

Introdução Inteligência Computacional

Profa. Dra. Ana Paula Abrantes de Castro e Shiguemori
anapaula.acs@ifsp.edu.br
Instituto Federal de Educação, Ciência e Tecnologia – IFSP

Dr. Elcio Hideiti Shiguemori
elcio@ieav.cta.br
Instituto de Estudos Avançados - IEAv

Curso: Análise e Desenvolvimento de Sistemas
4º. Semestre - IICI4 – 4 aulas Semanais



Perceptron Múltiplas Camadas

Histórico MLP

- › Depois das fortes críticas de Minsky e Pappert em 1969, as pesquisas em redes neurais foram drasticamente reduzidas durante toda a década de 70

- › Na década de 80 as redes neurais ressurgiram com trabalhos de alguns pesquisadores que pareciam mostrar que as afirmações de Minsky e Pappert não estavam corretamente embasadas

Histórico MLP

1969 Depois das fortes críticas de Minsky e Pappert, as pesquisas em redes neurais foram drasticamente reduzidas durante toda a década de 70

Na década de 80 as redes neurais ressurgiram com trabalhos de alguns pesquisadores que pareciam mostrar que as afirmações de Minsky e Pappert não estavam corretamente embasadas

Histórico MLP

- 1974** Paul Werbos conseguiu o maior progresso em termos de redes neurais desde o perceptron de Rosenblatt: ele lançou as bases do algoritmo de retro-propagação ("backpropagation"), que permitiu que redes neurais com múltiplas camadas apresentassem capacidade de aprendizado
- 1982** David Parker desenvolveu um método similar, de forma aparentemente independente. Contudo, a potencialidade deste método tardou a ser reconhecida

Histórico MLP

1986 – 1987

Foram publicados os primeiros resultados da retomada, com o trabalho de Rumelhart, Hinton e Williams do grupo de pesquisa PDP (Parallel and Distributed Processing) do MIT, onde ficou consagrada a técnica de treinamento por retropropagação do erro (backpropagation)

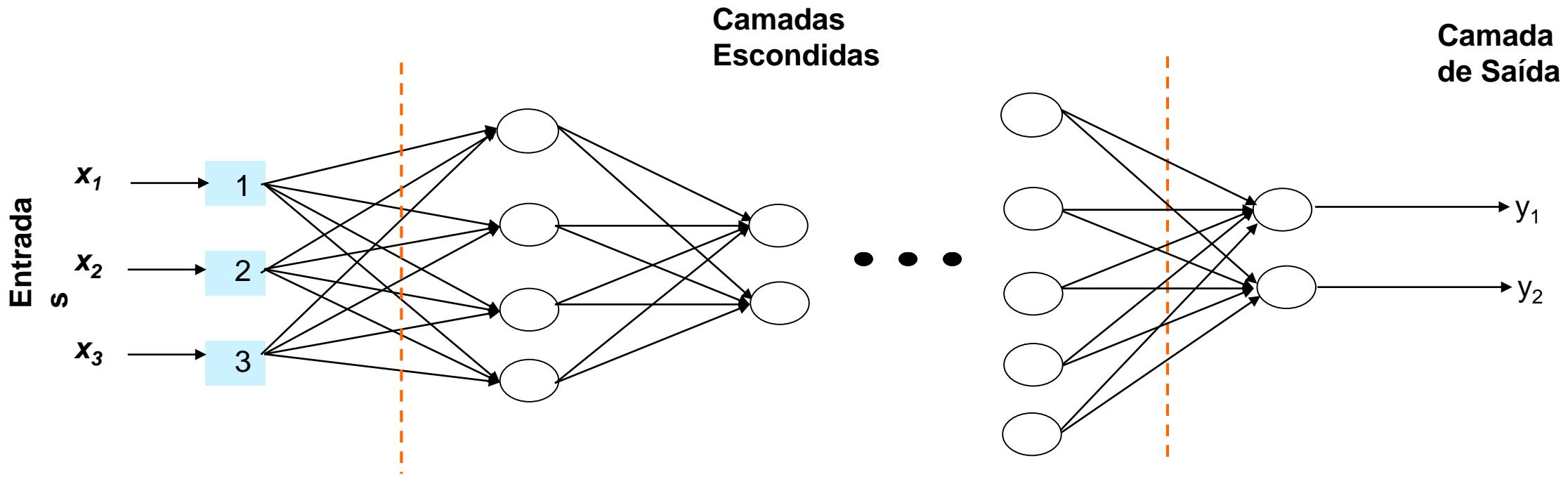
Introdução

- Diferentemente do Perceptron e Adaline, onde existe apenas um neurônio de saída {y}, a MLP pode relacionar o conhecimento a vários neurônios de saída.
- O algoritmo de aprendizado da MLP é chamado backpropagation ou regra Delta generalizada e é dividida em duas fases:
 1. Propagação adiante (forward): onde {x1, x2, ..., xn } são propagados até a saída, mantendo os pesos e limiares inalterados. Na saída para cada neurônio são obtidos os respectivos desvios (erros) entre as saídas desejadas e obtidas.
 2. Propagação reversa (backward): os ajustes são realizados a cada iteração, por todos os neurônios, produzindo o ajuste gradativo frente a saída desejada.
- Cada neurônio {j} pertence a uma camada{L}, onde g(.) representa a função de ativação de função logística ou hiperbólica.

Arquitetura

- A MLP é uma rede de múltiplas camadas e seu nome em inglês significa MultiLayer Perceptron (Perceptrons de múltiplas camadas)
- A MLP consiste em uma rede totalmente conectada com conexões feedforward
- É uma rede em que as camadas estão organizadas em uma ordem e os neurônios de uma camada estimulam todos os neurônios da camada seguinte (totalmente conectada)
- Nenhum neurônio pode estimular um neurônio da mesma camada ou de camadas anteriores (feedforward)

Arquitetura da MLP



- ▶ Uso – resolução de problemas difíceis e complexos
- ▶ Aprendizagem supervisionada - algoritmo de retropropagação do erro (Error Back-propagation, back-prop)

Características

- 1) Utiliza neurônios não lineares, com funções suaves (diferenciáveis)
- 2) Possui uma ou mais camadas escondidas
- 3) Alto grau de conectividade

