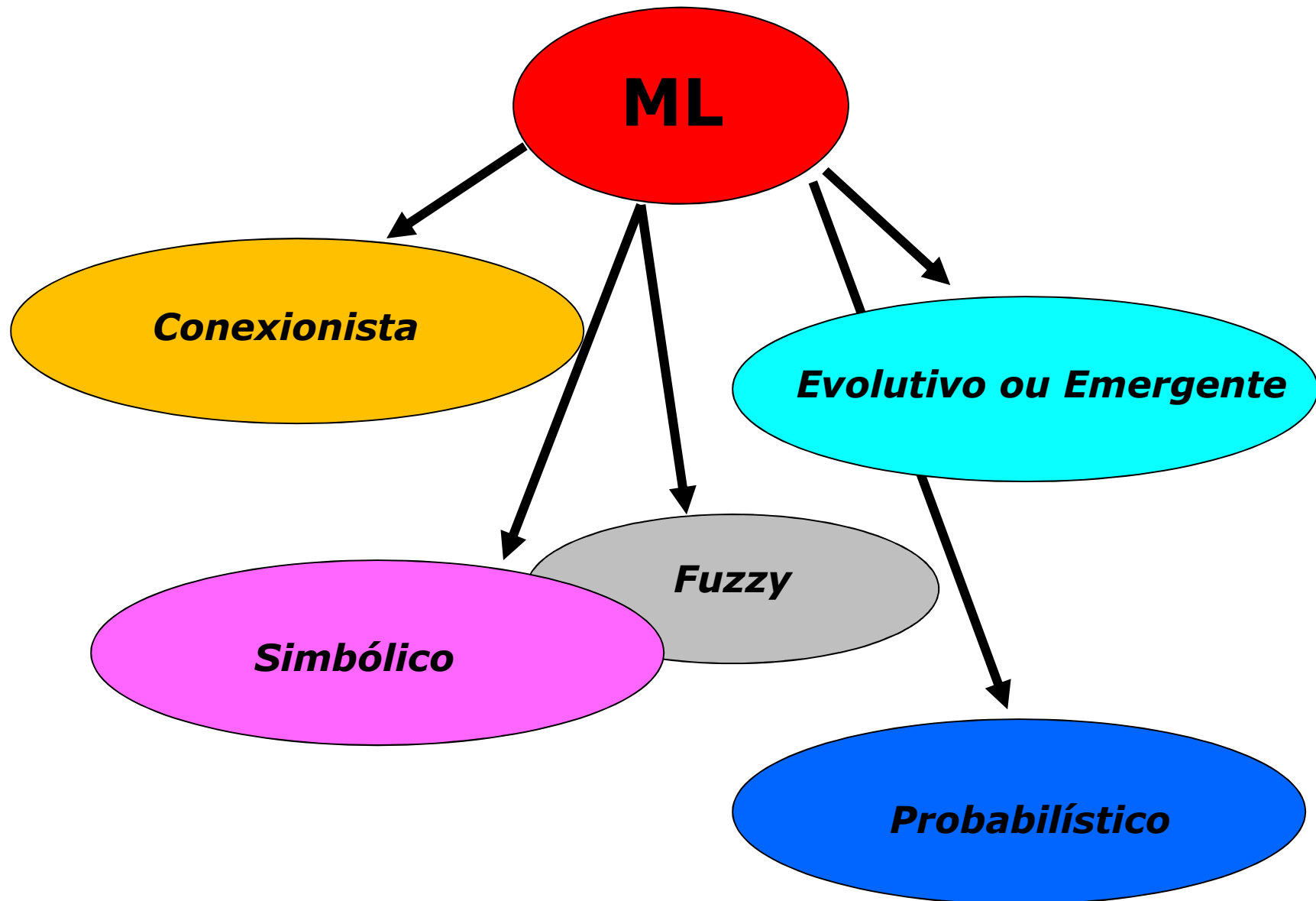

SISTEMAS INTELIGENTES 1

Aprendizado (*Learning*)

Aprendizado de Máquina (Machine Learning)



Aprendizado de Máquina (Machine Learning)

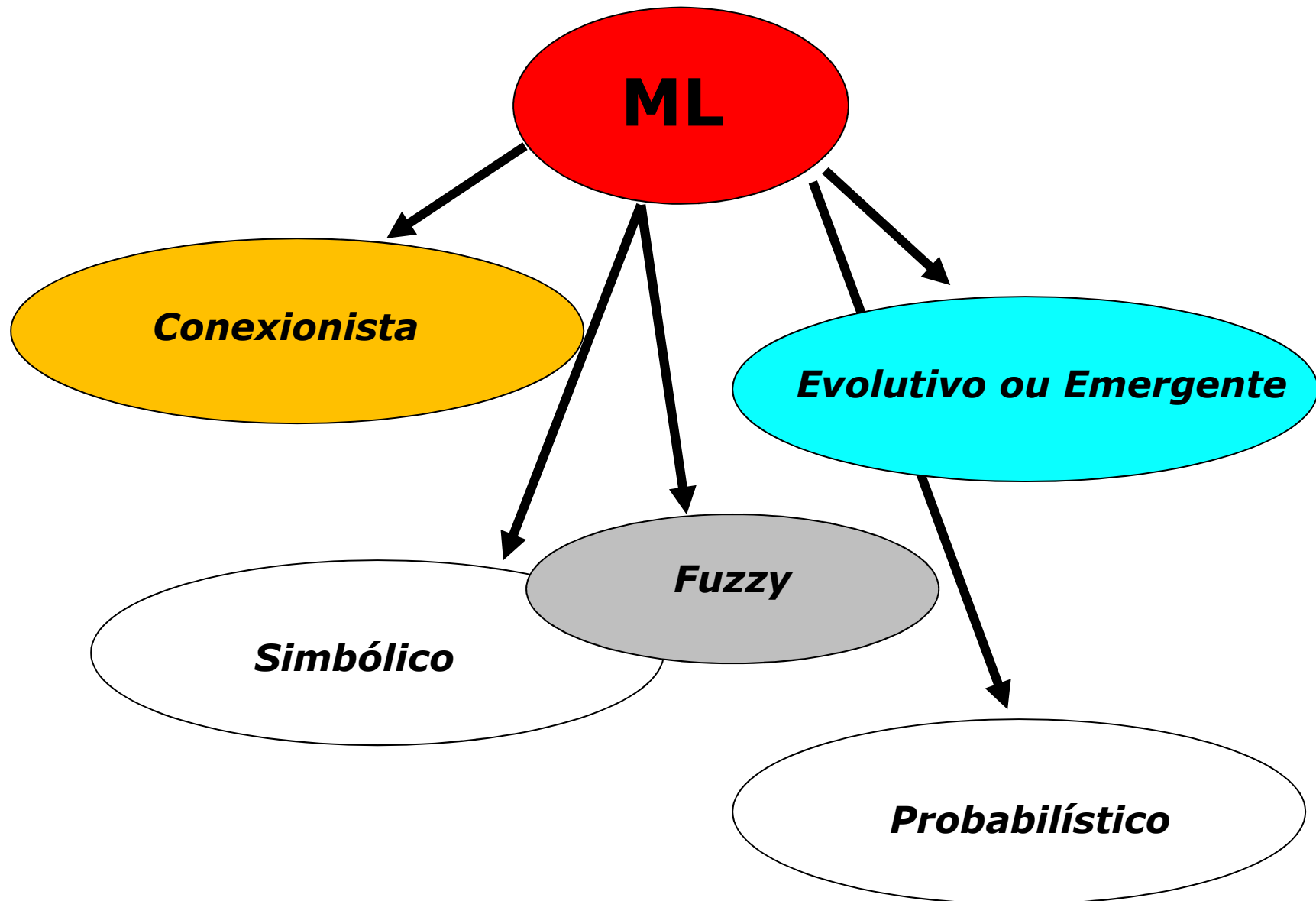
Questões Filosóficas envolvidas:

Generalização: extração de padrões invariantes dos dados de treinamento de forma a responder corretamente para padrões não treinados.

Viés Indutivo (*Inductive bias*): Este viés se refere a como o “modelador” usa sua própria intuição e heurística para projetar a representação, modelo ou algoritmo que dão suporte ao processo de aprendizagem. Isto pode ajudar mas também pode limitar o processo de aprendizado.

Dilema dos empiricistas (*Empiricist's Dilemma*): se não há nenhum viés indutivo, algo útil pode realmente ser aprendido? Em geral esta questão está associada a aprendizado não supervisionado e emergente.

Aprendizado de Máquina (Machine Learning)





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



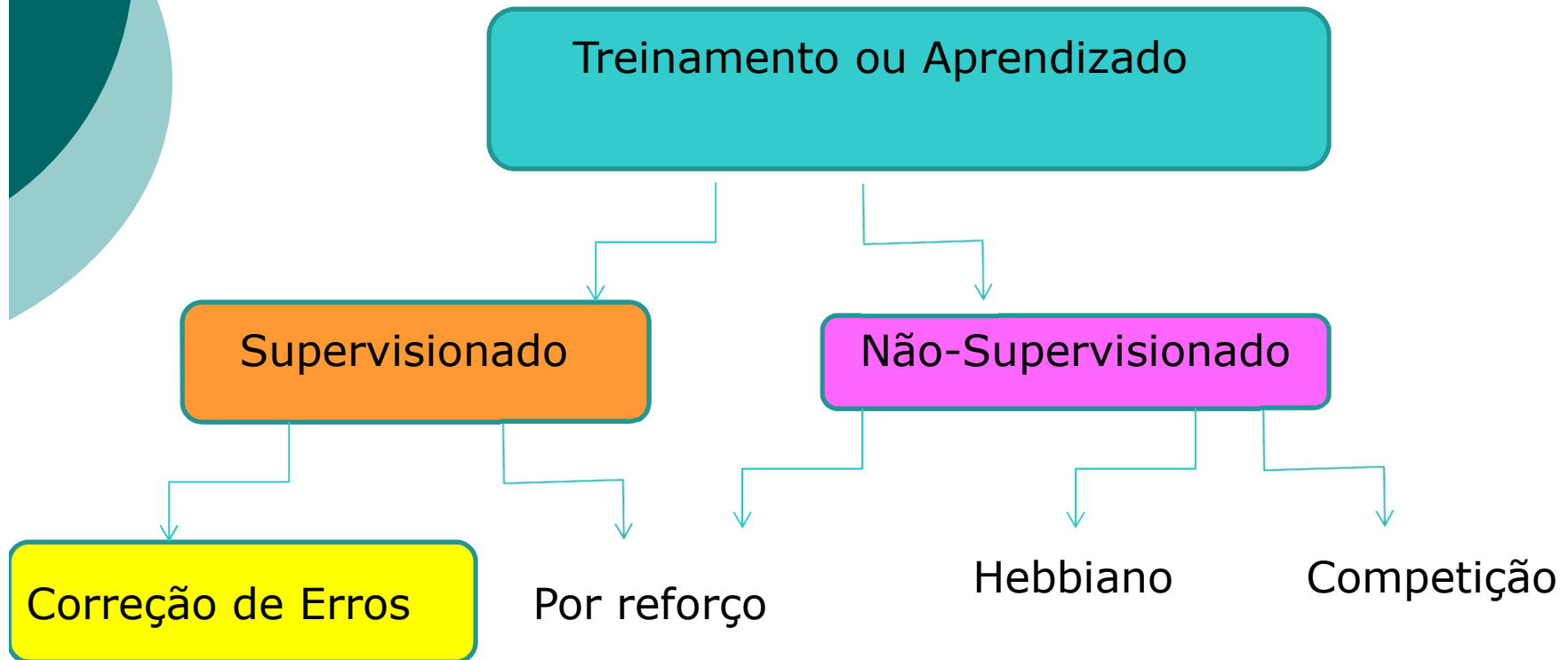
Redes Neurais Artificiais: Treinamento

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)
Exemplo: algoritmo Backpropagation

