Inteligência Artificial

Aula 28- Redes Neurais: Simulação MLP[1]

Sílvia M.W. Moraes

Faculdade de Informática - PUCRS

June 13, 2017

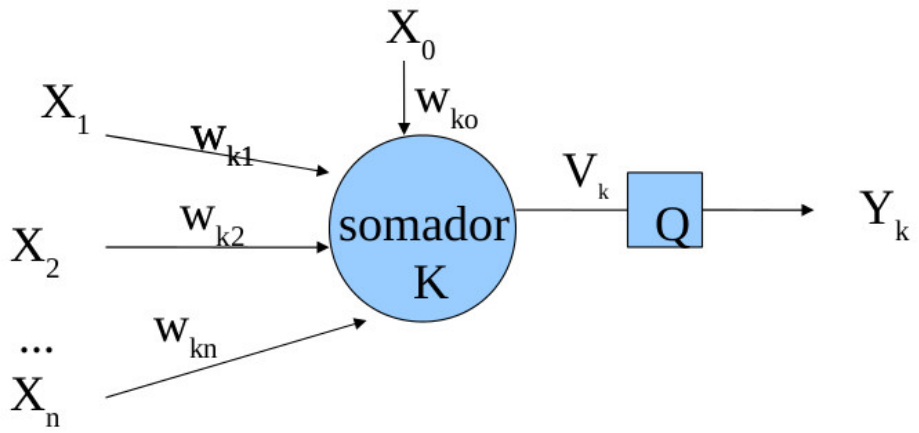
[1] Este material não pode ser reproduzido ou utilizado de forma parcial sem a permissão dos autores.

Sinopse

- Nesta aula, continuamos a estudar **redes neurais.**
- Este material foi construído com base nos algoritmos implementados.