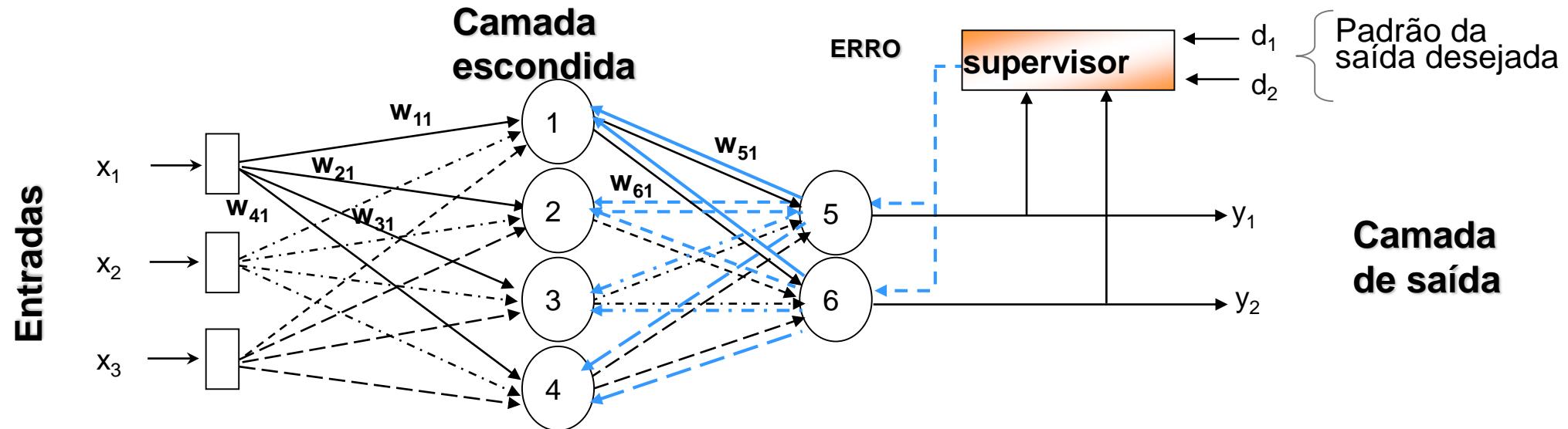
Algoritmo de Treinamento

- O algoritmo de treinamento de uma MLP mais famoso é o proposto por Rumelhart e McClelland em 1986, conhecido como back-propagation (retro-propagação)
- Este algoritmo consiste em um algoritmo supervisionado estático (não auto-organizável), ou seja, a arquitetura da rede deve ser previamente conhecida, e esta não é alterada durante o treinamento

Algoritmo de Treinamento

- Os únicos parâmetros alterados são os pesos da rede
- O algoritmo de treinamento consiste na generalização da Regra Delta
- A cada passo o algoritmo atualiza cada peso baseado na derivada parcial da função de erro com relação a este peso

Algoritmo Retropropagação do Erro



Passo 1: Ativação da rede (forward) - propagação da entrada para a saída

- › Os pesos não são alterados
- › Os sinais na camada são propagados camada por camada, neurônio por neurônio até que o vetor saída seja obtido na última camada

Passo 2: Retropoproragação (backward) - Propagação do erro e ajuste dos pesos

- › Iniciando na camada de saída
- › A resposta real da rede é subtraída da resposta desejada, para produzir um sinal de erro
- › Os pesos adaptados de acordo com a regra delta

Algoritmo Retropropagação do Erro

Valor Instantâneo do erro quadrático para o neurônio j :

$$\frac{1}{2} e_j^2(n)$$

Valor Instantâneo da soma dos erros quadráticos neurônios visíveis:

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

Erro Quadrático Médio para N padrões no conjunto de treinamento (função custo):

$$\varepsilon_{medio} = \frac{1}{N} \sum_{n=1}^N \varepsilon(n)$$

→ Aprendizagem ajusta os pesos e limiares para minimizar o erro quadrático médio (EQM)

Abordagem Padrão-a-Padrão

› Ajuste dos pesos como função dos erros calculados para cada padrão

› Usando Regra Delta:

$$w_j(n+1) = w_j(n) + \Delta w_j(n)$$

Gradiente Descendente:

$$\Delta w_j(n) = -\eta \frac{\partial \varepsilon(n)}{\partial w_j(n)}$$

$$\frac{\partial \varepsilon(n)}{\partial w_j(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_j(n)}$$

$$\frac{\partial \varepsilon(n)}{\partial w_j(n)} = -e_j(n) \varphi'_j(v_j(n)) v_i(n)$$

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \quad , j \text{ é o } \underline{\text{neurônio de saída}}$$

Algoritmo Backpropagation

(continuação)

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = -\sum_k e_k(n) \dot{\varphi}_k(v_k(n)) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n)$$

$$\delta_j(n) = \dot{\varphi}_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n) \quad , j \text{ é neurônio escondido}$$

$$\dot{\varphi}_j(v_j(n)) \quad \text{Depende apenas da função de ativação do neurônio}$$

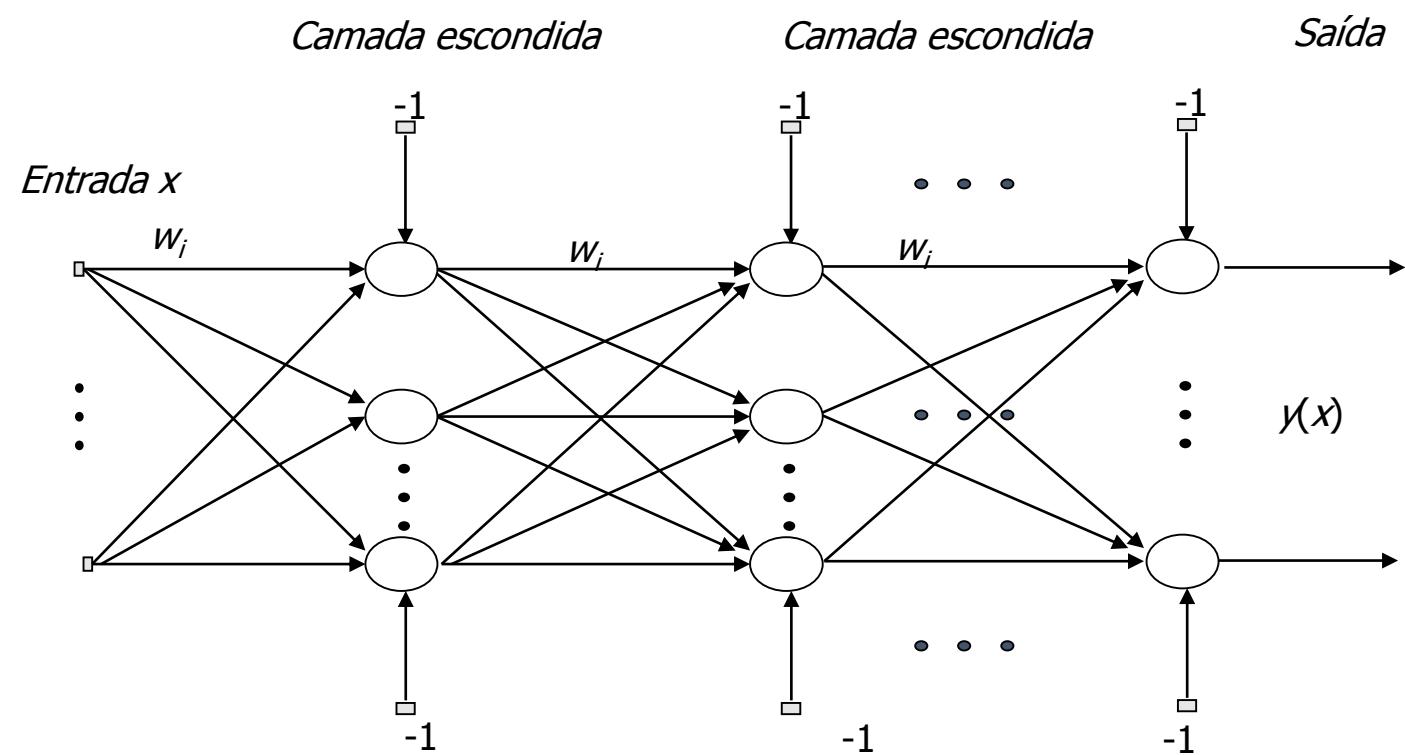
$$\delta_k(n) \quad \text{Requer conhecimento do sinal de erro para todos os neurônios na camada à direita da camada do neurônio } j$$

Regra de Atualização dos Pesos

$$\begin{pmatrix} \text{Correção} \\ \text{do Peso} \\ \Delta w_{ij}(n) \end{pmatrix} = \begin{pmatrix} \text{Taxa de} \\ \text{Aprendizagem} \\ \eta \end{pmatrix} \begin{pmatrix} \text{Gradiente} \\ \text{Local} \\ \delta_j(n) \end{pmatrix} \begin{pmatrix} \text{Sinal de} \\ \text{Entrada} \\ y_i(n) \end{pmatrix}$$