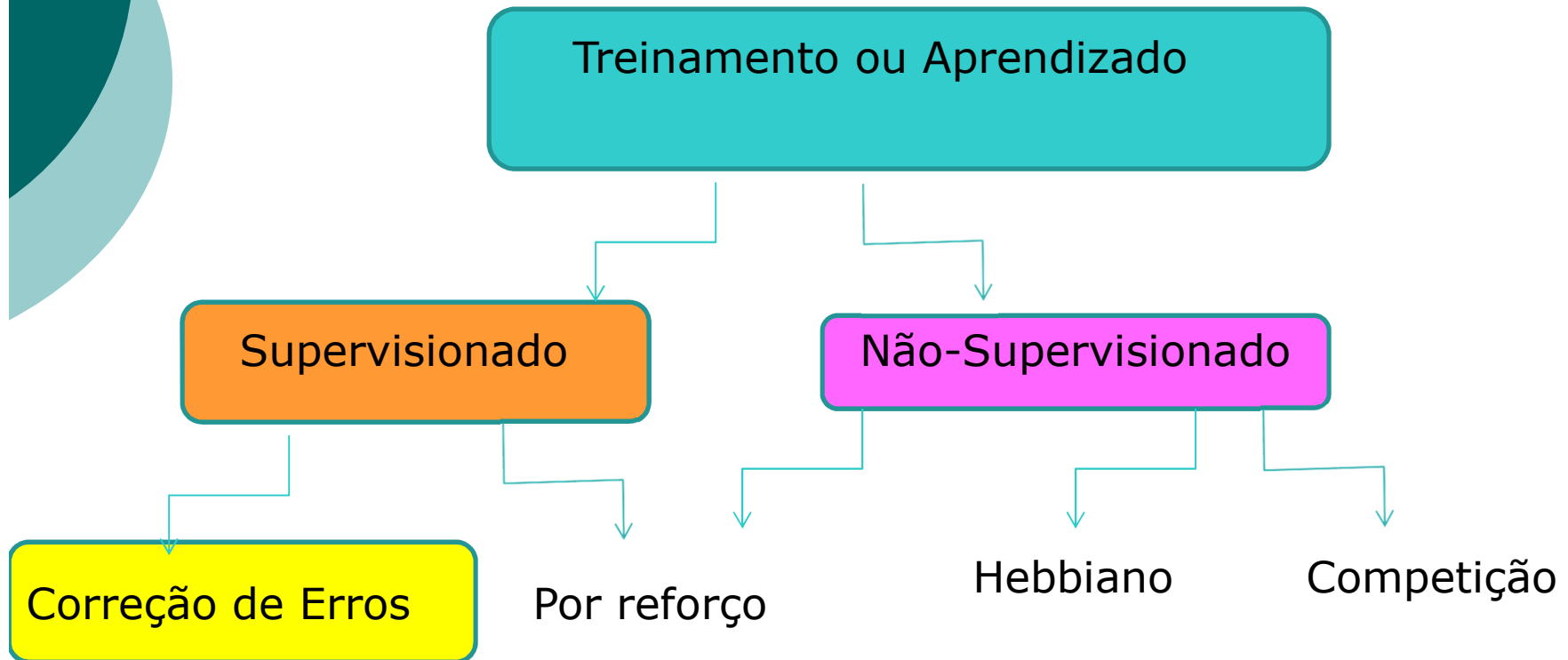
Aprendizado Não-supervisionado:

- Dados de entrada são fornecidos à rede
- Não há supervisor externo (professor)
- Exemplo: 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 Supervisionado:

O modelo mais conhecido de aprendizado supervisionado é 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**).



RNA: Aprendizado Supervisionado

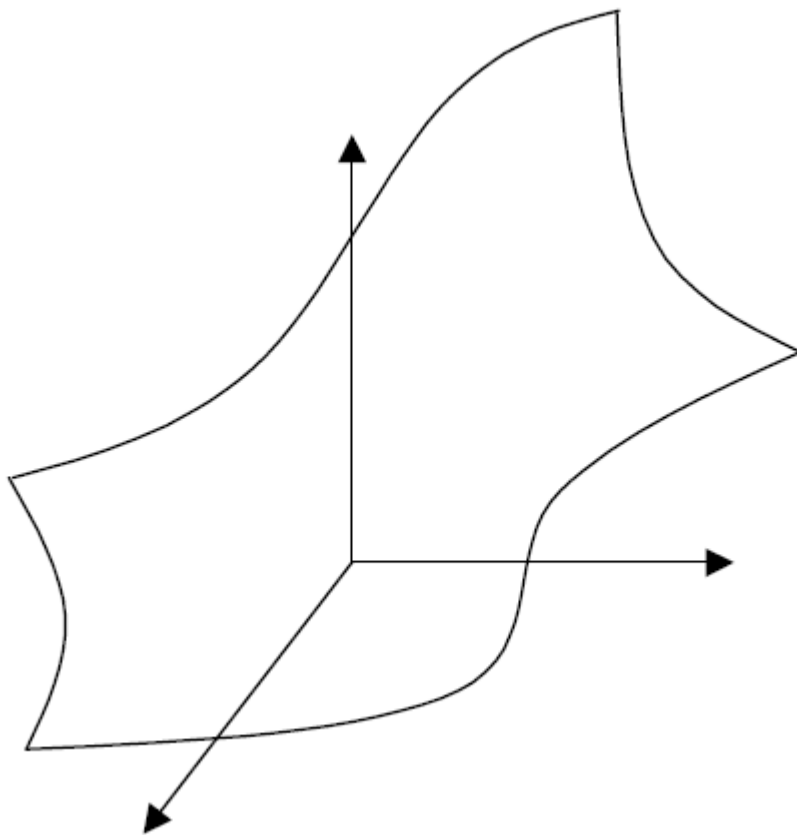
- Aprendizado Supervisionado e o Processo de Mapeamento Entrada Saída.

Em geral, três mapeamentos estão envolvidos:

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

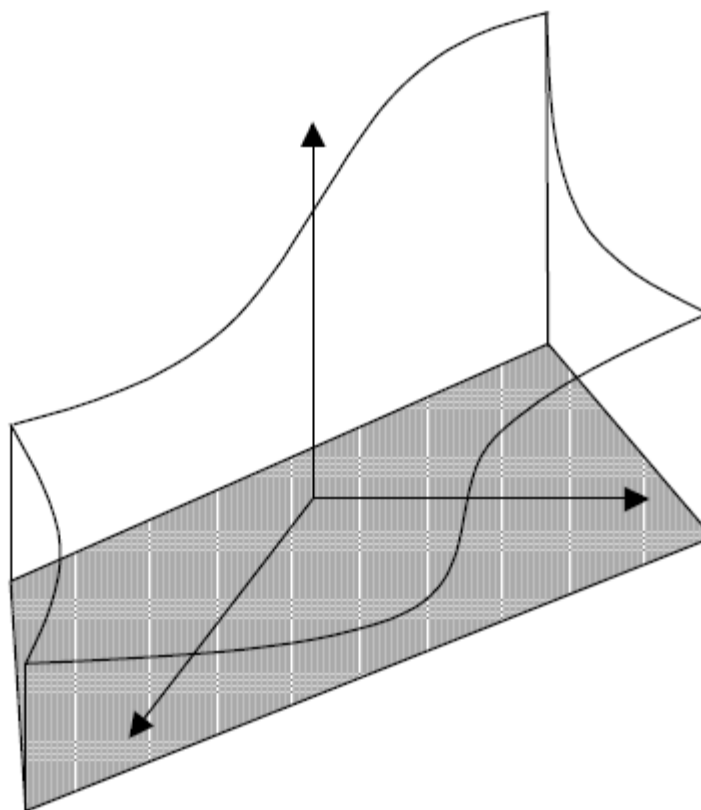
Aprendizado Supervisionado: Mapeamento I/O

Mapeamento a ser aproximado



Aprendizado Supervisionado: Mapeamento I/O

Mapeamento a ser aproximado (região de operação)



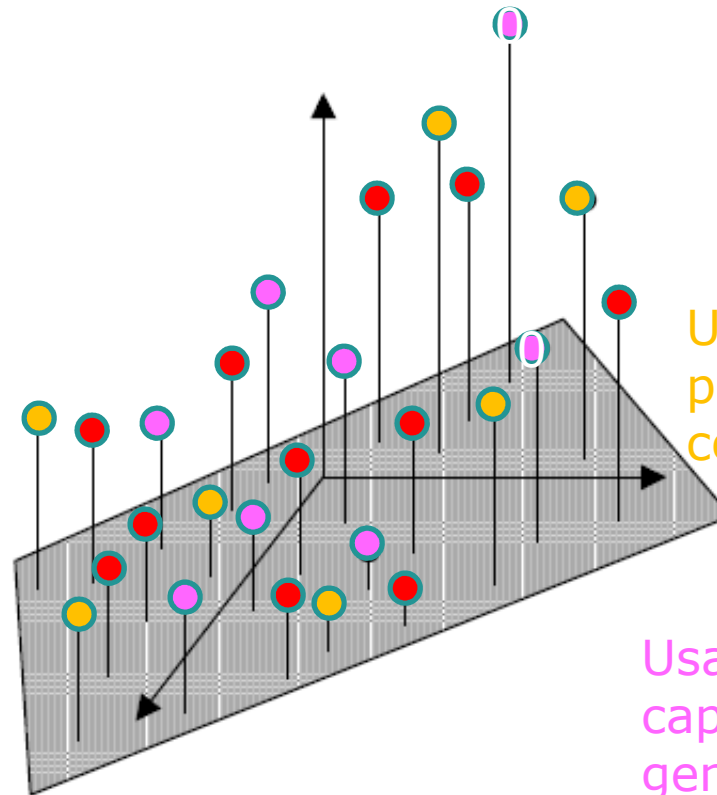
Aprendizado Supervisionado: Mapeamento I/O

Mapeamento a ser aproximado (dados amostrados)

