Universidade Tecnológica Federal do Paraná - Campus Curitiba

IA- Redes Neurais

Treinamento em Redes Neurais

Prof. Myriam Delgado

Redes Neurais Artificiais

- Modelo do neurônio: função de ativação
- Arquitetura da rede
- **Treinamento**

Aprendizado Conexionista

Aprendizado Conexionista é o processo no qual os parâmetros livres de uma Rede Neural Artificial (RNA) são alterados pela estimulação contínua causada pelo ambiente no qual a rede está inserida, da seguinte forma:

Estímulo → Adaptação → Novo comportamento

Redes Neurais Artificiais: Treinamento

O aprendizado em RNAs, também chamado de **treinamento**, é o processo iterativo de **ajuste dos parâmetros** da rede.

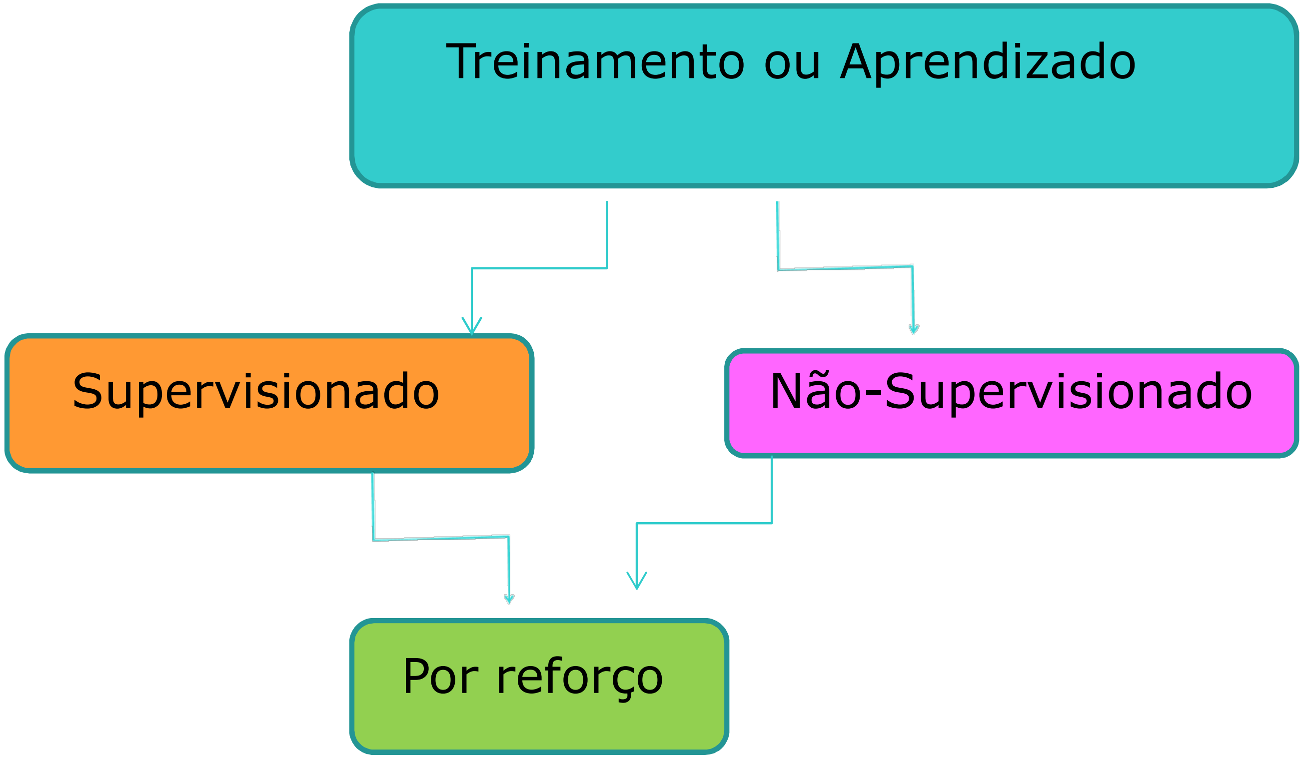
De forma geral o aprendizado em RNAs pode ser classificado em duas categorias principais (que resultam numa terceira – combinação das duas):

- Supervisionado

Por reforço

- Não-supervisionado

Redes Neurais Artificiais: Treinamento



Redes Neurais Artificiais: Aprendizado

Aprendizado Supervisionado:

Dados de entrada e saída desejada são fornecidos à rede
Há supervisor externo (professor)
Exemplos: Regra Delta, Backpropagation

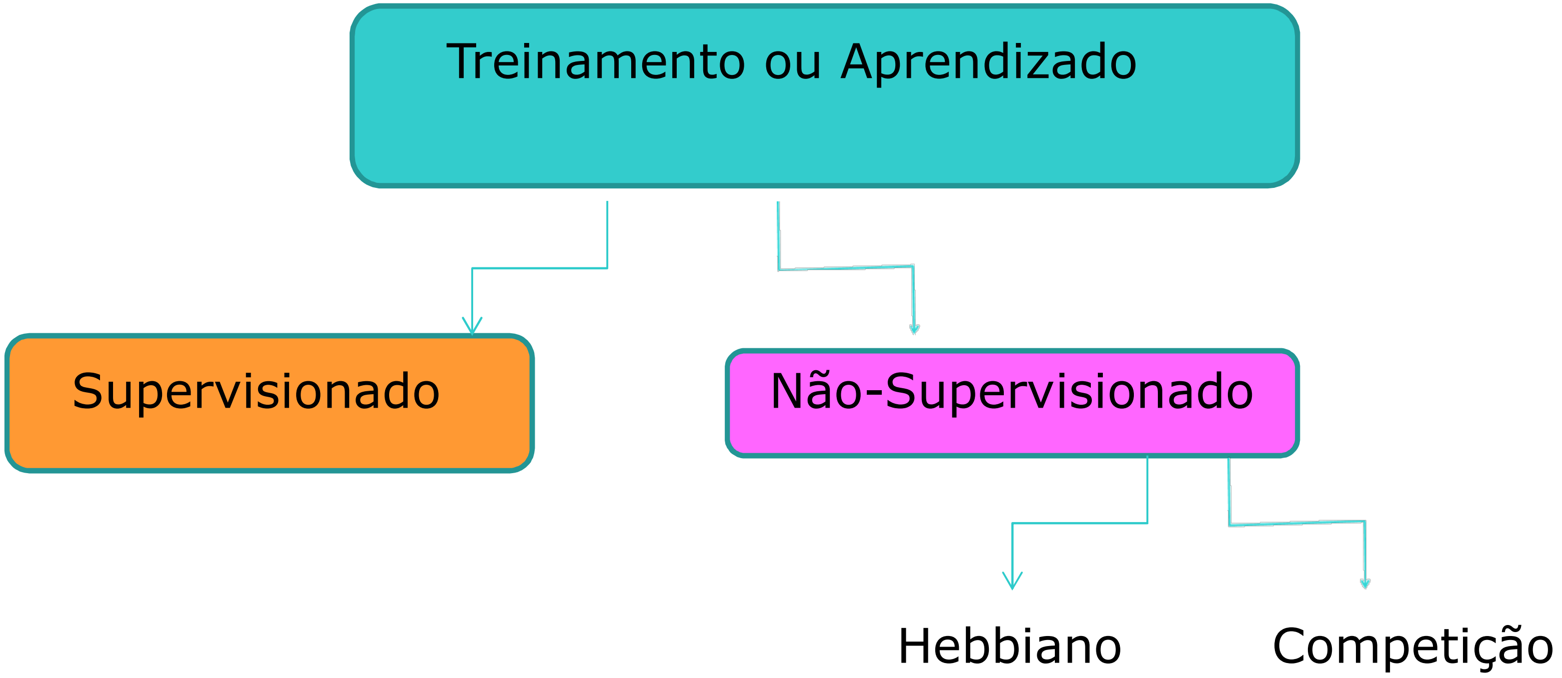
Aprendizado Não-supervisionado:

- Dados de entrada são fornecidos à rede
- Não há supervisor externo (professor)
- Exemplos: Regra de Hebb, Aprendizado por Competição

Aprendizado por Reforço:

Não há supervisor externo (professor) mas cada ação (resposta da rede) pode ser avaliada como positiva ou negativa, podendo receber recompensa ou punição, respectivamente. Exemplo: aprendizado por evolução

Redes Neurais Artificiais: Treinamento



Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado: Este tipo de aprendizado só é possível quando há redundância nos dados de entrada.