1. $\delta_j(n) = e_j(n)\varphi'_j(v_j(n))$ *j é um neurônio na camada de saída*
2. Se j é um neurônio escondido:
O gradiente é o produto da derivada da função de ativação pela soma dos gradientes computados nos neurônios da camada imediatamente à direita da camada do neurônio j

Regra de Atualização dos Pesos



- Ativação:

$$v_j(n) = \sum_{i=1}^p x_i w_{ij} + b_j$$

$$y_j(n) = \varphi_j(v_j(n))$$

- Aprendizado:

$$w_{ji}^L(n+1) = w_{ji}^L(n) + \eta \delta_j^L(n) y_i^{L-1}(n)$$

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial v_j(n)} \quad \varepsilon(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

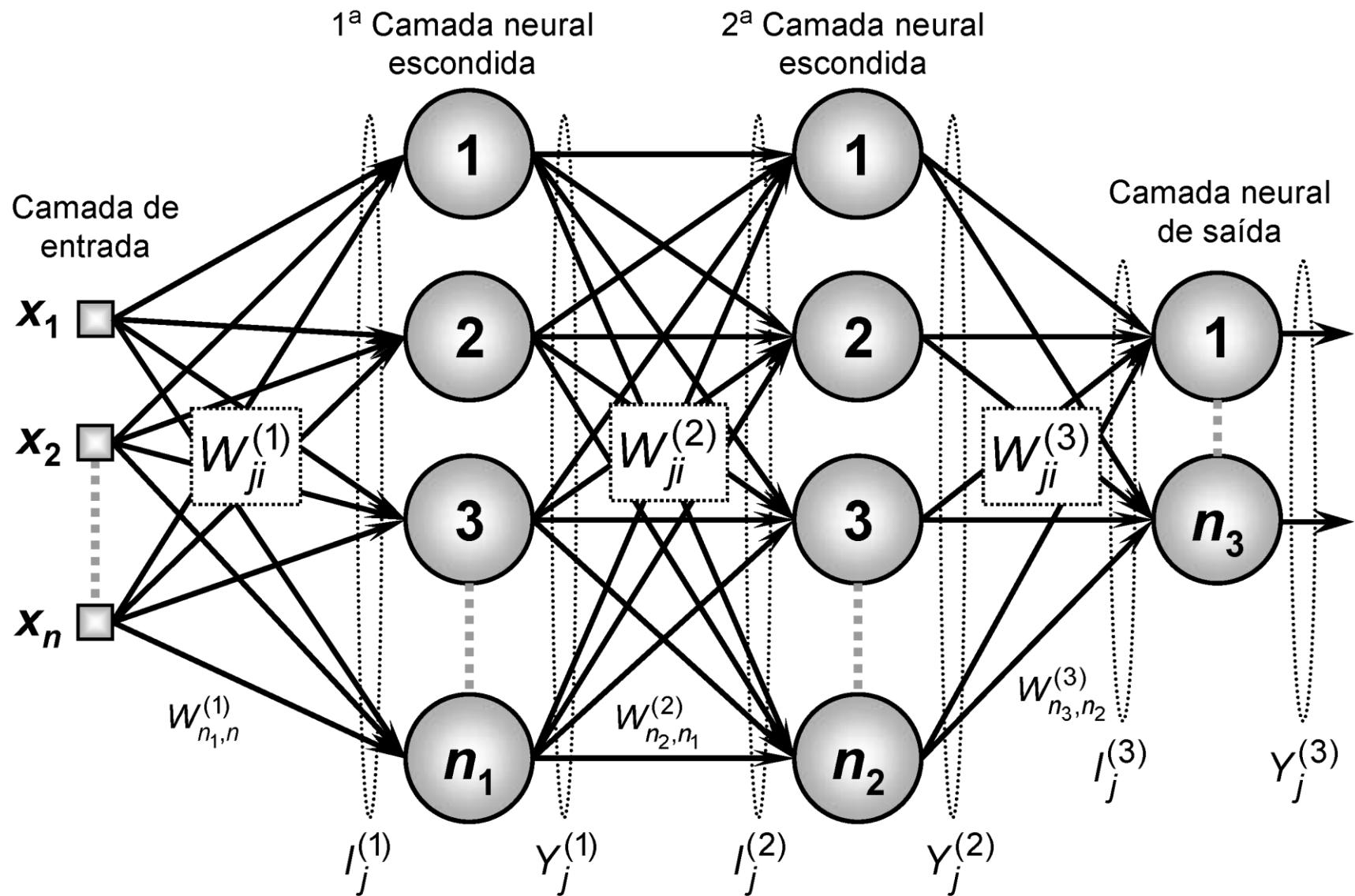
- Última Camada

$$\delta_j(n) = -e_j(n) \varphi'_j(v_j(n))$$

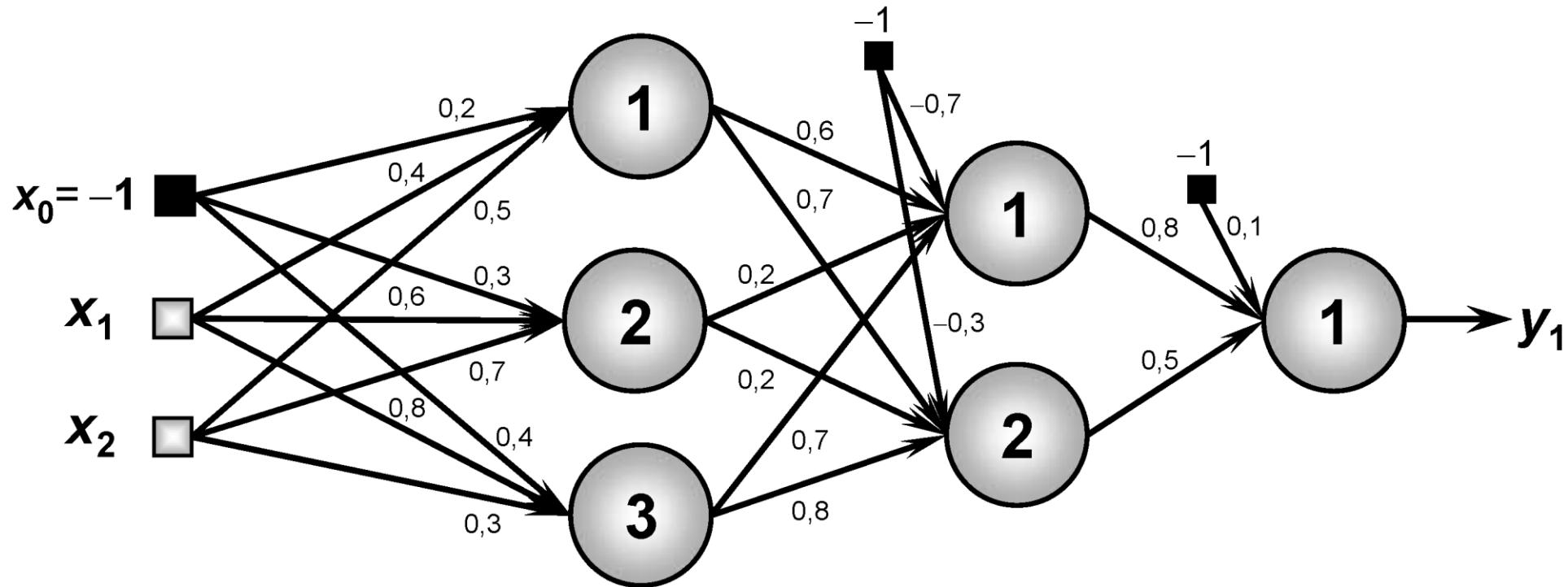
- Camadas Escondidas

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

Backpropagation – Arquitetura Exemplo



Backpropagation – Exemplo



$$W_{ji}^{(1)} = \begin{bmatrix} 0.2 & 0.4 & 0.5 \\ 0.3 & 0.6 & 0.7 \\ 0.4 & 0.8 & 0.3 \end{bmatrix} \quad W_{ji}^{(2)} = \begin{bmatrix} -0.7 & 0.6 & 0.2 & 0.7 \\ -0.3 & 0.7 & 0.2 & 0.8 \end{bmatrix} \quad W_{ji}^{(3)} = [0.1 \quad 0.8 \quad 0.5]$$