Dados de
treinamento

Dados de
validação

Dados de teste



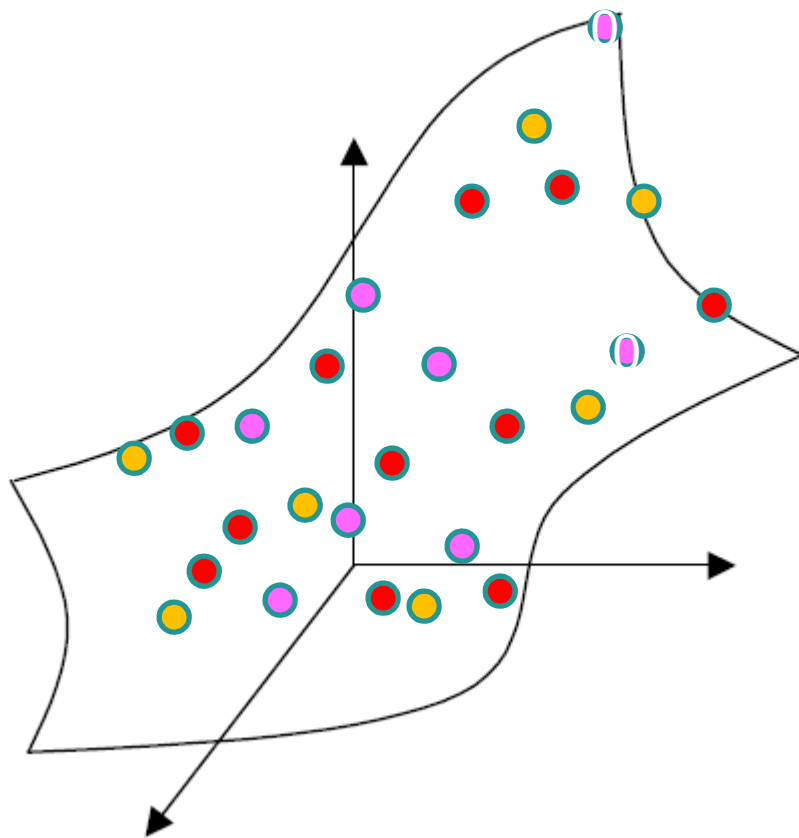
Usados no ajuste de
parâmetros da RN

Usados (em geral)
para determinar a
condição de parada

Usados para testar a
capacidade de
generalização da rede

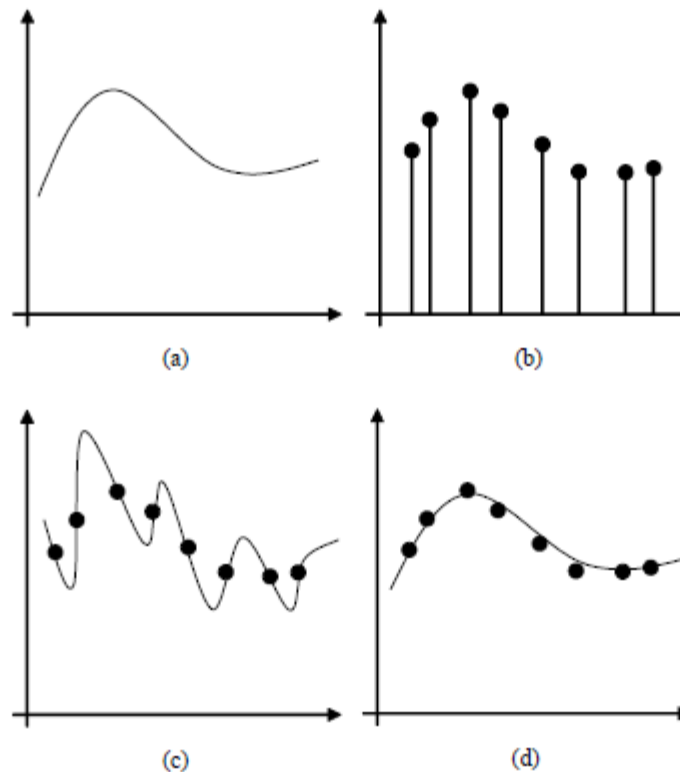
Aprendizado Supervisionado: Mapeamento I/O

Superfície de erros



Aprendizado Supervisionado: Mapeamento I/O

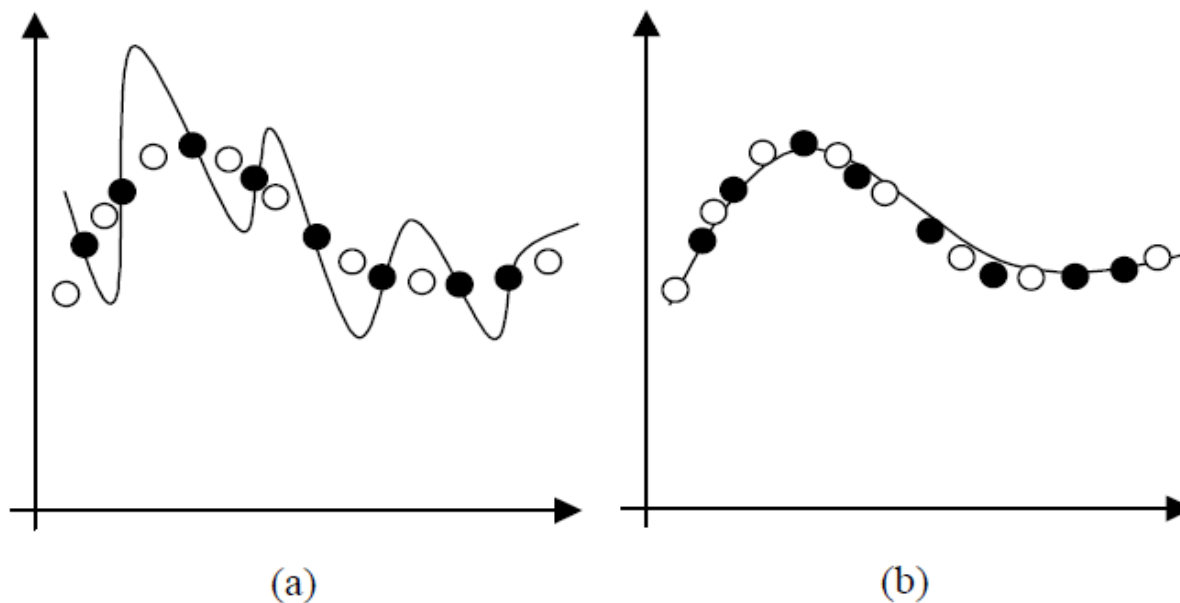
Interpolação x Aproximação



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

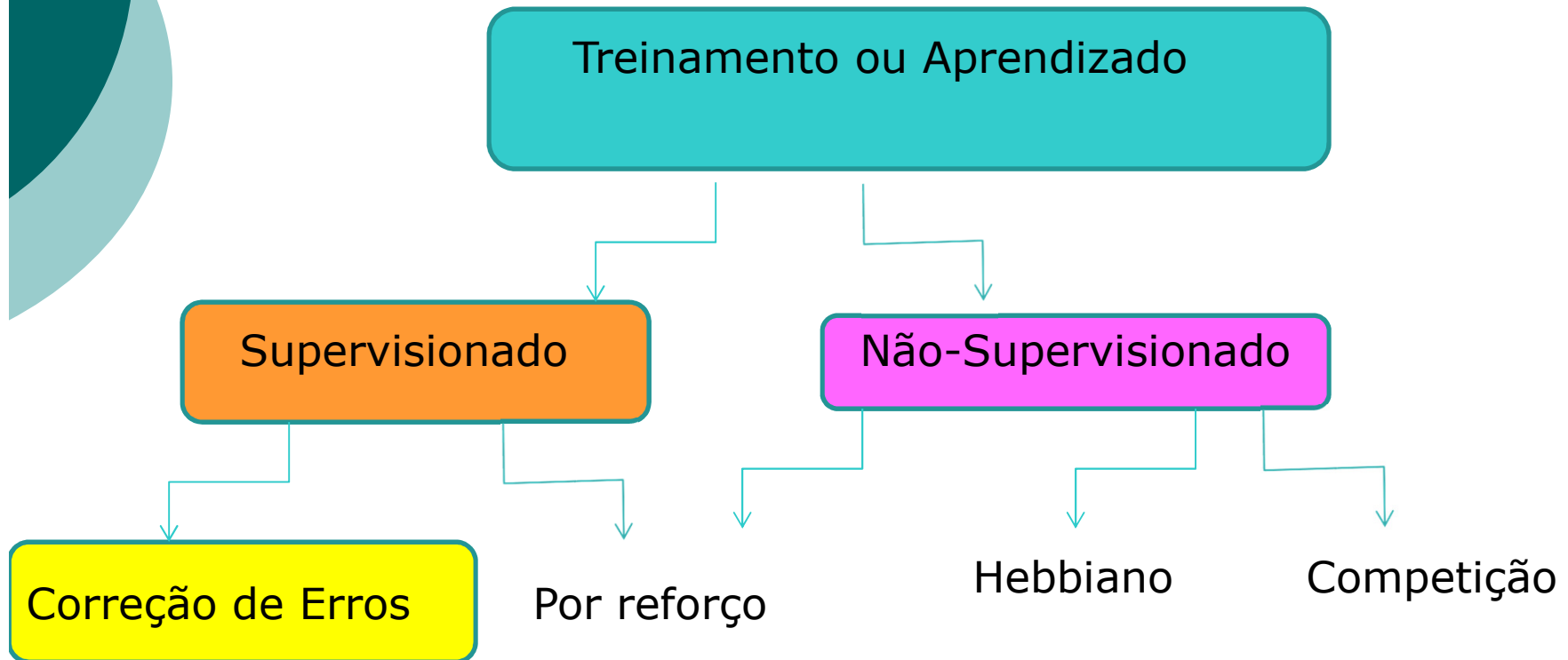
Aprendizado Supervisionado: Mapeamento I/O

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

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-supervisionado:

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”

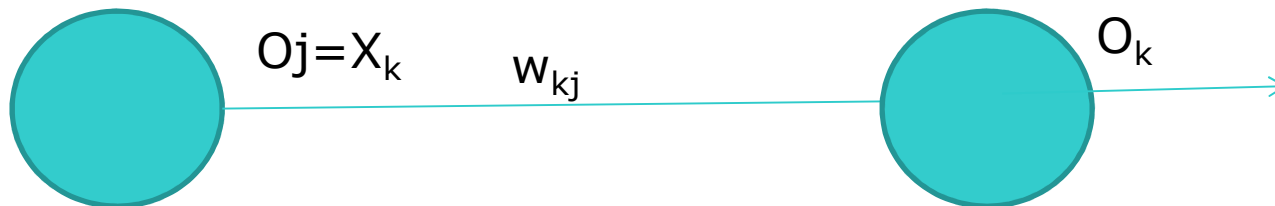
O aprendizado Hebbiano tem aplicação em diferentes modelos de rede.