Neste tipo a rede estabelece uma harmonia com as regularidades estatísticas das entradas sendo então capaz de formar representações internas para codificar características da entrada.

Há dois modelos principais de aprendizado não-supervisonado:

Hebbiano

Por competição

Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado Hebbiano:

Baseado na observação de sistemas biológicos:
“quando um neurônio contribui para o disparo
de outro, a conexão entre eles se fortalece”

O aprendizado Hebbiano tem aplicação em diferentes
modelos de rede.

O efeito de fortalecer a conexão entre dois neurônios pode
ser simulado matematicamente ajustando o peso de forma
proporcional ao sinal do produto entre suas saídas (ou entre
a saída do posterior e sua entrada).

$$w_{kj}(t+1) = w_{kj}(t) + \alpha(x_k o_k)$$

Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado Hebbiano:

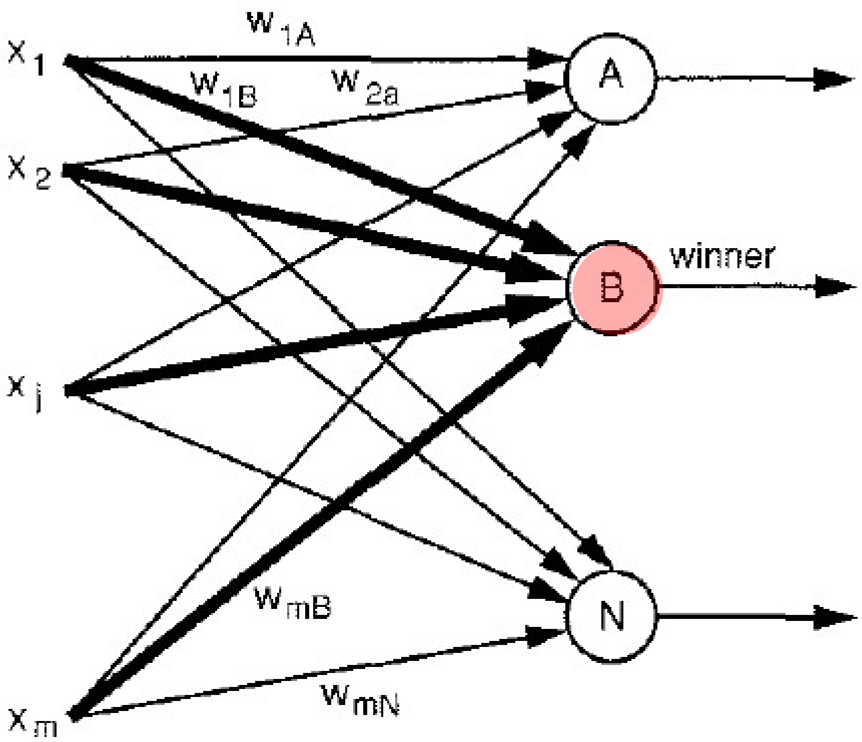
Regra de Hebb

Oj	Ok	$\Delta w_{kj}=\alpha OjOk$
+	+	+
+	-	-
-	+	-
-	-	+

Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado Por Competição:

Baseado na estratégia "winner-takes-all"



$(\mathbf{x} \cdot \mathbf{w}_B) > (\mathbf{x} \cdot \mathbf{w}_A)$

...

$(\mathbf{x} \cdot \mathbf{w}_B) > (\mathbf{x} \cdot \mathbf{w}_N)$

Luger, 2005

Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado Por Competição:
o ajuste leva o peso do vencedor (k) na direção da entrada

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha\left(\mathbf{x}_k - \mathbf{w}_k(t)\right)$$

Se o vetor $\mathbf{w}_k$ é normalizado, a definição do neurônio vencedor pode ser calculada de duas formas:

Produto escalar ou interno ($\mathbf{x} \cdot \mathbf{w}_k$)

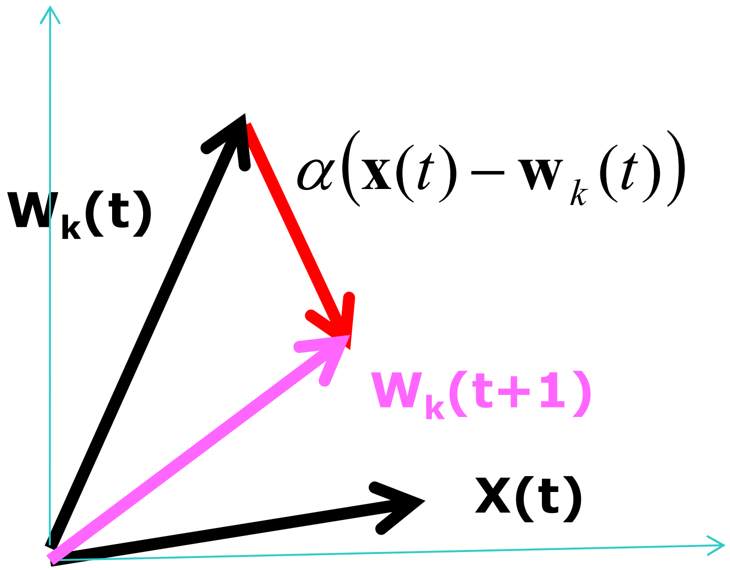
Norma Euclidiana (vale também para $\mathbf{w}_k$ não normalizado)

Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado Por Competição:

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha\big(\mathbf{x}(t) - \mathbf{w}_k(t)\big)$$

K: winner



Prof. Fernando J. Von Zuben
DCA/FEEC/Unicamp

Redes Neurais Artificiais: Treinamento

Treinamento ou Aprendizado

Supervisionado

RNA: Aprendizado Supervisionado

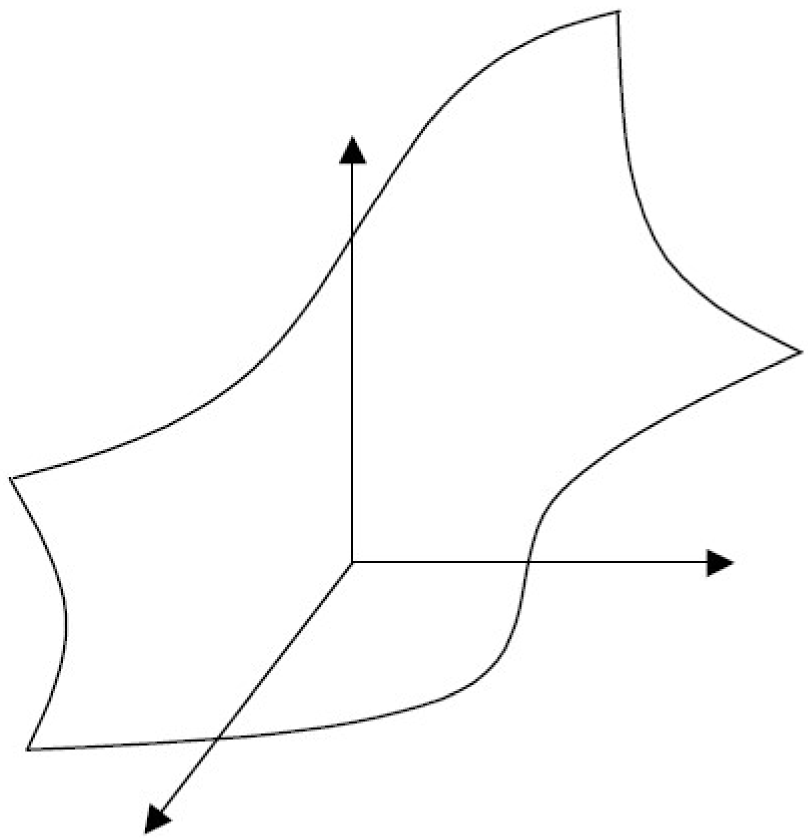
- Aprendizado Supervisionado e o Processo de Mapeamento Entrada Saída.