Sumário

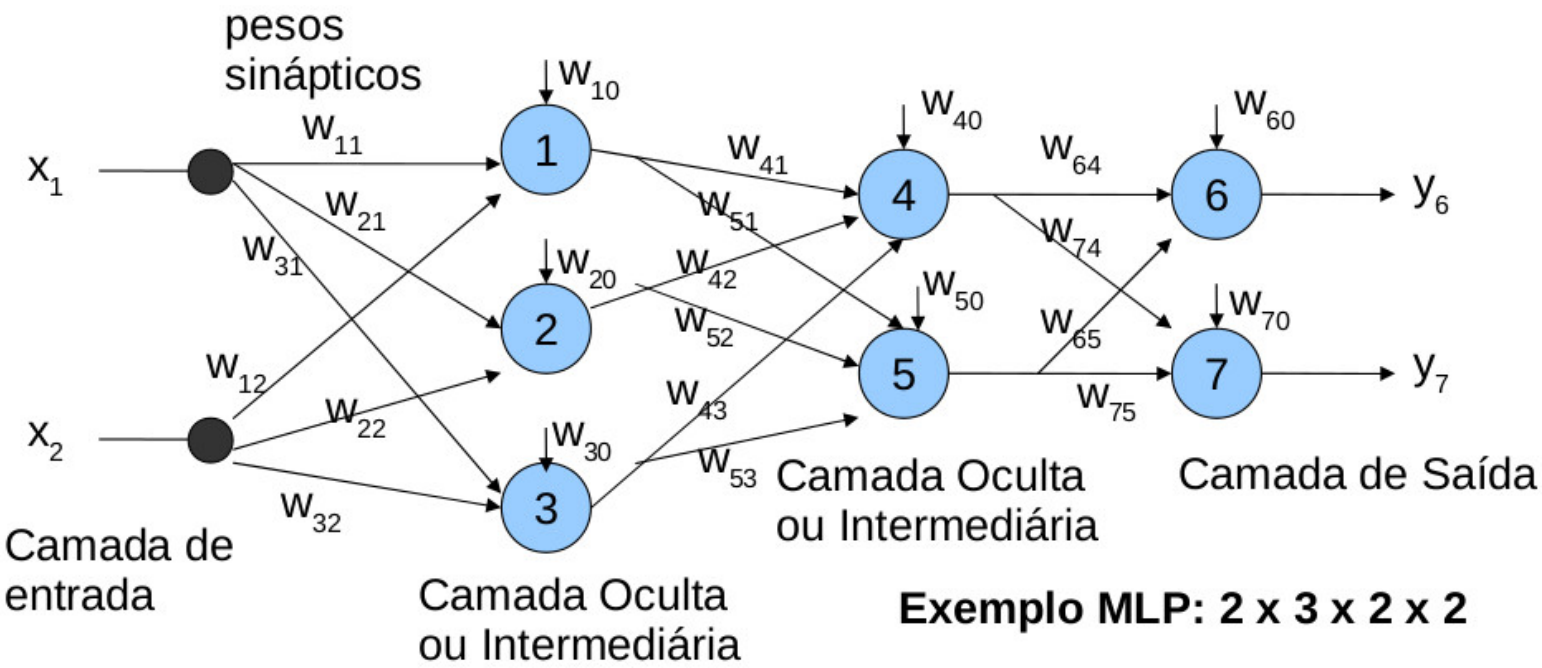
Relembrando: Neurônio Artificial



- $v_k = \sum_{i=0}^{n} w_{ki} \times x_i$
- $y_k = Q(v_k)$, onde a função de transferência pode ser:
  - limiar: $Q(v_k) = \begin{array}{ll} 1 & se\ v_k \geq 0 \\ 0 & caso\ contrário \end{array}$

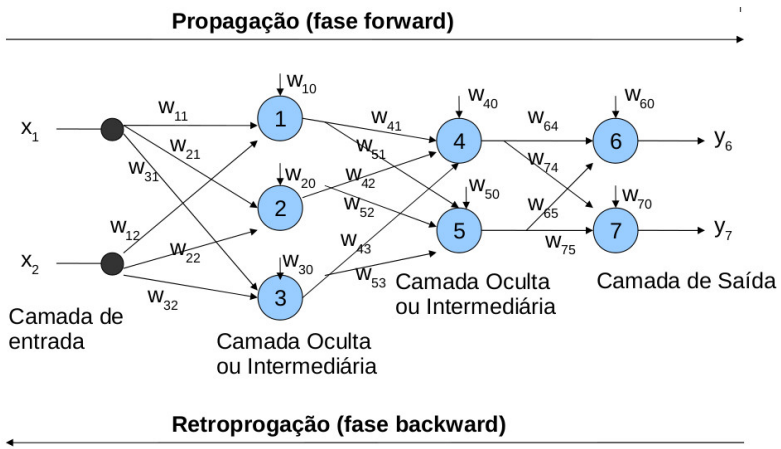
Relembrando: MultiLayer Perceptron (MLP)

- MLPs são redes **perceptron** de múltiplas camadas (feed forward), totalmente conectadas, **contendo uma ou mais camadas ocultas**.



Relembrando: MLP - Treinamento

- O algoritmo **Error Backpropagation é usado para treinar a rede**.
- Ele possui 2 etapas: **forward** e **backward**.
- Em cada etapa a rede é percorrida em um sentido.
  - A fase forward (para frente) – **propagação** - é utilizada para definir a saída da rede para um dado padrão de entrada.
  - A fase backward (para trás) – **retropropagação** - utiliza a saída desejada e a saída gerada pela rede para atualizar os pesos de suas conexões.



Relembrando: MLP - Treinamento

- **Funções de transferência** (em Java e C)
  - Limiar: recomenda-se seu uso apenas na camada de saída
  - Logística:
    - valores entre [0;1]
    - $Q(v_k) = \frac{1}{1+exp(-v_k)}$
  - Tangente hiperbólica:
    - valores entre [-1;1]
    - $Q(v_k) = tanh(v_k)$

MLP - Algoritmo de Treinamento

- **Algoritmo Error-Backpropagation (Retropropagação do Erro)**
  - Considere que a topologia da rede já está definida e que há um conjunto de Treino com N pares (X,D), onde:
    - X é o conjunto de entrada: $\{x_1, x_2, x_3, x_4, ..x_Z\}$, com Z entradas (atributos)
    - D é o conjunto de saídas desejadas:$\{d_1, d_2, d_3, d_4, ..x_M\}$, com M saídas (uma para cada neurônio da camada de saída)

MLP - Algoritmo de Treinamento

- **Algoritmo Error-Backpropagation (Retropropagação do Erro)**
  - **Etapas :**
    1. Iniciar os pesos da rede **arbitrariamente com valores não nulos.**
    2. Apresentar cada padrãon de entrada do conjunto de treino e **propagá-lo** até a saída da rede (geração dos y's da camada de saída), onde $n = 1$até N.
    - **Propagação:** Para cada neurônio k da rede.
      - $v_k(n) = \sum_{i=0}^{z} (w_{ki}(n) \times x_i(n))$, onde k é o neurônio e i a entrada, para $i = 0$ até z (total de entradas, ou seja, total de atributos de um padrão n do conjunto de treino). Quando o neurônio for de uma camada oculta, x_i será y_i(saida da neurônio i da camada anterior).

