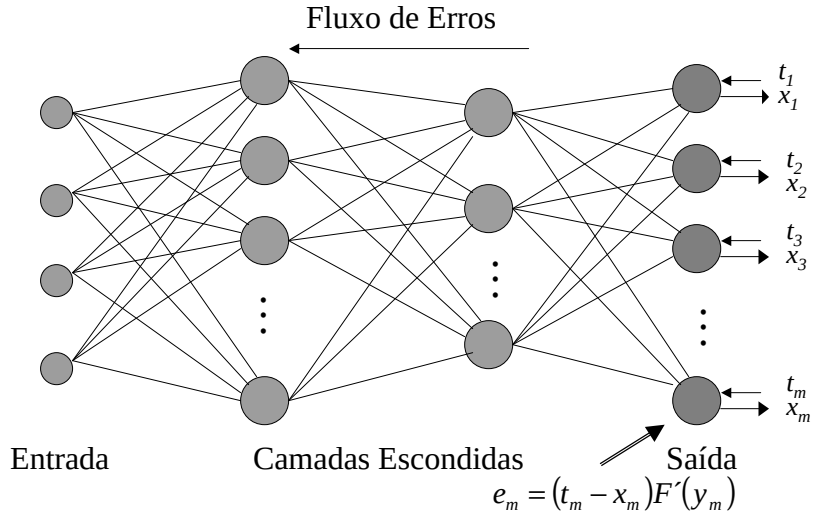
Ilustração Fase 1: Feed-Forward



PROCESSO DE APRENDIZADO

Ilustração Fase 1: Feed-Backward

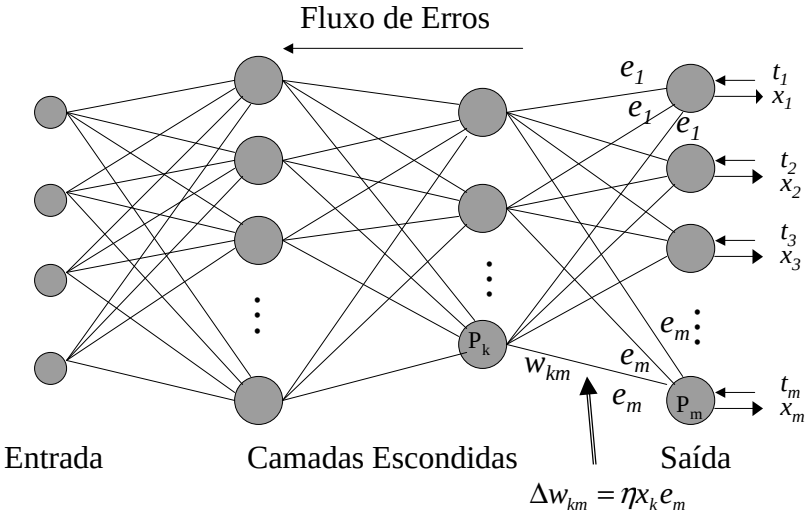
Cálculo do erro da camada de saída



PROCESSO DE APRENDIZADO

Ilustração Fase 1: Feed-Backward

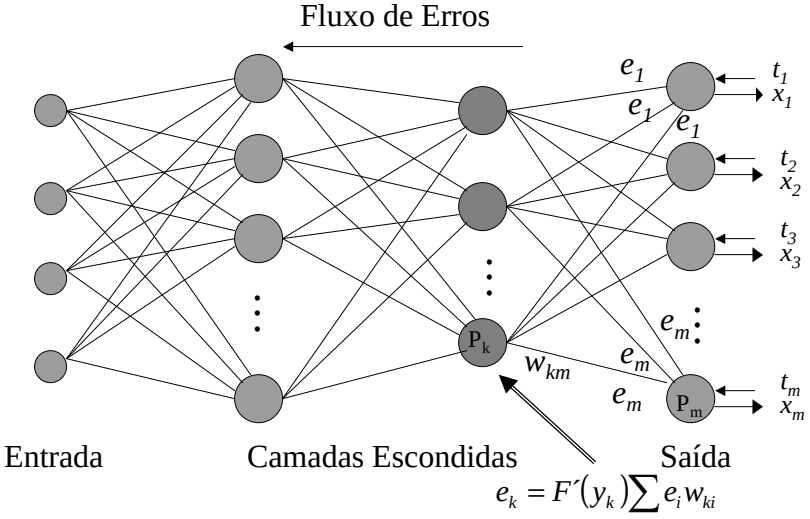
Atualização dos pesos da camada de saída