Em geral, três mapeamentos estão envolvidos:

- Mapeamento a ser aproximado (onde são conhecidos os dados amostrados)
- Mapeamento produzido pela rede
- Superfície de erros

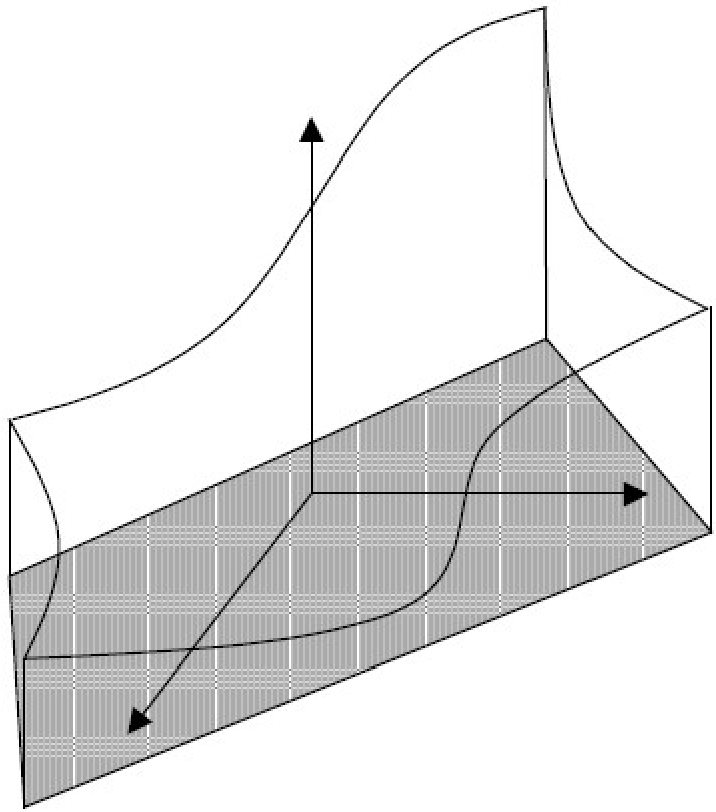
Aprendizado Supervisionado: Mapeamento I/O

Mapeamento a ser aproximado



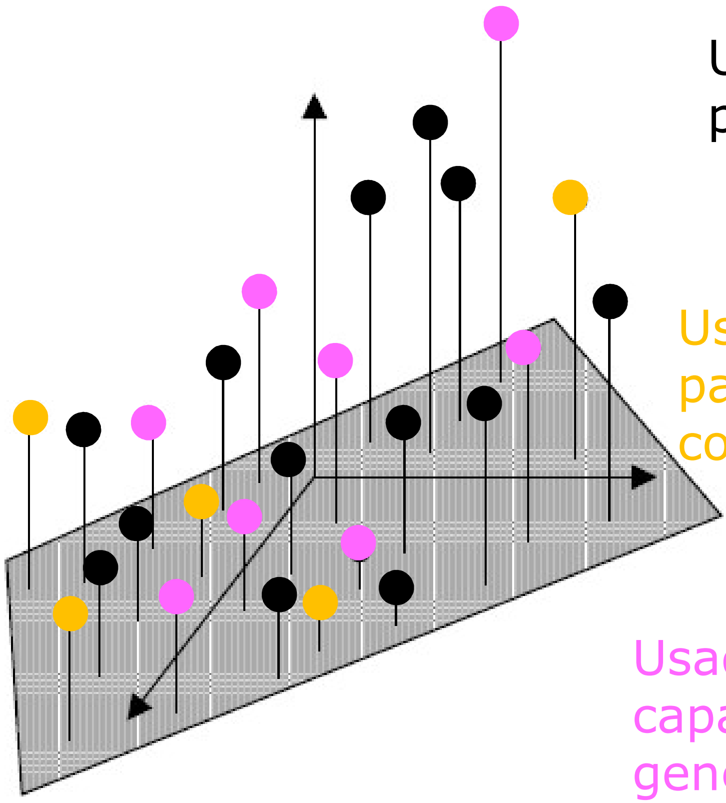
Aprendizado Supervisionado: Mapeamento I/O

Mapeamento a ser aproximado (região de operação)



Aprendizado Supervisionado: Mapeamento I/O

Mapeamento a ser aproximado (dados amostrados)



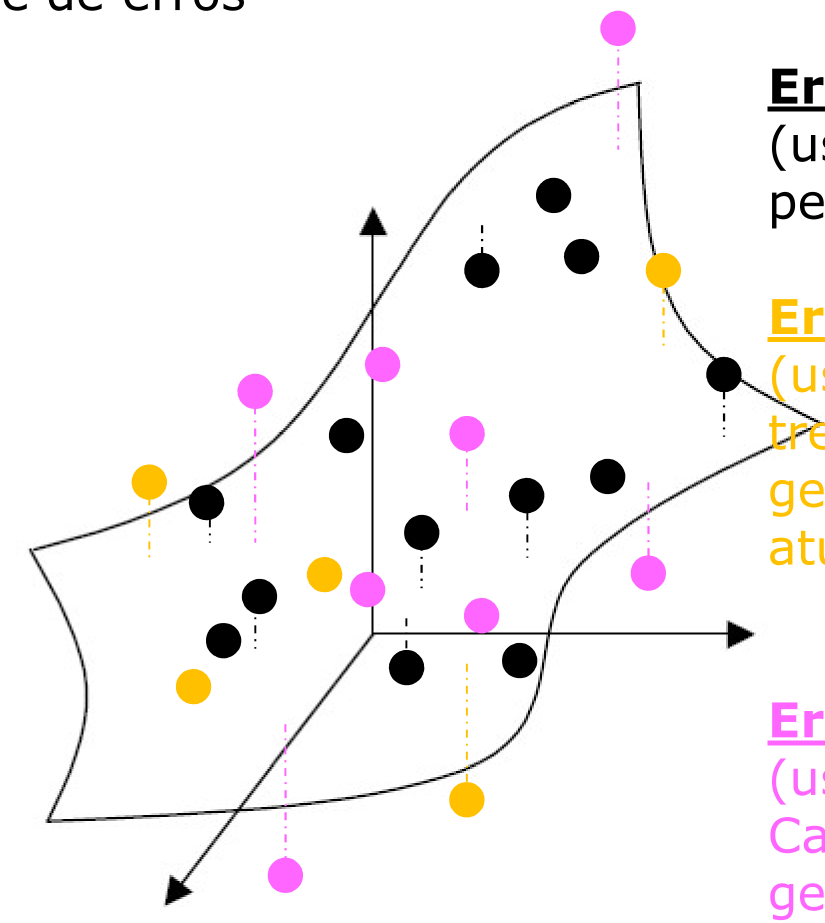
Dados de treinamento — Usados no ajuste de parâmetros da RN

Dados de validação — Usados (em geral) para determinar a condição de parada do treino

Dados de teste — Usados para testar a capacidade de generalização da rede

Aprendizado Supervisionado: Mapeamento I/O

Superfície de erros



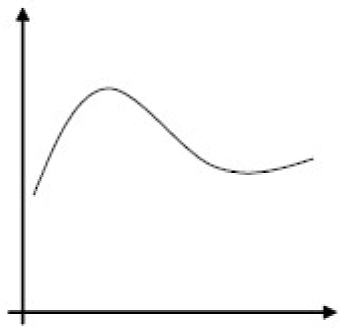
Erro de treinamento (usado para corrigir os pesos)

Erro de validação (usado para parar o treinamento – avalia a generalização da rede atual)

Erro de teste (usado para avaliar a Capacidade de generalização da rede final)

