Inteligência Artificial

IA conexionista

Conceitos básicos de Redes Neurais Artificiais

Prof. Paulo Martins Engel

Ciência da Computação

- No âmbito da Ciência da Computação, as Redes Neurais são estudadas na grande área de Inteligência Artificial (IA).
- Historicamente, a *IA clássica*, seguiu o paradigma da computação simbólica.
- As redes neurais deram origem à chamada *IA conexionista* e se encontram também dentro da grande área chamada de *Inteligência Computacional* (IC).

Abordagens Não-Simbólicas

- Técnicas de IC, como as *Redes Neurais Artificiais* (RN), são uma abordagem *bottom-up*.
- A semântica do domínio não precisa ser introduzida explicitamente no sistema.
- O sistema pode *induzir* este conhecimento, através de um processo de *aprendizagem*.
- Por outro lado, com as técnicas atuais, é muito ineficiente aprender adequadamente em ambientes complexos.
- O conhecimento aprendido não se torna facilmente interpretável pelo usuário.

Aplicações das Redes Neurais

- As habilidades que as redes neurais apresentam permitem o seu emprego numa grande diversidade de áreas de aplicação.
- As RN são aplicadas em várias áreas que envolvem *reconhecimento de padrões* e o aprendizado de funções não-lineares.
- Exemplos dessas áreas são o processamento de imagens, a visão robótica, o reconhecimento de voz, controle de sistemas dinâmicos, categorização de textos, mineração de dados, etc.

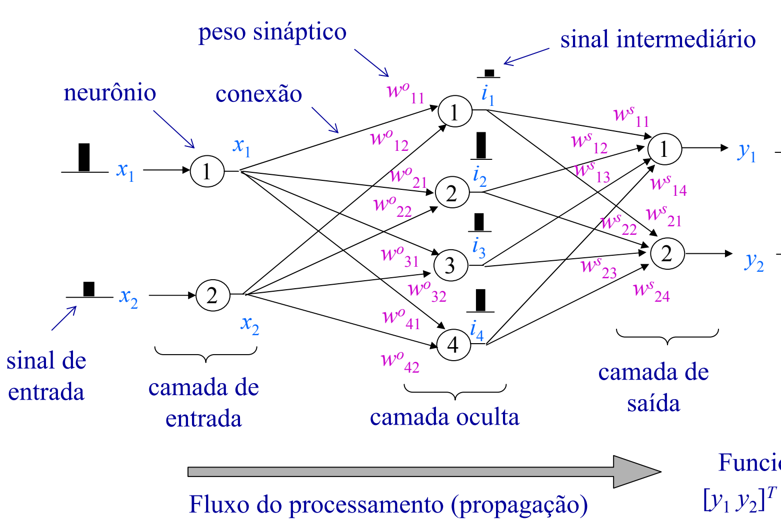
Conceitos básicos de Redes Neurais

- As Redes Neurais Artificiais (RN) são sistemas físicos celulares que podem adquirir, armazenar e utilizar conhecimento extraído da experiência, por meio de *algoritmos de aprendizagem*.
- O conhecimento está na forma de estados estáveis ou mapeamentos incorporados numa rede de processadores simples, os neurônios artificiais, interligados por elos que possuem parâmetros ajustáveis, os *pesos sinápticos*, que controlam a intensidade das conexões.
- A computação por RN é realizada por uma densa malha de nós processadores e conexões. O elemento processador básico é o *neurônio artificial*.

Conceitos

- **Topologia da rede:** grafo direcionado rotulado, determina a conectividade da rede.
- **Neurônio:** função elementar (nó). Calcula um valor de saída, função dos valores de entrada do nó e dos pesos sinápticos correspondentes.
- **Conexão:** determina um caminho para o fluxo de informação entre dois nós (elo).
- **Pesos sináptico:** determina a força de conexão entre dois nós da rede (rótulo) .
- **Funcionalidade da rede:** composição das funções elementares determinada pela conectividade da rede.

