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

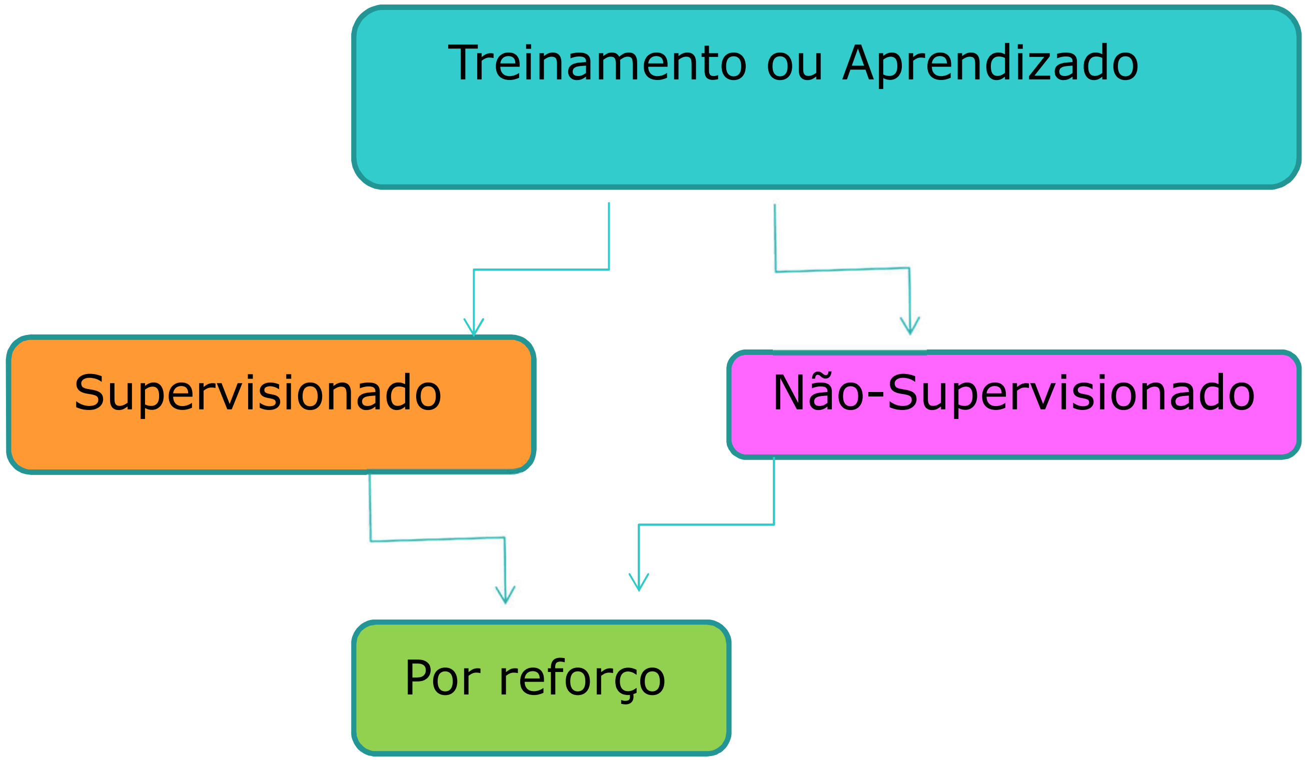
Redes Neurais Artificiais: Treinamento

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:

- Supervisionado
- 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

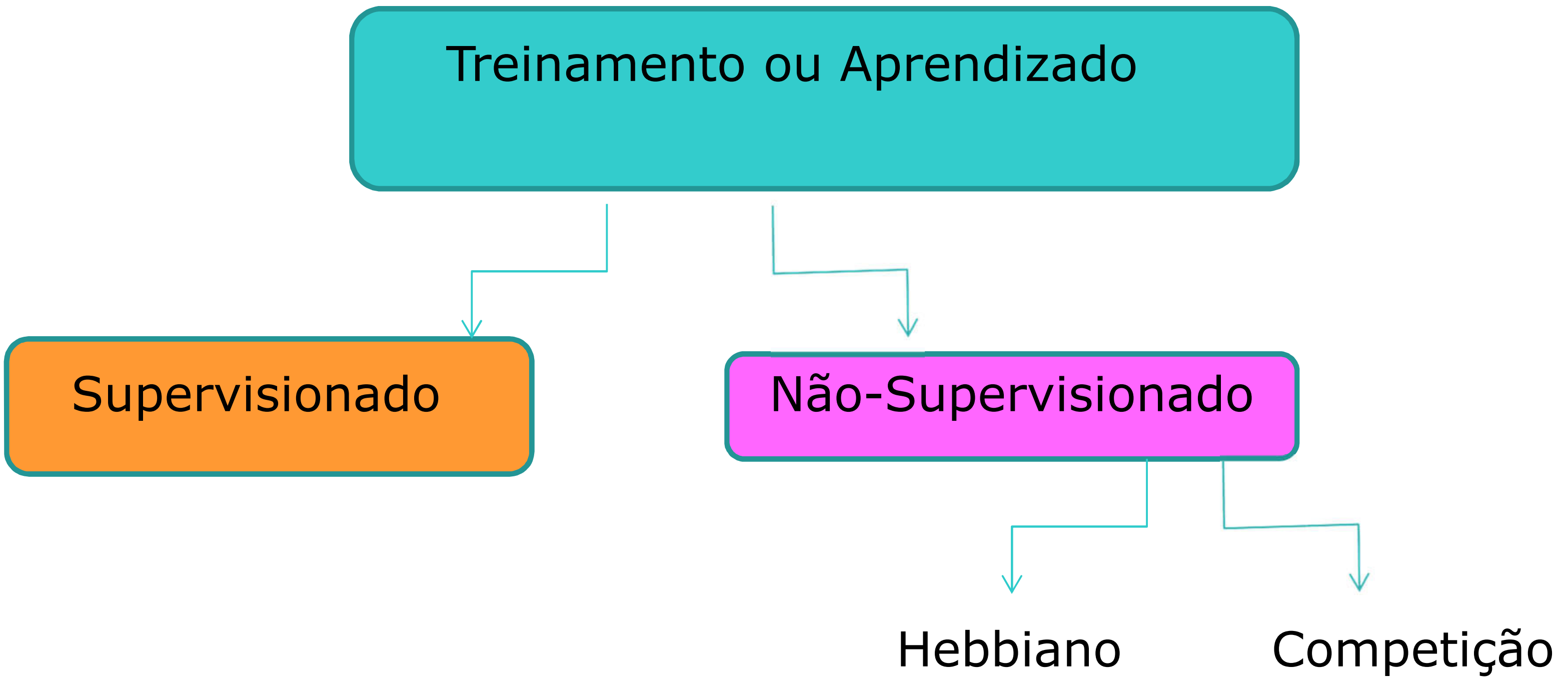
Aprendizado Não-supervisionado:

- 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”

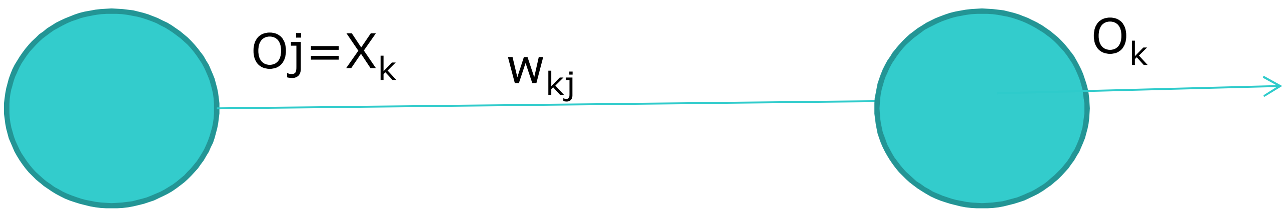
O aprendizado Hebbiano tem aplicação em diferentes modelos de rede.

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:



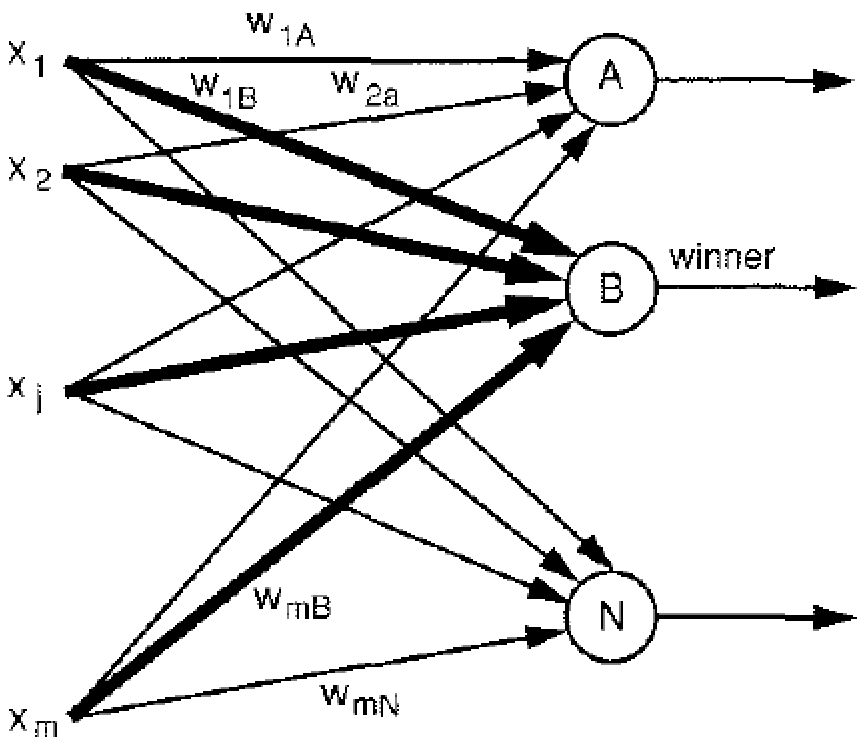
Regra de Hebb

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

Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado Por Competição:

Baseado na estratégia “winner-takes-all”



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

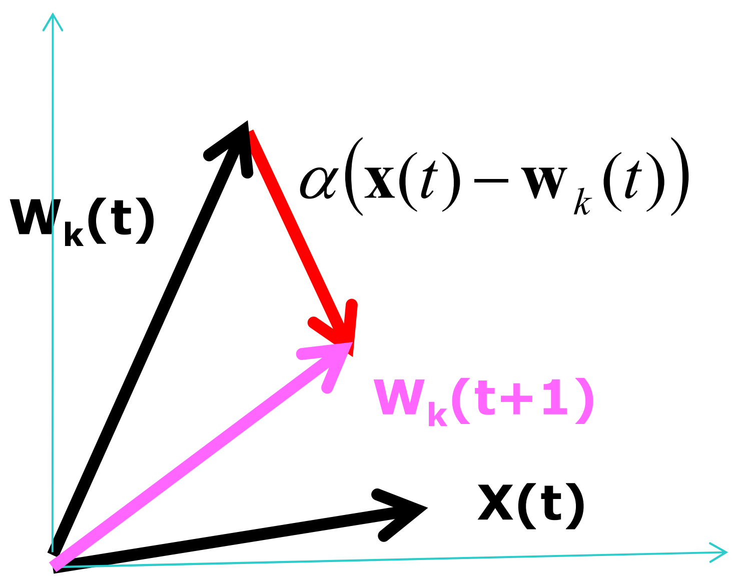
Norma Euclidiana (vale também para $\mathbf{w}_k$ não normalizado)

Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado Por Competição:

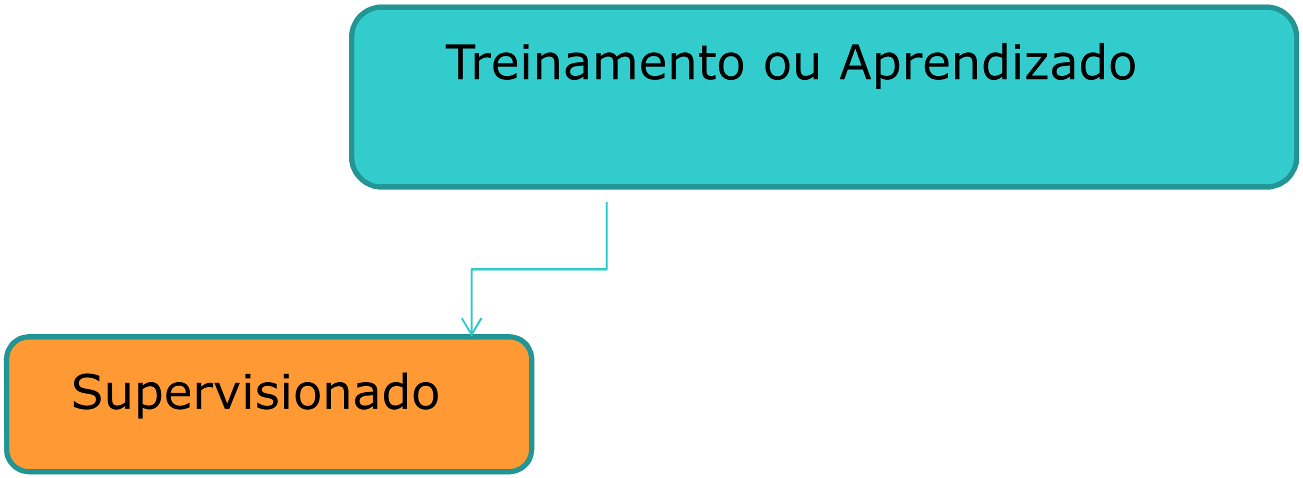
$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha\big(\mathbf{x}(t) - \mathbf{w}_k(t)\big)$$

K: winner



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

Redes Neurais Artificiais: Treinamento



RNA: Aprendizado Supervisionado

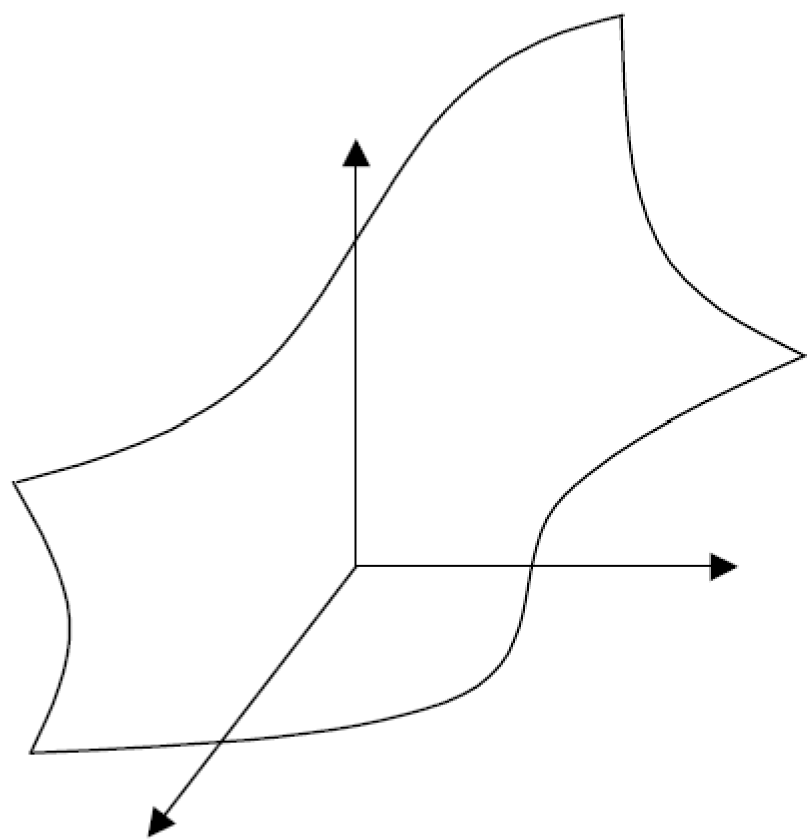
- Aprendizado Supervisionado e o Processo de Mapeamento Entrada Saída.