Aprendizado Supervisionado: Mapeamento I/O

Interpolação x Aproximação

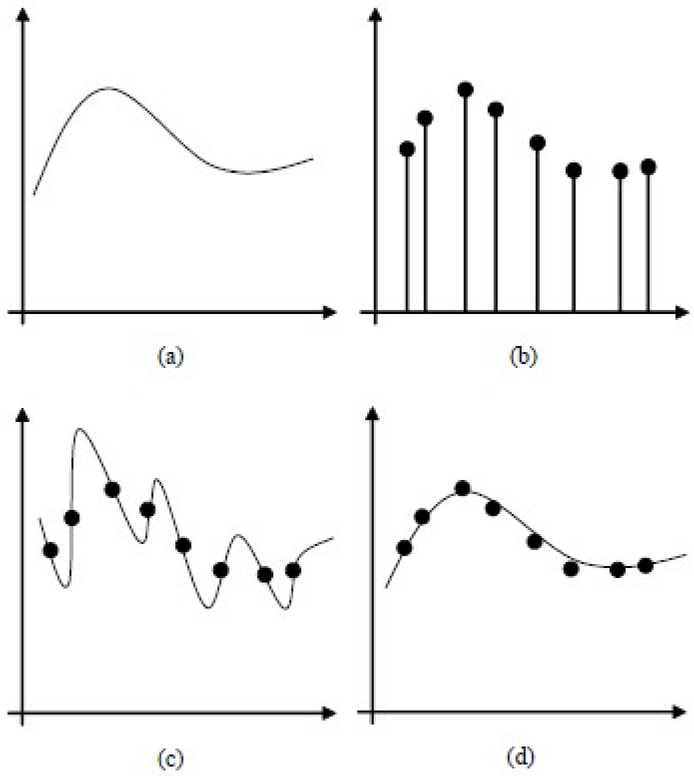


(a)

(a) Função a ser aproximada;

Aprendizado Supervisionado: Mapeamento I/O

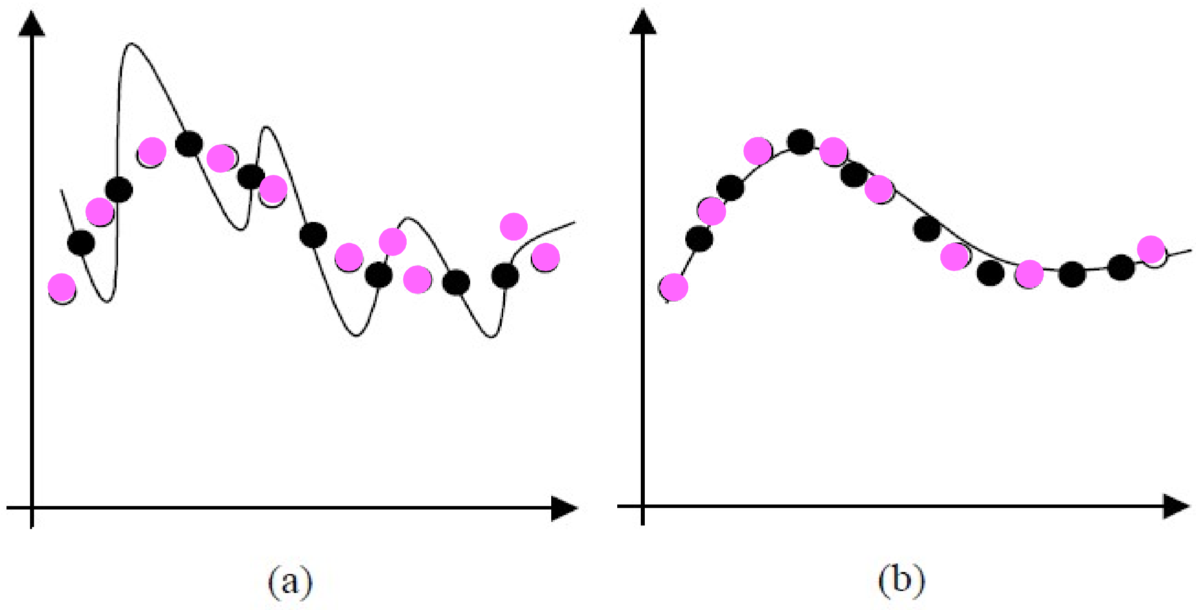
Interpolação x Aproximação



(a) Função a ser aproximada; (b) Amostras disponíveis; (c) Resultado de um processo de interpolação; (d) Resultado de um processo de aproximação.

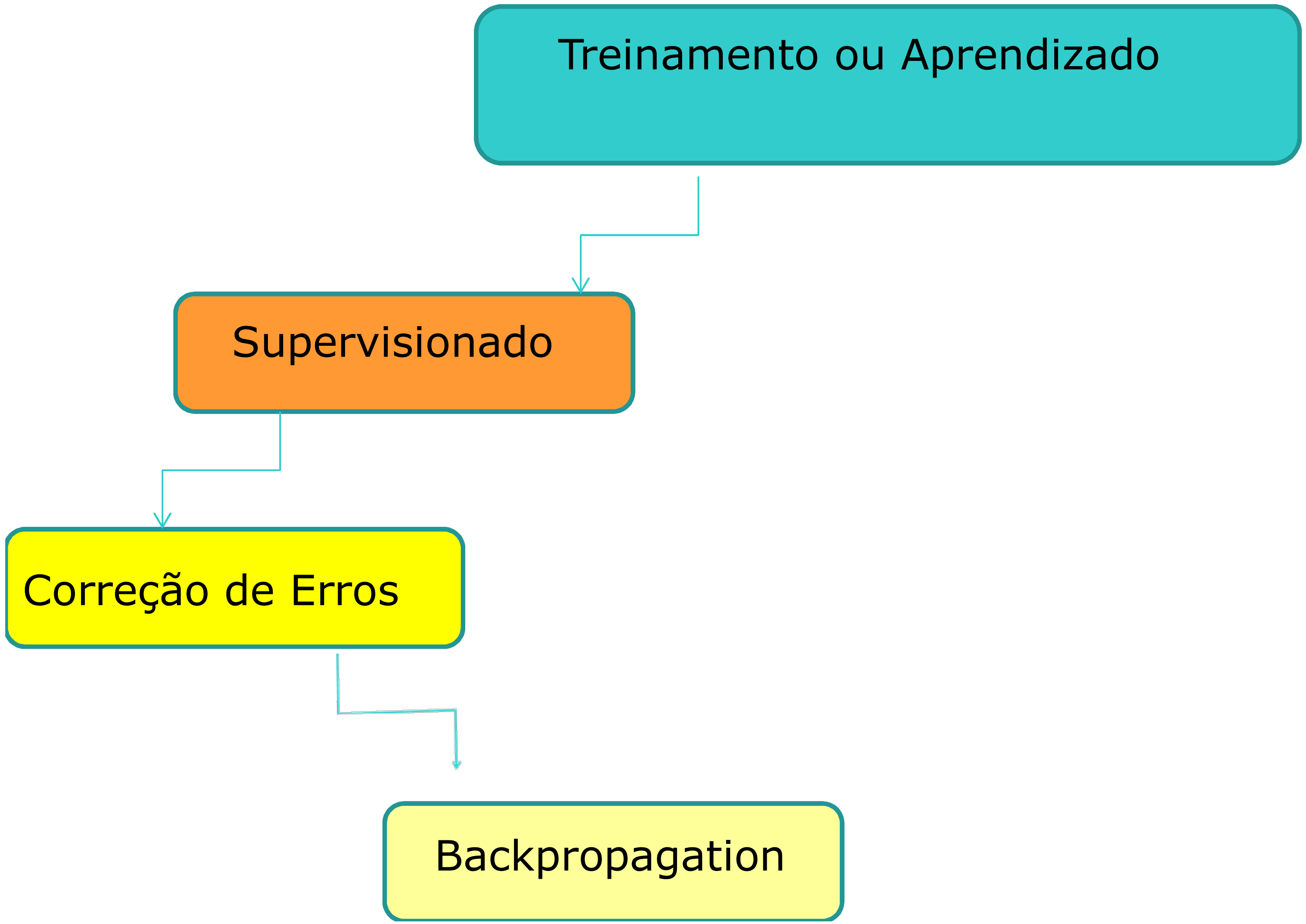
Aprendizado Supervisionado: Mapeamento I/O

Interpolação x Aproximação



Comparação de desempenho para dados de treinamento e teste, de modo a medir a capacidade de generalização dos mapeamentos produzidos.

Redes Neurais Artificiais: Treinamento



Redes Neurais Artificiais: Treinamento

Backpropagation: Algoritmo de treinamento ou aprendizado **supervisionado** por **correção de erros**.