PROCESSO DE APRENDIZADO

Ilustração Fase 1: Feed-Backward

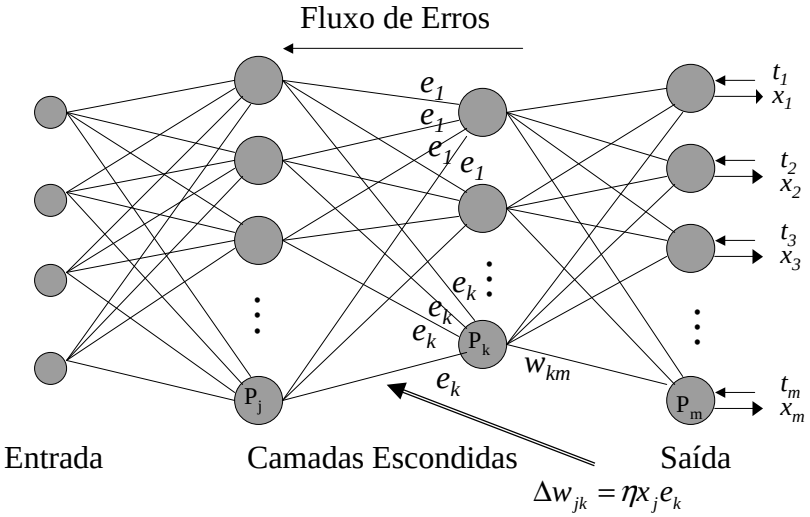
Cálculo do erro da 2º camada escondida



PROCESSO DE APRENDIZADO

Ilustração Fase 1: Feed-Backward

Atualização dos pesos da 2º camada escondida

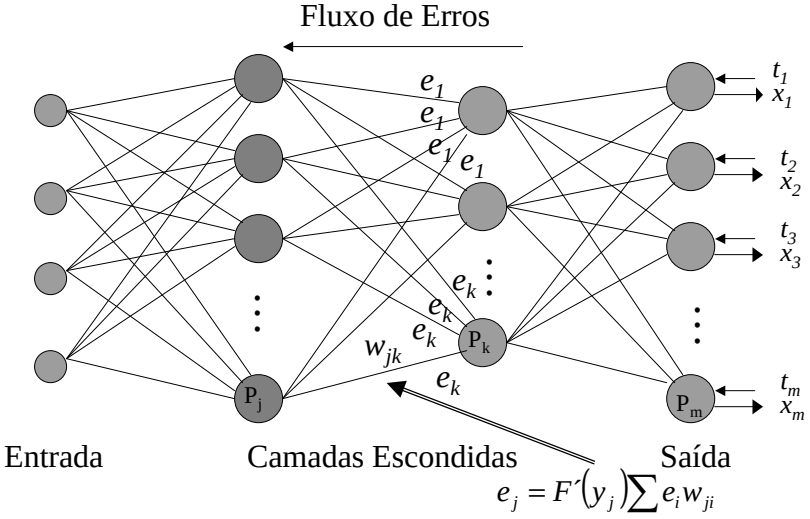


$$\Delta w_{jk} = \eta x_j e_k$$

PROCESSO DE APRENDIZADO

Ilustração Fase 1: Feed-Backward

Cálculo do erro da 1º camada escondida

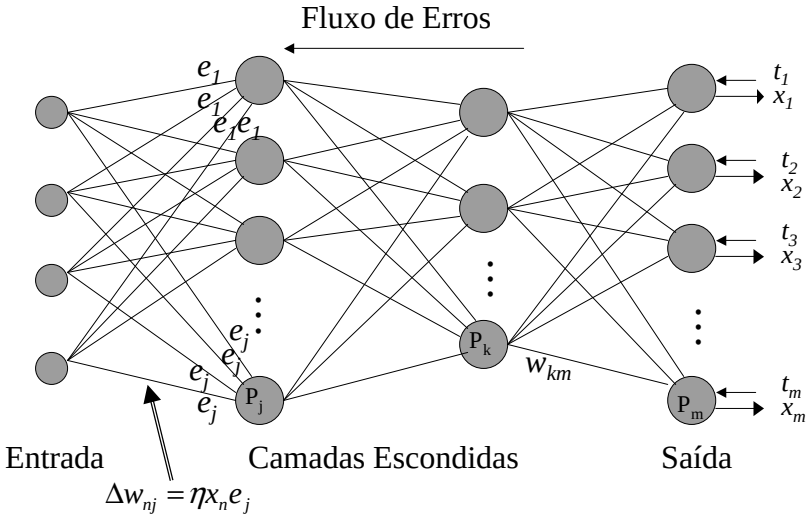


$$e_j = F'(y_j)\sum e_i w_{ji}$$

PROCESSO DE APRENDIZADO

Ilustração Fase 1: Feed-Backward

Atualização dos pesos da 1º camada escondida



ALGORITMO

Este procedimento de aprendizado é repetido diversas vezes, até que *para todos processadores de camada de saída e para todos padrões de treinamento*, o erro seja menor do que o especificado.