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:

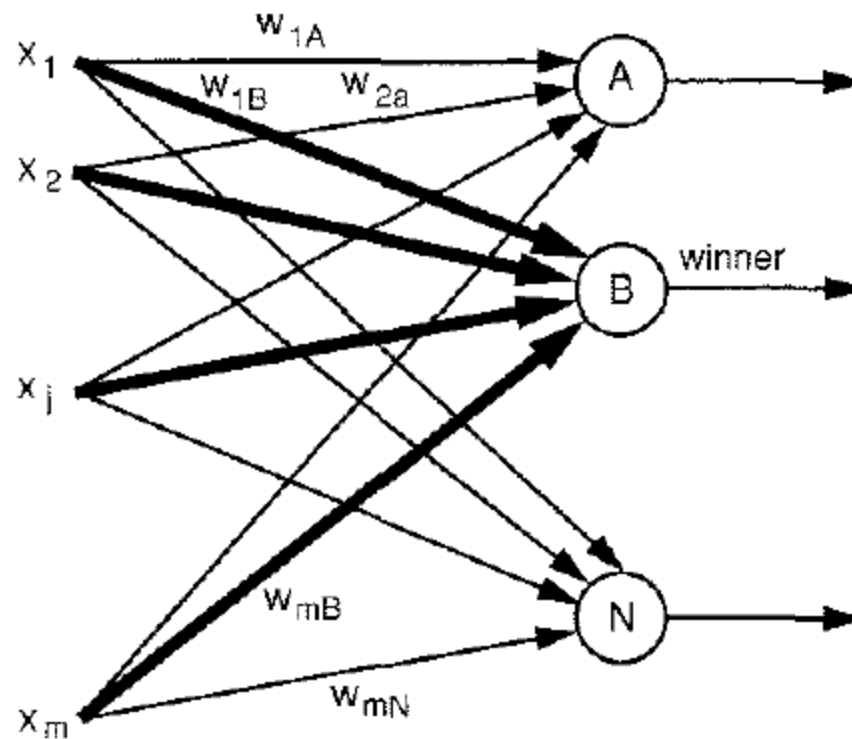


O_j	O_k	$\Delta w_{kj} = \alpha O_j O_k$
+	+	+
+	-	-
-	+	-
-	-	+

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

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

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

Produto escalar ou interno

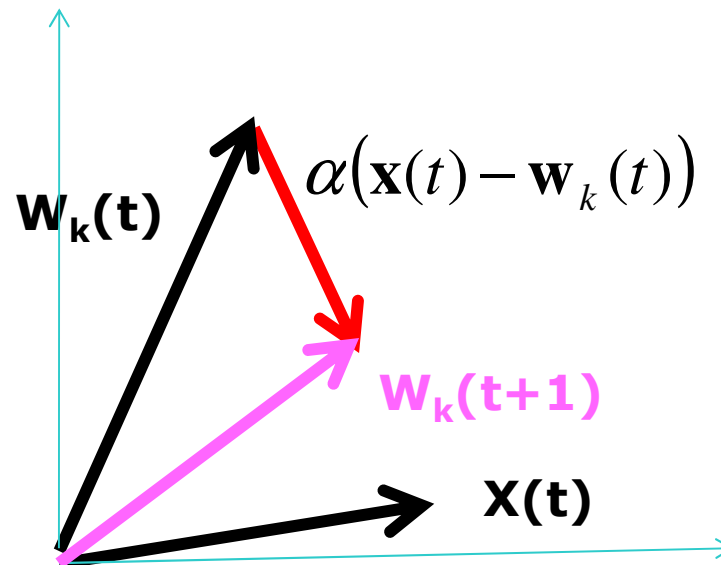
Norma Euclidiana (vale também para \mathbf{w}_k não normalizado)

Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado Por Competição:

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

K: winner





Treinamento por Competição e Mapas de Kohonen

Um mapa de Kohonen é um arranjo de neurônios, geralmente restrito a espaços de dimensão 1 ou 2, que procura estabelecer e preservar noções de vizinhança (preservação topológica).

Se estes mapas apresentarem propriedades de auto-organização, então eles podem ser aplicados a problemas de clusterização e ordenação espacial de dados.

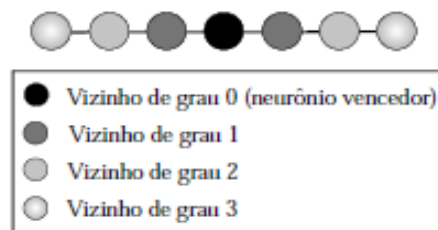
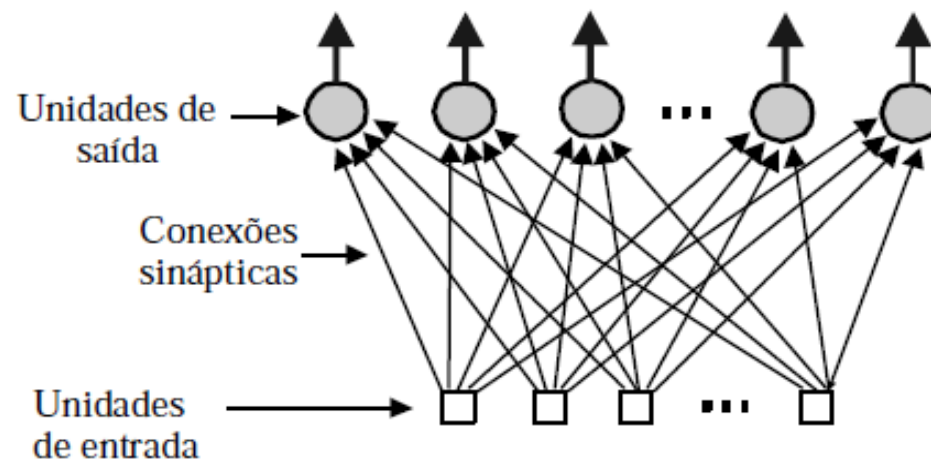
Neste caso podem ser treinados por aprendizado não supervisionado (por competição).

Este treinamento gera um conjunto de pesos que realiza um mapeamento do espaço original (em que os dados se encontram) para o espaço em que está definido o arranjo de neurônios

Mapas de Kohonen

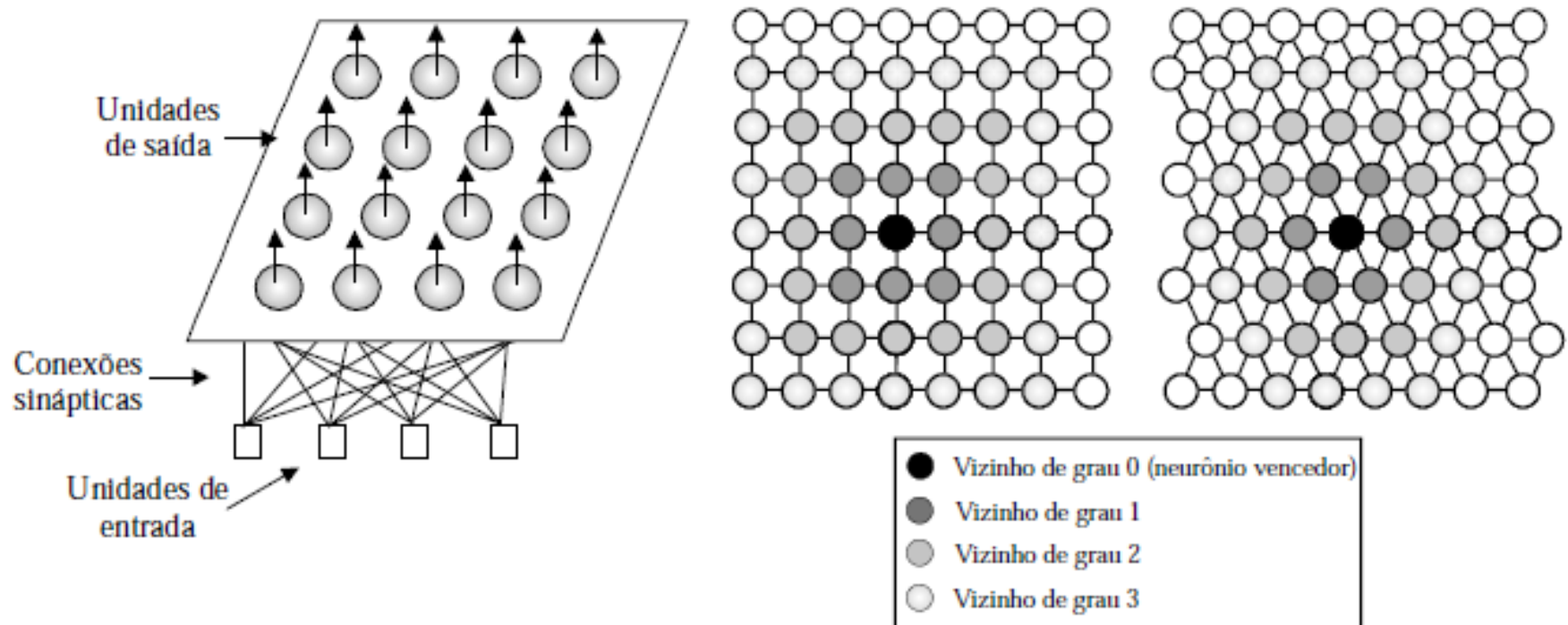
Arranjo unidimensional

- um mapa de Kohonen unidimensional é dado por uma seqüência ordenada de neurônios lineares, sendo que o número de pesos é igual ao número de entradas.



Mapas de Kohonen

Arranjo multidimensional

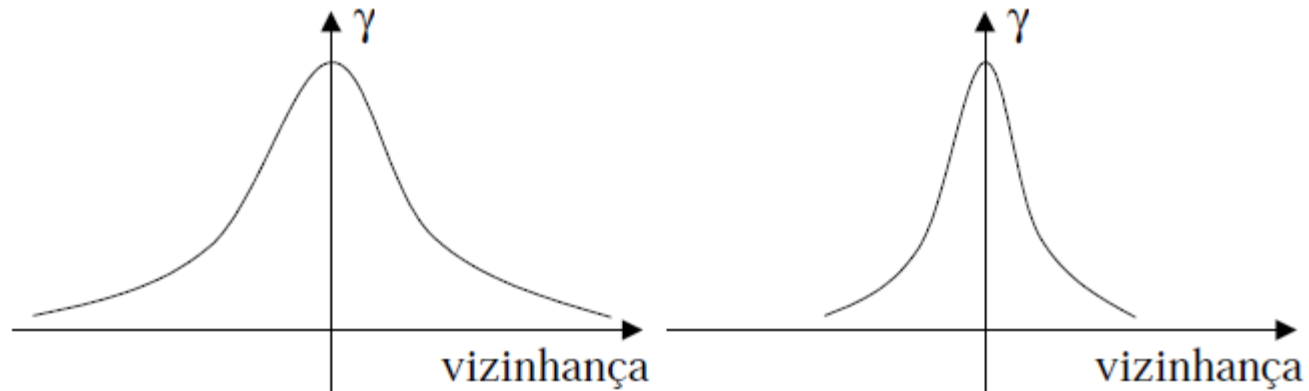


Redes Neurais Artificiais: Treinamento

Treinamento Não-supervisionado Por Competição:

- caso existam múltiplos representantes para cada agrupamento de dados, então é interessante ajustar o neurônio vencedor e seus vizinhos mais próximos.

Implementação



- é importante que a influência de cada neurônio vencedor seja ampla no início do processo e sofra uma redução continuada com o decorrer das iterações.