Em geral, três mapeamentos estão envolvidos:

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

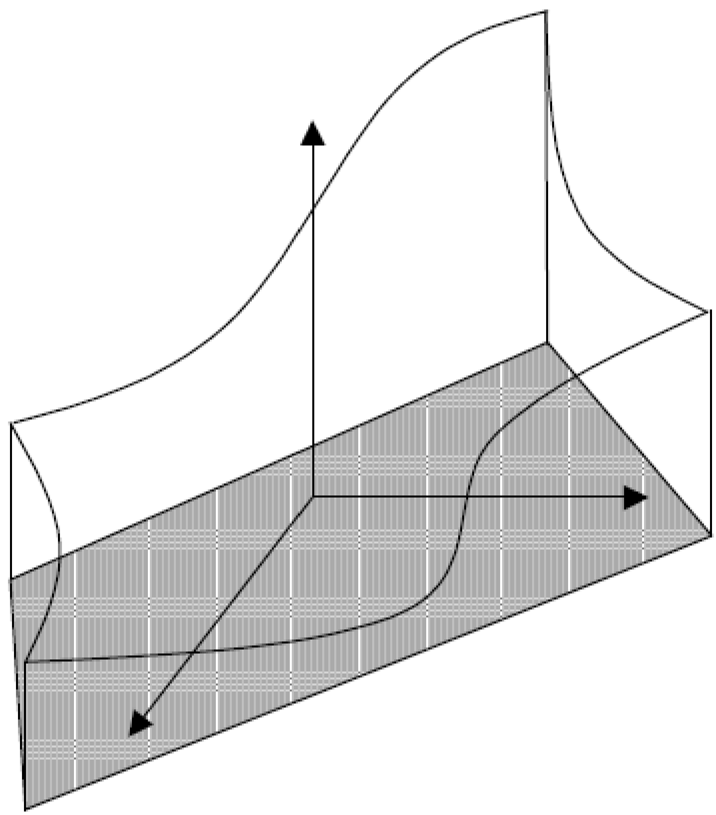
Aprendizado Supervisionado: Mapeamento I/O

Mapeamento a ser aproximado



Aprendizado Supervisionado: Mapeamento I/O

Mapeamento a ser aproximado (região de operação)

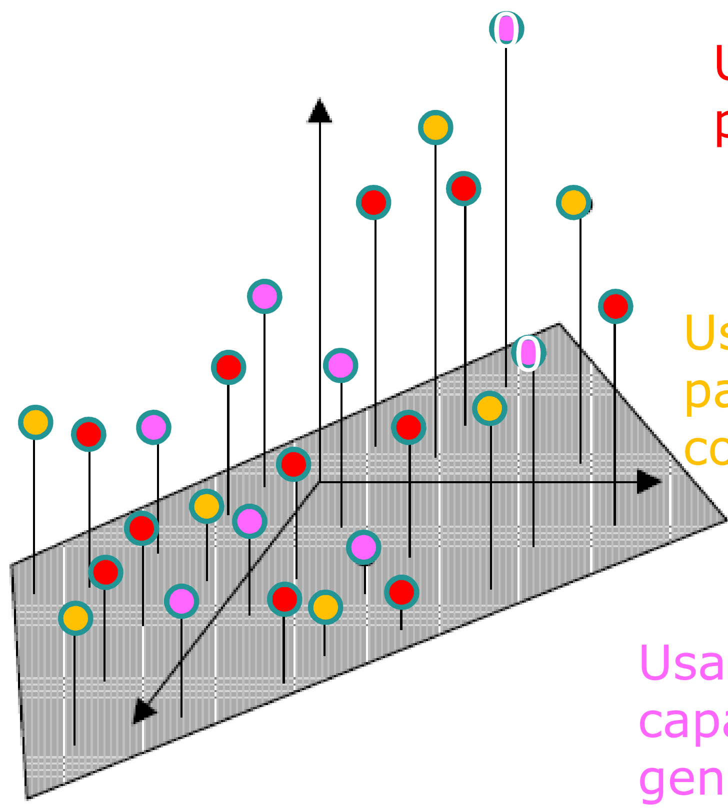


Aprendizado Supervisionado: Mapeamento I/O

Mapeamento a ser aproximado (dados amostrados)

Dados de treinamento

Usados no ajuste de parâmetros da RN



Dados de validação

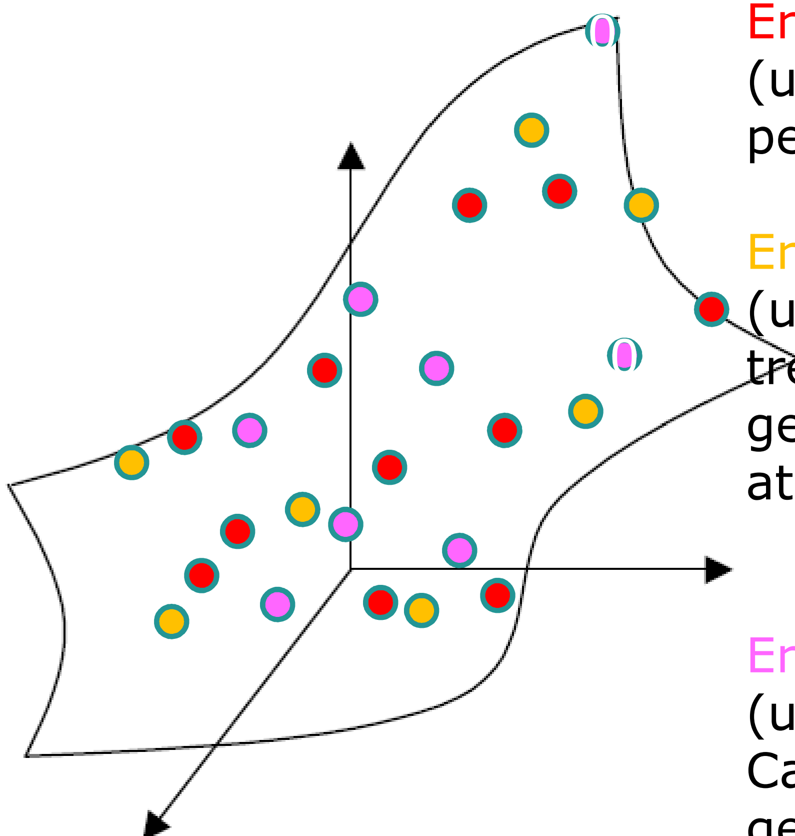
Usados (em geral) para determinar a condição de parada do treino

Dados de teste

Usados para testar a capacidade de generalização da rede

Aprendizado Supervisionado: Mapeamento I/O

Superfície de erros



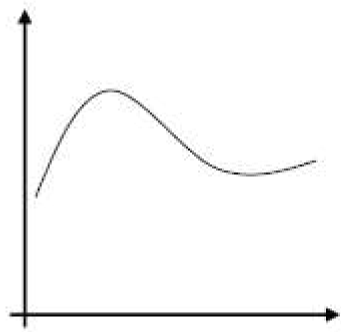
Erro de treinamento
(usado para corrigir os pesos)

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

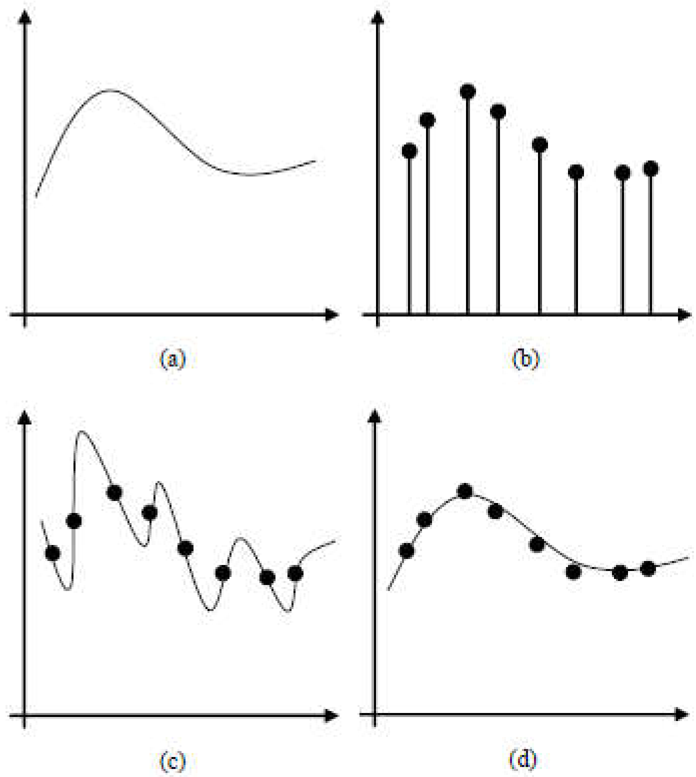
Interpolação x Aproximação



(a) Função a ser aproximada;

Aprendizado Supervisionado: Mapeamento I/O

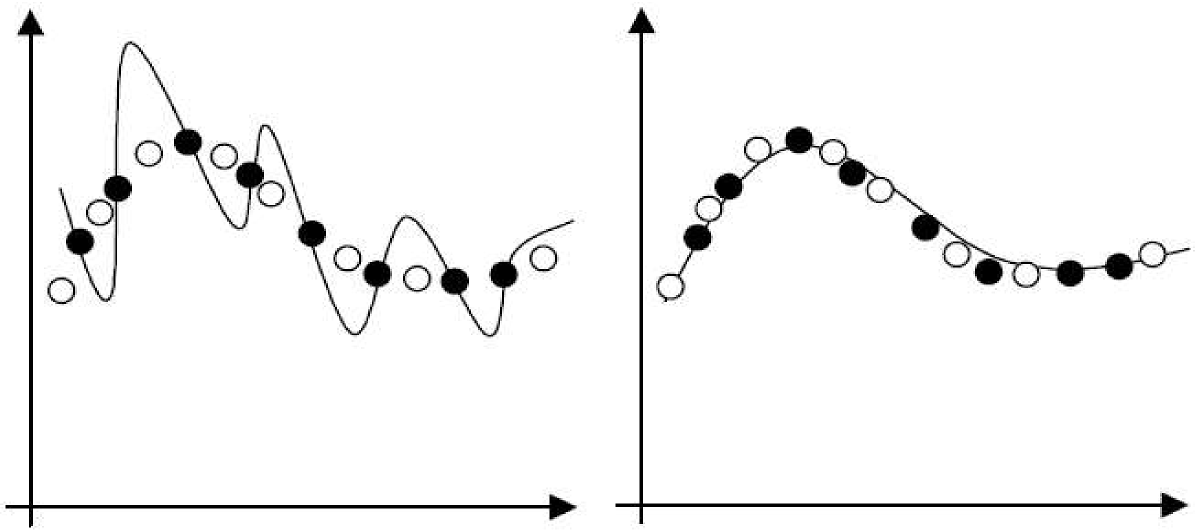
Interpolação x Aproximação



(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



(a) (b)