ALGORITMO

Inicialização

pesos iniciados com valores aleatórios e pequenos ($|w_{ij}| < 0.1$)

Treinamento

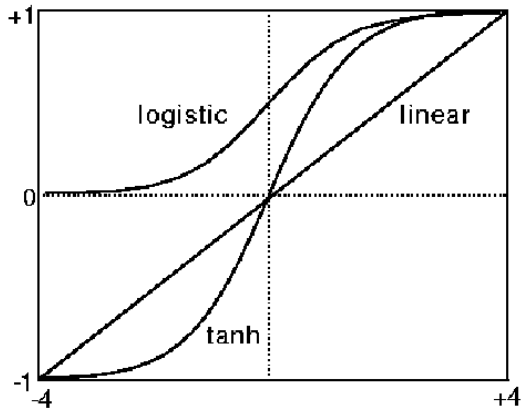
loop até que o erro de cada processador de saída seja <= tolerância

para todos os padrões de conjunto de treinamento

1) Aplica se um padrão de entrada x_i e seu respectivo vetor de saída t_i desejado;
2) calcule se as saídas dos processadores, começando da primeira camada escondida até a camada de saída;
3) calcule se o erro para cada processador da camada de saída, se erro <= tolerância, para todos os processadores, volta para 1)
4) atualiza os pesos de cada processador, começando pela camada de saída, até a camada de entrada;
5) volta ao passo 1).

DICAS PARA BP

O uso da função de ativação simétrica geralmente acelera o treinamento

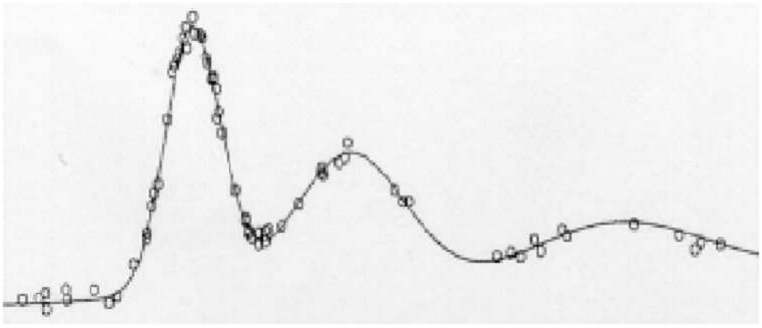


DICAS PARA BP

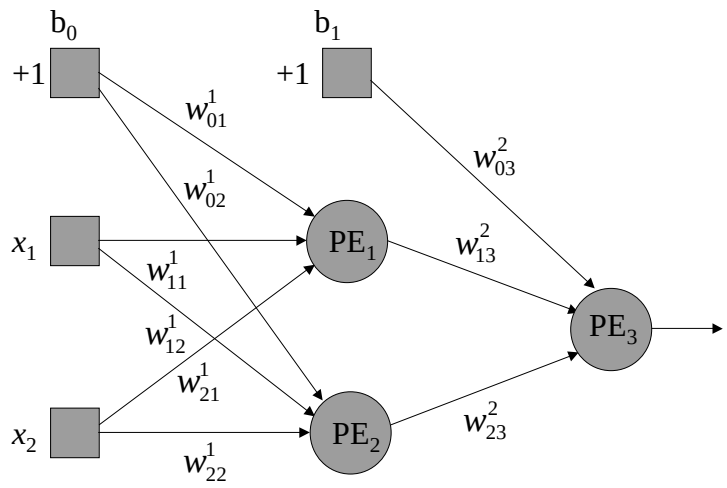
- Os pesos devem ser inicializados com valores uniformemente distribuídos
- A ordem de apresentação dos padrões de treinamento deve ser alterada a cada época
  - ordem aleatória
- Todos os neurônios devem aprender na mesma taxa

DICAS PARA BP

O processo de aprendizado pode ser visto como um problema de *curve fitting* ou *interporação*



EXEMPLO



EXEMPLO

Entrada: (x_1= 1, x_2 = 0) Saída Desejada: t_3 = 1

Pesos iniciais: $w_{ij}(0) = 0$ Taxa de Aprendizagem: $\eta = 0.5$

Função de Ativação:

$$F(y_i) = \frac{1}{1+\exp(-y_i)}$$

Derivada da Função de Ativação:

$$F'(y_i) = \frac{\exp(-y_i)}{[1+\exp(-y_i)]^2}$$

EXEMPLO

Algoritmo de Aprendizado:

$$w_{ij} = w_{ij} + \eta x_i e_j$$

$$e_j = (t_j - x_j) F'(y_j) \qquad \text{Camada de Saída}$$

$$e_j = F'(y_j) \sum_k e_k w_{jk} \qquad \text{Camada Escondida}$$

EXEMPLO

Primeiro Circulo de Treinamento:

$b_1 = 1$

$y_1 = 1\times0 + 1\times0 + 0\times0 = 0$ $\quad x_1 = 0.5$

$y_2 = 1\times0 + 1\times0 + 0\times0 = 0$ $\quad x_2 = 0.5$

$y_3 = 1\times0 + 0.5\times0 + 0.5\times0 = 0$ $\quad x_3 = 0.5$

Então, $t_3 - x_3 = 1 - 0.5 = 0.5$ $\quad e_3 = 0.5 \times 0.25 = 0.125$

$$w_{03}^2 = 0 + 0.5\times1\times0.125 = 0.0625$$

$$w_{13}^2 = 0 + 0.5\times0.5\times0.125 = 0.0313$$

$$w_{23}^2 = 0 + 0.5\times0.5\times0.125 = 0.0313$$

EXEMPLO

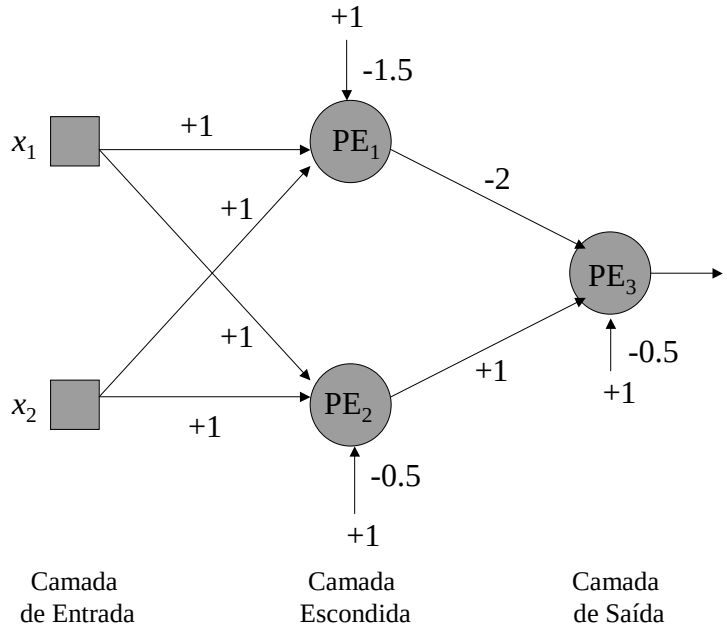
Primeiro Circulo de Treinamento (cont.):

$e_1 = 0.125 \times 0.0313 = 0.0039$

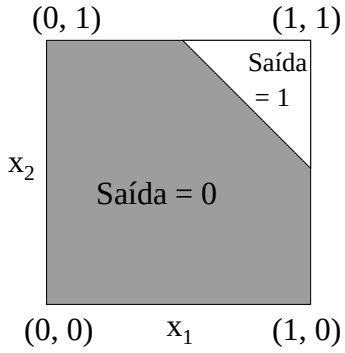
$e_2 = 0.125 \times 0.0313 = 0.0039$

$$w_{01}^1 = 0 + 0.5 \times 1 \times 0.0039 \times 0.25 = 0.0004875$$
$$w_{02}^1 = 0 + 0.5 \times 1 \times 0.0039 \times 0.25 = 0.0004875$$
$$w_{11}^1 = 0 + 0.5 \times 1 \times 0.0039 \times 0.25 = 0.0004875$$
$$w_{12}^1 = 0 + 0.5 \times 1 \times 0.0039 \times 0.25 = 0.0004875$$
$$w_{21}^1 = 0 + 0.5 \times 0 \times 0.0039 \times 0.25 = 0$$
$$w_{22}^1 = 0 + 0.5 \times 0 \times 0.0039 \times 0.25 = 0$$

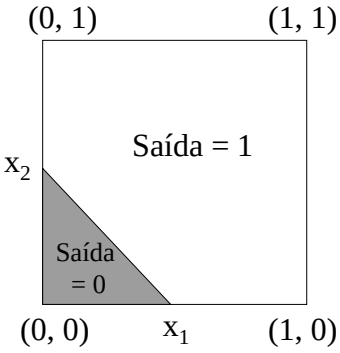
PROBLEMA XOR



PROBLEMA XOR

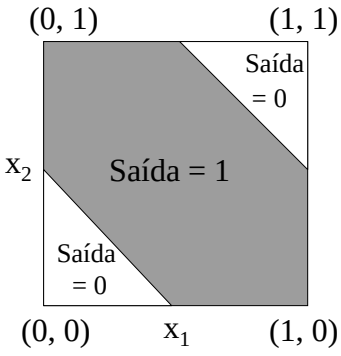


Borda de decisão construída pelo 1ª neurônio escondido



Borda de decisão construída pelo 2ª neurônio escondido

PROBLEMA XOR



Borda de decisão construída pela rede completa