Mapas de Kohonen

Algoritmo de ajuste dos pesos

```
while <condição de parada> é falso,  
    ordene aleatoriamente os  $N$  padrões de entrada;  
    for  $i=1$  até  $N$ ,  
         $j = \arg \min_j \|\mathbf{x}_i - \mathbf{w}_j\|$   
         $\forall J \in \text{Viz}(j)$  do:  
             $\mathbf{w}_J = \mathbf{w}_J + \gamma(\text{dist}(j, J))(\mathbf{x}_i - \mathbf{w}_J);$   
        end do  
    end for  
    atualize a taxa de aprendizado  $\gamma$ ;  
    atualize a vizinhança;  
    avalie a condição de parada;
```

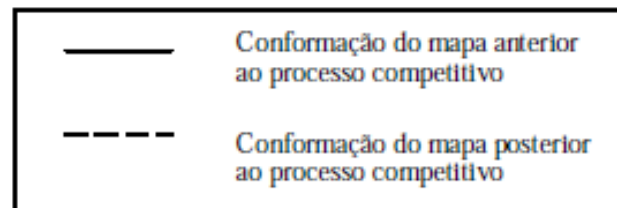
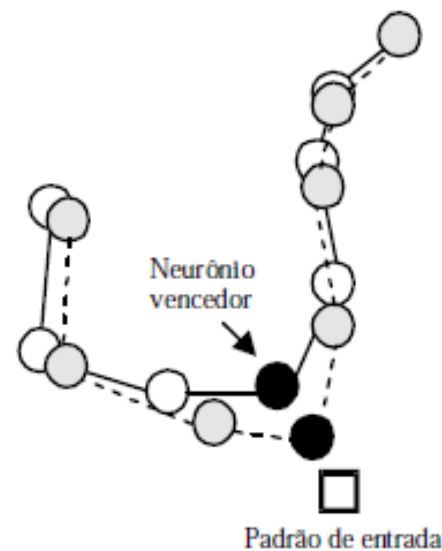
```
end while
```

$$\gamma(\text{dist}(j, J)) = \exp\left(-\frac{\text{dist}(j, J)^2}{2\sigma^2(t)}\right),$$

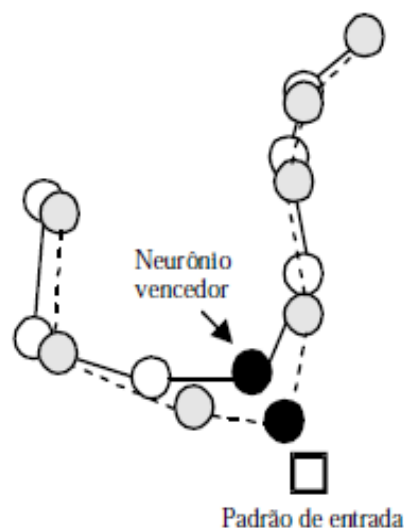
$$\text{onde } \sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_1}\right) \text{ e } \text{dist}(j, J)^2 = \|\mathbf{r}_j - \mathbf{r}_J\|^2$$

Mapas de Kohonen

Treinamento: Não supervisionado por competição



Mapas de Kohonen e Problema do Caixeiro Viajante



Problema do Caixeiro Viajante (PCV) é um problema De Otimização que consiste em determinar a menor rota para visitar um número de cidades percorrendo cada uma delas pelo menos uma vez.



Mapas de Kohonen e Problema do Caixeiro Viajante

O mapa de Kohonen se auto organiza, partindo de valores aleatórios iniciais em uma rede elástica.

As cidades representadas por coordenadas (x,y) puxam a rede elástica (um nó ou grupo por vez), através da atualização dos pesos dos neurônios, modificando seu formato a cada iteração.

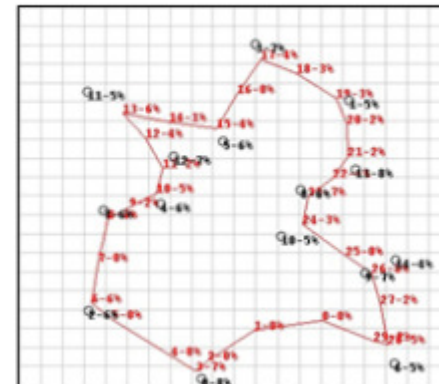
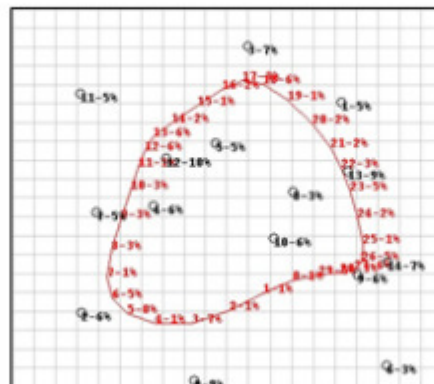
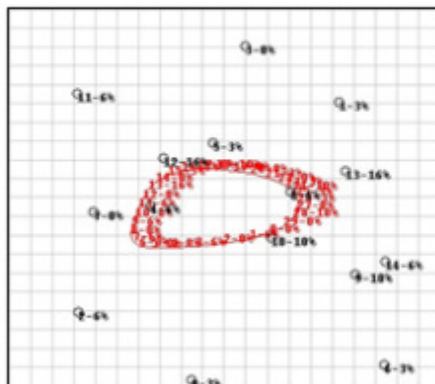
Após um ciclo completo de atualizações, o formato da rede elástica converge para um caminho passando por todas as cidades.

Mapas de Kohonen e Problema do Caixeiro Viajante

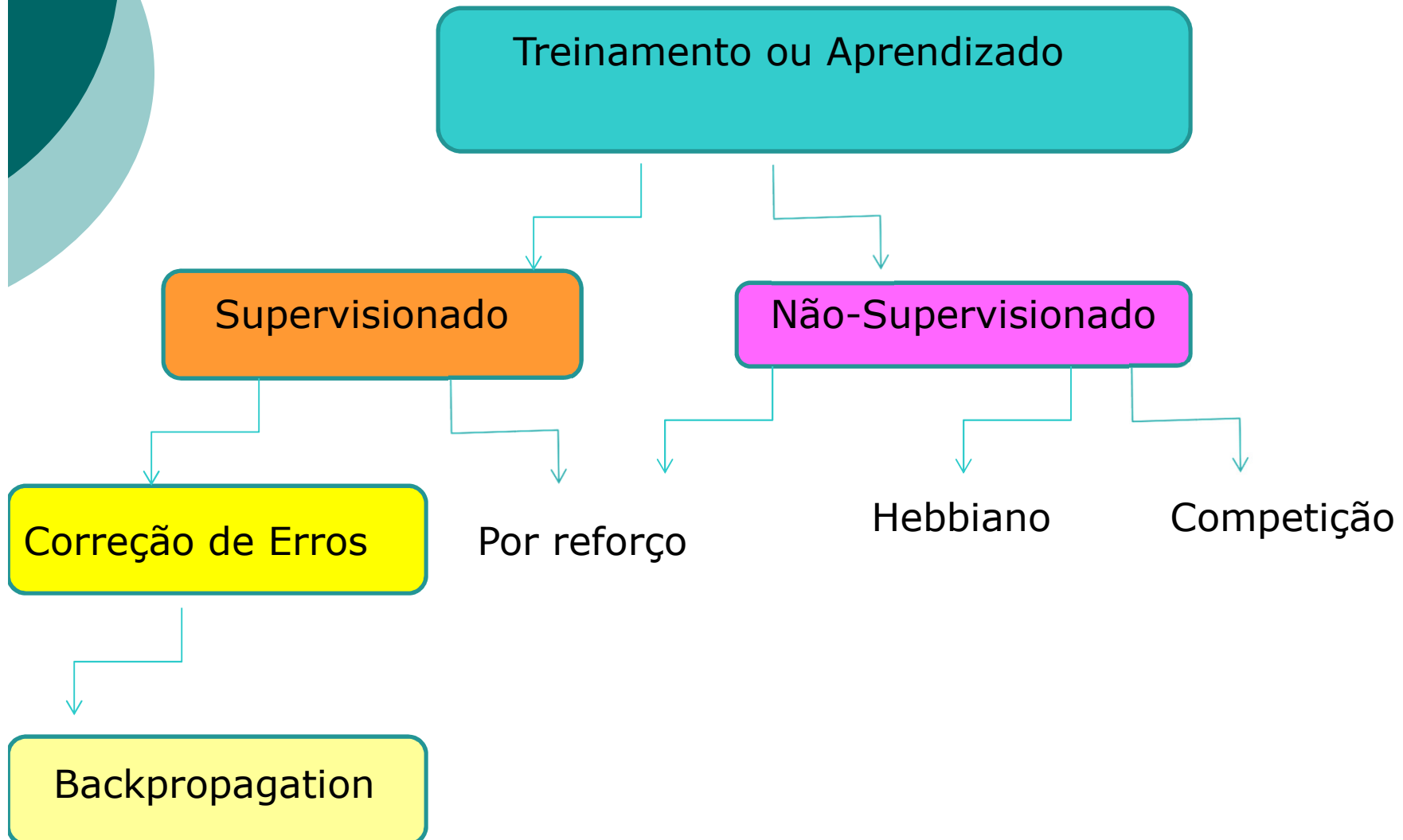
O mapa de Kohonen se auto organiza, partindo de valores aleatórios iniciais em uma rede elástica.

As cidades representadas por coordenadas (x,y) puxam a rede elástica (um nó ou grupo por vez), através da atualização dos pesos dos neurônios, modificando seu formato a cada iteração.

Após um ciclo completo de atualizações, o formato da rede elástica converge para um caminho passando por todas as cidades.



Redes Neurais Artificiais: Treinamento





Redes Neurais Artificiais: Trein Supervisionado

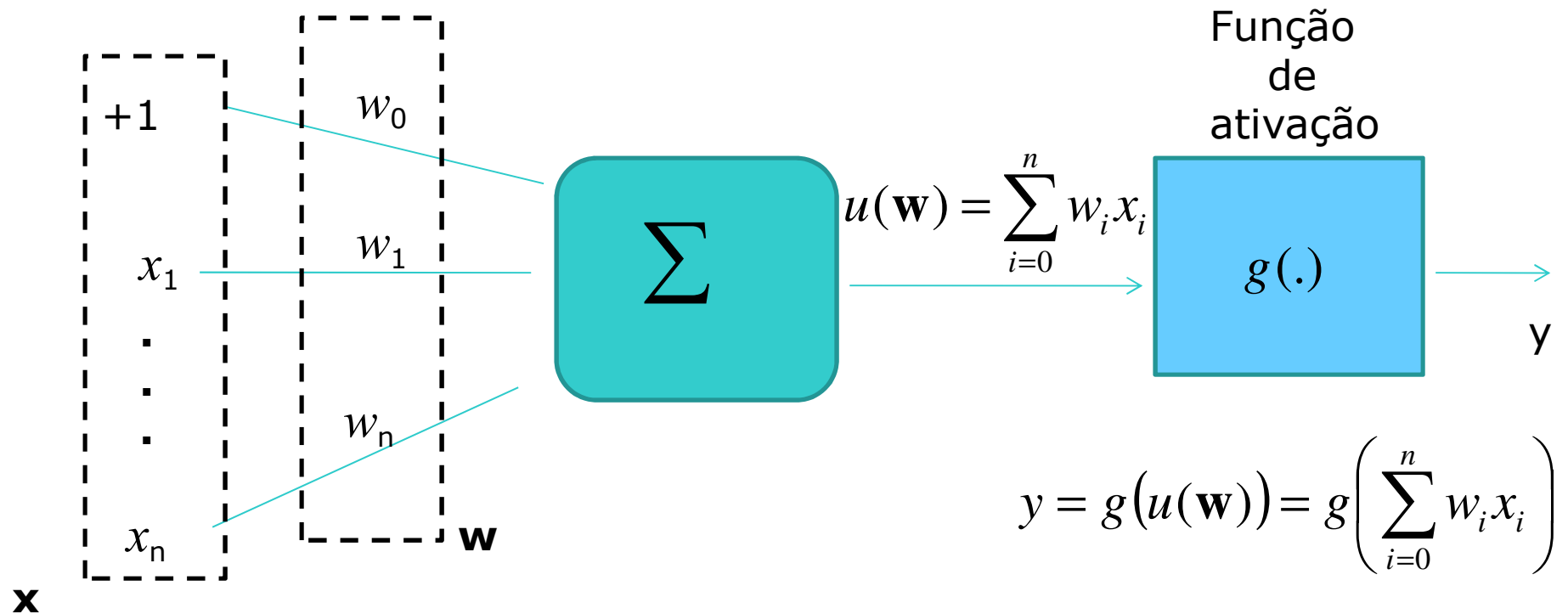
No processo de treinamento, normalmente são **ajustados os pesos** das conexões, os quais guardam (ao final do treinamento) **o conhecimento** que a rede adquiriu do ambiente no qual está operando.