MLP - Algoritmo de Treinamento

- **Algoritmo Error-Backpropagation (Retropropagação do Erro)**
  - **Etapas :**

    ..

    2. Apresentar cada padrãon de entrada do conjunto de treino e **propagá-lo** até a saída da rede (geração dos y's da camada de saída), onde $n = 1$até N.

    - **Propagação:** Para cada neurônio k da rede.
      - $v_k(n) = \sum_{i=0}^{z} (w_{ki}(n) \times x_i(n))$, ...
      - $y_k(n) = Q(v_k(n))$, a saída é gerada pela aplicação da função de transferência Q sobre o campo local induzido v.

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation : :**

  ..

  3. Iniciar a **Retropropagação**.

  a) Calcular o erro. Para cada neurônio k da camada de saída: $erro_k(n) = d_k(n) - y_k(n)$, onde d_k é a saída desejada, o rótulo (a classe) do padrão n.

  b) Calcular a energia do erro instantâneo para o padrão n propagado. $\xi(n) = \frac{1}{2} \sum_{k=1}^{s} erro_k(n)^2$, onde n identifica o padrão e $k = 1$até s (o número de neurônios da camada de saída). O erro instantâneo combina os erros de todos neurônios da camada de saída para o padrão então propagado.

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**

  ..

  3. Iniciar a **Retropropagação**.

  ...

  c) Calcular os gradientes δ da camada de saída.

  - Para cada neurônio k da camada de saída.

  $\delta_k(n) = Q'(v_k(n)) \times erro_k(n)$

  d) Calcular o ajuste dos pesos de k:

  $\triangle w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$, onde é a η taxa de aprendizagem – intervalo típico (0;1].

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**

  ..

  3. Iniciar a **Retropropagação**.

  ...

  e) Ajustar os pesos dos neurônios da camada de saída:
  $w_{ki}(n+1) = w_{ki}(n) + \triangle w_{ki}(n)$.
  - Pode-se usar ainda a **constante de momento** α, intervalo típico [0;1]: $w_{ki}(n+1) = w_{ki}(n) + \triangle w_{ki}(n) + \alpha \times w_{ki}(n-1)$

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**

  ..

  3. Iniciar a **Retropropagação**.

  ...

  f) Calcular os gradientes δ das camadas ocultas: Para cada neurônio k de uma camada oculta:

  $\delta_k(n) = Q'(v_k(n)) * \sum_{j=1}^{t} (\delta_j(n) * w_{jk}(n+1))$, onde j são os neurônios com os quais o neurônio k tem conexão à direita. Como a camada à direita já sofreu retropropagação, seus pesos já foram atualizados, por isso aparece $n+1$.

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**

  ..

  3. Iniciar a **Retropropagação**.

  ...

  g) Da mesma forma calcular o ajuste dos pesos de k:
  $\triangle w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$
  h) Também da mesma forma, ajustar os pesos dos neurônios das camadas ocultas:
  $w_{ki}(n+1) = w_{ki}(n) + \triangle w_{ki}(n)$.
  - Pode-se usar ainda a **constante de momento** α, intervalo típico [0;1]: $w_{ki}(n+1) = w_{ki}(n) + \triangle w_{ki}(n) + \alpha \times w_{ki}(n-1)$

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**
  - ...
  - Ao final de uma época, **calcular o Erro Médio Quadrado (EMQ):** Média aritmética dos erros instântaneos.
    - $EMQ = (\sum_{i=1}^{N} \xi_i)/N$
    - É usado como critério de parada.
    - Pode oscilar no inicio da aprendizagem, mas deve decrescer ao longo do treinamento.