Exemplo de uma rede multicamadas



Funcionalidade: $[y_1\ y_2]^T = \mathbf{F}(x_1, x_2)$

Mais conceitos

- A topologia da rede determina uma funcionalidade genérica adequada para uma certa classe de aplicações.
- A funcionalidade específica (mapeamento) de uma RN depende dos valores dos seus pesos sinápticos.
- Os valores dos pesos sinápticos são ajustados ao problema específico através de procedimentos iterativos de treinamento (processo de aprendizagem indutiva).
- Para o processo de aprendizagem é necessário se dispor de um conjunto de exemplos do mapeamento desejado (aprendizagem supervisionada).
- Alternativamente, uma RN pode ser treinada para extrair o modelo de um conjunto de dados de forma autônoma (aprendizagem não-supervisionada).

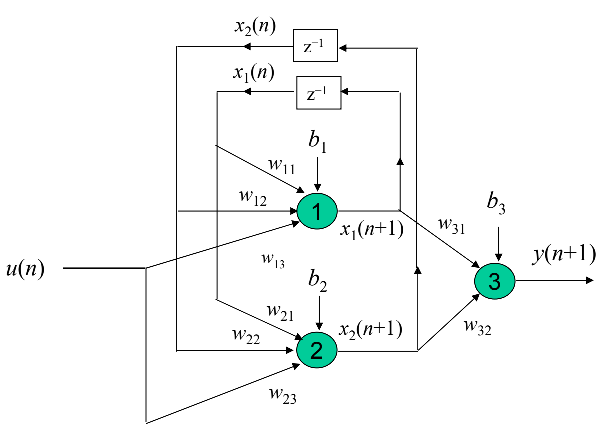
Computação realizada por uma RN

- O mapeamento que uma rede neural implementa pode envolver valores binários de saída, ou então valores contínuos.
- No primeiro caso, normalmente o conjunto de saídas binárias da rede é interpretado como rótulo de classe e a RN desempenha o papel de um *classificador*.
- No segundo caso, a rede atua como um *regressor*, fornecendo estimativas de valores do mapeamento aprendido (os valores de saída correspondentes a valores apresentados na entrada).
- Normalmente, os valores de entrada para a rede devem pertencer a um intervalo numérico adequado previamente definido, o que usualmente exige algum tipo de pré-processamento dos valores do domínio da aplicação.

Computação realizada

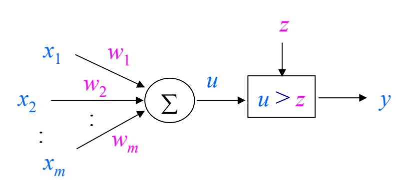
- Uma rede conectada apenas para frente (*feedforward*), sem realimentações, implementa mapeamentos estáticos.
- Exemplos de mapeamentos estáticos são máquinas combinacionais (sistemas discretos), modelos de sistemas estáticos (contínuos), além dos classificadores e regressores.
- Uma rede com conexões realimentadoras (*feedback*), ou *recorrente*, cria dependências temporais entre os sinais.
- Uma rede recorrente implementa mapeamentos dinâmicos, como máquinas seqüenciais, modelos de sistemas dinâmicos e regressores dinâmicos.

Exemplo de uma rede recorrente



Modelo de neurônio artificial

- A funcionalidade básica de um *neurônio artificial* é a de uma porta lógica genérica da *lógica de limiar* (LTU, *Logic Threshold Unit*).
- No modelo de neurônio artificial o valor de saída depende do valor (u) da *soma ponderada* entre as entradas do neurônio e os seus pesos (sinápticos).
- No modelo binário, o valor de saída corresponde ao resultado da comparação de u com um *limiar* z.
- A operação de comparação é modelada por uma *função de ativação*, $\varphi(u)$.
- Em geral, cada neurônio artificial tem uma entrada extra utilizada para ajustar o seu limiar. O valor *default* do limiar de um neurônio é zero.

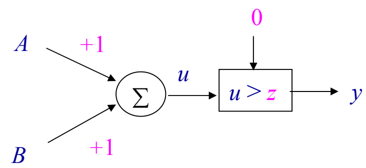


$$u = \sum_{j=1}^{m} w_j x_j$$

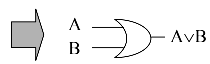
$$y = \begin{cases} 1 & \text{se } u > z \\ 0 & \text{se } u \leq z \end{cases}$$

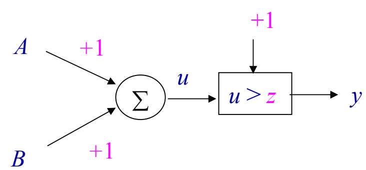
Implementação de funções booleanas por perceptrons