Seja $w(t)$ o peso sináptico de um neurônio no instante t . A adaptação (ou ajuste) $\Delta w(t)$ é aplicada ao peso $w(t)$ no instante t , gerando o valor corrigido (ou adaptado) no instante $t+1$, na forma

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

Aprendizado: neurônio com função de ativação $g(\cdot)$

Ajustando os pesos de um neurônio com função de ativação (diferenciável) g





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 $(\mathbf{x}_p, y_{d_p})_{p=1, \dots, P}$, a função de erro quadrático

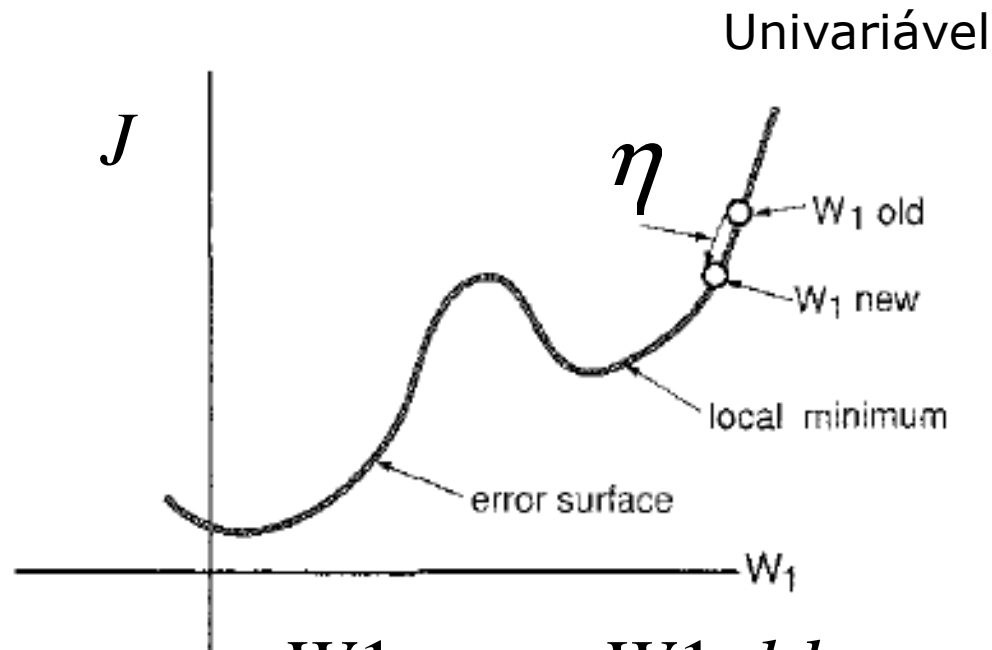
$$J(\mathbf{w}) = \sum_{p=1}^P \frac{1}{2} \left(y_p(\mathbf{w}) - y_{d_p} \right)^2 = \sum_{p=1}^P \frac{1}{2} \left(g(u_p(\mathbf{w})) - y_{d_p} \right)^2 = J_p(\mathbf{w})$$

seja minimizada.

Redes Neurais Artificiais: Treinamento

Problema de minimização do Erro

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



$$W_{1new} = W_{1old} - \eta \, dJ / dW_1$$

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



Redes Neurais Artificiais: Treinamento

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

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

Onde

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



Redes Neurais Artificiais: Treinamento

O ajuste do vetor de pesos

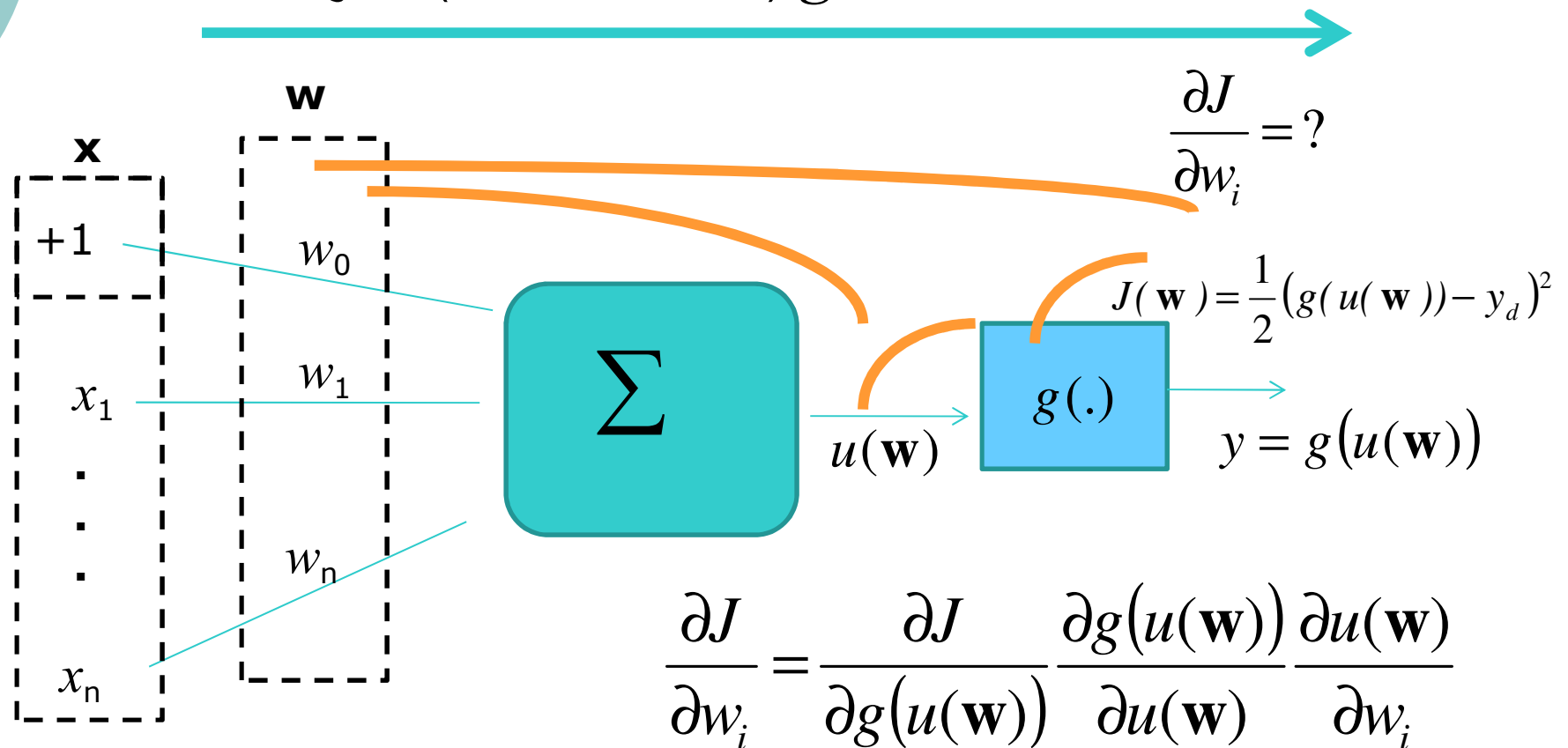
$$\mathbf{w}(t+1) = \mathbf{w}(t) - \eta \nabla J(\mathbf{w}(t))$$

Pode ser calculado para cada elemento w_i , $i=1\dots n$, do vetor \mathbf{w} como

$$w_i(t+1) = w_i(t) - \eta \frac{\partial J}{\partial w_i}$$

Aprendizado: neurônio com função de ativação $g(\cdot)$

Ajustando os pesos de um neurônio com função de ativação (diferenciável) g



Aprendizado: Neurônio com função de ativação $g(\cdot)$

O ajuste do vetor de pesos

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

Pode ser calculado para cada w_i , $i=1\dots n$, elemento do vetor \mathbf{w} como

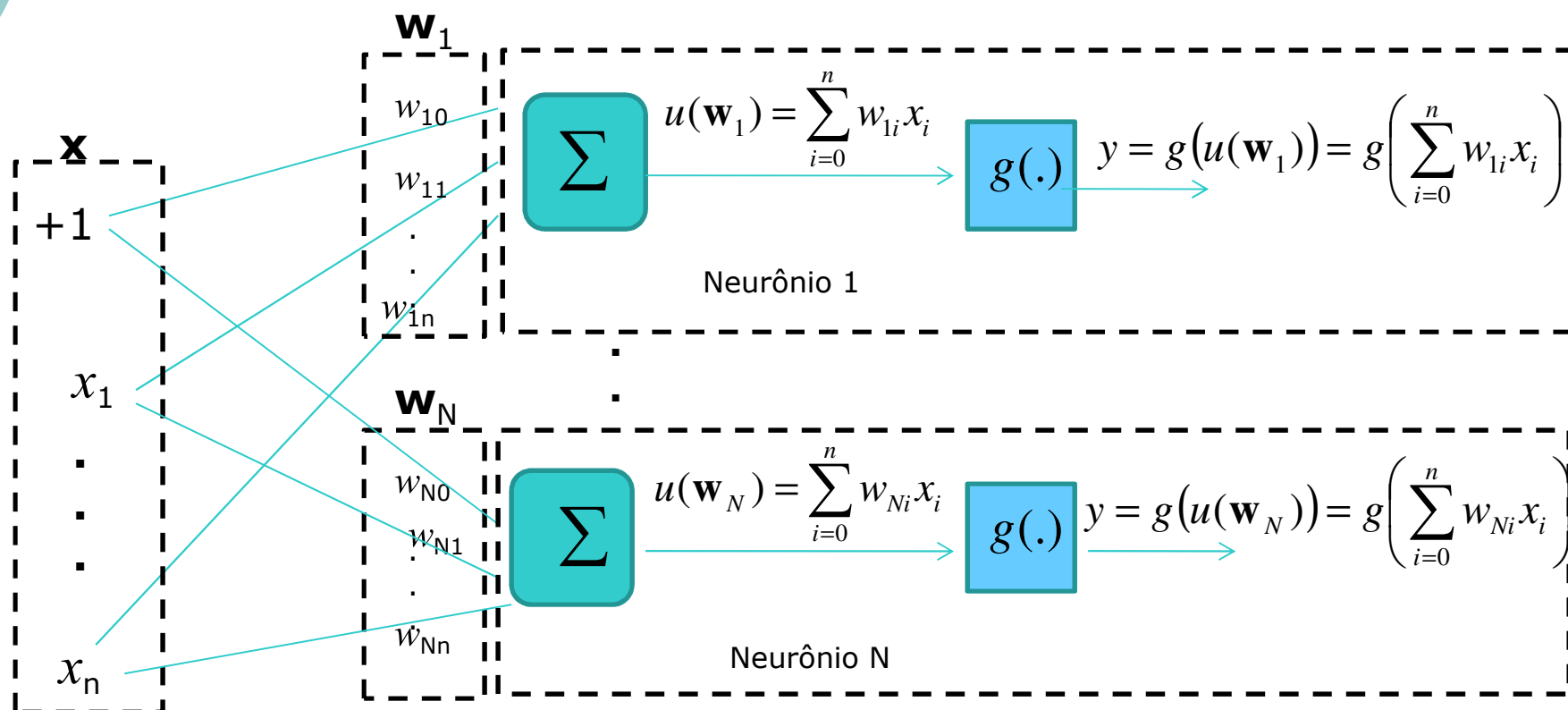
$$w_i(t+1) = w_i(t) - \eta \frac{\partial J}{\partial w_i} = w_i(t) - \eta \left(\frac{\partial J}{\partial g(u(\mathbf{w}))} \frac{\partial g(u(\mathbf{w}))}{\partial u(\mathbf{w})} \frac{\partial u(\mathbf{w})}{\partial w_i} \right)$$

$$w_i(t+1) = w_i(t) - \eta \left((g(u(\mathbf{w})) - y_d) g'(u(\mathbf{w})) x_i \right)$$

$$w_i(t+1) = w_i(t) - \eta (\text{erro } g'(u(\mathbf{w})) x_i)$$

Treinamento: rede de camada única

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



Exemplo de Algoritmo de treinamento (por batelada) de redes de **camada única** (função de ativação $g(\cdot)$)