Backpropagation – Exemplo

Exemplo, $x_1 = 0,3$ e $x_2 = 0,7$:

$$I_j^{(1)} = \begin{bmatrix} I_1^{(1)} \\ I_2^{(1)} \\ I_3^{(1)} \end{bmatrix} = \begin{bmatrix} W_{1,0}^{(1)} \cdot x_0 + W_{1,1}^{(1)} \cdot x_1 + W_{1,2}^{(1)} \cdot x_2 \\ W_{2,0}^{(1)} \cdot x_0 + W_{2,1}^{(1)} \cdot x_1 + W_{2,2}^{(1)} \cdot x_2 \\ W_{3,0}^{(1)} \cdot x_0 + W_{3,1}^{(1)} \cdot x_1 + W_{3,2}^{(1)} \cdot x_2 \end{bmatrix} = \begin{bmatrix} (0,2 \cdot (-1)) + (0,4 \cdot 0,3) + (0,5 \cdot 0,7) \\ (0,3 \cdot (-1)) + (0,6 \cdot 0,3) + (0,7 \cdot 0,7) \\ (0,4 \cdot (-1)) + (0,8 \cdot 0,3) + (0,3 \cdot 0,7) \end{bmatrix} = \begin{bmatrix} 0,27 \\ 0,37 \\ 0,05 \end{bmatrix}$$

$$Y_j^{(1)} = \begin{bmatrix} Y_1^{(1)} \\ Y_2^{(1)} \\ Y_3^{(1)} \end{bmatrix} = \begin{bmatrix} g(I_1^{(1)}) \\ g(I_2^{(1)}) \\ g(I_3^{(1)}) \end{bmatrix} = \begin{bmatrix} \tanh(0,27) \\ \tanh(0,37) \\ \tanh(0,05) \end{bmatrix} = \begin{bmatrix} 0,26 \\ 0,35 \\ 0,05 \end{bmatrix} \xrightarrow{Y_0^{(1)} = -1} Y_j^{(1)} = \begin{bmatrix} Y_0^{(1)} \\ Y_1^{(1)} \\ Y_2^{(1)} \\ Y_3^{(1)} \end{bmatrix} = \begin{bmatrix} -1 \\ 0,26 \\ 0,35 \\ 0,05 \end{bmatrix}$$

tanh é a tangente hiperbólica e os argumentos estão em radianos.

Backpropagation – Exemplo

Exemplo, $x_1 = 0,3$ e $x_2 = 0,7$:

$$l_j^{(2)} = \begin{bmatrix} l_1^{(2)} \\ l_2^{(2)} \end{bmatrix} = \begin{bmatrix} W_{1,0}^{(2)} \cdot x_0 + W_{1,1}^{(2)} \cdot x_1 + W_{1,2}^{(2)} \cdot x_2 + W_{1,3}^{(2)} \cdot x_3 \\ W_{2,0}^{(2)} \cdot x_0 + W_{2,1}^{(2)} \cdot x_1 + W_{2,2}^{(2)} \cdot x_2 + W_{2,3}^{(2)} \cdot x_3 \end{bmatrix} = \begin{bmatrix} 0,96 \\ 0,59 \end{bmatrix}$$

$$Y_j^{(2)} = \begin{bmatrix} Y_1^{(2)} \\ Y_2^{(2)} \end{bmatrix} = \begin{bmatrix} g(l_1^{(2)}) \\ g(l_2^{(2)}) \end{bmatrix} = \begin{bmatrix} \tanh(0,96) \\ \tanh(0,59) \end{bmatrix} = \begin{bmatrix} 0,74 \\ 0,53 \end{bmatrix} \xrightarrow{Y_0^{(2)} = -1} Y_j^{(2)} = \begin{bmatrix} Y_0^{(2)} \\ Y_1^{(2)} \\ Y_2^{(2)} \end{bmatrix} = \begin{bmatrix} -1 \\ 0,74 \\ 0,53 \end{bmatrix}$$

tanh é a tangente hiperbólica e os argumentos estão em radianos.

Backpropagation – Exemplo

Exemplo, $x_1 = 0,3$ e $x_2 = 0,7$:

$$l_j^{(3)} = [l_1^{(3)}] = [W_{1,0}^{(3)} \cdot x_0 + W_{1,1}^{(3)} \cdot x_1 + W_{1,2}^{(3)} \cdot x_2] = [0,76]$$

$$Y_j^{(3)} = [0,76] = [g(l_1^{(3)})] = [\tanh(0,76)] = [0,64]$$

Encontrado o y_1 .

Próximo passo é medir o desvio para iniciar a derivação do **backpropagation**.

$$\delta_j^{(3)} = (d_j - Y_j^{(3)}) \cdot g'(l_j^{(3)})$$

$$W_{ji}^{(3)}(t+1) = W_{ji}^{(3)}(t) + \eta \cdot \delta_j^{(3)} Y_i^{(2)}$$

Backpropagation – Exemplo

Camada Intermediária

$$\delta_j^{(2)} = \left(\sum_{k=1}^{n_3} \delta_k^{(3)} \cdot W_{kj}^{(3)} \right) \cdot g'(l_j^{(2)})$$

$$W_{ji}^{(2)}(t+1) = W_{ji}^{(2)}(t) + \eta \cdot \delta_j^{(2)} Y_i^{(1)}$$

Camada Inicial

$$\delta_j^{(1)} = \left(\sum_{k=1}^{n_2} \delta_k^{(2)} \cdot W_{kj}^{(2)} \right) \cdot g'(l_j^{(1)})$$

$$W_{ji}^{(1)}(t+1) = W_{ji}^{(1)}(t) + \eta \cdot \delta_j^{(1)} x_i$$

Algoritmo

Fase de Operação e Treinamento:

Início {Algoritmo PMC – Fase de Operação}

- <1> Obter uma amostra { x };
- <2> Assumir $W_{ji}^{(1)}$, $W_{ji}^{(2)}$ e $W_{ji}^{(3)}$ já ajustadas no treinamento;
- <3> Execute as seguintes instruções:
 - <3.1> Obter $I_j^{(1)}$ e $Y_j^{(1)}$; {conforme (5.1) e (5.4)}
 - <3.2> Obter $I_j^{(2)}$ e $Y_j^{(2)}$; {conforme (5.2) e (5.5)}
 - <3.3> Obter $I_j^{(3)}$ e $Y_j^{(3)}$; {conforme (5.3) e (5.6)}
- <4> Disponibilizar as saídas da rede, as quais são dadas pelos elementos contidos em $Y_j^{(3)}$