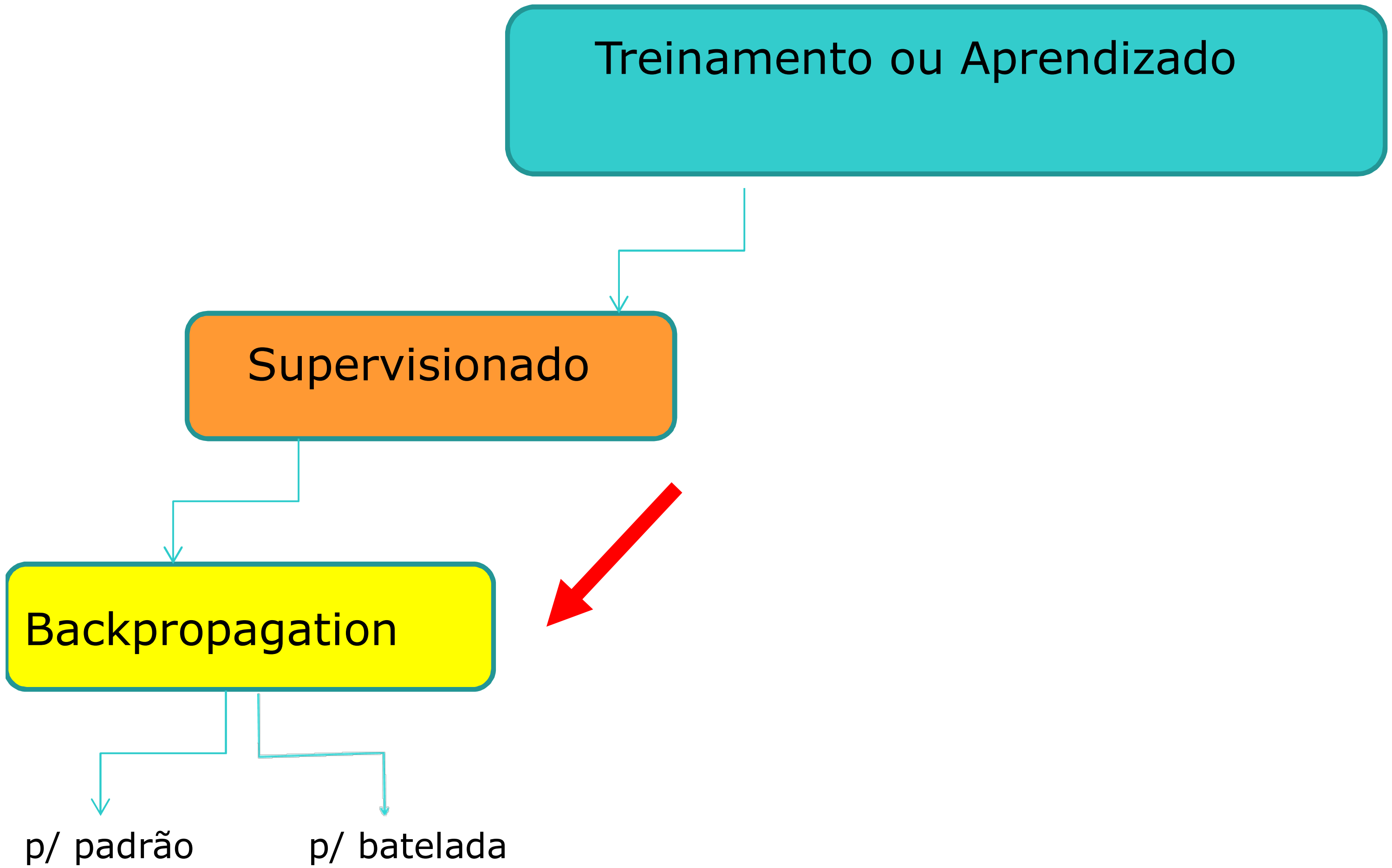
A cada padrão apresentado, compara-se a saída produzida pela rede com a **saída desejada**. Calcula-se o ajuste nos pesos de forma a minimizar o erro na saída.

O esquema de correção pode ser feito de duas formas:

•**Individualmente a cada padrão**: os pesos são corrigidos após a apresentação de cada padrão.

•**Por batelada**: as mudanças nos pesos são calculadas para cada padrão mas a alteração ocorre somente após todo o conjunto ser apresentado à rede (este ciclo de apresentação de todos os padrões é chamado de **época**).

Redes Neurais Artificiais: Treinamento



Redes Neurais Artificiais: Treinamento

Treinamento ou Aprendizado

Supervisionado

Backpropagation

p/ padrão

Cada padrão *p* modifica $\Delta\mathbf{w}_p$

$\Delta\mathbf{w}_p =$ *prop ao erro/grad_erro*

E na sequência também o(s) peso(s)

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}_p$$

Redes Neurais Artificiais: Treinamento

Treinamento ou Aprendizado

Supervisionado

Backpropagation

p/ batelada
(batch)

Cada padrão *p* modifica $\Delta\mathbf{w}$

$\Delta\mathbf{w}_p =$ *prop ao erro/grad_erro*

Só ao final da batelada (apresentação de todos os padrões do batch) o(s) peso(s) são modificados

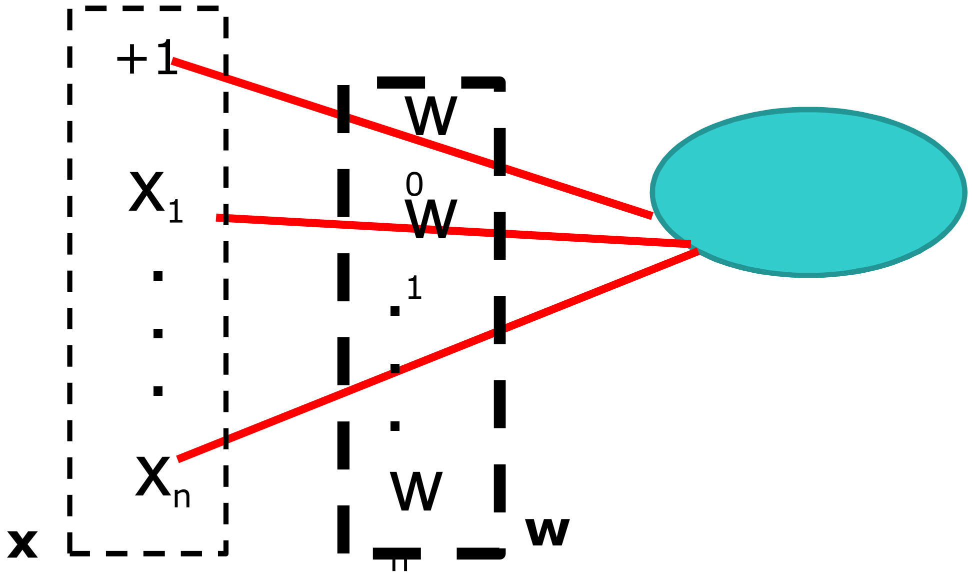
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Sigma_p \, \Delta\mathbf{w}_p(t)$$

MLP: Backpropagation Clássico

Seja **w**(t) o vetor de peso sinápticos de um neurônio no instante t.

A adaptação (ou ajuste) Δ**w**(t) é aplicada ao vetor **w**(t) no instante t, gerando um vetor corrigido (ou adaptado) no instante t+1, na forma

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$$



Redes Neurais Artificiais: Treinamento

O problema do treinamento (ajuste dos pesos) pode ser formulado da seguinte forma:

Dado um estado inicial $\mathbf{w}_0$ do vetor de pesos**,** conduzir o neurônio para um estado final $\mathbf{w_f}$

tal que, para um determinado conjunto de dados de entrada-saída $(\mathbf{x}_p, yd_p)_{\,p=1,\dots,P}$ **,** a função de custo do erro quadrático

$$J(\mathbf{w}) = \sum_{p=1}^{P} \frac{1}{2}\left(y_{d_p} - y_p(\mathbf{w})\right)^2 = \sum_{p=1}^{P} \frac{1}{2}\left(y_{d_p} - g\left(u_p(\mathbf{w})\right)\right)^2$$

seja minimizada.