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

Redes Neurais Artificiais: Treinamento

Treinamento ou Aprendizado

Supervisionado

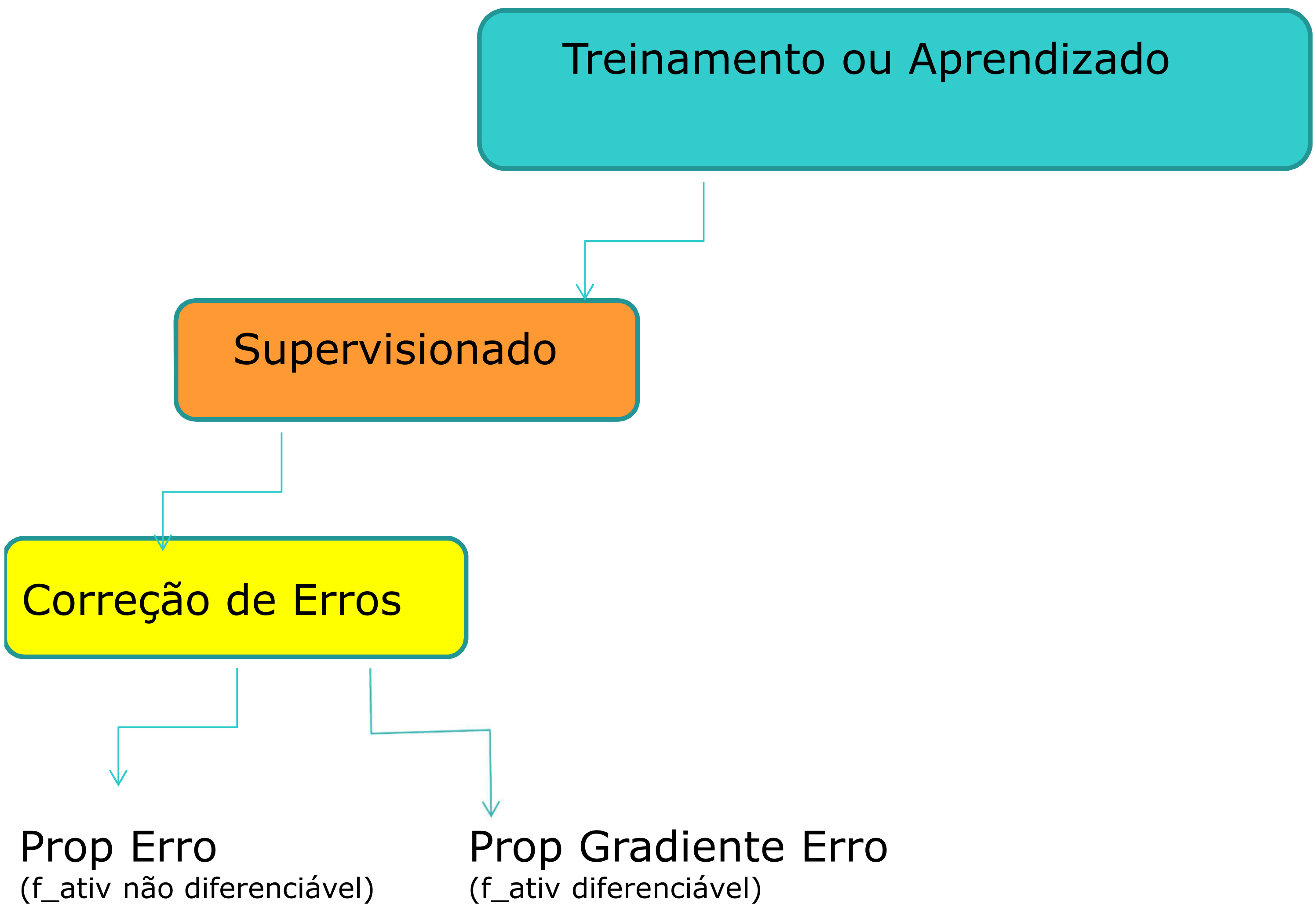
Correção de Erros

Regra Delta

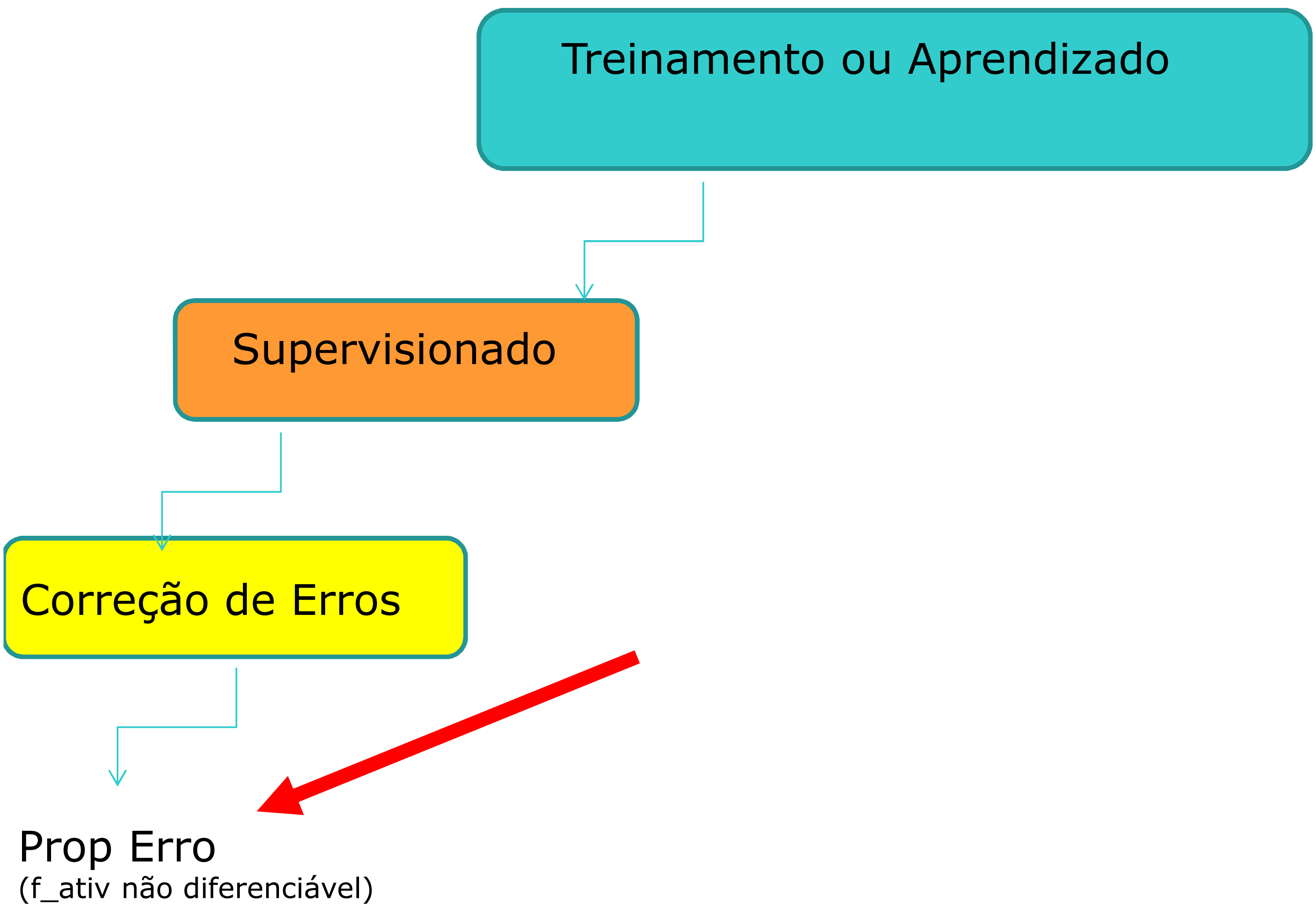
$$w_{\mathrm{i}}(t+1) = w_{\mathrm{i}}(t) + \eta \; Error \;\; xi$$

Onde Error = $(y_d - y)$

Redes Neurais Artificiais: Treinamento



Redes Neurais Artificiais: Treinamento

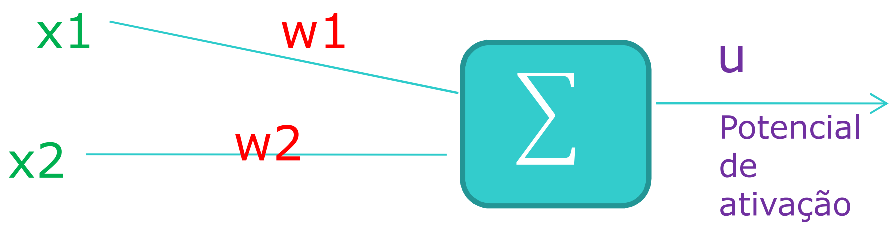


RNAs: Treino Prop. ao Erro (Pot Ativação u)

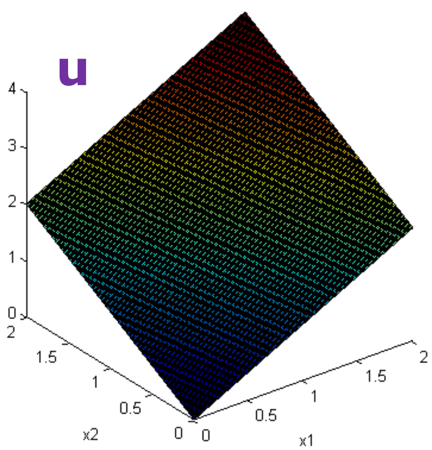
Neurônio

x=(x1,x2)

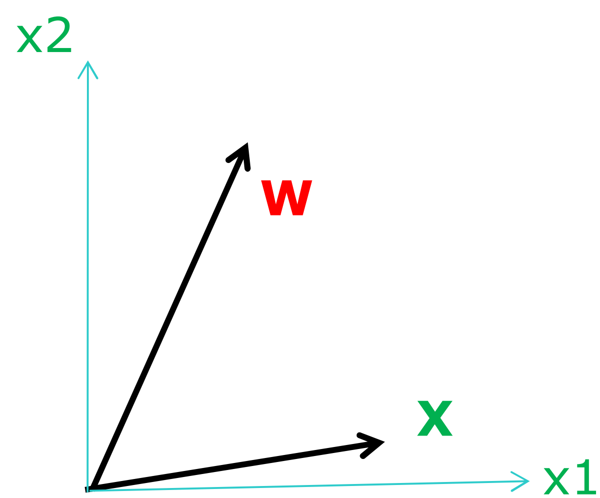
w=(w1,w2)



Potencial de ativação u $= \sum_{i=1}^{2} w_i x_i$ ***versus*** Produto escalar $< \mathbf{w} , \mathbf{x} >$



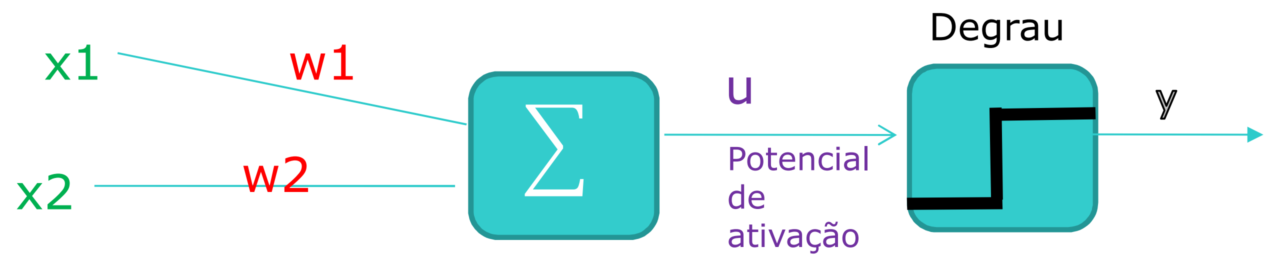
$$\sum_{i=1}^{2} w_i x_i = < \mathbf{w} , \mathbf{x} >$$



RNAs: Treino Prop. ao Erro (função degrau)

$\mathbf{x}=(x1,x2)$

$\mathbf{w}=(w1,w2)$



$$y = \begin{cases} +1 & \text{se } \sum_{i=1}^{2} w_i x_i \geq 0 \\ 0 & \text{se } \sum_{i=1}^{2} w_i x_i < 0 \end{cases}$$

Saída é função do **somatório** das entradas ponderadas pelos pesos

Neurônio MCP

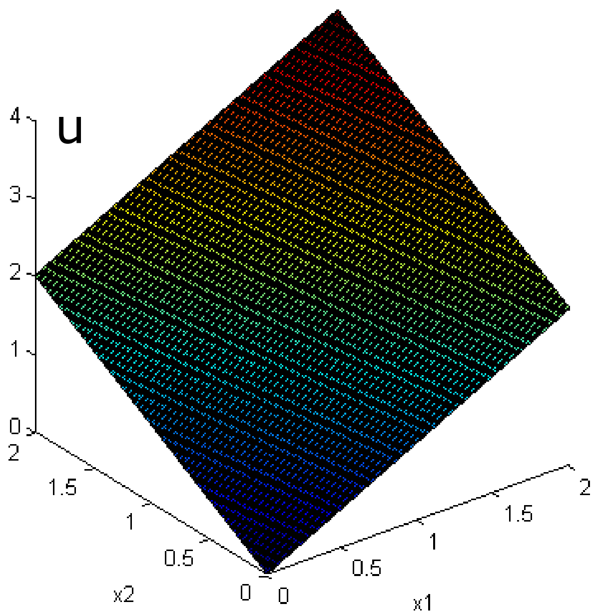
$$y = \begin{cases} +1 & \text{se } \langle \mathbf{w}, \mathbf{x} \rangle \mathbf{>= 0} \\ 0 & \text{se } \langle \mathbf{w}, \mathbf{x} \rangle \mathbf{< 0} \end{cases}$$

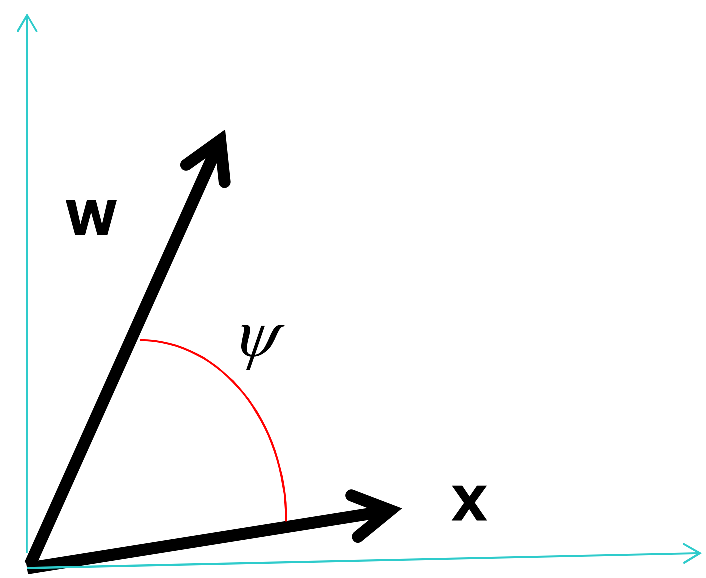
Saída função do **produto escalar** entre o vetor de entradas e o vetor de pesos

RNAs: Produto Escalar entre pesos e entrada

Pot Ativação (u) = **Produto escalar**: <**w**, **x**> entre pesos e entrada

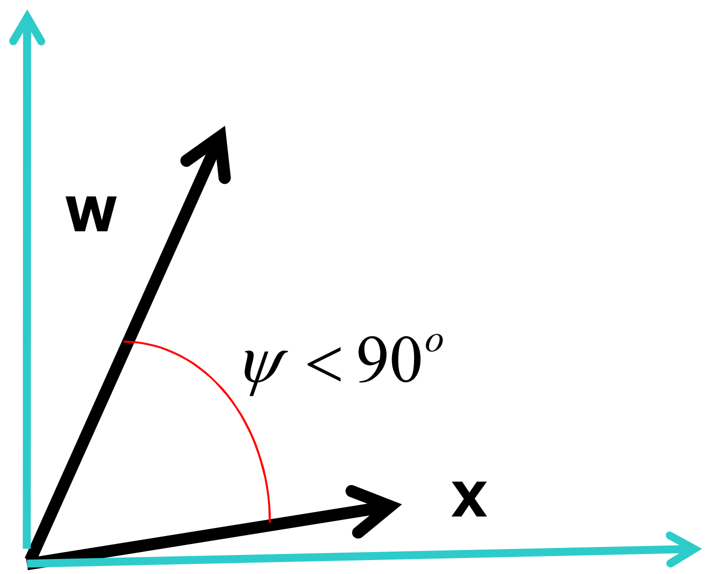
$$<\mathbf{w},\mathbf{x}> = \left(\sum_{i=0}^{n} w_i x_i\right) = \mathbf{w}^T\mathbf{x} = \|\mathbf{w}\|_2 \|\mathbf{x}\|_2 \cos\psi$$