- Um neurônio artificial é uma porta lógica universal que pode implementar as diversas funções booleanas elementares, que formam a base da lógica booleana, apenas alterando o seu conjunto de pesos e limiar:

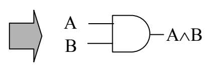


A	B	u	y
0	0	0	0
0	+1	+1	+1
+1	0	+1	+1
+1	+1	+2	+1





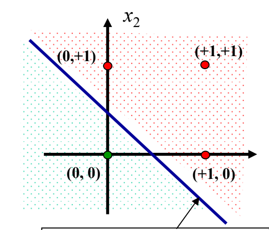
A	B	u	y
0	0	0	0
0	+1	+1	0
+1	0	+1	0
+1	+1	+2	+1



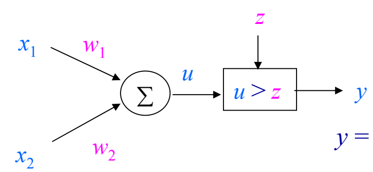
A superfície de decisão do perceptron

- A funcionalidade do perceptron equivale a uma tomada de decisão segundo uma superfície de decisão linear no espaço de entrada.
- Isto se deve ao fato de que a soma ponderada na entrada do perceptron representa uma reta, no espaço bidimensional, um plano no espaço 3-D, e um hiperplano em espaços de alta dimensionalidade.

	x_1	x_2	OU
$\mathbf{x}(1)$	0	0	0
$\mathbf{x}(2)$	0	+1	+1
$\mathbf{x}(3)$	+1	0	+1
$\mathbf{x}(4)$	+1	+1	+1



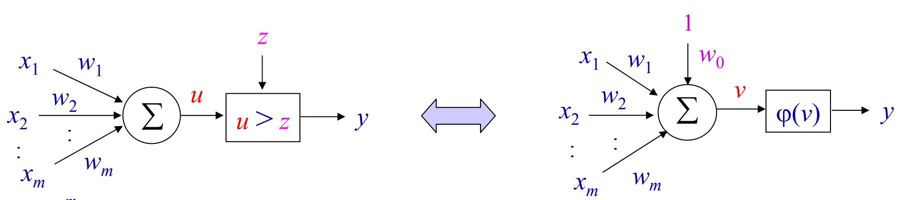
$w_1x_1 + w_2x_2 - z = 0$



$$y = \begin{cases} 1 & \text{se } u > z \\ 0 & \text{se } u \le z \end{cases}$$

Modelo de neurônio artificial com bias

- O limiar da função de ativação pode ser substituído por um peso extra, que recebe o nome de *bias*, mantendo a mesma funcionalidade do modelo.
- A vantagem é que neste modelo, só existem pesos para serem ajustados.



$$u = \sum_{j=1}^{m} w_j x_j \qquad y = \begin{cases} 1 & \text{se } u > z \\ 0 & \text{se } u \le z \end{cases}$$

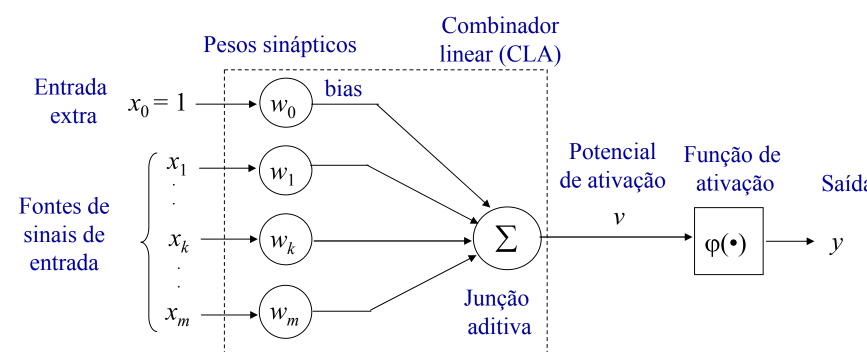
$$v = \sum_{j=0}^{m} w_j x_v \qquad y = \begin{cases} 1 & \text{se } v > 0 \\ 0 & \text{se } v \le 0 \end{cases}$$

Função de ativação: $u > z$

Função de ativação: $v > 0$
com $v = u + w_0$

$$u > z \;\Rightarrow\; u - z > 0 \;\Rightarrow\; u + w_0 > 0 \text{ com } w_0 = -z$$