Fim {Algoritmo PMC – Fase de Operação}

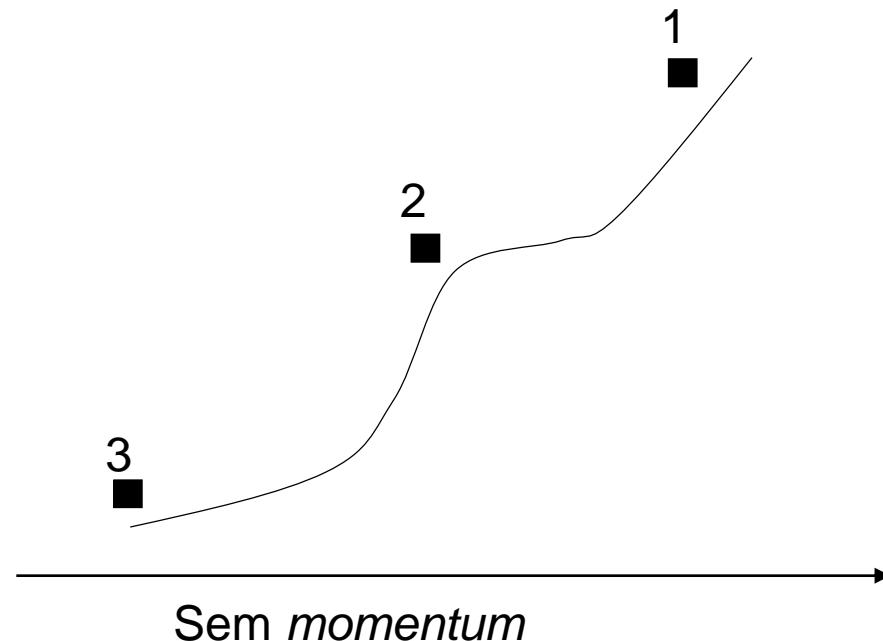
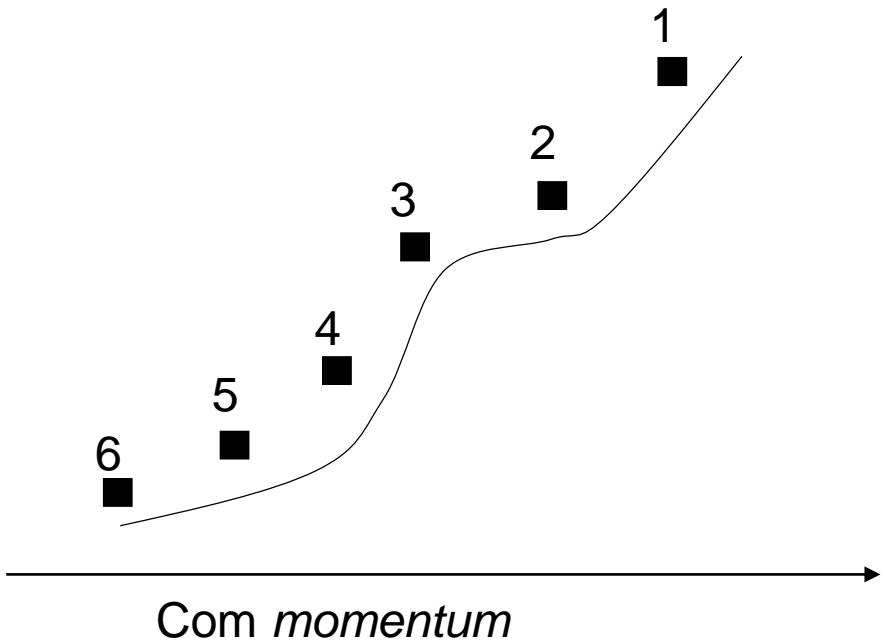
Início {Algoritmo PMC – Fase de Treinamento}

- <1> Obter o conjunto de amostras de treinamento { $x^{(k)}$ };
- <2> Associar o vetor de saída desejada { $d^{(k)}$ } para cada amostra;
- <3> Iniciar $w_{ji}^{(1)}$, $W_{ji}^{(2)}$ e $W_{ji}^{(3)}$ com valores aleatórios pequenos;
- <4> Especificar taxa de aprendizagem { η } e precisão requerida { ε };
- <5> Iniciar o contador de número de épocas {época $\leftarrow 0$ };
- <6> Repetir as instruções:
 - <6.1> $E_M^{\text{anterior}} \leftarrow E_M$; {conforme (5.8)}
 - <6.2> Para todas as amostras de treinamento { $x^{(k)}$, $d^{(k)}$ }, fazer:
 - <6.2.1> Obter $I_j^{(1)}$ e $Y_j^{(1)}$; {conforme (5.1) e (5.4)}
 - <6.2.2> Obter $I_j^{(2)}$ e $Y_j^{(2)}$; {conforme (5.2) e (5.5)}
 - <6.2.3> Obter $I_j^{(3)}$ e $Y_j^{(3)}$; {conforme (5.3) e (5.6)}
 - <6.2.4> Determinar $\delta_j^{(3)}$; {conforme (5.15)}
 - <6.2.5> Ajustar $W_{ji}^{(3)}$; {conforme (5.17)}
 - <6.2.6> Determinar $\delta_j^{(2)}$; {conforme (5.26)}
 - <6.2.7> Ajustar $W_{ji}^{(2)}$; {conforme (5.28)}
 - <6.2.8> Determinar $\delta_j^{(1)}$; {conforme (5.37)}
 - <6.2.9> Ajustar $W_{ji}^{(1)}$; {conforme (5.39)}
 - <6.3> Obter $Y_j^{(3)}$ ajustado; {conforme <6.2.1>, <6.2.2> e <6.2.3>}
 - <6.4> $E_M^{\text{atual}} \leftarrow E_M$; {conforme (5.8)}
 - <6.5> $\text{época} \leftarrow \text{época} + 1$;
Até que: $|E_M^{\text{atual}} - E_M^{\text{anterior}}| \leq \varepsilon$

Fim {Algoritmo PMC – Fase de Treinamento}

Momentum

- ▶ É uma técnica bem simples e que diminui o tempo de treinamento de uma rede
- ▶ Foi introduzido não apenas para que o algoritmo convergisse mais rápido, mas também para se evitar problemas como por exemplo, onde a derivada é zero e os mínimos locais



Passo para trás (*Backward Pass*) ou Retropropagação do erro:

- › Iniciando na camada de saída, calcula-se o gradiente local neurônio
- › Os pesos passam por mudanças de acordo com a regra delta
- › O vetor de entrada é fixo durante todo o processo

Regra Delta Generalizada:

$$\Delta w_{ji}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

$\alpha > 0$, momento: aumenta a taxa de aprendizagem e diminui o risco de instabilidade

Modos de Aprendizagem

1) Padrão:

- › Pesos atualizados a cada padrão apresentado (época)
- › Apresentação aleatória do padrões

2) Batch:

- Pesos atualizados após a apresentação de todos os padrões

$$\mathcal{E}_{medio} = \frac{1}{2N} \sum_{n=1}^N \sum_{j \in c} e_j^2(n) \quad , N \text{ é o número de padrões}$$

$$\Delta w_{ji} = -\eta \frac{\partial \mathcal{E}_{medio}}{\partial w_{ji}} = -\frac{\eta}{N} \sum_{n=1}^N e_j(n) \frac{\partial e_j(n)}{\partial w_{ji}}$$