MLP - Algoritmo de Treinamento

- **Critérios de parada** do algoritmo backpropagation:
  - número pré-definido de épocas;
  - valor pré-definido como desejado para o erro médio quadrado;
  - variação do erro médio quadrado nas últimas x épocas inferior a um valor pré-definido (convergência);
  - número de padrões corretamente classificados não se alterar;
  - combinação desses critérios.

MultiLayer Perceptron (MLP): Generalização

- **Generalização**:
  - Após o treinamento, os pesos que mapeiam os padrões de entrada nas saídas desejadas foram encontrados.
  - A generalização consistem em propagar pela rede, usando os pesos encontrados, os padrões pré-processados do conjunto de teste e analisar os resultados gerados pela rede.
  - Visto que as funções de transferência geram valores contínuos é comum um pós-processamento da saída gerada.
    - O pós-processamento é uma regra de decisão, geralmente baseada em algum valor limiar que auxilia a definir a classe do padrão de teste.
    - Ex: se y >= 0.8 então y=1

MultiLayer Perceptron (MLP): Simulação XOR

- Conjunto de Treino: (-1 : Falso; 1- Verdadeiro)

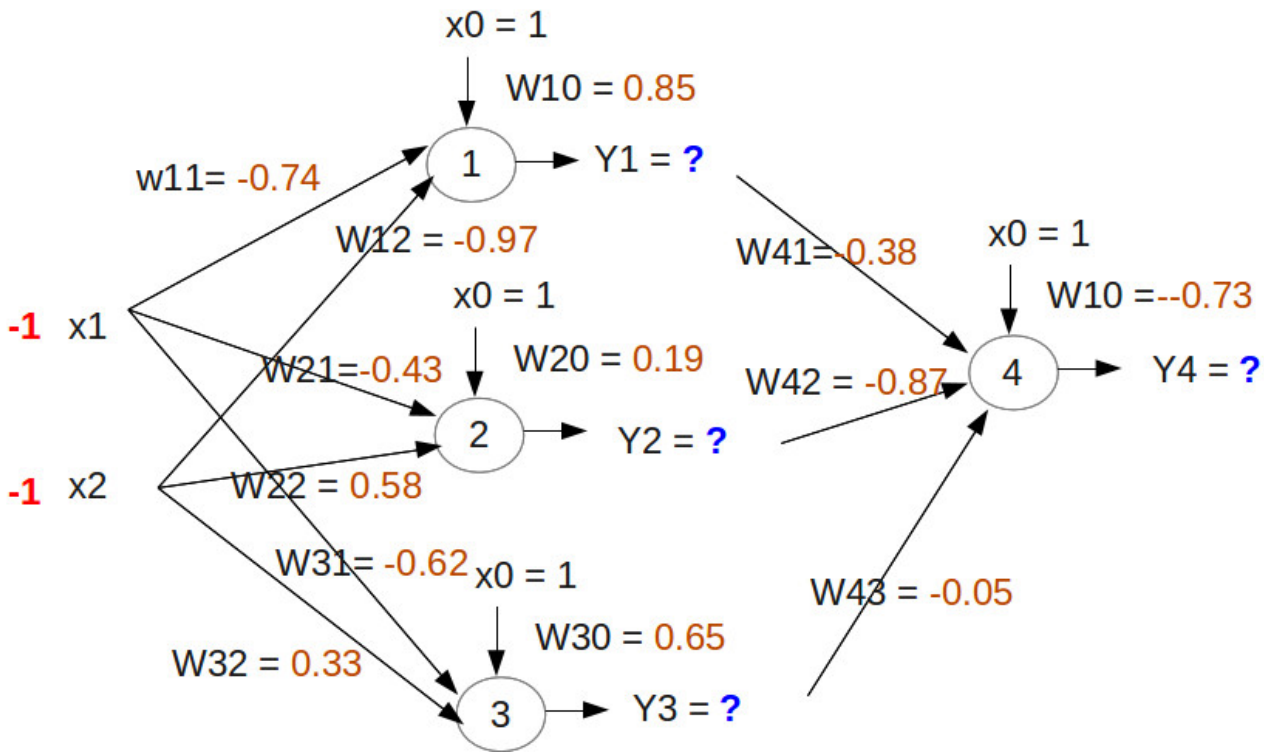
x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