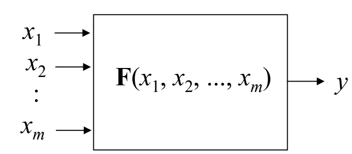
Modelo estático de um neurônio



O neurônio pode ser descrito matematicamente pelas seguintes equações:

Potencial de ativação $\longrightarrow$ $v = \sum_{k=0}^{m} w_k \cdot x_k$

Sinal de saída do neurônio $\longrightarrow$ $y = \varphi(v)$



Classificação de padrões por um perceptron

- O psicólogo Frank Rosenblatt [1958] propôs o *perceptron* como um modelo para realizar tarefas de classificação.
- A *classificação* é uma tarefa de previsão, que consiste em mapear (classificar) um determinado item, representado por um vetor de características (de entrada), para uma entre várias classes pré-definidas.
- Inicialmente, é necessário construir o *classificador* que implementa a função de mapeamento dos vetores de entrada para os rótulos numéricos de saída.
- No caso de um problema de classificação linear, o classificador pode ser implementado por um perceptron elementar.
- Neste caso, o mapeamento desejado se dá por aprendizado a partir de um conjunto de treinamento, com itens previamente classificados.
- Para problemas de classificação não linear o modelo de rede neural mais indicado é o *perceptron de múltiplas camadas* (MLP), que implementa um classificador universal.
- Quando o valor de saída do perceptron é contínuo, a sua funcionalidade é de um aproximador universal de funções.

Algoritmos de treinamento

- Uma das formas mais eficientes de se ajustar os pesos de uma rede neural é através do algoritmo do *mínimo quadrado médio*, ou *least mean square* (*LMS*).
- O algoritmo LMS é um processo iterativo no qual, a cada instante discreto de tempo n, um dos vetores de treinamento, representado por $\mathbf{x}(n)$, é propagado através do neurônio gerando a saída correspondente $y(n)$.
- O *erro quadrado instantâneo* na saída é então calculado, correspondente à diferença entre o valor desejado d e a saída obtida $y(n)$, representado por:

$$e^2(n) = \left(\left(d(n) - y(n)\right)\right)^2$$

- O vetor de pesos é então ajustado de modo a diminuir $e^2(n)$:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta\, e(n).\mathbf{x}(n)$$

- O processo é repetido para todos os vetores de treinamento.
- A apresentação do conjunto completo de treinamento é denominada de *época*.
- O processo de apresentação de vetores de treinamento e ajuste de pesos é repetido por várias épocas até que o erro médio quadrado se torne suficientemente pequeno ou então que ele não se modifique mais.

O método da decida mais íngreme

- O algoritmo LMS se baseia na minimização de uma função de custo $E(\mathbf{w})$, proporcional ao erro quadrático instantâneo:

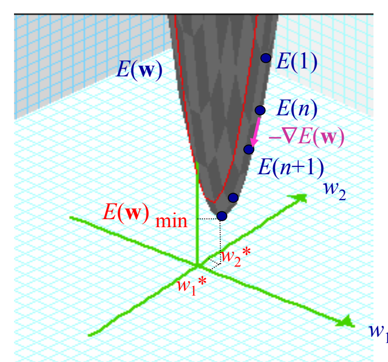
$$E(\mathbf{w}) = \frac{1}{2}\, e^2(n)$$

- $E(\mathbf{w})$ representa uma superfície de erro no espaço dos pesos.
- Os pesos são ajustados de modo que o erro quadrático instantâneo $e^2(n)$ diminua. Isto é feito ajustando-se os pesos na direção oposta ao *vetor gradiente* $\nabla E(\mathbf{w})$.
- O gradiente $\nabla E(\mathbf{w})$ é um vetor composto pelas derivadas parciais da função de erro (quadrático) em relação a cada peso. Para um determinado valor de $\mathbf{w}$, $\nabla E(\mathbf{w})$ aponta na direção de máximo crescimento do erro.

$$\nabla E(\mathbf{w}) \equiv \frac{\partial E(n)}{\partial \mathbf{w}(n)}$$

- Ajustando-se os pesos na direção de $-\nabla E(\mathbf{w})$ nós estaremos nos deslocando na direção da *descida mais íngreme* da superfície de erro.