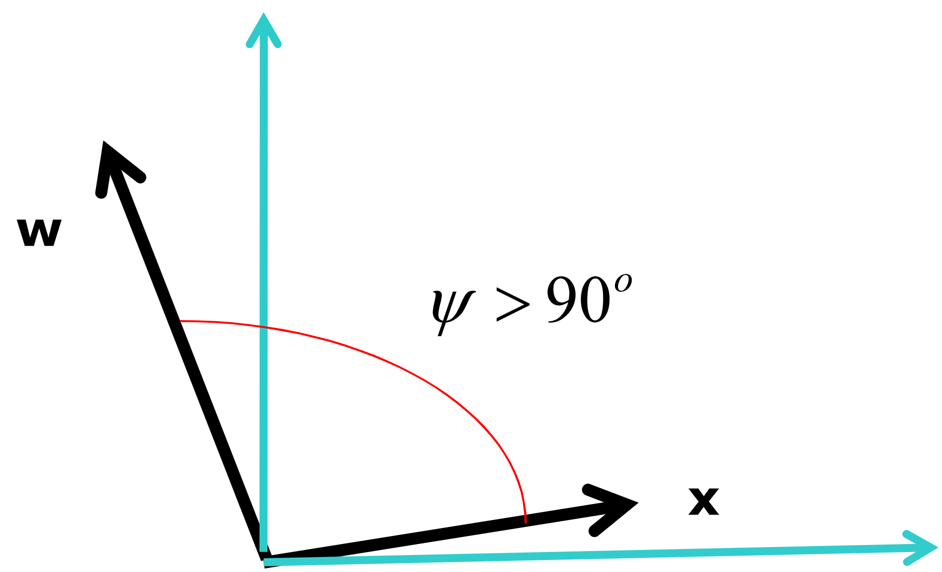




RNAs: Produto Escalar entre pesos e entrada

$$< \mathbf{w}, \mathbf{x} > = \left(\sum_{i=0}^{n} w_i x_i \right) = \mathbf{w}^T \mathbf{x} = \| \mathbf{w} \|_2 \, \| \mathbf{x} \|_2 \cos \psi$$



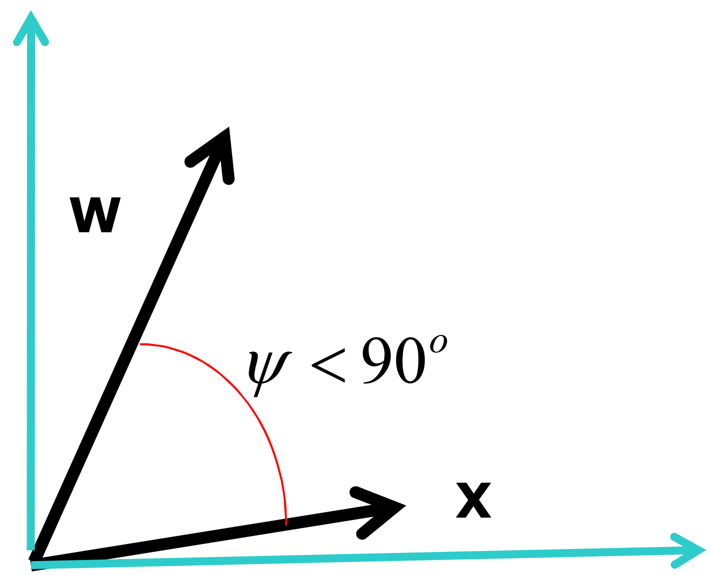


RNAs: Produto Escalar entre pesos e entrada

Produto Escalar: quem define o sinal de u é o ângulo ψ

$$< \mathbf{w}, \mathbf{x} > = \left(\sum_{i=0}^{n} w_i x_i \right) = \| \mathbf{w} \|_2 \| \mathbf{x} \|_2 \cos \psi$$

$$\psi < 90^o \rightarrow \cos \psi > 0 \rightarrow \left(\sum_{i=0}^{n} w_i x_i \right) > 0$$



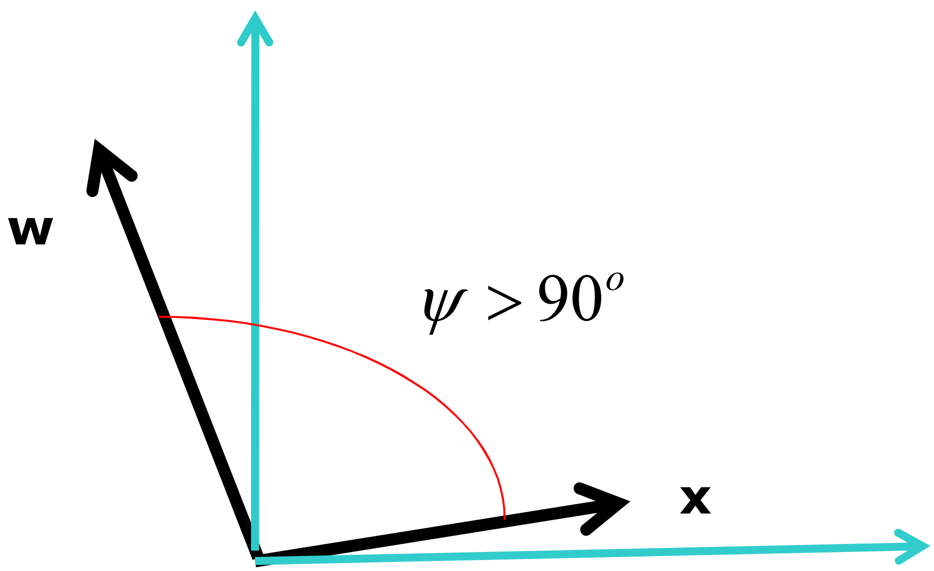
$$y = \begin{cases} \boxed{+1} & \text{se } \sum_{i=0}^{n} w_i x_i \geq 0 \\ 0 & \text{se } \sum_{i=0}^{n} w_i x_i < 0 \end{cases}$$

RNAs: Produto Escalar entre pesos e entradas

Produto Escalar: quem define o sinal de u é o ângulo ψ

$$< \mathbf{w}, \mathbf{x} > = \left(\sum_{i=0}^{n} w_i x_i \right) = \| \mathbf{w} \|_2 \, \| \mathbf{x} \|_2 \cos \psi$$

$$\psi > 90^o \rightarrow \cos \psi < 0 \;\; \rightarrow \;\; \left(\sum_{i=0}^{n} w_i x_i \right) < 0$$



$$y = \begin{cases} +1 & \text{se } \sum_{i=0}^{n} w_i x_i \geq 0 \\ \boxed{0} & \text{se } \sum_{i=0}^{n} w_i x_i < 0 \end{cases}$$

RNAs: Correção de Erros (Regra Delta)

Neurônio MCP

$$y = \begin{cases} +1 & \text{se } \sum_{i=0}^{n} w_i x_i \geq 0 \\ 0 & \text{se } \sum_{i=0}^{n} w_i x_i < 0 \end{cases}$$

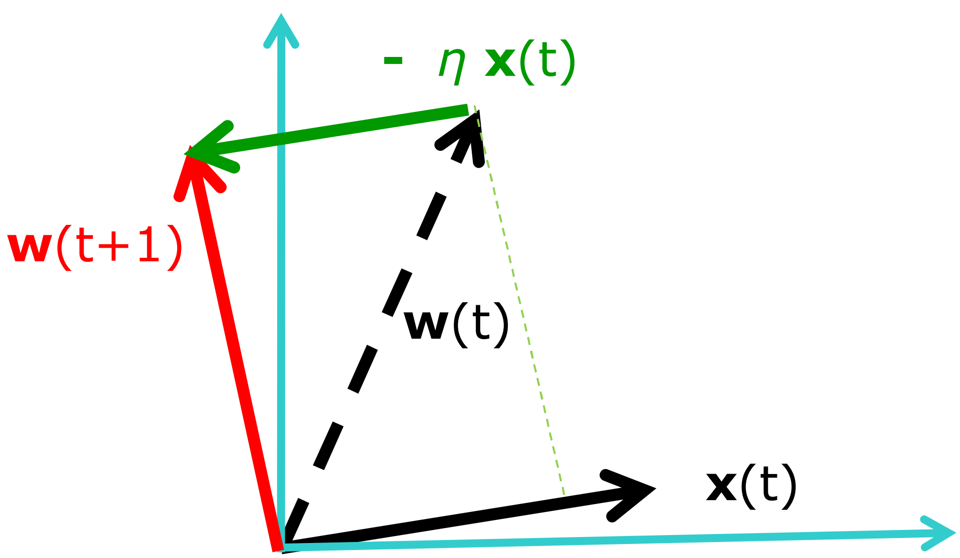
Correção

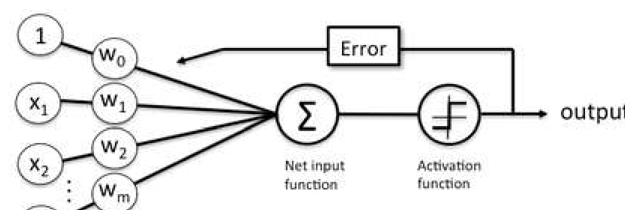
w(t+1) = **w**(t) **-** *η* **x**(t)

$$y = 1 \quad \rightarrow \quad y = 0$$

$$\text{erro} = yd - y = -1$$

w(t+1) = **w**(t) **+** *η erro* **x**(t)





Perceptron rule.

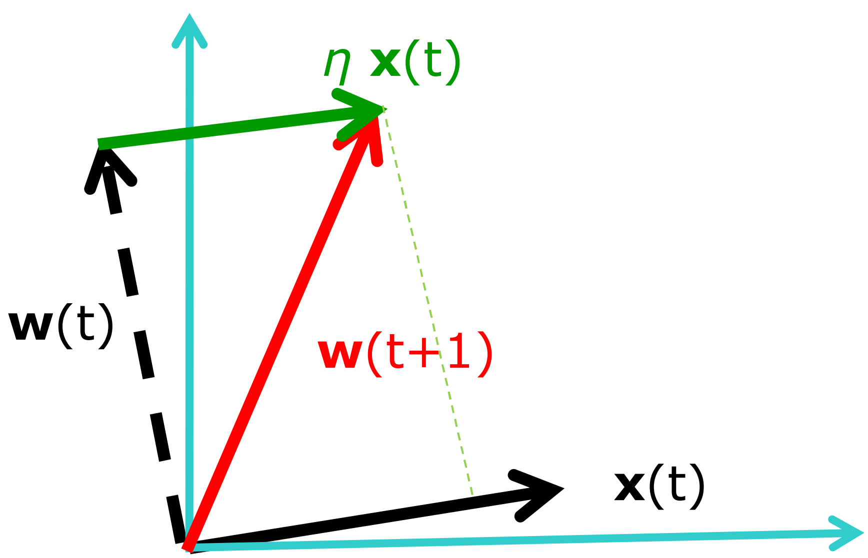
RNAs: Correção de Erros (Regra Delta)

Neurônio MCP

$$y = \begin{cases} +1 & \text{se } \sum_{i=0}^{n} w_i x_i \geq 0 \\ 0 & \text{se } \sum_{i=0}^{n} w_i x_i < 0 \end{cases}$$

Correção

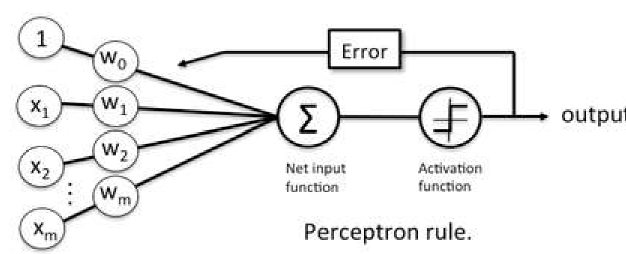
$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta\, \mathbf{x}(t)$