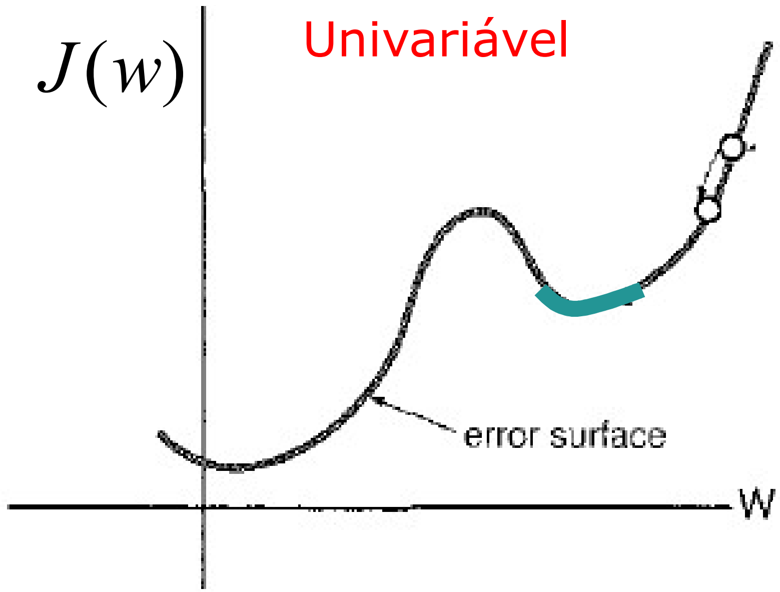
Redes Neurais Artificiais: Treinamento

Problema de minimização do Erro

$$J(w) = \sum_{p=1}^{P} \frac{1}{2}\left(y_{d_p} - y_p(w)\right)^2$$

$$\min_{\mathbf{w}} \; J(\mathbf{w})$$

para $\mathbf{w} = w$



$$W_{new} = W_{old} - \eta \; dJ / dW$$

O sinal negativo busca alcançar o mínimo do Erro

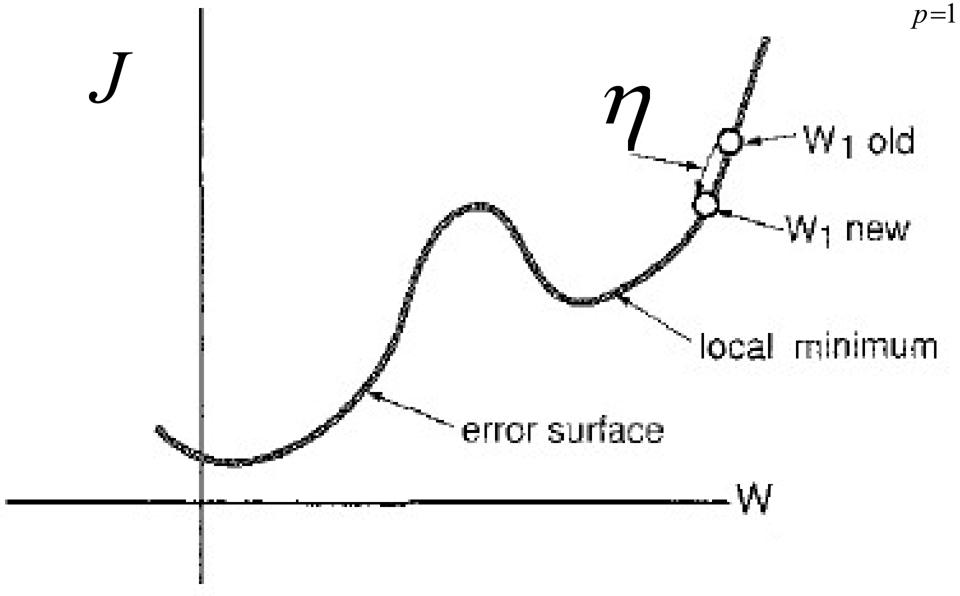
Redes Neurais Artificiais: Treinamento

Problema de minimização do Erro

Univariável

$$J(w) = \sum_{p=1}^{P} \frac{1}{2}\left(y_{d_p} - y_p(w)\right)^2$$

$$\min_{\mathbf{w}} \; J(\mathbf{w})$$



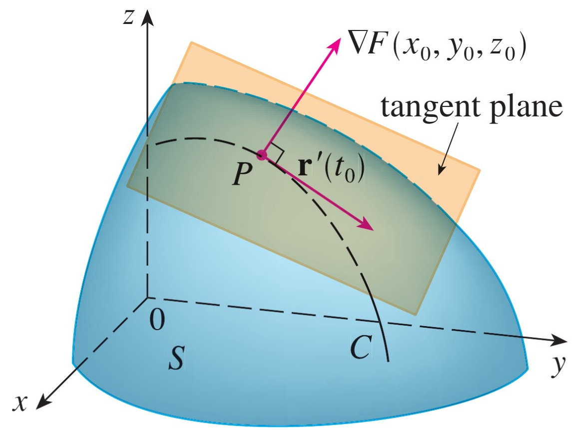
$$J(\mathbf{w}) = \sum_{p=1}^{P} \frac{1}{2}\left(y_{d_p} - y_p(\mathbf{w})\right)^2$$

Multivariável?

Treinamento baseado no gradiente

Resposta: ajustar os pesos $\mathbf{w}=[w_0, w_1, \ldots, w_n]$ *no* *sentido oposto* *ao do* *vetor gradiente* *da função custo J (do erro Q.) em relação ao vetor de pesos* **w**, com um passo denominado taxa de aprendizado η

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla J(\mathbf{w}(t))$$



Onde

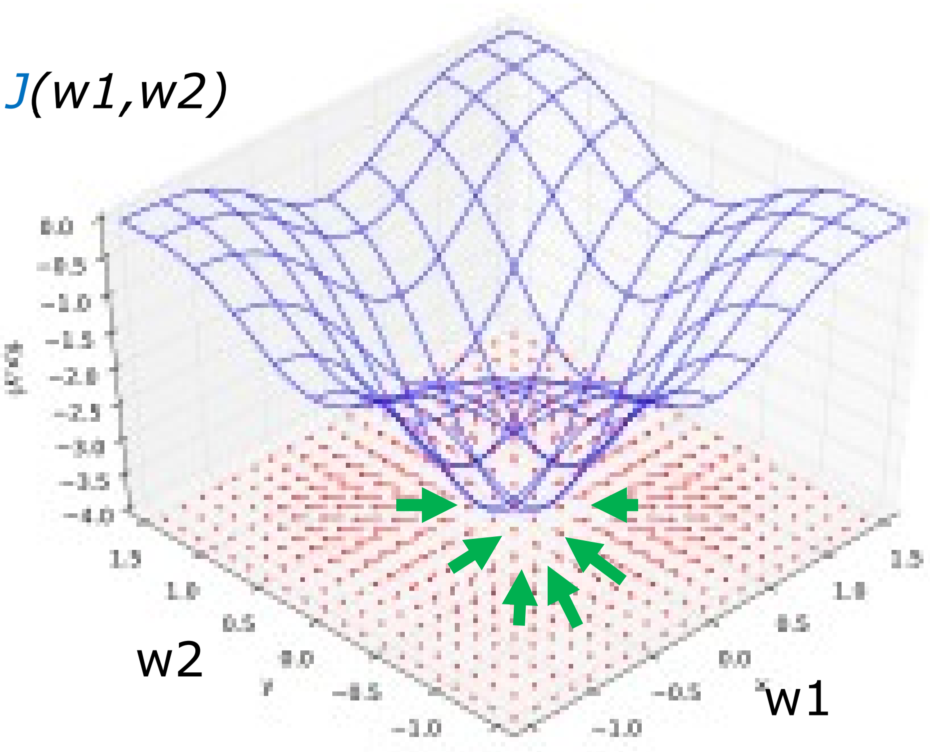
$$\nabla J(\mathbf{w}) = \left[\frac{\partial J(\mathbf{w})}{\partial w_0} \; \frac{\partial J(\mathbf{w})}{\partial w_1} \cdots \frac{\partial J(\mathbf{w})}{\partial w_n} \right]^T$$