Resumo da Aprendizagem

- › A rede aprende um conjunto predefinido de pares de exemplos de entrada/saída em ciclos de propagação/adaptação.
- › A entrada é propagada por cada uma das outras camadas até que a saída seja gerada.
- › A saída é comparada com a saída desejada e um sinal de erro é calculado para cada elemento de saída

Resumo da Aprendizagem

- › O sinal de erro é retro-propagado da camada de saída para cada elemento da camada intermediária anterior que contribui diretamente para a formação da saída

- › Cada elemento da camada intermediária recebe apenas uma porção do sinal de erro total, proporcional apenas à contribuição relativa de cada elemento na formação da saída original

Resumo da Aprendizagem

- O processo se repete, camada por camada, até que cada elemento da rede receba um sinal de erro que descreva sua contribuição relativa para o erro total
- Os pesos das conexões são então atualizados para cada elemento de modo a fazer a rede convergir para um estado que permita a codificação de todos os padrões do conjunto de treinamento

Resumo do Algoritmo

1. Inicialização do pesos e limiares (números aleatórios pequenos uniformemente distribuídos)

Obs.: A escolha dos valores iniciais é importante para acelerar o processo. Se houver conhecimento a priori disponível

2. Apresentação dos exemplos de treinamento (em ordem ou aleatoriamente) à primeira camada

3. Ativação da rede (feedforward)

4. Calcular os gradientes e atualização dos pesos e limiares

5. Nova iteração (se a condição de parada não for satisfeita)

Observações

Taxa de aprendizagem η

pequena → aprendizagem lenta

grande → aprendizagem mais rápida, mas pode haver instabilidade

- Uso da constante de momento → acelera o processo e diminui a instabilidade (regra Delta Generalizada)
- η não precisa ser constante para todos os neurônios.
- Deseja-se que os neurônios aprendam na mesma taxa. As últimas camadas tendem a ter valores maiores para os gradientes locais, portanto, deveria ser menor nestas camadas

η

Observações

Inicialização:

- › Escolha dos valores iniciais influencia o comportamento da rede
- › Na ausência de conhecimento a priori, os pesos e bias devem ser inicializados para valores aleatórios pequenos uniformemente distribuídos
- › Saturação prematura - o erro se mantém quase constante durante um certo número de épocas (não é um mínimo local)
- › Saturação - saída do neurônio é igual a um dos limites da função
- › A saturação incorreta (saída igual +1 e saída desejada -1)
 - › É menos provável quando o no. de neurônios é pequeno
 - › Não ocorre quando os neurônios da rede operam na faixa linear

Generalização

No treinamento de uma rede perceptron em camadas múltiplas:

- › Necessita-se de um conjunto de pares de entrada / saídas desejadas
- › Na prática os dados disponíveis devem ser divididos (aleatoriamente) em:

Treinamento

- › Usados para computar os pesos e bias da rede

Teste

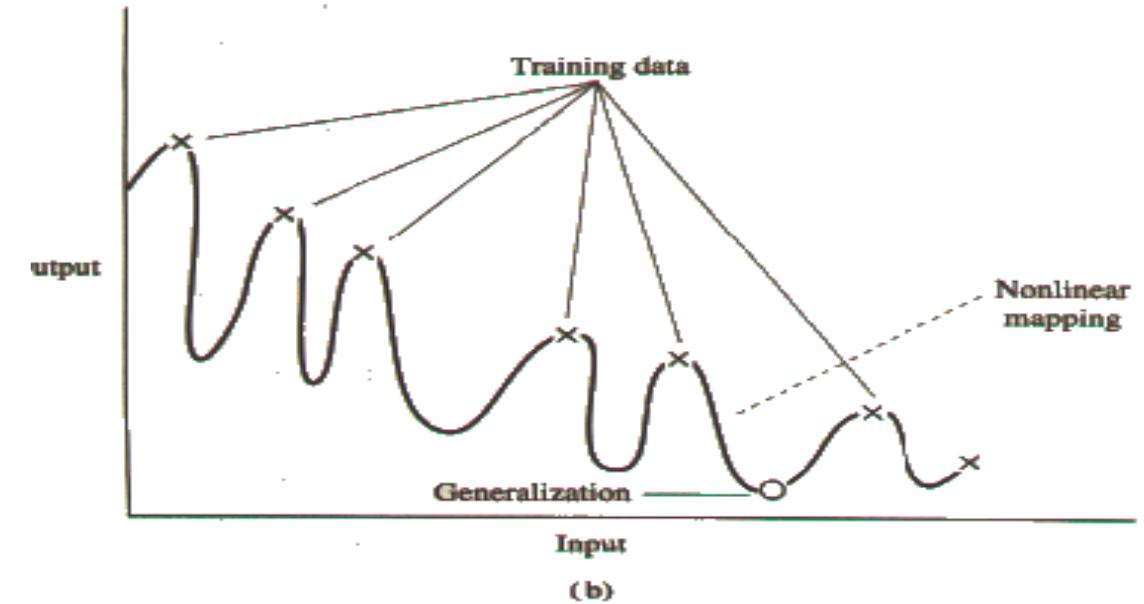
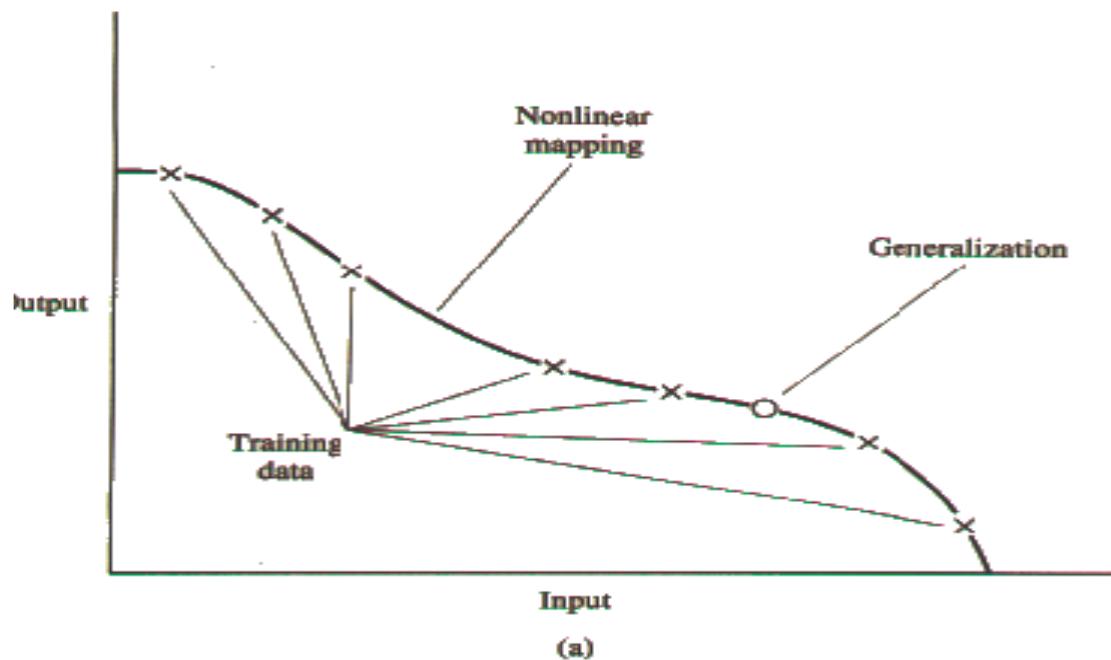
- › Usados para testar a capacidade de aprendizagem da rede (generalização)
- › Generalização boa → a rede responde adequadamente para dados não utilizados

Generalização

Processo de aprendizagem como um processo de ajuste de curva:

rede → mapeamento não linear da entrada e saída

generalização → efeito de uma boa interpolação não linear de dados.