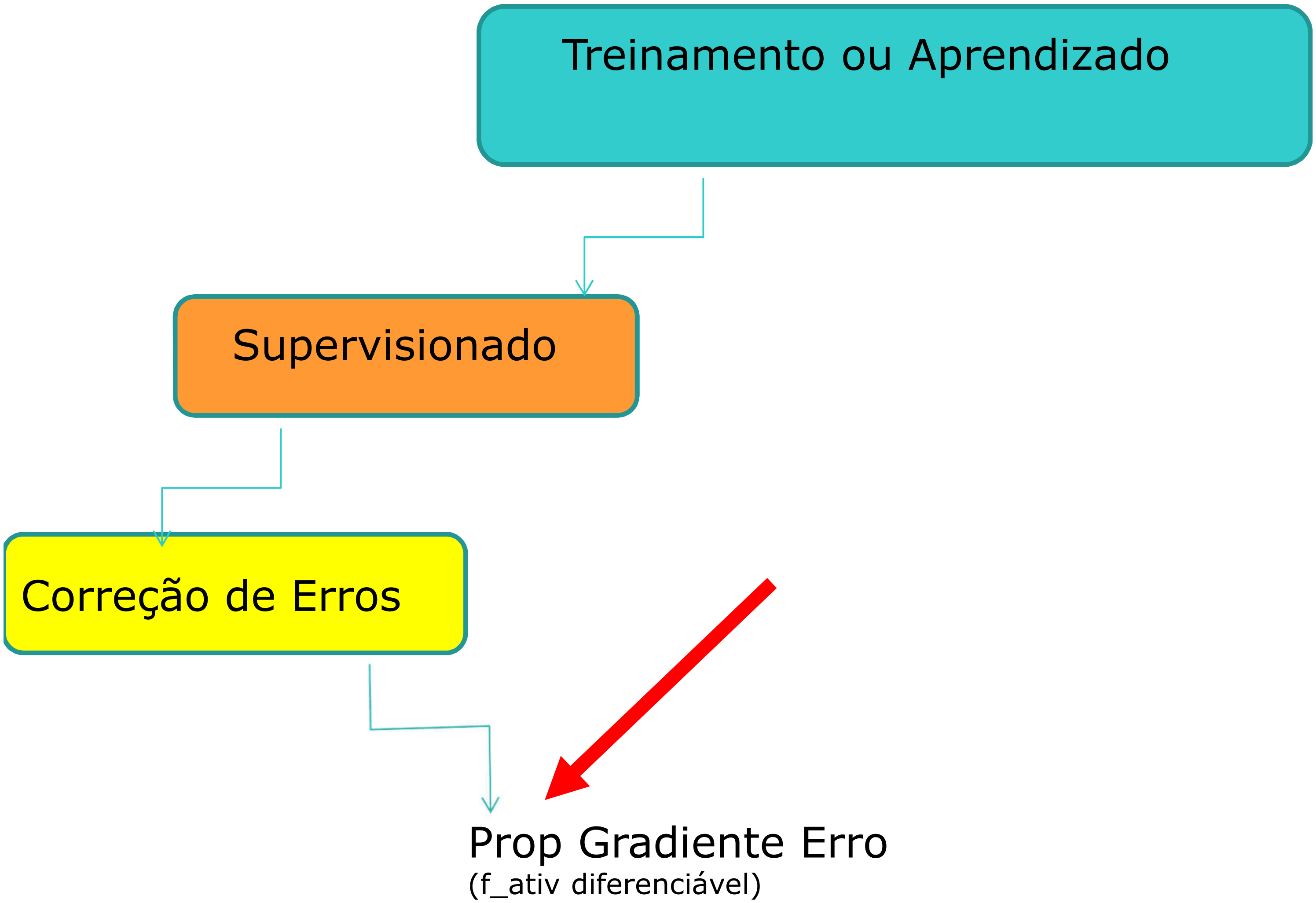
Y=0 → Y=1

erro = 1

$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta\ erro\ \mathbf{x}(t)$



Redes Neurais Artificiais: Treinamento

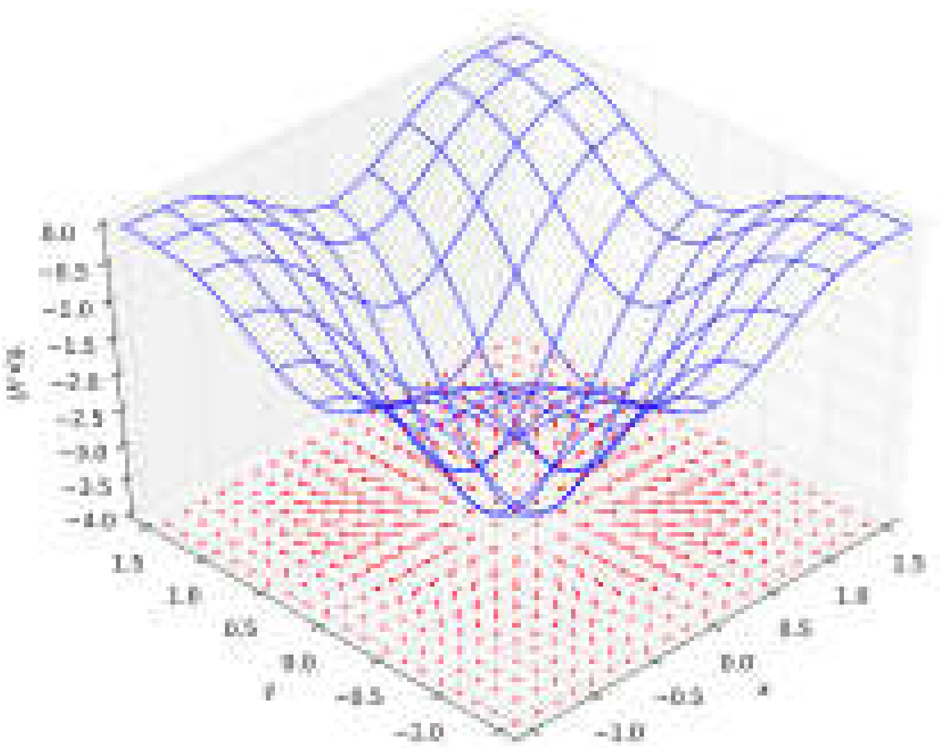


definida num intervalo de números reais.

RNAs: Treinamento por Correção de Erros

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

Δw função do Gradiente do Erro $(\nabla Erro)$



a maior taxa de variação do Erro ocorre na direção e sentido do vetor gradiente $\nabla Erro$

definida num intervalo de números reais.

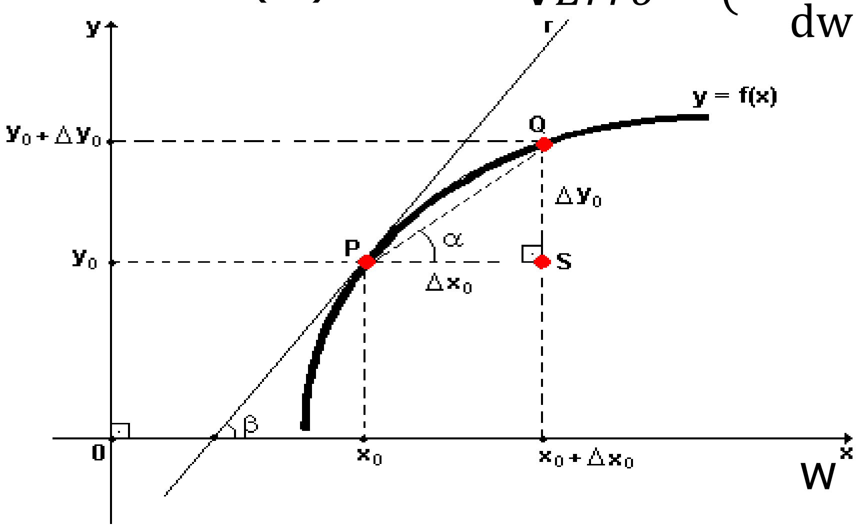
RNAs: Treinamento por Correção de Erros

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

Δw função do Vetor Gradiente: $\nabla Erro = \frac{\partial Erro}{\partial \mathbf{w}} = (\frac{\partial Erro}{\partial w_1}, \frac{\partial Erro}{\partial w_2}, \dots, \frac{\partial Erro}{\partial w_n})$

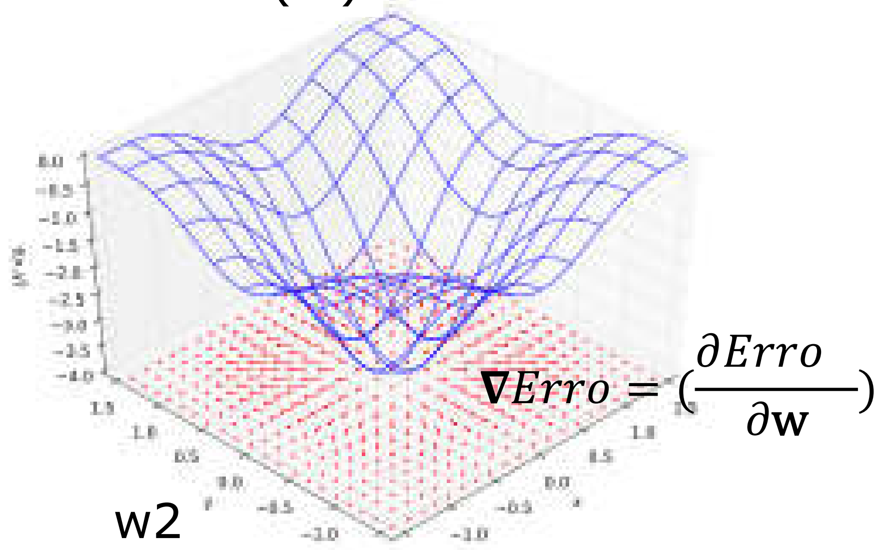
Erro = f(w)

$\nabla Erro = (\frac{dErro}{dw})$



Derivada (1 peso w)

Erro = f(**w**)



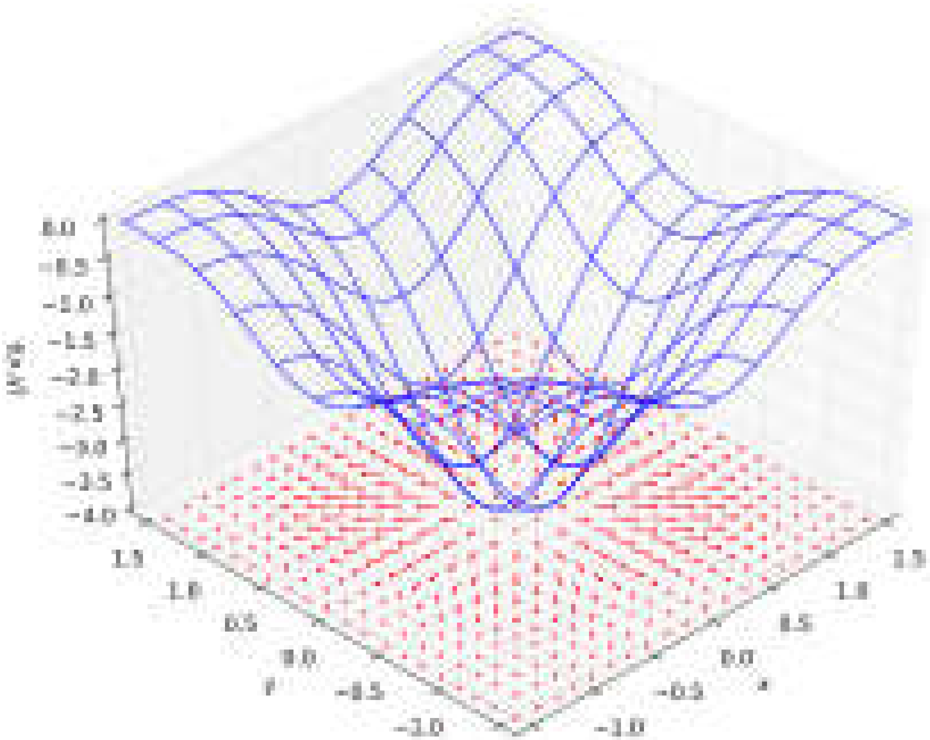
Gradiente (vetor de pesos **w**)

definida num intervalo de números reais.

RNAs: Treinamento por Correção de Erros

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

Δw função do Gradiente do Erro $(\nabla Erro)$



a maior taxa de variação do Erro ocorre na direção e sentido do vetor gradiente $\nabla Erro$

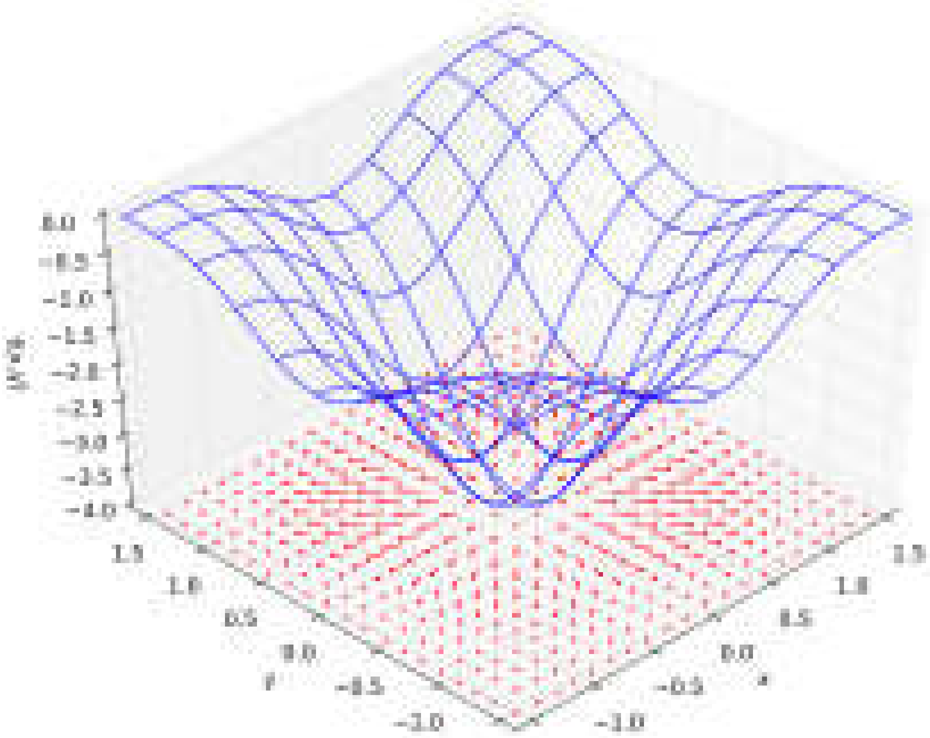
Só é possível quando a função de ativação é diferenciável

definida num intervalo de números reais.