- O método da descida mais íngreme é um processo iterativo que, a partir de um ponto inicial sobre a superfície de erro, procura o mínimo global, modificando os pesos a cada iteração na direção do gradiente descendente, $-\nabla E(\mathbf{w})$.



Com base neste método, o ajuste de pesos é feito pela expressão:

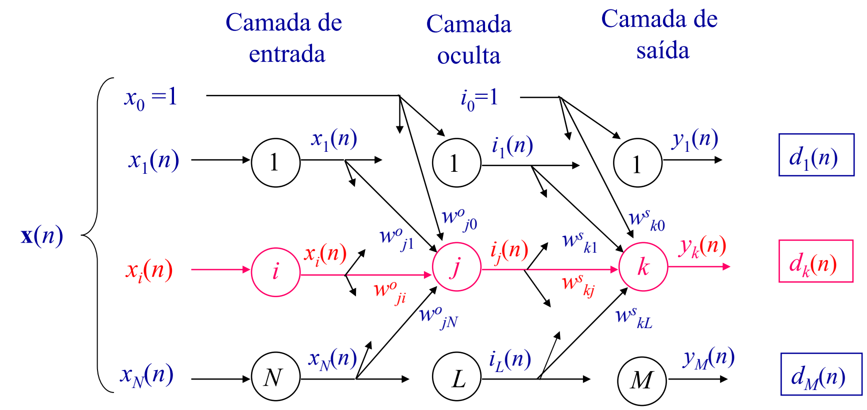
$$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta \nabla E(\mathbf{w})$$

onde η é o parâmetro da taxa de aprendizagem, que determina o passo da descida.

O perceptron de múltiplas camadas (MLP)

- A solução para problemas não-linearmente separáveis está baseada na rede MLP (Multilayer Perceptron), ou *perceptron de múltiplas camadas*, que apresenta uma organização topológica em pelo menos 3 camadas:
  - **Camada de entrada:** composta de neurônios sensoriais; distribui o vetor de entrada para todos os neurônios da camada oculta;
  - **Camada oculta:** composta de neurônios computacionais; realiza um mapeamento intermediário do problema, gerando vetores linearmente separáveis para a camada de saída;
  - **Camada de saída:** composta de neurônios computacionais; realiza rotulação das classes ou o mapeamento desejado.
- Alternativamente, o mapeamento intermediário do problema pode ser realizado por sucessivas camadas ocultas.
- Pode-se provar que qualquer problema pode ser solucionado por um MLP de 3 camadas, com um número suficiente de neurônios na camada oculta.

A topologia do Perceptron de Múltiplas Camadas MLP



$i_j(n)$: valor de saída do neurônio genérico (j) da camada oculta gerado por $\mathbf{x}(n)$.
$y_k(n)$: valor de saída do neurônio genérico (k) da camada de saída gerado por $\mathbf{x}(n)$.
$d_k(n)$: valor de saída desejado do neurônio k correspondente a $\mathbf{x}(n)$.
w^o_{ji} e w^s_{kj} : pesos genéricos da camada oculta e de saída, respectivamente.

O processamento de informação na rede MLP

O processamento de informação em MLPs acontece em duas fases:

- a fase de *propagação*, onde o sinal de entrada é propagado através de toda a rede, camada por camada. Esta fase é responsável pela atuação da rede e, portanto, ocorre *on-line*.
- a fase de *adaptação*, onde ocorrem os ajustes dos pesos da rede. Nesta fase, o fluxo de informação se dá da camada de saída em direção à camada de entrada. As diferenças entre os valores de saída da rede e os valores desejados causam parcelas individuais de erro para cada neurônio, que são usadas para corrigir os pesos, segundo o algoritmo *backpropagation*. Esta fase é utilizada apenas durante o treinamento da rede, que é realizado *off-line*, ou seja, sem que a rede atue no ambiente.