17 (a) Properly fitted data (good generalization). (b) Overfitted data

Generalização

Generalização boa:

- mapeamento correto, inclusive na presença de pequenas alterações

Generalização ruim:

- ocorre quando a rede aprende muitas relações entre as entradas e as saídas (ou seja, foram super treinadas), a rede pode memorizar os dados de entrada e com isso diminui a capacidade de generalização para padrões de entrada/saída não observados no treinamento.

Fatores que influenciam a generalização:

- tamanho e eficiência do conjunto de treinamento (controlável)
- arquitetura da rede (controlável)
- complexidade física do problema a ser resolvido (não há controle)

Perspectivas

- ▶ Arquitetura fixa (compatível com a complexidade do problema) e determina-se o tamanho do conjunto de treinamento adequado para uma generalização boa (mais comum)
- ▶ Não há receita
- ▶ Na prática pode haver grandes discrepâncias entre a necessidade real e a estimativa feita pela expressão acima
- ▶ O tamanho do conjunto de treinamento é fixo e determina-se a melhor arquitetura da rede para alcançar um generalização boa.
- ▶ Envolve várias alterações de arquitetura e re-treinamentos

Vantagens e Desvantagens

- › Propriedades:
 - › Computação local simples
 - › Realiza uma busca por gradiente descendente estocástico no espaço de pesos
- › Conexionismo
 - › Similar ao sistema neurofisiológico no funcionamento
- › Unidades escondidas
 - › Papel fundamental na aprendizagem
 - › Agem como detectores de características
- › Convergência lenta
 - › Superfície de erro muito plana
 - › Gradiente apontando para direções opostas à do mínimo → é caro computacionalmente



Exemplo

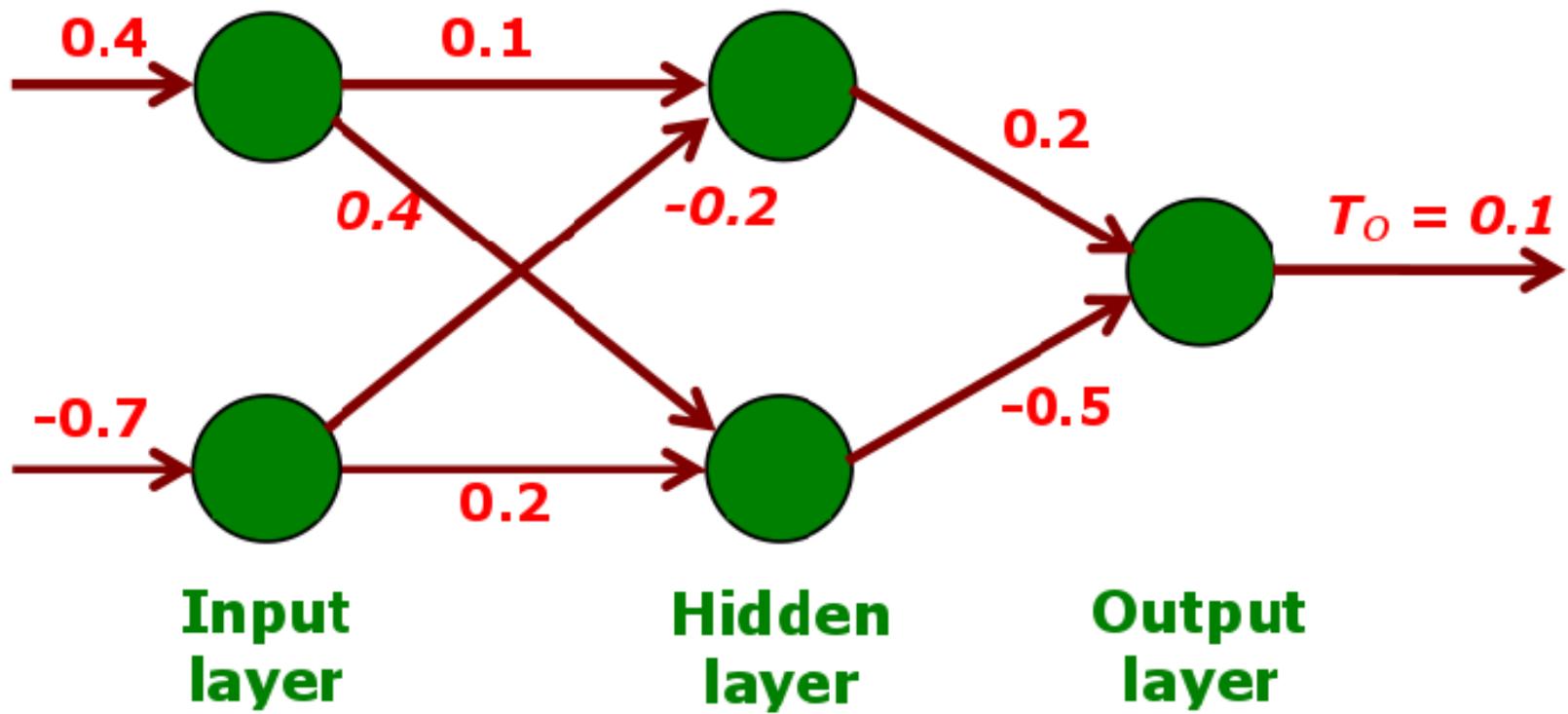
Considere o uso de uma Rede Neural Artificial para solucionar um problema de regressão. Os dados de treinamento são 5 amostras, conforme a tabela abaixo.

Amostra	Atributo 1	Atributo 2	Atributo Alvo (Saída)
1	0.4	-0.7	0.1
2	0.3	-0.5	0.05
3	0.6	0.1	0.3
4	0.2	0.4	0.25
5	0.1	-0.2	0.12



Exemplo

A topologia da rede apresenta dois neurônios na camada escondida e os pesos já foram definidos, conforme a figura abaixo. Para o exercício foi utilizada uma taxa de aprendizado (η) de 0.6.



AI

Exemplo

1. Estimular os neurônios da camada escondida com primeira amostra de treinamento (Equação 5.1 dos slides):

$$I_j^{(1)} = \sum_{i=0}^n w_{ji}^{(1)} x_i$$

$$I_j^{(1)} = \begin{bmatrix} 0.1 & -0.2 \\ -0.4 & 0.2 \end{bmatrix} \begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix} = \begin{bmatrix} 0.18 \\ 0.02 \end{bmatrix}$$

AI Exemplo

2. Utilizar a função de ativação (função logística) para combinar os estímulos (Equação 5.4 dos slides):

$$Y_j^{(1)} = g(I_{ji}^{(1)})$$

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

$$Y_1^{(1)} = \begin{bmatrix} \frac{1}{(1+e^{-(0.18)})} \\ \frac{1}{(1+e^{-(0.02)})} \end{bmatrix} = \begin{bmatrix} 0.5448 \\ 0.505 \end{bmatrix}$$

AI

Exemplo

3. Combinar as saídas dos neurônios das camadas intermediárias (passo anterior) com os pesos até a saída (Equação 5.2):

$$I_j^{(2)} = \sum_{i=0}^n w_{ji}^{(2)} x_i$$

$$I_j^{(2)} = [0.2 \quad -0.5] \begin{bmatrix} 0.5448 \\ -0.505 \end{bmatrix} = [-0.1435]$$