RNAs: Treinamento por Descida do Gradiente

$$\mathbf{w}(t+1)=\mathbf{w}(t)+\Delta\mathbf{w}$$

$\Delta \boldsymbol{w}$ função do Gradiente do Custo do Erro Quadrático (∇J)

J(w1,w2)



A maior taxa de variação do Custo do Erro Quadrático ($J(\boldsymbol{w})$) ocorre na direção do vetor gradiente (sentido contrário)

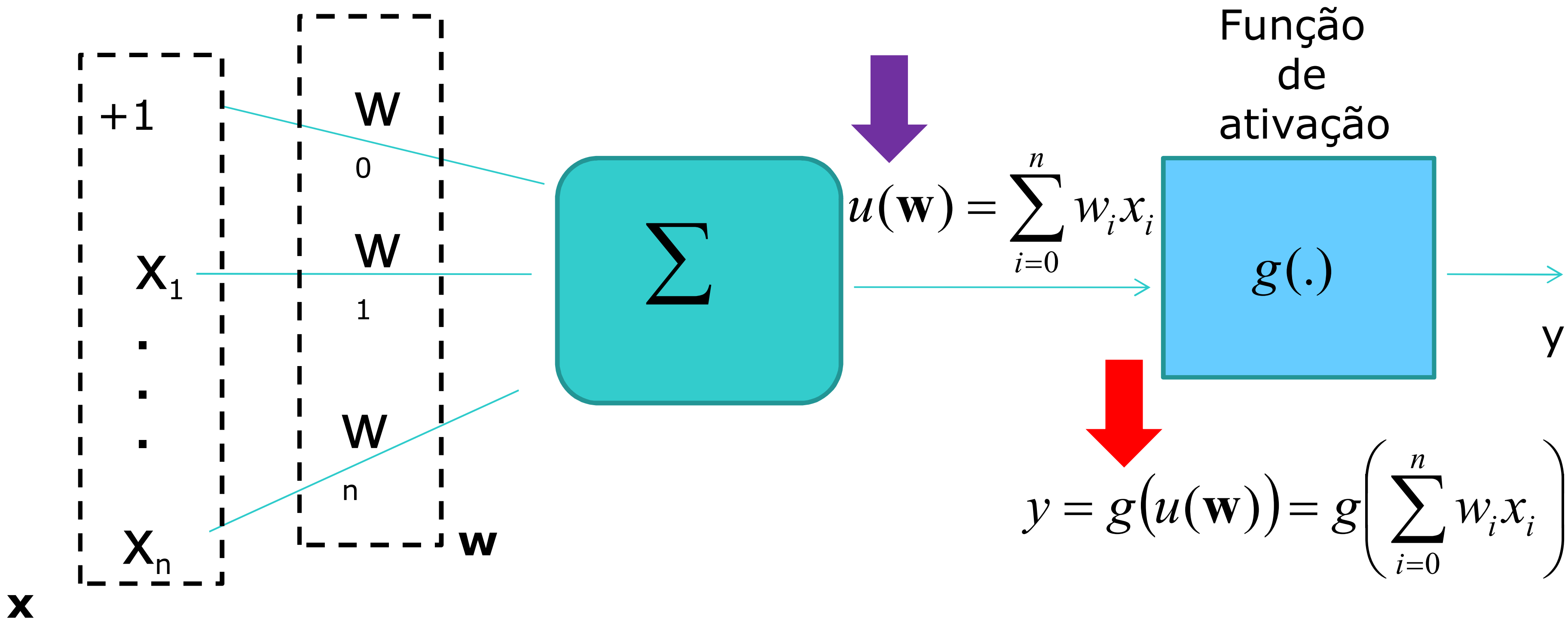
Só é possível quando o $J(\boldsymbol{w})$ é diferenciável em relação aos pesos.

$$J(\mathbf{w}) = \frac{1}{2}\left(y_{d_p} - y_p(\mathbf{w})\right)^2 = \frac{1}{2}\left(y_{d_p} - g\left(u_p(\mathbf{w})\right)\right)^2$$

Como o custo depende da função de ativação (g) dos neurônios, a função de ativação também deve ser diferenciável

Aprendizado: neurônio com função de ativação g(.)

Ajustando os pesos de um neurônio com função de ativação g (diferenciável) com derivada em relação ao potencial de ativação dada por g´



Descida do Gradiente do Erro_q
Backpropagation

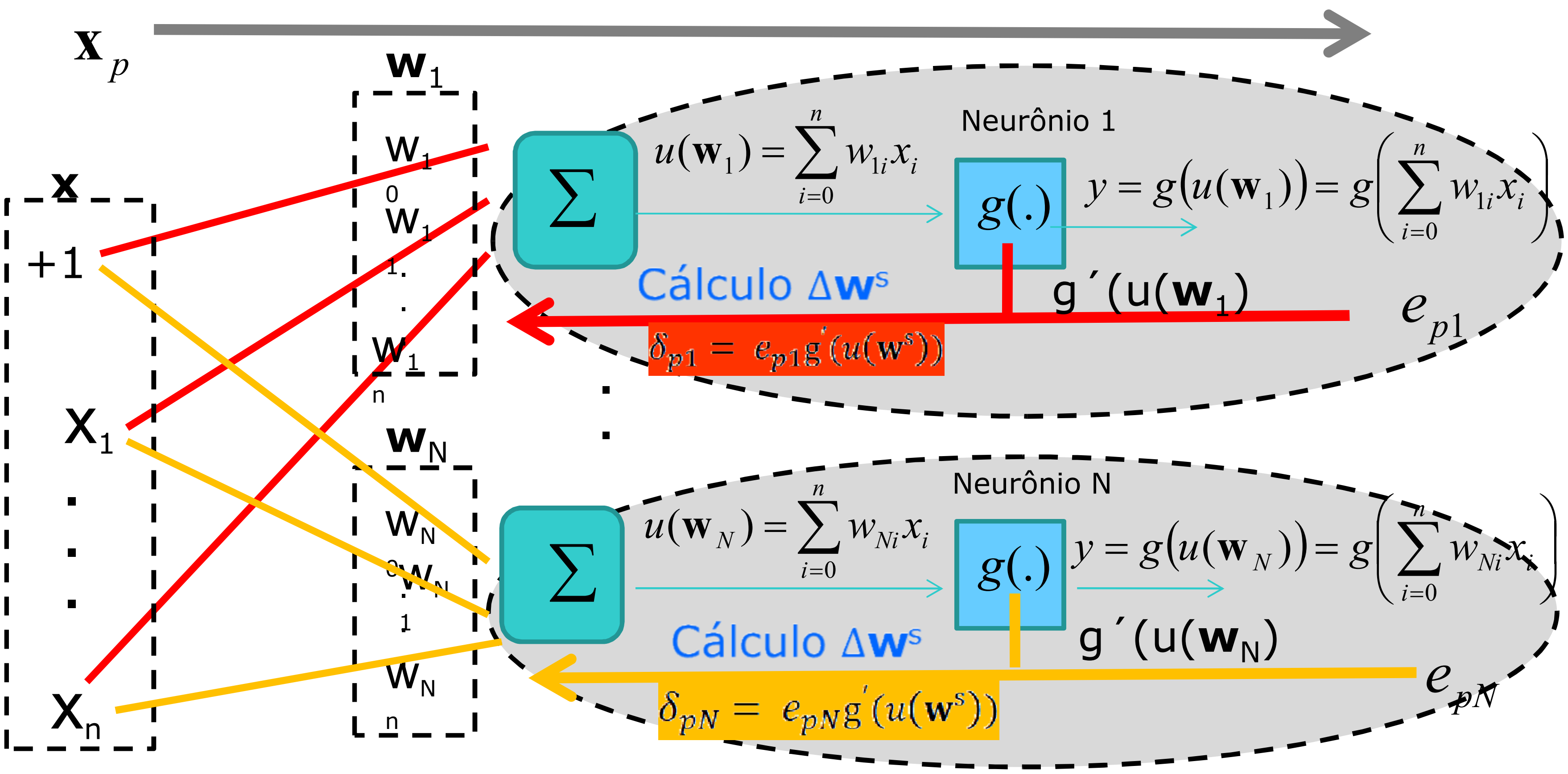
Backpropagation $\Delta \mathbf{w} = -\eta \nabla J(\mathbf{w}) = \eta \ \text{erro}\, g'(u(\mathbf{w}))\, x$