- Função de Transferência
  - Camada oculta: tangente hiperbólica
  - Camada de saída: tangente hiperbólica
- Taxa de Aprendizagem: $\eta = 0.3$
- Constante de momentum: $\alpha = 0$
- Critérios de parada:
  - EMQ = 0.01 ou
  - Épocas: 1000

MultiLayer Perceptron (MLP): Simulação XOR

- Topologia: $2 \times 3 \times 1$

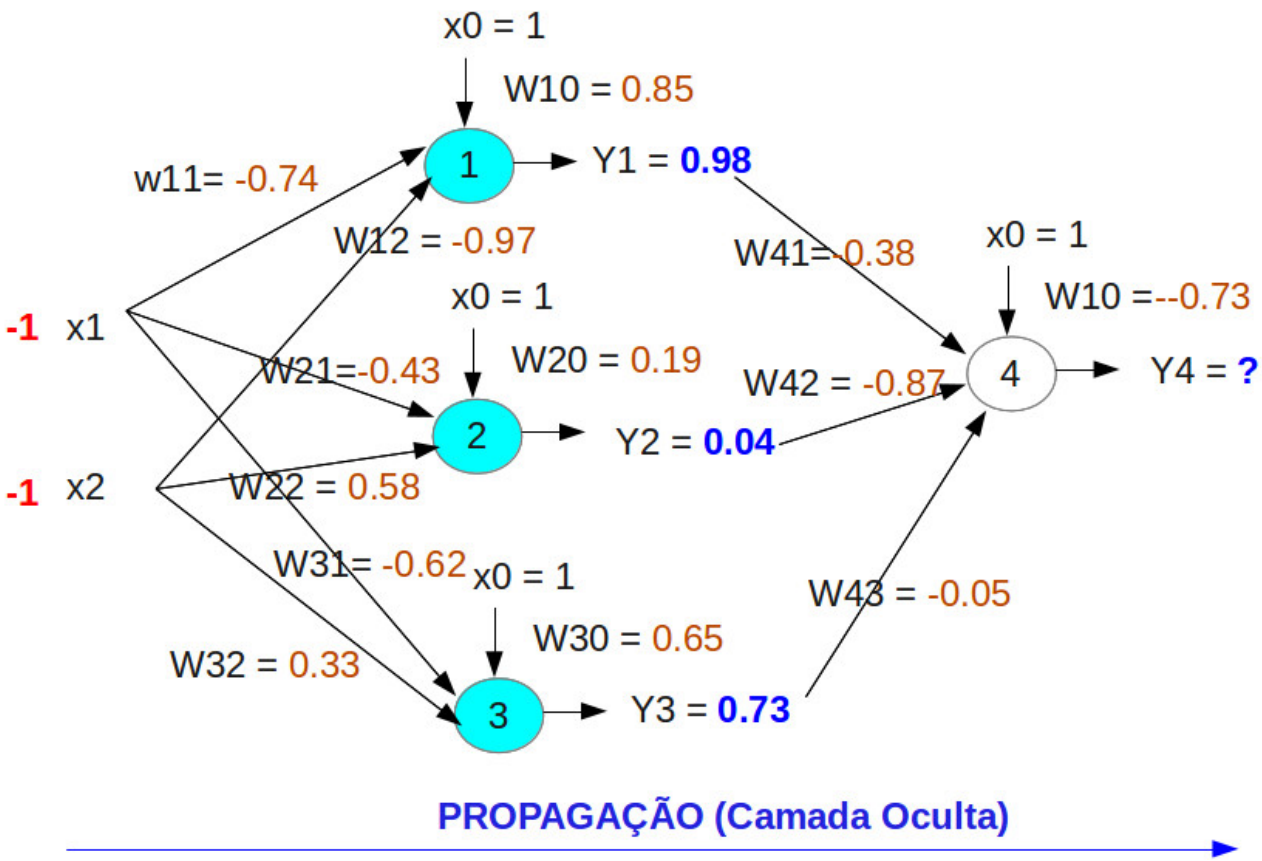
x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