O algoritmo *Backpropagation*

Os MLP são treinados pelo algoritmo de *retropropagação* de erros, que é baseado na regra delta de aprendizado por correção de erro [Paul Werbos 74].
O algoritmo *Backprop* pode ser visto como uma generalização do algoritmo LMS (Least Mean Square) desenvolvido para um único neurônio.
Como existem vários neurônios na camada de saída, deve-se definir a soma instantânea dos quadrados dos erros em cada nó de saída da rede, $E(n)$, quando o n-ésimo vetor de treinamento $\mathbf{x}(n)$ é apresentado na entrada da rede:

$$E(n) = \frac{1}{2}\sum_{k=1}^{M} e^2_k(n)$$

Com o erro quadrado instantâneo na unidade k de saída definido por:

$$e^2_k(n) = \left(d_k(n) - y_k(n)\right)^2$$

e_k é o erro numa unidade de saída k, quando o vetor $\mathbf{x}(n)$ é propagado pela rede:

$$e_k(n) = d_k(n) - y_k(n)$$

$d_k(n)$ é a saída desejada, correspondente a $\mathbf{x}(n)$, e $y_k(n)$ é a saída instantânea obtida no neurônio de saída k, pela propagação de $\mathbf{x}(n)$.

Resumo do treinamento BP

1. Inicializar os pesos com valores arbitrários não nulos.
2. Apresentar um padrão de entrada $\mathbf{x}(n)$ e propagá-lo até a saída da rede.
3. Calcular os erros instantâneos na saída da rede, $e_k(n)$.
4. Calcular os gradientes locais dos neurônios da camada de saída, $\delta^s_k(n)$.
5. Ajustar os pesos da camada de saída pela expressão:

$$w^s_{kj}(n+1) = w^s_{kj}(n) + \eta\, \delta^s_k(n) . i_j(n)$$

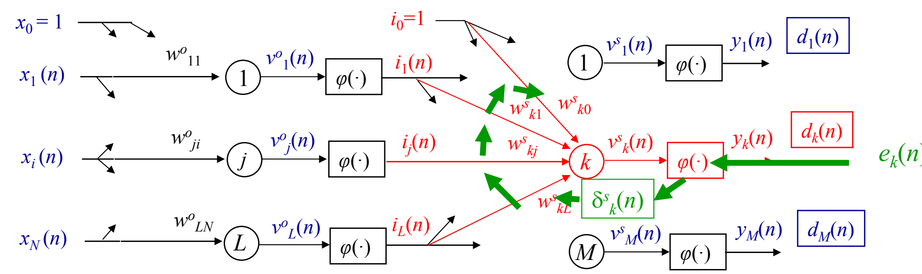
6. Calcular os gradientes locais dos neurônios da camada oculta, $\delta^o_j(n)$.
7. Ajustar os pesos da camada oculta pela expressão:

$$w^o_{ji}(n+1) = w^o_{ji}(n) + \eta\, \delta^o_j(n) . x_i(n)$$

8. Repetir os passos de 2 a 7 para todos os padrões de treinamento (1 época).
9. Calcular o erro médio quadrado (EMQ) para o arquivo de treinamento.
10. Se o EMQ for maior que o valor desejado, repetir o passo 8.

Gradiente do erro em relação a um peso de saída

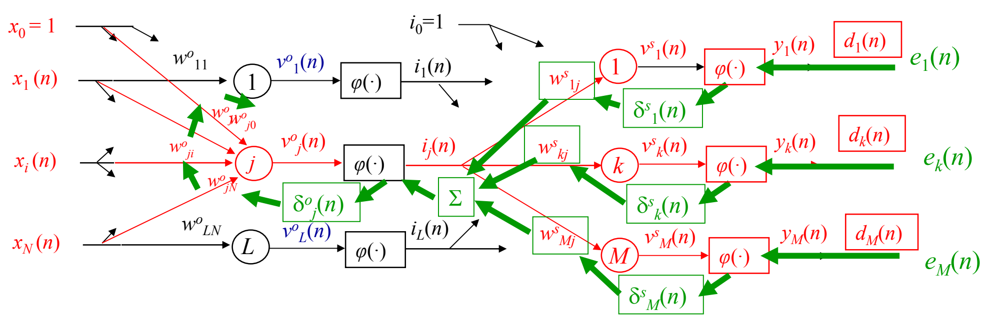
$$\delta^s_k(n) \equiv e_k(n)\, f_s'(v^s_k(n))$$



Ajuste de um vetor de pesos na camada oculta

$$\delta^o_j(n) \equiv f_o'(v^o_j(n))\, (\mathbf{w}^s_k(n)^T \cdot \delta^s(n))$$

$$\mathbf{w}^o_j(n+1) = \mathbf{w}^o_j(n) + \eta\, \delta^o_j(n) . \mathbf{x}(n)$$



Soluções do problema do jogo de tênis por BP