Defina η e inicialize os vetores de pesos dos N neurônios da rede : $\mathbf{w}_k, k=1, \dots, N$

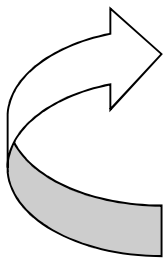
$t = 1$;

repita

para cada neurônio $k \quad k=1, \dots, N$ faça

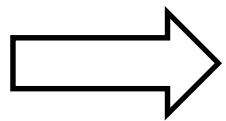
para cada peso $w_i \quad i=0, \dots, n$ do neurônio k faça

para cada par $(\mathbf{x}^p, y_d^p) \quad p=1, \dots, P$ faça



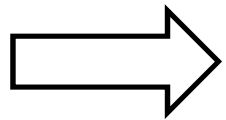
$$\frac{\partial J_p}{\partial w_{ki}} = \left(g(u_p(\mathbf{w}_k)) - y_{d_p} \right) \frac{\partial g(u_p(\mathbf{w}_k))}{\partial u_p(\mathbf{w}_k)} x_{pi} = e_{pk} g'(\mathbf{w}_k) x_{pi}$$

fim para



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

$$\frac{\partial J}{\partial w_{ki}} = \sum_{p=1}^P \frac{\partial J_p}{\partial w_{ki}}$$



Atualize o i -ésimo peso do neurônio k

$$w_{ki}(t+1) = w_{ki}(t) - \eta \frac{\partial J}{\partial w_{ki}}$$

fim para

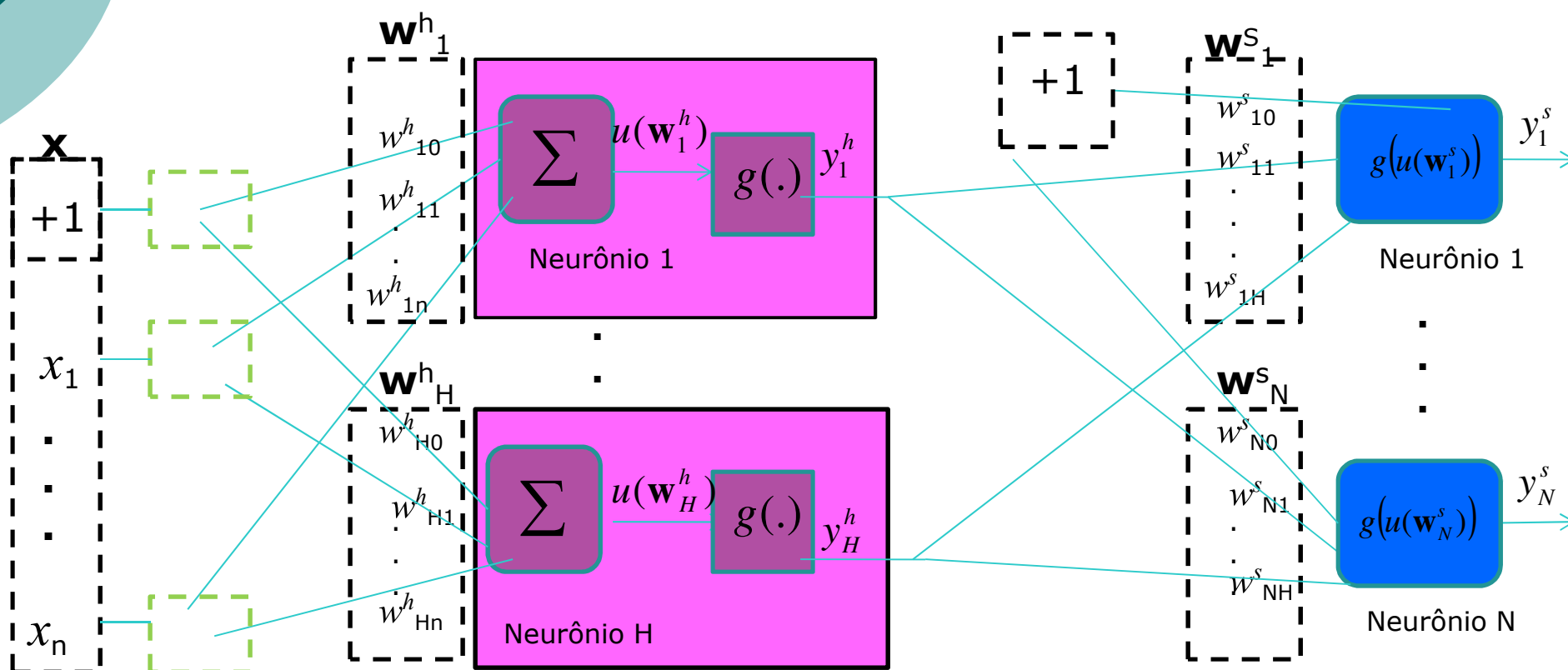
fim para

$t = t + 1$;

até atingir condição de **parada** (max_epocas ou aumento_erro_val ou $\text{erro_trein} < \xi$)

Treinamento: rede de múltiplas camadas

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





Treinamento BP: rede de múltiplas camadas

Ajustar os pesos de forma a minimizar a função

$$J(\mathbf{w}) = \frac{1}{2} \sum_{p=1}^P \sum_{k=1}^N \left(y^s_{pk}(\mathbf{w}) - y_{d_{pk}} \right)^2 = \frac{1}{2} \sum_{p=1}^P \sum_{k=1}^N \left(g(u_p(\mathbf{w})) - y_{d_{pk}} \right)^2 = \sum_{p=1}^P \sum_{k=1}^N J_{pk}(\mathbf{w})$$

Solução proposta: (mesma para a rede de 1 camada)

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

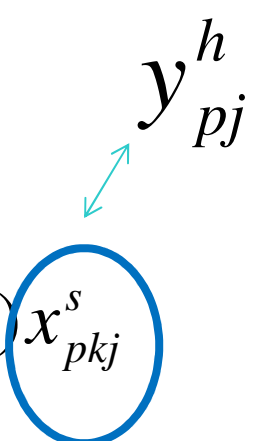
Treinamento BP: rede de múltiplas camadas

Camada de saída: (mesmo ajuste feito para 1 camada)

Calcular para cada peso w_{kj}^s , $k = 1, \dots, N$, $j = 1, \dots, H$

$$\frac{\partial J}{\partial w_{kj}^s} = \frac{\partial}{\partial w_{kj}^s} \left[\sum_{p=1}^P J_p \right] = \sum_{p=1}^P \frac{\partial J_p}{\partial w_{kj}^s} = \sum_{p=1}^P \frac{\partial J_p}{\partial g(u_p(\mathbf{w}_k^s))} \frac{\partial g(u_p(\mathbf{w}_k^s))}{\partial w_{kj}^s} =$$

$$\frac{\partial J}{\partial w_{kj}^s} = \sum_{p=1}^P \frac{\partial J_p}{\partial g(u_p(\mathbf{w}_k^s))} \frac{\partial g(u_p(\mathbf{w}_k^s))}{\partial u_p(\mathbf{w}_k^s)} \frac{\partial u_p(\mathbf{w}_k^s)}{\partial w_{kj}^s}$$

$$\frac{\partial J}{\partial w_{kj}^s} = \sum_{p=1}^P \left(g(u_p(\mathbf{w}_k^s)) - y_{d_{pk}} \right) g'(\mathbf{w}_k^s) x_{pkj}^s = \sum_{p=1}^P e_{pk} g'(\mathbf{w}_k^s) x_{pkj}^s$$


Treinamento: BP rede de múltiplas camadas

Camada intermediária: Calcular para cada peso

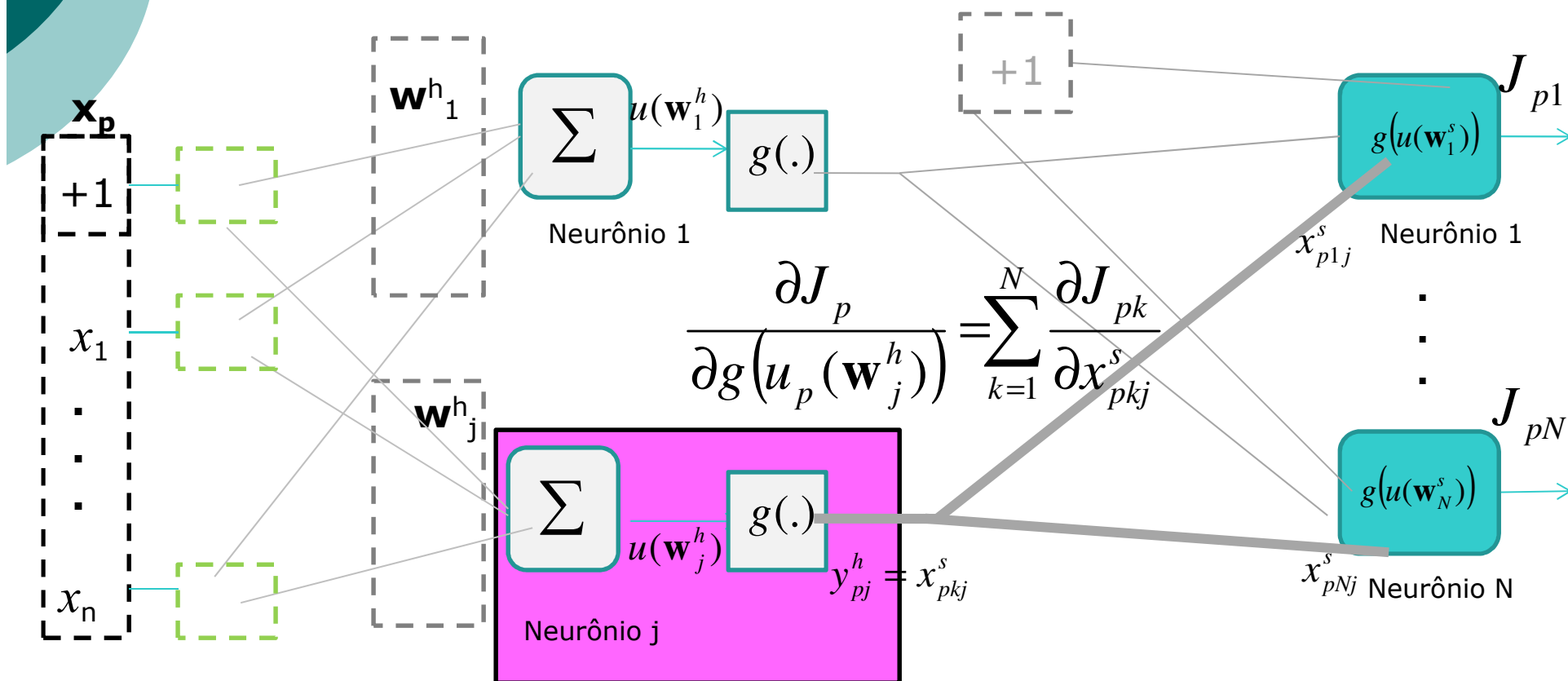
$$w_{ji}^h, \quad j = 1, \dots, H, \quad i = 1, \dots, n$$