RNAs: Treinamento por Correção de Erros

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

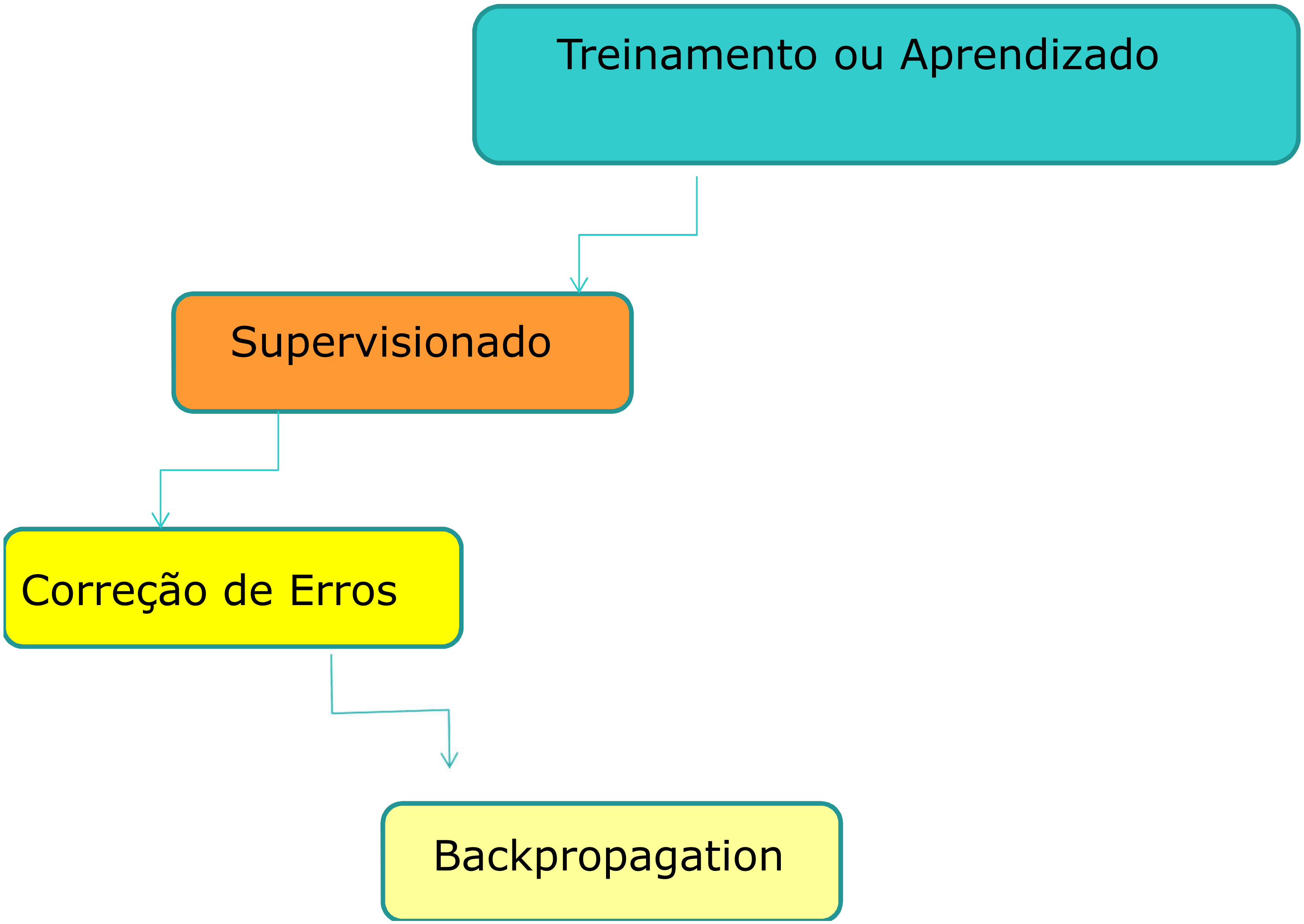
***Método de descida do* Gradiente do Erro** $(\nabla Erro)$



a maior taxa de variação do Erro ocorre na direção e sentido do vetor gradiente $\nabla Erro$

Então para minimizar o erro Deve-se caminhar no sentido contrário ao Gradiente

Redes Neurais Artificiais: Treinamento



Redes Neurais Artificiais: Treinamento

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

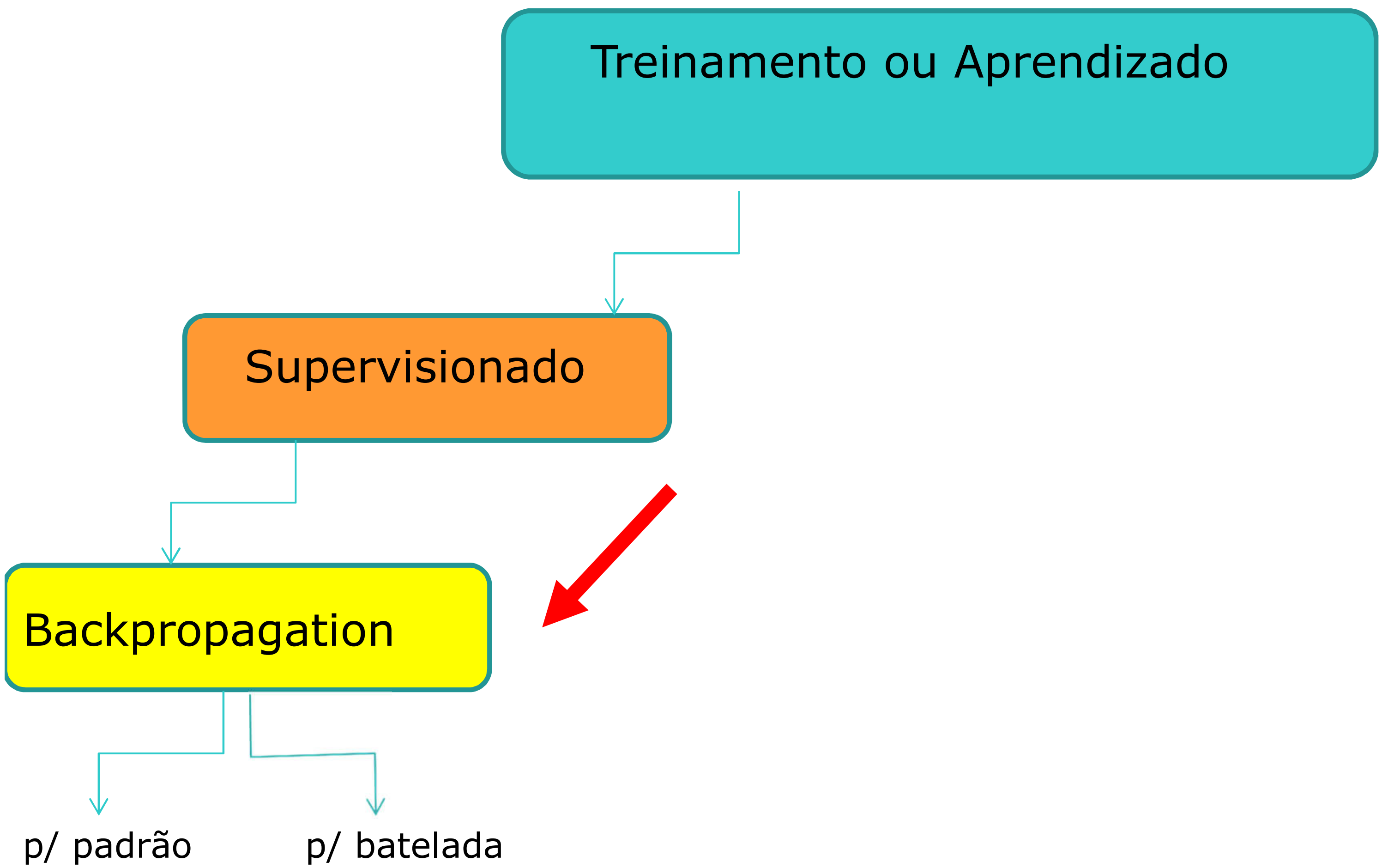
A cada padrão apresentado, compara-se a saída produzida pela rede com a **saída desejada**. Calcula-se o ajuste nos pesos de forma a minimizar o erro na saída.

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

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) o(s) peso(s) são modificados

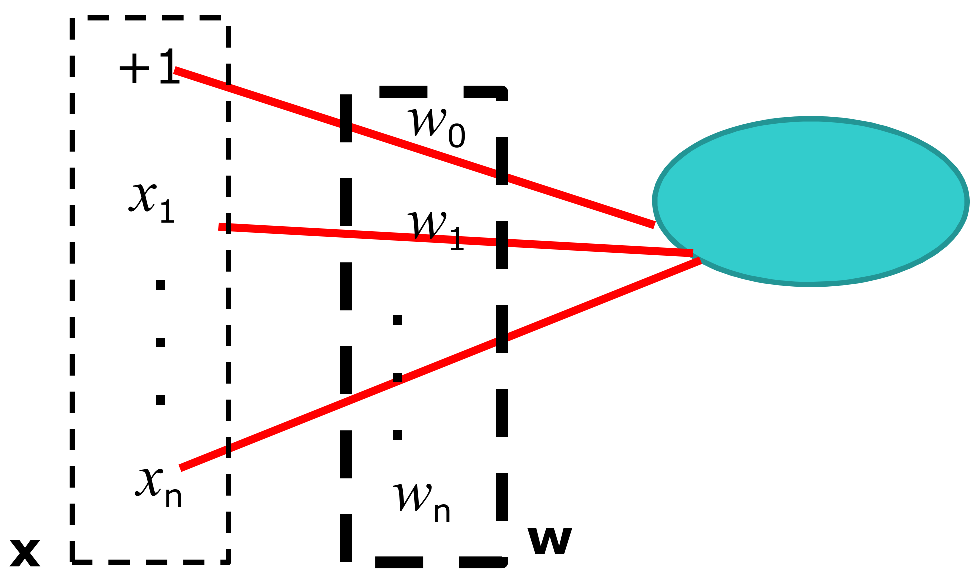
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \sum_{p=1}^{P} \Delta\mathbf{w}_p$$

MLP: Backpropagation Clássico

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

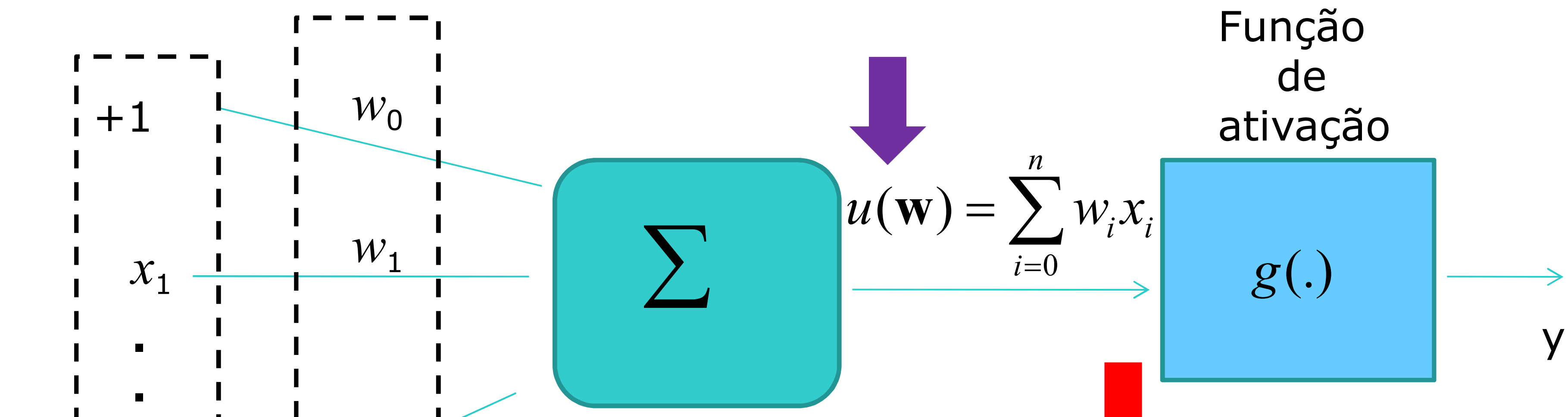
A adaptação (ou ajuste) Δ**w**(t) é aplicada ao vetor **w**(t) no instante t, gerando um vetor corrigido (ou adaptado) no instante t+1, na forma

w(t+1) = **w**(t) + Δ**w**(t)



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'



$$y = g(u(\mathbf{w})) = g\left(\sum_{i=0}^{n} w_i x_i\right)$$

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 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.

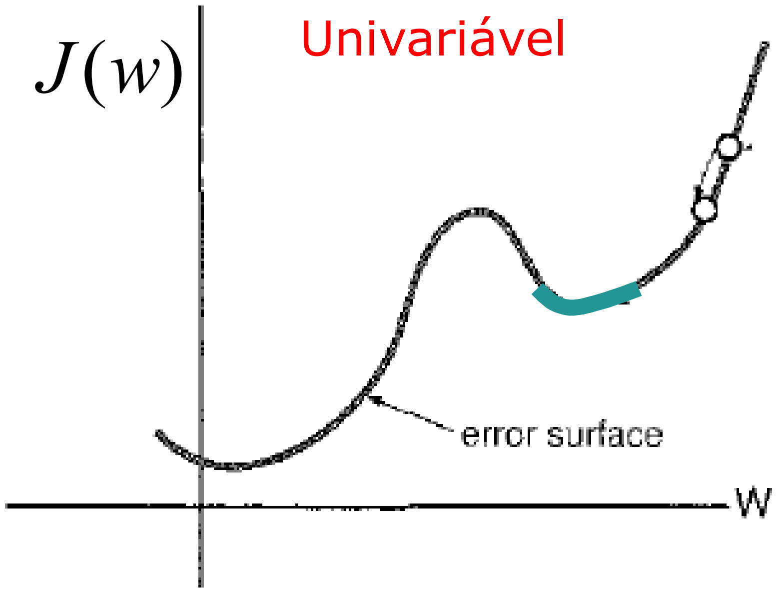
Redes Neurais Artificiais: Treinamento

Problema de minimização do Erro

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

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

$$\text{para } \mathbf{w} = w$$



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

O sinal negativo busca alcançar o mínimo do Erro

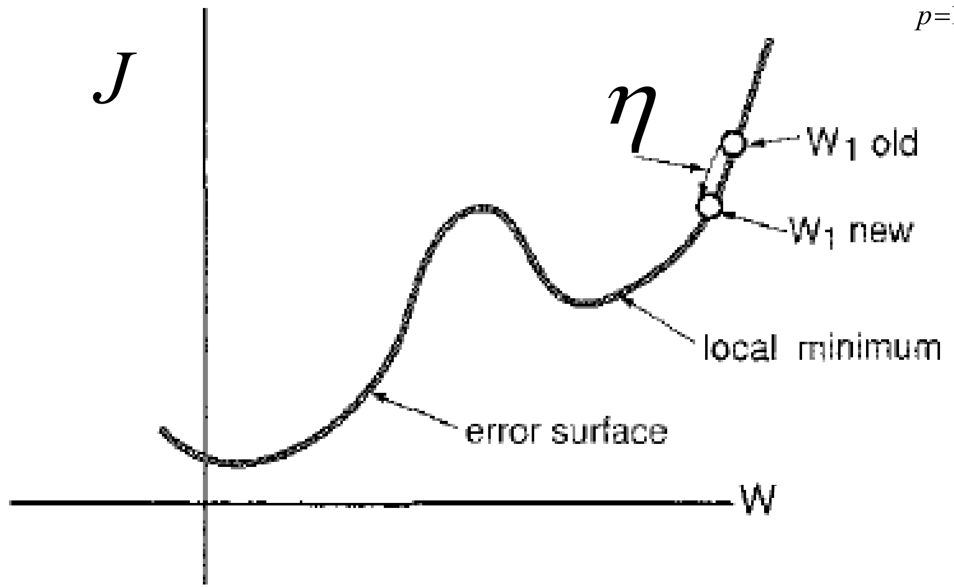
Redes Neurais Artificiais: Treinamento

Problema de minimização do Erro

Univariável

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

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

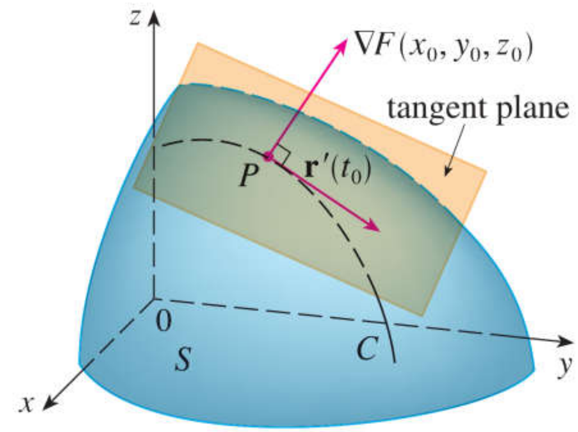


$$J(\mathbf{w}) = \sum_{p=1}^{P} \frac{1}{2}\left(y_{d_p} - y_p(\mathbf{w})\right)^2$$

Multivariável?