Exemplo

4. Utilizar a função de ativação para a obtenção da saída predita (Equação 5.5):

$$Y_j^{(2)} = g(I_{ji}^{(2)})$$

$$Y_j^{(2)} = \left[\frac{1}{(1+e^{-(0.1435)})} \right] = [0.4642]$$



Exemplo

5. Calculando o erro da predição (Equação 5.8):

$$E_M = \frac{1}{p} \sum_{k=1}^p E(k)$$

$$E_M = (0.1 - 0.4642)^2 = 0.13264$$

AI

Exemplo

6. Encontrando o δ (Equação 5.15) usando abordagens de diferenças, considere T_o como o valor desejado:

$$\delta_j^{(2)} = (T_o - Y_j^{(2)})(Y_j^{(2)})\left(1 - Y_j^{(2)}\right)$$

$$\delta_j^{(2)} = (0.1 - 0.4642)(0.4642)(0.5358) = -0.09054$$

$$[Y] = (Y_1^{(1)})(\delta_j^{(2)}) = \begin{bmatrix} 0.5448 \\ 0.505 \end{bmatrix} (-0.0905) = \begin{bmatrix} -0.0493 \\ -0.0457 \end{bmatrix}$$

AI Exemplo

7. Encontrando o novo peso da última camada:

$$w_{ji}^{(2)} = w_{ji}^{(2)} + \eta \delta_j^{(2)}$$

$$w_{ji}^{(2)} = \begin{bmatrix} 0.5448 \\ 0.505 \end{bmatrix} + (0.6) \begin{bmatrix} -0.0493 \\ -0.0457 \end{bmatrix} = \begin{bmatrix} -0.0295 \\ -0.0274 \end{bmatrix}$$

AI

Exemplo

8. Encontrando o δ para a camada anterior:

$$\delta_j^{(1)} = \begin{bmatrix} (-0.01811)(0.5448)(1 - 0.5448) \\ (-0.00452)(0.505)(1 - 0.505) \end{bmatrix} = \begin{bmatrix} -0.00449 \\ -0.01132 \end{bmatrix}$$

9. Encontrando novos pesos:

$$w_{ji}^{(1)} = \begin{bmatrix} 0.4 \\ -0.7 \end{bmatrix} + (0.6) \begin{bmatrix} -0.00449 & 0.01132 \end{bmatrix} = \begin{bmatrix} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{bmatrix}$$

AI Exemplo

10. Ajustando os pesos da primeira camada:

$$w_{ji}^{(1)} = \begin{bmatrix} 0.1 & 0.4 \\ -0.2 & 0.2 \end{bmatrix} + \begin{bmatrix} -0.001077 & 0.002716 \\ 0.001885 & -0.004754 \end{bmatrix} = \begin{bmatrix} -0.0989 & 0.0402 \\ 0.1981 & -0.1952 \end{bmatrix}$$

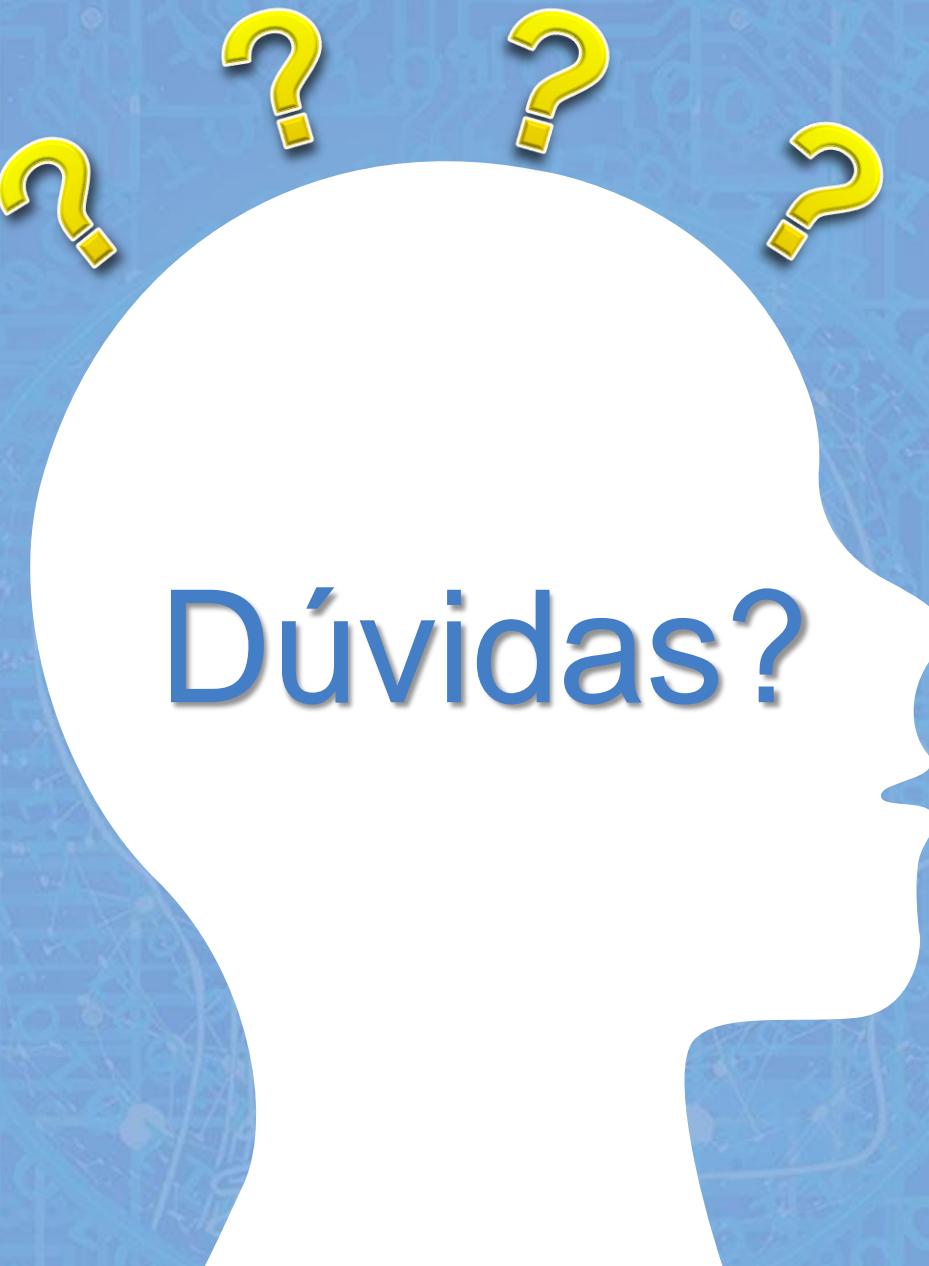
$$w_{ji}^{(2)} = \begin{bmatrix} 0.2 \\ -0.5 \end{bmatrix} + \begin{bmatrix} -0.0295 \\ -0.0274 \end{bmatrix} = \begin{bmatrix} 0.1704 \\ -0.0527 \end{bmatrix}$$

AI Exemplo

11. Com os pesos atualizados o erro é calculado novamente e uma nova amostra é treinada.
12. Toda a base deve ser treinada até ser atingido o nível de “tolerância” para o erro.

Reference:

http://www.myreaders.info/03_Back_Propagation_Network.pdf

A white silhouette of a human head profile facing right, centered against a blue background with a faint circuit board pattern. Four yellow question marks are floating around the top of the head.

Dúvidas?



Referências Bibliográficas

- **José Demídio Simões da Silva – Notas de Aula**
- **Ana Paula A. C. Shiguemori – Notas de Aula**
- **Elcio Hideiti Shiguemori – Notas de Aula**