MultiLayer Perceptron (MLP): Simulação XOR

- Topologia: $2 \times 3 \times 1$

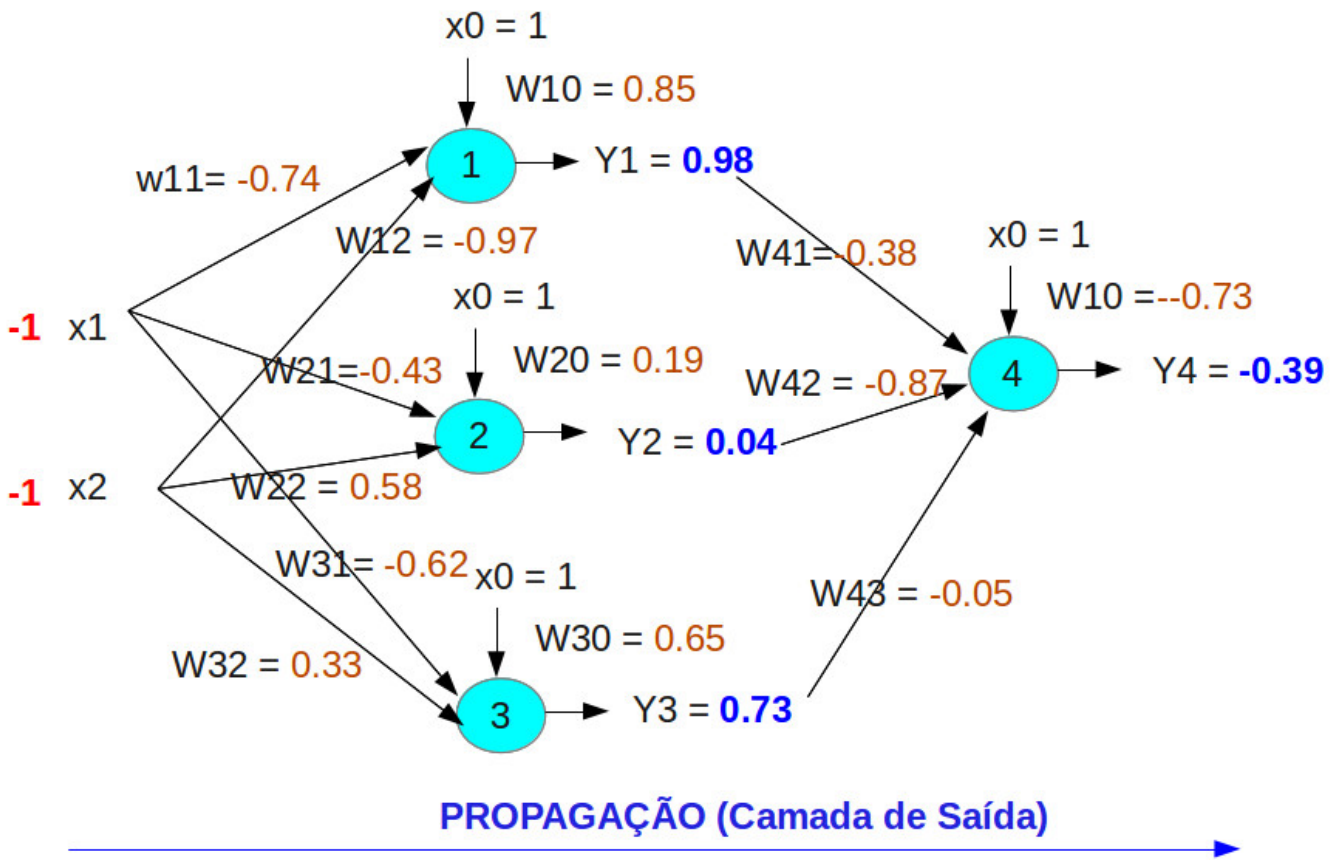
x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



MultiLayer Perceptron (MLP): Simulação XOR

- Topologia: $2 \times 3 \times 1$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