$$\begin{aligned} \frac{\partial J}{\partial w_{ji}^h} &= \frac{\partial}{\partial w_{ji}^h} \left[\sum_{p=1}^P J_p \right] = \sum_{p=1}^P \frac{\partial J_p}{\partial w_{ji}^h} = \sum_{p=1}^P \frac{\partial J_p}{\partial g(u_p(\mathbf{w}_j^h))} \frac{\partial g(u_p(\mathbf{w}_j^h))}{\partial w_{ji}^h} = \\ &= \sum_{p=1}^P \frac{\partial J_p}{\partial g(u_p(\mathbf{w}_j^h))} \frac{\partial g(u_p(\mathbf{w}_j^h))}{\partial u_p(\mathbf{w}_j^h)} \frac{\partial u_p(\mathbf{w}_j^h)}{\partial w_{ji}^h} \end{aligned}$$

$$\frac{\partial J}{\partial w_{ji}^h} = \sum_{p=1}^P \frac{\partial J_p}{\partial g(u_p(\mathbf{w}_j^h))} g'(\mathbf{w}_j^h) x_{pji}^h$$

Treinamento BP: rede de múltiplas camadas

Calculo da derivada parcial em relação ao j-ésimo neurônio



Treinamento BP: rede de múltiplas camadas

Camada intermediária:

$$\frac{\partial J_p}{\partial g(u_p(\mathbf{w}_j^h))} = \sum_{k=1}^N \frac{\partial J_{pk}}{\partial x_{pkj}^s}$$

$$\frac{\partial J_{pk}}{\partial x_{pkj}^s} = \frac{\partial J_{pk}}{\partial g(u_p(\mathbf{w}_k^s))} \frac{\partial g(u_p(\mathbf{w}_k^s))}{\partial u_p(\mathbf{w}_k^s)} \frac{\partial u_p(\mathbf{w}_k^s)}{\partial x_{pkj}^s}$$

$$\frac{\partial J_p}{\partial g(u_p(\mathbf{w}_j^h))} = \sum_{k=1}^N \frac{\partial J_{pk}}{\partial x_{pkj}^s} = \sum_{k=1}^N (g(u_p(\mathbf{w}_k^s)) - y_{d_{pk}}) g'(\mathbf{w}_k^s) w_{kj}^s$$

Treinamento BP: rede de múltiplas camadas

Camada intermediária: w_{ji}^h , $j = 1, \dots, H$, $i = 1, \dots, n$

$$\frac{\partial J}{\partial w_{ji}^h} = \sum_{p=1}^P \frac{\partial J_p}{\partial g(u_p(\mathbf{w}_j^h))} \frac{\partial g(u_p(\mathbf{w}_j^h))}{\partial u_p(\mathbf{w}_j^h)} \frac{\partial u_p(\mathbf{w}_j^h)}{\partial w_{ji}^h} = \sum_{p=1}^P \frac{\partial J_p}{\partial g(u_p(\mathbf{w}_j^h))} g'(\mathbf{w}_j^h) x_{pji}^h$$

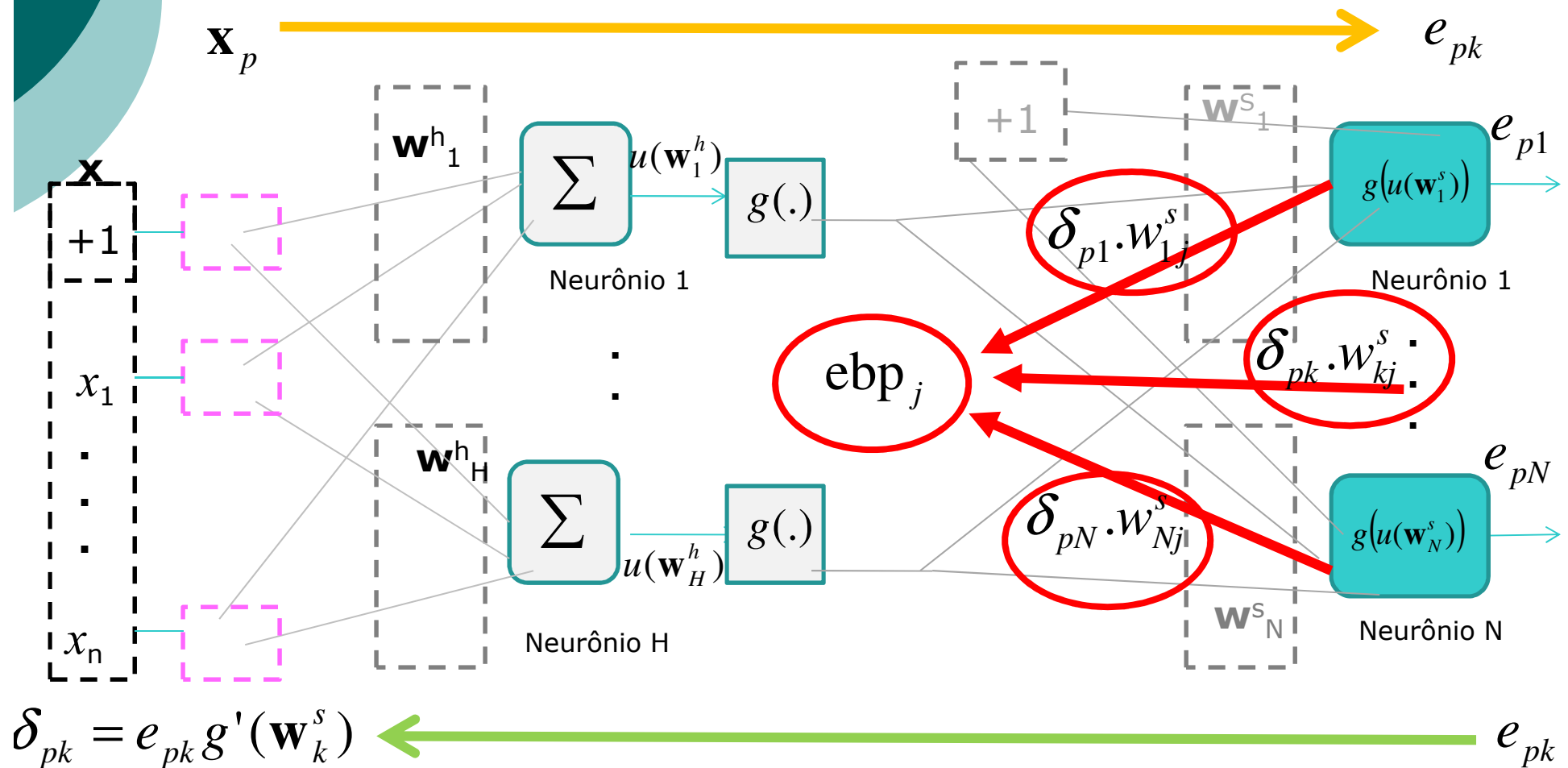
$$\frac{\partial J}{\partial w_{ji}^h} = \sum_{p=1}^P \left[\sum_{k=1}^N (g(u_p(\mathbf{w}_k^s)) - y_{d_{pk}}) \frac{\partial g}{\partial u_p(\mathbf{w}_k^s)} w_{kj}^s \right] g'(\mathbf{w}_j^h) x_{pji}^h$$

$$\frac{\partial J}{\partial w_{ji}^h} = \sum_{p=1}^P \left[\sum_{k=1}^N e_{pk} g'(\mathbf{w}_k^s) w_{kj}^s \right] g'(\mathbf{w}_j^h) x_{pji}^h$$

$$\frac{\partial J}{\partial w_{ji}^h} = \sum_{p=1}^P \left[\sum_{k=1}^N \delta_{pk} w_{kj}^s \right] g'(\mathbf{w}_j^h) x_{pji}^h = \sum_{p=1}^P \text{ebp}_{pj} g'(\mathbf{w}_j^h) x_{pji}^h$$

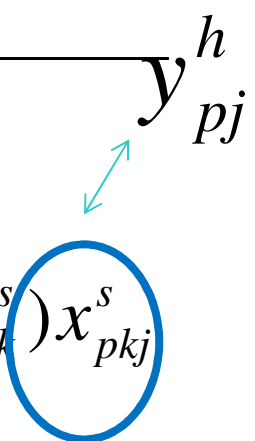
Treinamento BP: rede de múltiplas camadas

Duas fases: **propaga entrada** e **retropropaga o erro**

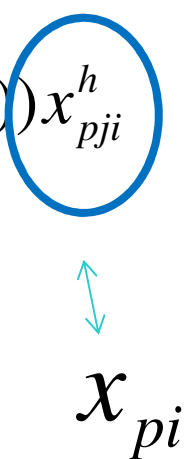


Treinamento BP: rede de duas camadas

Camada de saída:

$$\frac{\partial J}{\partial w_{kj}^s} = \sum_{p=1}^P \left(g(u_p(\mathbf{w}_k^s)) - y_{d_{pk}} \right) g'(\mathbf{w}_k^s) x_{pkj}^s = \sum_{p=1}^P e_{pk} g'(u(\mathbf{w}_k^s)) x_{pkj}^s$$


Camada intermediária:

$$\frac{\partial J}{\partial w_{ji}^h} = \sum_{p=1}^P \left[\sum_{k=1}^N \delta_{pk} w_{kj}^s \right] g'(u(\mathbf{w}_j^h)) x_{pji}^h = \sum_{p=1}^P \text{ebp}_{pj} g'(u(\mathbf{w}_j^h)) x_{pji}^h$$


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

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

$t = 1$;

Repita

para cada neurônio $k \quad k=1,\dots,N$ da **CAMADA DE SAÍDA** faça

para o j -ésimo peso $j=0,\dots,H$ do neurônio k faça

para cada par $(\mathbf{x}^p, y_{d_p}) \quad p=1,\dots,P$ faça

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

fim para

fim para

fim para

para cada neurônio $j \quad j=0,\dots,H$ da **CAMADA INTERMEDIÁRIA** faça

para o i -ésimo peso $i=0,\dots,n$ do neurônio j faça

para cada par $(\mathbf{x}^p, y_{d_p}) \quad p=1,\dots,P$ faça

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

fim para

fim para

fim para

Atualize o peso $j \quad (j=1,\dots,H)$ do neurônio $k \quad k=1,\dots,N$.

Atualize o peso $i \quad (i=1,\dots,n)$ do neurônio $j \quad j=1,\dots,H$

$t=t+1$;

até atingir condição de parada

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

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



Algoritmo Backpropagation:

Dicas práticas:

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

Normalizar as entradas:

$[0.1, 0.9]$ sigmoide

$[-0.9, 0.9]$ tangente hiperbólica

Utilizar valores pequenos para taxa de aprendizado

η in $[0.01 \text{ a } 0.1]$



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 mínimos 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 $[0.8, 0.9]$