Treinamento baseado no gradiente

Resposta: ajustar os pesos **w**=[w_0, w_1, …,w_n] *no* *sentido oposto ao do vetor gradiente da função custo J (ou erro) 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

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

Treinamento baseado no gradiente

O ajuste do vetor de pesos

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

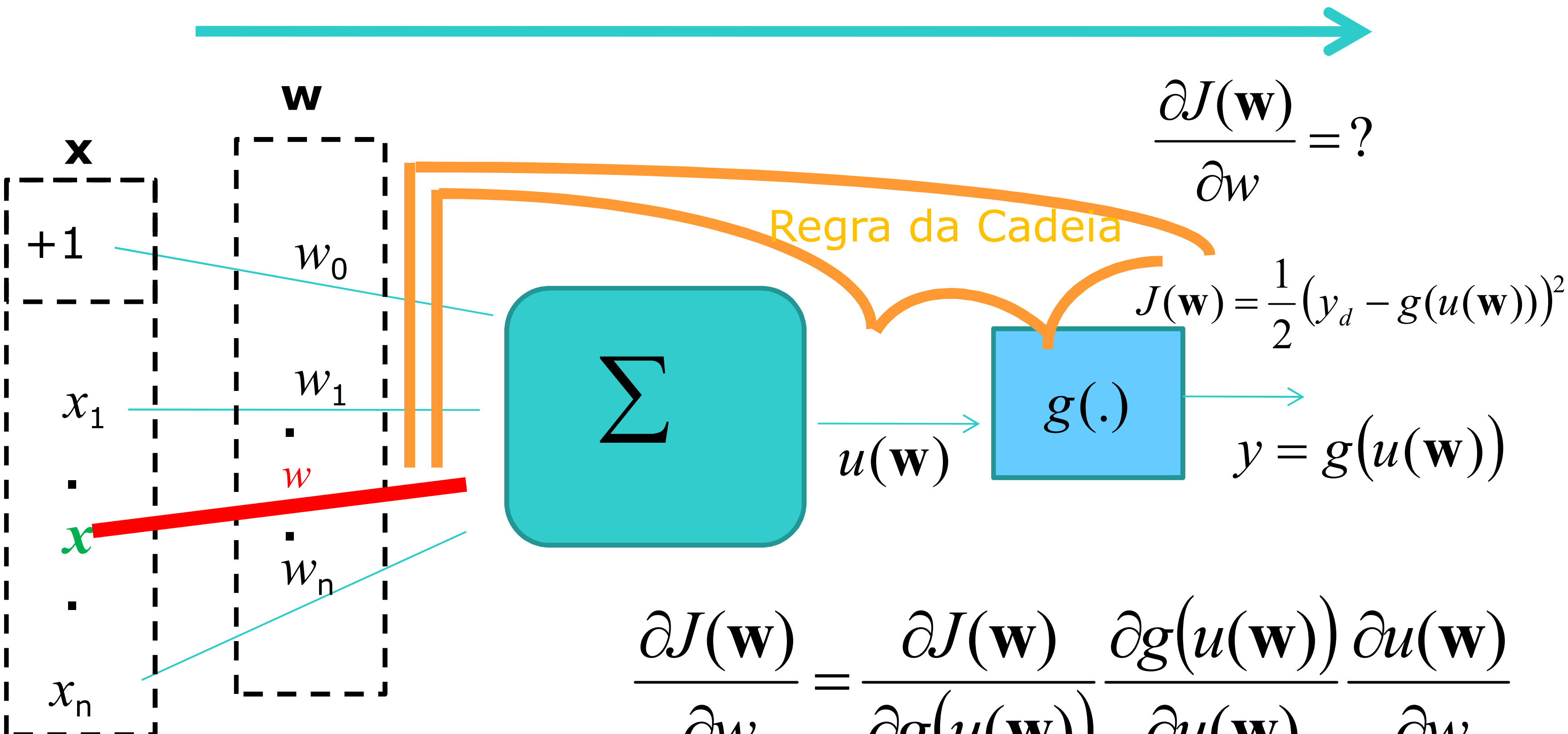
Pode ser calculado para cada elemento w do vetor ***w*** *como*

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

$$\Delta w(t) = -\eta \frac{\partial J(\mathbf{w})}{\partial w} = \eta \left(-\frac{\partial J(\mathbf{w})}{\partial w} \right) \qquad -\frac{\partial J(\mathbf{w})}{\partial w} \text{ ?}$$

Ajuste do peso: Regra da Cadeia

Ajustando o peso w conectado à entrada x de um neurônio qualquer com função de ativação (diferenciável) g



$$\frac{\partial J(\mathbf{w})}{\partial w} = \frac{\partial J(\mathbf{w})}{\partial g(u(\mathbf{w}))} \frac{\partial g(u(\mathbf{w}))}{\partial u(\mathbf{w})} \frac{\partial u(\mathbf{w})}{\partial w}$$

Ajuste do peso depende do erro, derivada e entrada

$$\frac{\partial J(\mathbf{w})}{\partial w} = \left(\frac{\partial J(\mathbf{w})}{\partial g(u(\mathbf{w}))} \frac{\partial g(u(\mathbf{w}))}{\partial u(\mathbf{w})} \frac{\partial u(\mathbf{w})}{\partial w} \right)$$

$$-\frac{\partial J(\mathbf{w})}{\partial w} = \mathrm{erro} \; g'(u(\mathbf{w})) \; x$$

Camadas intermediárias: *ebp*

Camada saída: y_d - y

Ajuste do peso depende do erro, derivada e entrada

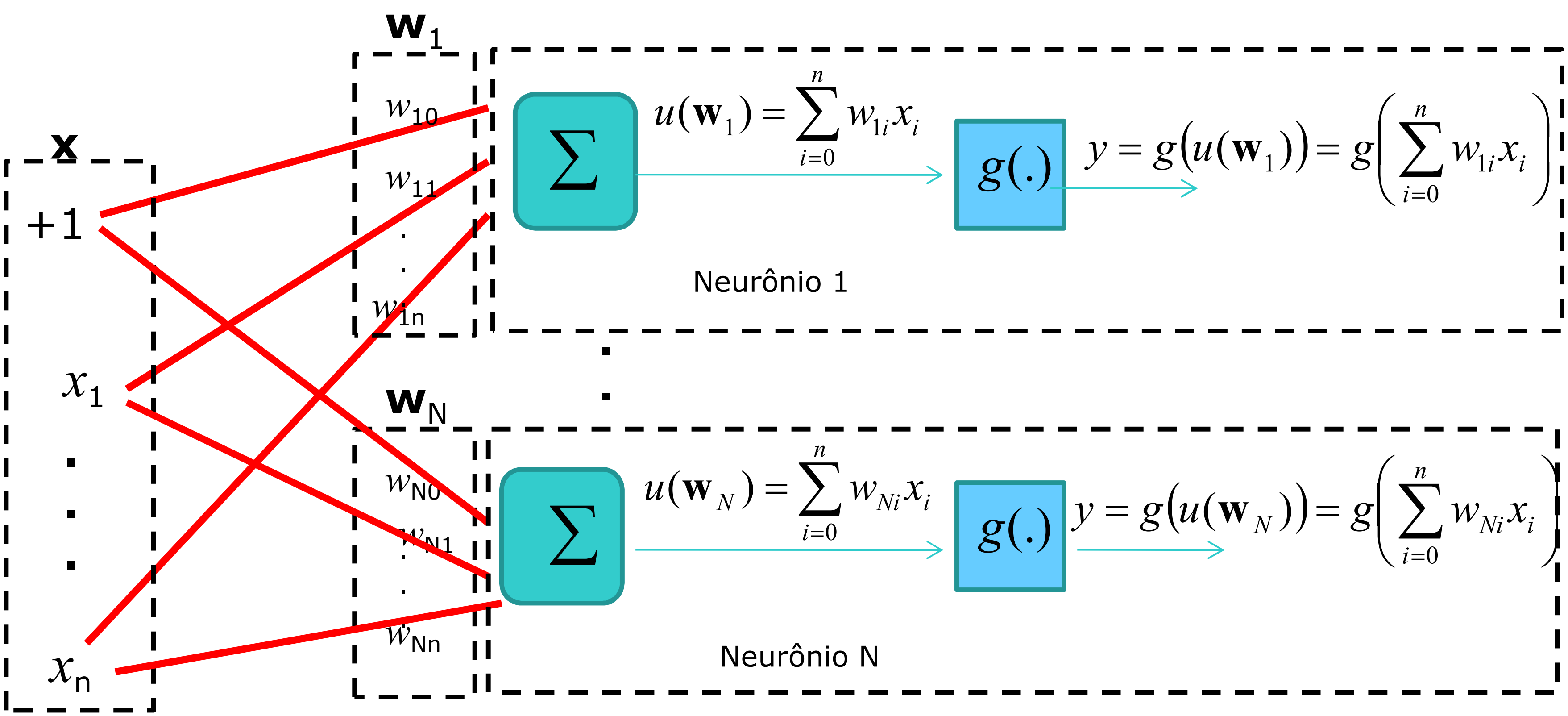
$$\frac{\partial J(\mathbf{w})}{\partial w} = \left(\frac{\partial J(\mathbf{w})}{\partial g(u(\mathbf{w}))} \cdot \frac{\partial g(u(\mathbf{w}))}{\partial u(\mathbf{w})} \cdot \frac{\partial u(\mathbf{w})}{\partial w} \right)$$

$$-\frac{\partial J(\mathbf{w})}{\partial w} = \text{erro} \quad g'(u(\mathbf{w})) \quad x$$

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



Exemplo de BackPropagation (por batelada) de redes de camada única (função de ativação $g(.)$)

Defina η e inicialize os vetores de pesos dos N neurônios da rede : $\mathbf{w}_k$, k=1,...,N
t = 1; //inicializa época
repita
para cada neurônio k k=1,...,N faça
para cada peso w_{ki} i=0,...,n do neurônio k faça
para cada par $(\mathbf{x}^p, y^p_d)$ p=1,...,P faça

$$-\frac{\partial J_p}{\partial w_{ki}} = \left(y_{d_p} - g\left(u_p(\mathbf{w}_k)\right)\right)\frac{\partial g\left(u_p(\mathbf{w}_k)\right)}{\partial u_p(\mathbf{w}_k)} x_{pi} = e_{pk} g'(\mathbf{w}_k) x_{pi}$$

fim para
Calcule a atualização geral (**batelada**) para o i-ésimo peso do neurônio k

$$\Delta w_{ki} = \eta \sum_{p=1}^{p} e_{pk} g'(\boldsymbol{w}_k) x_{pi}$$

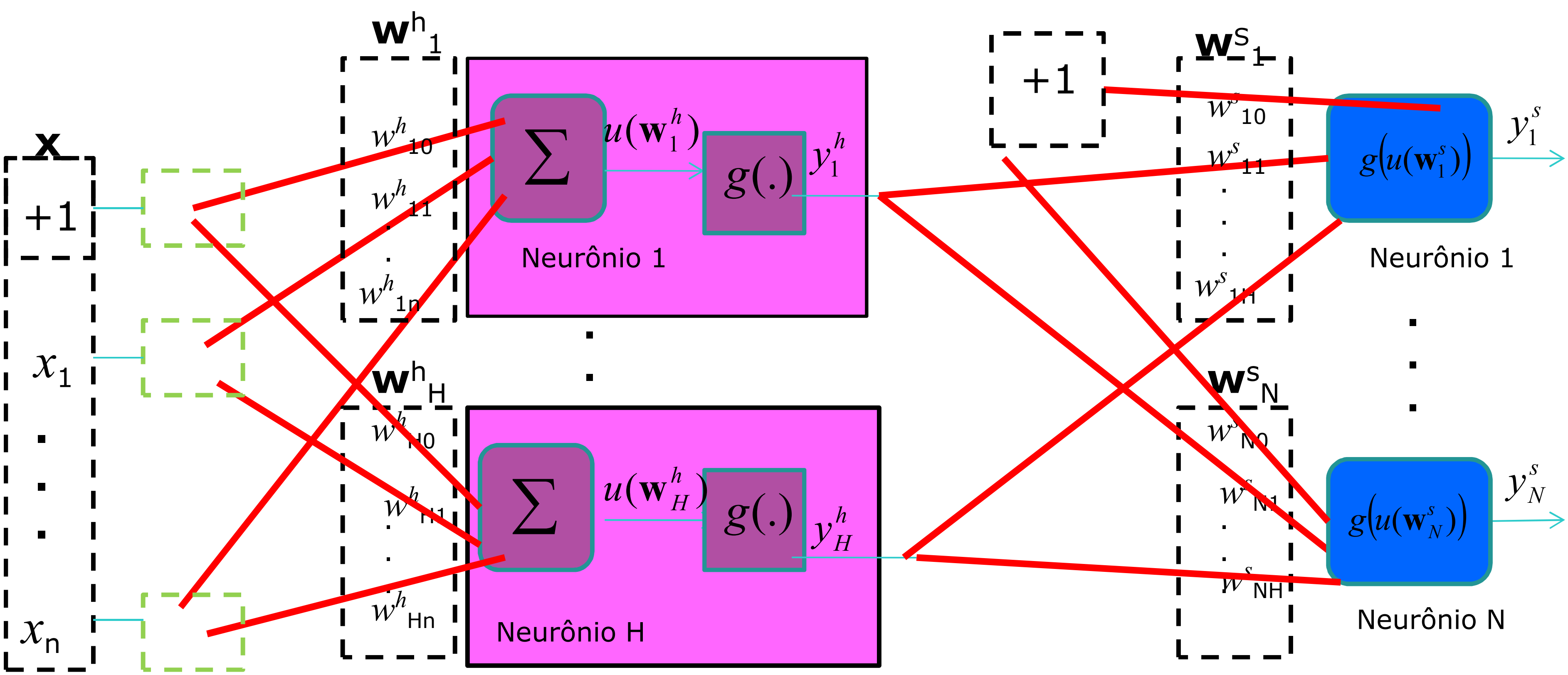
fim para
fim para
para cada neurônio k k=1,...,N faça
para cada peso w_{ki} i=0,...,n do neur k faça
Atualize o i-ésimo peso do neuronio k