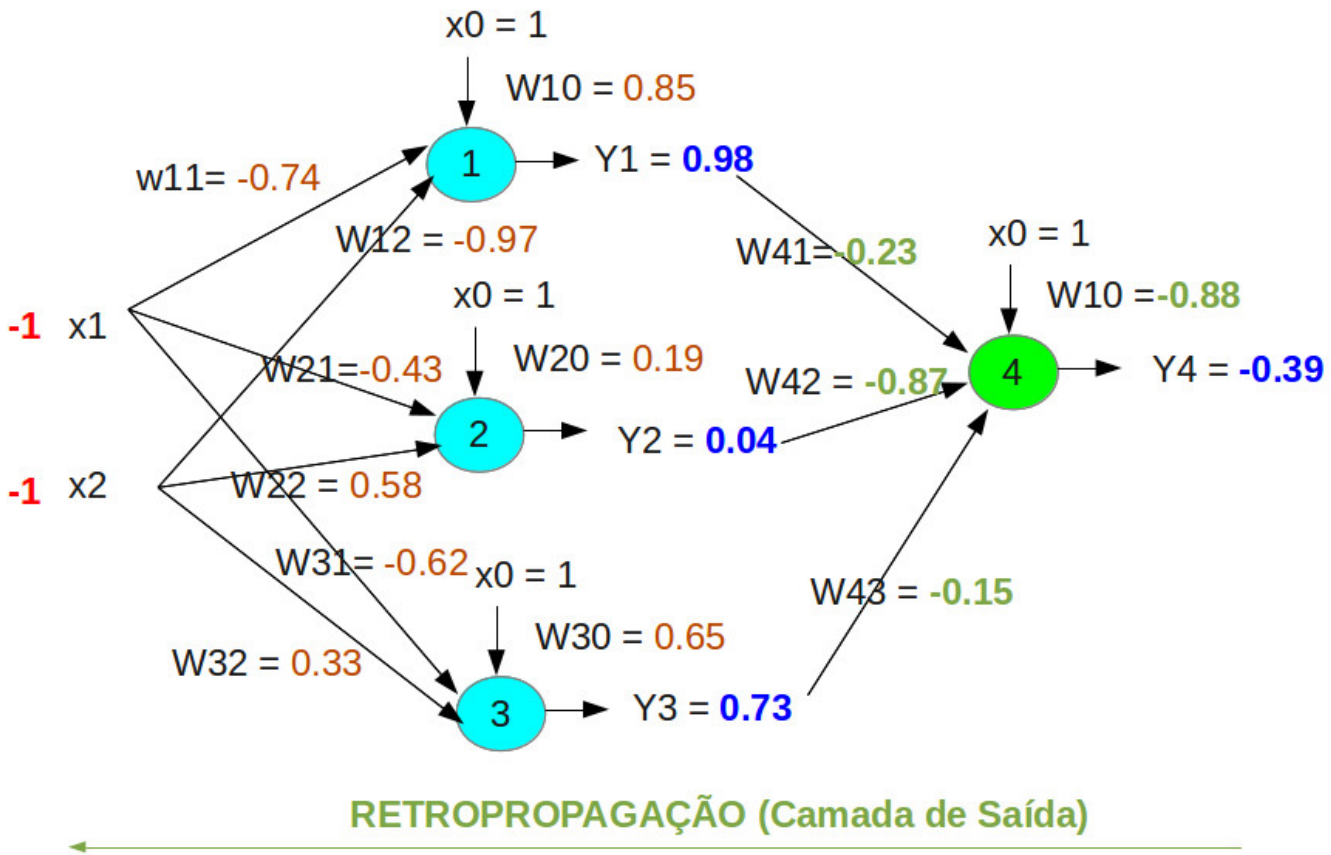
MultiLayer Perceptron (MLP): Simulação XOR

- Topologia: $2 \times 3 \times 1$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Neurônio 4 (Camada de Saida)
Erro: -0.6015309621707821
Erro Instantaneo:
0.1809197492250534
Gradiente =