Camada saída: y_d - y

BackPropagation: rede de camada única

Ajustando os pesos de uma rede com camada única e neurônios com função de ativação (diferenciável) g

$$\Delta \mathbf{w} = -\eta \nabla J(\mathbf{w}) = \eta \underbrace{\boxed{\text{erro}\, g'(u(\mathbf{w}))}}_{\delta} x$$



Descida do Gradiente do Custo do Erro_q
Backpropagation

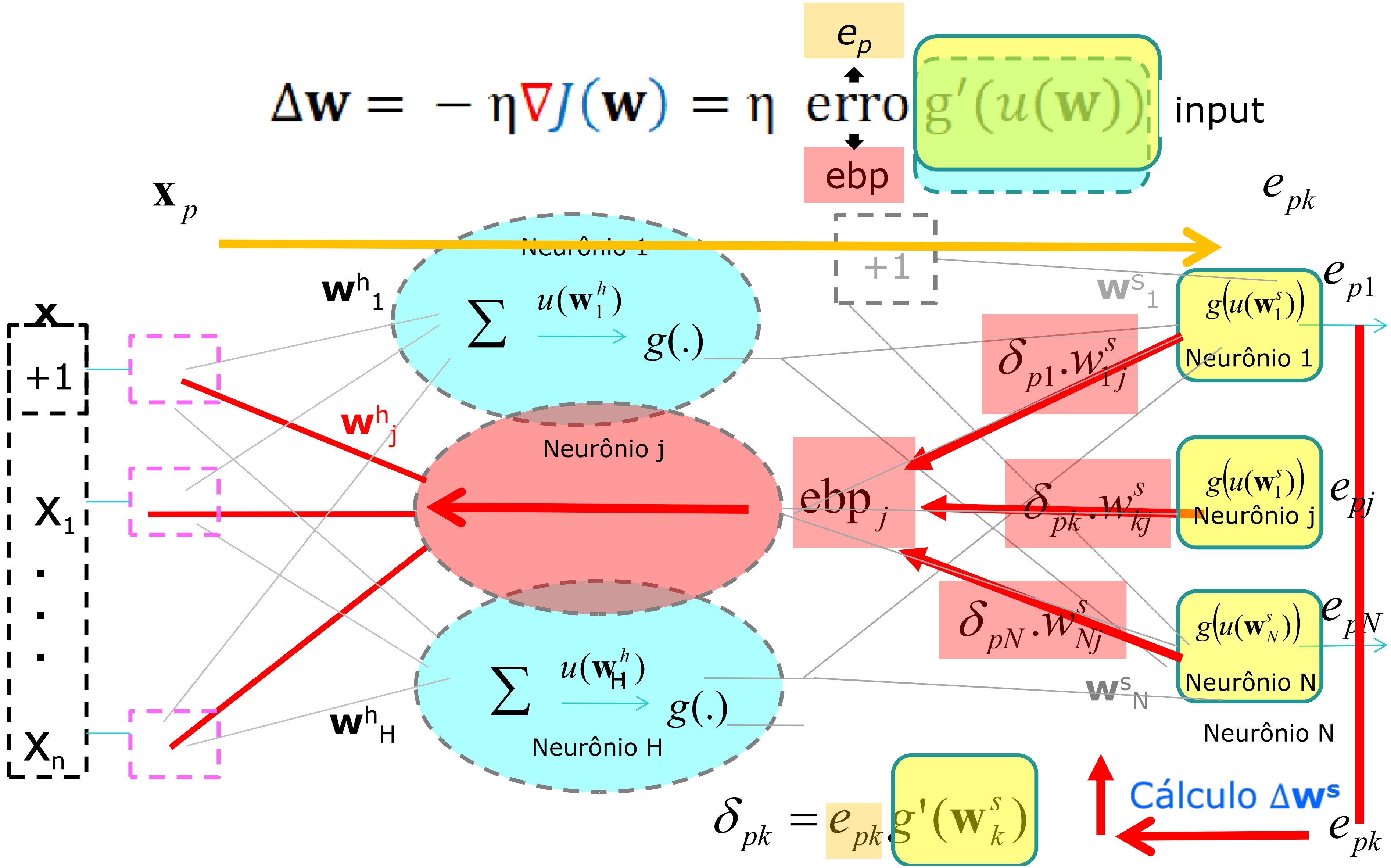
Backpropagation $$\Delta \mathbf{w} = -\eta \nabla J(\mathbf{w}) = \eta \, \overbrace{\text{erro} \; g'(u(\mathbf{w}))}^{\delta} \, x$$

Camadas intermediárias: *ebp*

Camada saída: y_d - y

BackPropagation : rede de múltiplas camadas

Duas fases: propaga entrada e retroprogapa o erro



Exemplo do algoritmo backpropagation de redes de 2 camadas (função de ativação g(.)): batelada

Defina η, Inicialize os pesos dos H+N neurônios: $\mathbf{w}^h{}_j$, j=1,...,H, $\mathbf{w}^s{}_k$, k=1,...,N.

epoca = 1;

Repita

para cada par $(\mathbf{x}^p, y^p{}_d)$ p=1,...,P faça

apresente o padrão $\mathbf{x}^p$ à entrada da rede e faça a informação fluir até a saída

para cada neurônio k k=1,...,N da **CAMADA DE SAÍDA** faça

calcule o erro $e_{pk} = y^p{}_d - y_{pk}$

para o j-ésimo peso j=0,...,H do neurônio k faça

$$\Delta w^s_{kj}(p) = \eta e_{pk} g'\left(u_p(\mathbf{w}^s_k)\right) x^s_{pkj} = \eta \delta_{pk} x^s_{pkj}$$

Saida da camada interm

fim para

fim para

para cada neurônio j j=0,...,H da **CAMADA INTERMEDIÁRIA** faça

para o i-ésimo peso i=0,...,n do neurônio j faça

$$\Delta w^h_{ji}(p) = \eta\ erro\_bkp\ g'\left(u_p(\mathbf{w}^h_j)\right) x^h_{pji} = \eta \sum_{k=1}^{N} \delta_{pk} w^s_{kj}\ g'\left(u_p(\mathbf{w}^h_j)\right) x^h_{pji}$$

fim para

fim para

fim para

Atualize o peso j (j=1,..H) do neur k k=1,...,N.

$$w^s_{kj}(t+1) = w^s_{kj}(t) + \frac{1}{P}\sum_{p=1}^{P} \Delta w^s_{kj}(p)$$

Atualize o peso i (i=1,..N) do neur j j=1,...,H

$$w^h_{ji}(t+1) = w^h_{ji}(t) + \frac{1}{P}\sum_{p=1}^{P} \Delta w^h_{ji}(p)$$

epoca=epoca+1;

até atingir condição de parada

Algoritmo Backpropagation:

Dicas práticas: (treinamento por batelada ou por padrão)

Inicializar aleatoriamente os pesos no intervalo [-1,1]

(lembrar de incluir os limiares no conjunto de pesos associados com entradas fixa em +1)

Normalizar as entradas:

[0.1,0.9] sigmoide

[-0.9,0.9] tangente hiperbólica

Utilizar valores pequenos para taxa de aprendizado

η in [0.01 a 0.1]

Algoritmo Backpropagation: mínimos locais

BP puro: sujeito a ficar preso nos mínimos locais

Como escapar de mínimos locais?

– Utilizar o Backpropagation com momento:

Usa na atualização dos pesos um termo proporcional a última direção de alteração do peso. (Alteração do Peso no passo anterior do algoritmo BP) - idéia de inércia ou um "empurrão" para sair dos minimos locais.

Algoritmo Backpropagation com Momento

BP puro:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t) = \mathbf{w}(t) - \eta\nabla J(\mathbf{w}(t))$$

BP com momento:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t) + \gamma\Delta\mathbf{w}(t-1)$$

Dica prática: utilizar γ in [0.8, 0.9]