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

fim para
fim para
t = t + 1; //incrementa época
até atingir condição de parada (max_epocas ou aumento_erro_val ou erro_trein<ξ)

BackPropagation: rede de múltiplas camadas

Ajustando os pesos de uma rede com duas camadas e neurônios com função de ativação (diferenciável) g



MLP:Ajuste do peso função do erro, derivada e entrada

$$\frac{\partial J(\mathbf{w})}{\partial w} = \left(\frac{\partial J(\mathbf{w})}{\partial g(u(\mathbf{w}))} \cdot \frac{\partial g(u(\mathbf{w}))}{\partial u(\mathbf{w})} \cdot \frac{\partial u(\mathbf{w})}{\partial w} \right)$$

Camada de saída

$$-\frac{\partial J(\mathbf{w})}{\partial w} = \text{erro} \; g'(u(\mathbf{w})) \; x$$

Camada saída: y_d - y

Similar à rede de 1 camada

MLP:Ajuste do peso função do erro, derivada e entrada

$$\frac{\partial J(\mathbf{w})}{\partial w} = \left(\frac{\partial J(\mathbf{w})}{\partial g(u(\mathbf{w}))} \frac{\partial g(u(\mathbf{w}))}{\partial u(\mathbf{w})} \frac{\partial u(\mathbf{w})}{\partial w} \right)$$

Camada de saída

$$-\frac{\partial J(\mathbf{w})}{\partial w} = \text{erro} \; g'(u(\mathbf{w})) \; x$$

entrada (x) = saída (y^h) camada anterior

Camada saída: y_d - y

MLP:Ajuste do peso função do erro, derivada e entrada

$$\frac{\partial J(\mathbf{w})}{\partial w} = \left(\frac{\partial J(\mathbf{w})}{\partial g(u(\mathbf{w}))} \frac{\partial g(u(\mathbf{w}))}{\partial u(\mathbf{w})} \frac{\partial u(\mathbf{w})}{\partial w} \right)$$

Camada de saída

$$-\frac{\partial J(\mathbf{w})}{\partial w} = \text{erro} \;\; g'(u(\mathbf{w})) \;\; x$$

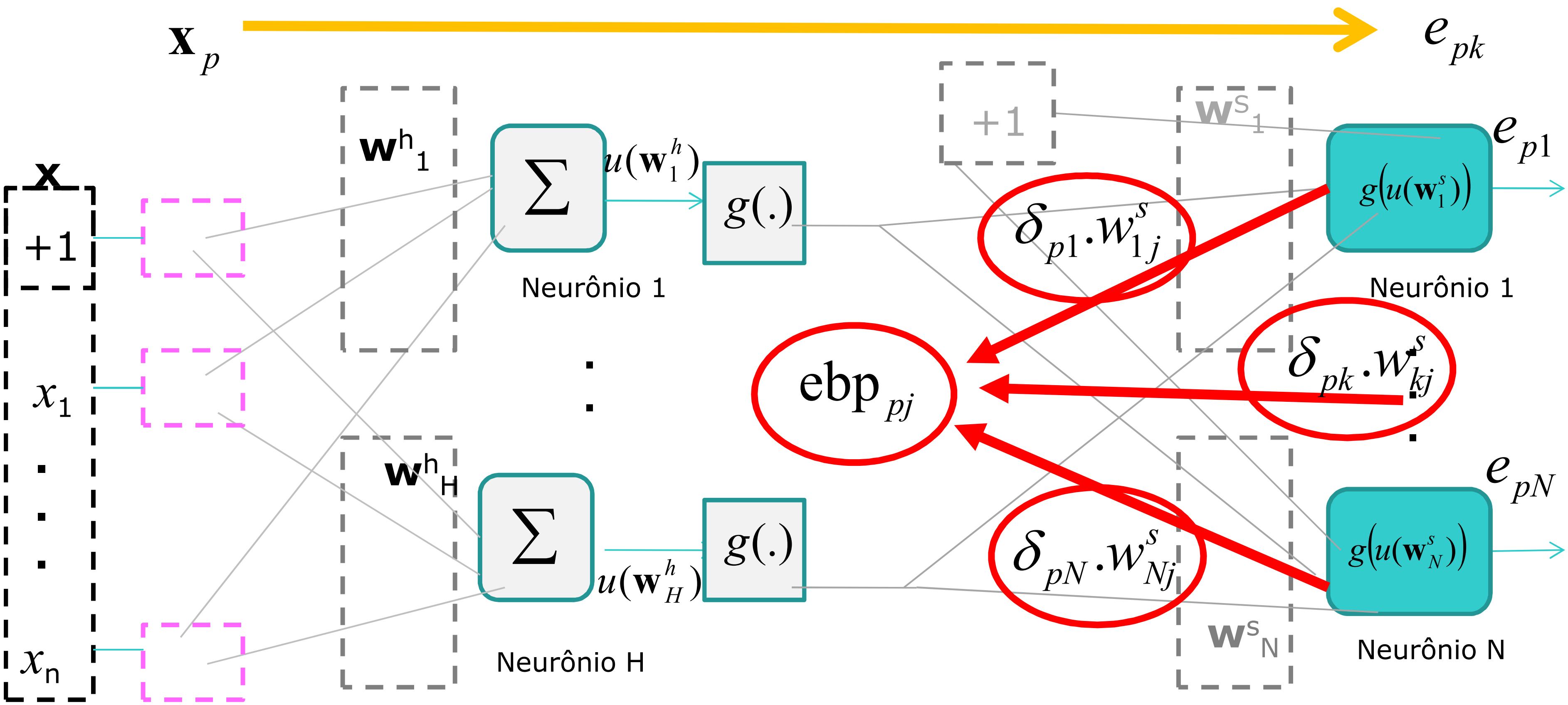
entrada (x) = saída (y^h) camada anterior

Camada saída: y_d - y

(fator δ que ponderado pelos pesos é back propagado) para a camada anterior

BackPropagation : rede de múltiplas camadas

Duas fases: propaga entrada e retroprogapa o erro



$$\delta_{pk} = e_{pk}\, g'(\mathbf{w}_k^s) \qquad e_{pk}$$

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

CAMADA DE SAÍDA

$$-\frac{\partial J_p}{\partial w_{kj}^s} = \left(y_{d_{pk}} - g\left(u_p(\mathbf{w}_k^s)\right)\right)\frac{\partial g\left(u_p(\mathbf{w}_k^s)\right)}{\partial u_p(\mathbf{w}_k^s)} x_{pkj} = (y_{pk} - y_{d_{pk}})\, g'(u_p(\mathbf{w}_k^s))\,\mathrm{x}_{pkj}^s = e_{pk}\, g'(u_p(\mathbf{w}_k^s))\,\mathrm{x}_{pkj}^s = \delta_{pk}\,\mathrm{x}_{pkj}^s$$

CAMADA INTERMEDIÁRIA

$$-\frac{\partial J_p}{\partial w_{ji}^h} = \left[\sum_{k=1}^{N} \left(y_{d_{pk}} - g\left(u_p(\mathbf{w}_k^s)\right)\right)\frac{\partial g\left(u_p(\mathbf{w}_k^s)\right)}{\partial u_p(\mathbf{w}_k^s)} w_{kj}^s\right]\frac{\partial g\left(u_p(\mathbf{w}_j^h)\right)}{\partial u_p(\mathbf{w}_j^h)} x_{pji}^h = \left[\overbrace{\sum_{k=1}^{N} \delta_{pk}\, w_{kj}^s}^{\text{ebp}_j}\right] g'(u_p(\mathbf{w}_j^h))\, x_{pji}^h$$

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

CAMADA DE SAÍDA

$$-\frac{\partial J_p}{\partial w_{kj}^s} = \left(y_{d_{pk}} - g\left(u_p(\mathbf{w}_k^s)\right)\right)\frac{\partial g\left(u_p(\mathbf{w}_k^s)\right)}{\partial u_p(\mathbf{w}_k^s)} x_{pkj} = (y_{pk} - y_{d_{pk}})g'(u_p(\mathbf{w}_k^s))\mathrm{x}_{pkj}^s = \underline{e_{pk}\,g'(u_p(\mathbf{w}_k^s))\mathrm{x}_{pkj}^s} = \delta_{pk}\mathrm{x}_{pkj}^s$$

CAMADA INTERMEDIÁRIA

ebp$_j$

$$-\frac{\partial J_p}{\partial w_{ji}^h} = \left[\sum_{k=1}^{N}\left(y_{d_{pk}} - g\left(u_p(\mathbf{w}_k^s)\right)\right)\frac{\partial g\left(u_p(\mathbf{w}_k^s)\right)}{\partial u_p(\mathbf{w}_k^s)} w_{kj}^s\right]\frac{\partial g\left(u_p(\mathbf{w}_j^h)\right)}{\partial u_p(\mathbf{w}_j^h)} x_{pji}^h = \underline{\left[\underline{\sum_{k=1}^{N}\delta_{pk}w_{kj}^s}\right]g'(u_p(\mathbf{w}_j^h))\,x_{pji}^h}$$

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

CAMADA DE SAÍDA

$$-\frac{\partial J_p}{\partial w^s_{kj}} = \left(y_{d_{pk}} - g\left(u_p(\mathbf{w}^s_k)\right)\right)\frac{\partial g\left(u_p(\mathbf{w}^s_k)\right)}{\partial u_p(\mathbf{w}^s_k)} x_{pkj} = (y_{pk} - y_{d_{pk}})\,g'(u_p(\mathbf{w}^s_k))\mathbf{x}^s_{pkj} = \underline{e_{pk}\,g'(u_p(\mathbf{w}^s_k))\mathbf{x}^s_{pkj}} = \delta_{pk}\mathbf{x}^s_{pkj}$$

CAMADA INTERMEDIÁRIA

$$-\frac{\partial J_p}{\partial w^h_{ji}} = \left[\sum_{k=1}^{N}\left(y_{d_{pk}} - g\left(u_p(\mathbf{w}^s_k)\right)\right)\frac{\partial g\left(u_p(\mathbf{w}^s_k)\right)}{\partial u_p(\mathbf{w}^s_k)} w^s_{kj}\right]\frac{\partial g\left(u_p(\mathbf{w}^h_j)\right)}{\partial u_p(\mathbf{w}^h_j)} x^h_{pji} = \overbrace{\left[\underline{\sum_{k=1}^{N}\delta_{pk} w^s_{kj}}\right]}^{ebp_j} g'(u_p(\mathbf{w}^h_j))\,x^h_{pji}$$

$$w^s_{kj}(t+1) = w^s_{kj}(t) - \eta\sum_{p=1}^{P}\boxed{\frac{\partial J_p}{\partial w^s_{kj}}}$$

$$w^h_{ji}(t+1) = w^h_{ji}(t+1) - \eta\sum_{p=1}^{P}\boxed{\frac{\partial J_p}{\partial w^h_{ji}}}$$

Treinamento por batelada

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

Defina η, Inicialize os pesos dos H+N: $\mathbf{w}^h_j$, j=1,...,H, $\mathbf{w}^s_k$, k=1,...,N.

t = 1;

Repita

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

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

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

$$-\frac{\partial J_p}{\partial w^s_{kj}} = \left(y_{d_{pk}} - g\left(u_p(\mathbf{w}^s_k)\right)\right)\frac{\partial g\left(u_p(\mathbf{w}^s_k)\right)}{\partial u_p(\mathbf{w}^s_k)} x_{pkj} = (y_{pk} - y_{d_{pk}})g'(u_p(\mathbf{w}^s_k))\mathrm{x}^s_{pkj} = \underline{e_{pk}g'(u_p(\mathbf{w}^s_k))\mathrm{x}^s_{pkj}} = \delta_{pk}\mathrm{x}^s_{pkj}$$

fim para

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

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

$$-\frac{\partial J_p}{\partial w^h_{ji}} = \left[\sum_{k=1}^{N}\left(y_{d_{pk}} - g\left(u_p(\mathbf{w}^s_k)\right)\right)\frac{\partial g\left(u_p(\mathbf{w}^s_k)\right)}{\partial u_p(\mathbf{w}^s_k)} w^s_{kj}\right]\frac{\partial g\left(u_p(\mathbf{w}^h_j)\right)}{\partial u_p(\mathbf{w}^h_j)} x^h_{pji} = \underline{\overbrace{\left[\sum_{k=1}^{N}\delta_{pk}w^s_{kj}\right]}^{\text{ebp}_j} g'(u_p(\mathbf{w}^h_j))x^h_{pji}}$$

fim para

fim para

fim para

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

$$\boxed{w^s_{kj}(t+1) = w^s_{kj}(t) - \eta\sum_{p=1}^{P}\frac{\partial J_p}{\partial w^s_{kj}}}$$

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

$$\boxed{w^e_{ji}(t+1) = w^e_{ji}(t) - \eta\sum_{p=1}^{P}\frac{\partial J_p}{\partial w^e_{ji}}}$$

t=t+1;

até atingir condição de parada

Abstração: algoritmo backpropagation de redes de 3 camadas (função de ativação $g(.)$): **batelada**

Defina η, Inicialize os pesos dos H+N: $\mathbf{w}^h_j$, j=1,...,H, $\mathbf{w}^s_k$, k=1,...,N.

t = 1;

Repita

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

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

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

$$-\frac{\partial J_p}{\partial w^s_{kj}} = e_{pk} g'(u_p(\mathbf{w}^s_k)) \mathrm{x}^s_{pkj} = \delta_{pk} \mathrm{x}^s_{pkj}$$

fim para

fim para

fim para

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

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

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

$$-\frac{\partial J_p}{\partial w^h_{ji}} = \left[\sum_{k=1}^{N} \delta_{pk} w^s_{kj}\right] g'(u_p(\mathbf{w}^h_j)) x^h_{pji}$$

fim para

fim para

fim para Atualize o peso j (j=1,..H) do neuronio k.

$$\boxed{w^s_{kj}(t+1) = w^s_{kj}(t) - \eta \sum_{p=1}^{P} \frac{\partial J_p}{\partial w^s_{kj}}}$$

Atualize o peso i (i=1,..n) do neur j j=1,...,HB

$$\boxed{w^{hB}_{ji}(t+1) = w^{hB}_{ji}(t) - \eta \sum_{p=1}^{P} \frac{\partial J_p}{\partial w^{hB}_{ji}}}$$

t=t+1;

até atingir condição de parada

CADAMADA INTERMEDIARIA h**A** ????

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]