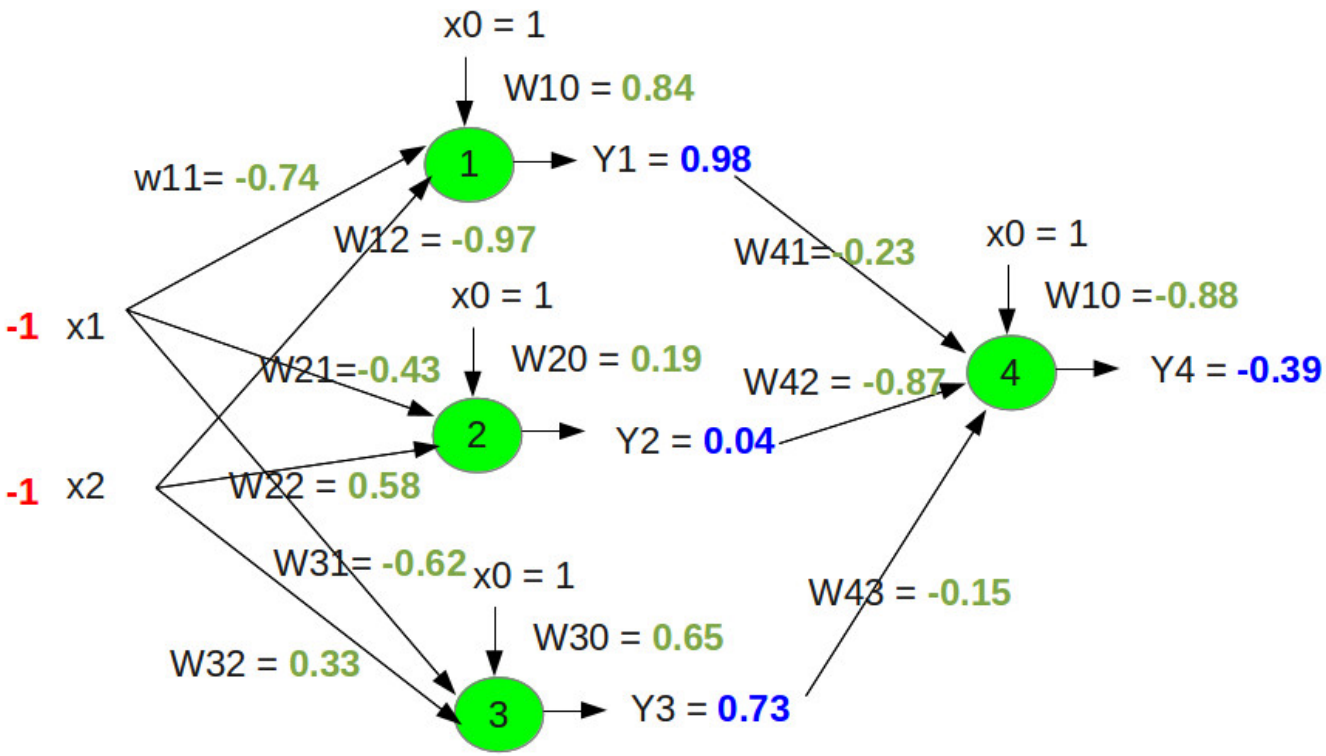
-0.5060213352461277



MultiLayer Perceptron (MLP): Simulação XOR

- Topologia: $2 \times 3 \times 1$

x1	x2	d
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



- Camada Oculta
- Gradiente Neurônio 1: -0.0027818424102683944
- Gradiente Neurônio 2: 0.4444999763647362
- Gradiente Neurônio 3: 0.03803417896970686

MultiLayer Perceptron (MLP): Simulação XOR

- Época: 0 - Erro médio quadrado: 0.6382619197564264
- Época: 1 - Erro médio quadrado: 0.6217486395123959
- Época: 2 - Erro médio quadrado: 0.5710208319681408
- Época: 3 - Erro médio quadrado: 0.5473148706283855
- Época: 4 - Erro médio quadrado: 0.5355908980006899
- Época: 5 - Erro médio quadrado: 0.5285727922709779
- ...
- Epoca: 446 - Erro médio quadrado: 0.009972902630117393

MultiLayer Perceptron (MLP): Simulação XOR

- **Pesos Finais:**
  - Camada Oculta
    - Neuronio: 1: w[10]=2.545363016862031, w[11]=-1.8662421473793356, w[12]=-1.7588858258650666
    - Neuronio: 2: w[20]=0.1914996892738543, w[21]=-0.4332485003836084, w[22]=0.5833743630800029
    - Neuronio: 3: w[30]=0.655200823402003 w[31]=-0.6209562373205424 w[32]=0.3376845283666482
  - Camada de Saida
    - Neuronio: 4: w[40]=0.3607155700762586 5, w[41]=2.388002191368662, w[42]=4.357544216452374, w[43]=-5.498527652915548 5

MultiLayer Perceptron (MLP): Simulação XOR

- Generalização: A rede está pronta para teste ou uso.
  1. Carregar a topologia com os pesos encontrados;
  2. Alimentar a rede com novas entradas e realizar apenas a propagação;
  3. Aplicar o pós-processamento às saídas geradas;
  4. Exibir o resultado.
- Em caso de teste, usar métricas para medir e avaliar os resultados obtidos.