1. Construa um MLP(2, 2, 2) de acordo com a arquitetura mostrada abaixo, para classificar as frutas Maçã e Laranja. As frutas serão identificadas através de duas características (features), tamanho (0,5 = Maçã e 0,8 = Laranja) e textura (lisa = 0,2 = Maçã e áspera = 0,6 = Laranja). Então, como amostras iniciais, considere Maça = [0,5; 0,2] e Laranja = [0,8; 0,6]. A saída: (i) se for Maçã, deve ser [1, 0] e (ii) se for Laranja, deve ser [0, 1].

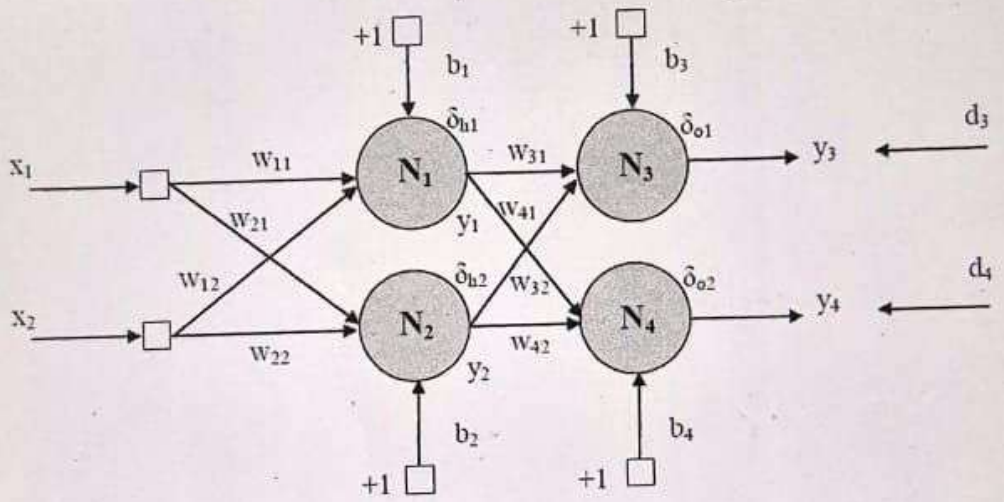


Figura Rede neural MLP(2, 2, 2).

Primeira Parte

1.1 Utilize o algoritmo de backpropagation para executar a primeira época de treinamento da Rede Neural;

1.2 Utilize taxa de aprendizado (eta) igual a 0,5;

1.3 Utilize taxa de momento (alfa) igual a 0,1;

1.4 Utillize função sigmóide em todas as camadas;

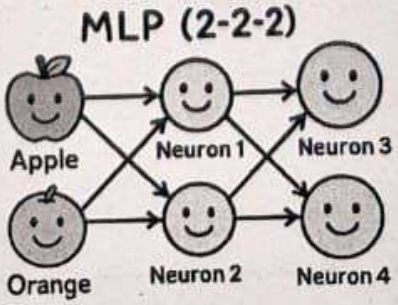
1.5 Inicie com todos os pesos e bias iguais a 0,1;

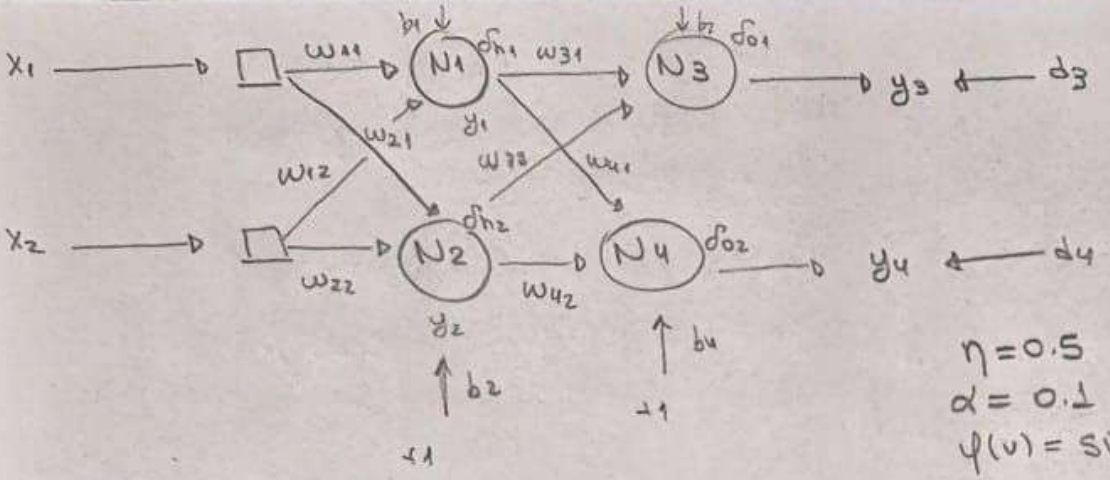
1.6 Mostre todas as contas para a primeira época de treinamento;

1.7 Depois de fazer todas as contas, gere uma rotina em Python para replicar os cálculos realizados e construir uma curva do RMSE [você define para quantas épocas e o tamanho do erro].

Segunda Parte

1.8 Refaça a Primeira Parte utilizando a função de ativação ReLU na camada oculta.





$\eta = 0.5$
$\alpha = 0.1$
$\varphi(v)$ = sigmóide
$w_{ij} = 0.1 \quad \forall i,j$
$b_i = 0.1$

Primeiras épocas: Maçã.

A primeira entrada: $\begin{cases} x_1 = 0.5 & d_3 = 1 \\ x_2 = 0.2 & d_4 = 0 \end{cases}$

Cálculo das saídas de N_1:

$v_1 = x_1 \cdot w_{11} + x_2 \cdot w_{12} + b_1 \cdot (+1)$

$v_1 = 0.5 \cdot 0.1 + 0.2 \cdot 0.1 + 1 \cdot 0.1$

$v_2 = 0.05 + 0.02 + 0.10$

$v_2 = 0.17 \parallel$

$\varphi(v) = \frac{1}{1+\exp(0.17)} = \frac{1}{1+0.844} = 0.542 \parallel$

$y_1 = 0.542 \parallel$

Cálculo das saídas de N_2:

$v_2 = x_2 \cdot w_{22} + x_1 \cdot w_{21} + 1 \cdot b_2$

$v_2 = 0.2 \cdot 0.1 + 0.5 \cdot 0.1 + 0.1$

$v_2 = 0.02 + 0.05 + 0.1 = 0.17 \parallel$

$\varphi(v) = \frac{1}{1+\exp(-0.17)} = \frac{1}{1+0.844} = 0.542 \parallel$

$y_2 = 0.542 \parallel$

Cálculo das saídas de N_3:

$v_3 = y_1 \cdot w_{31} + y_2 \cdot w_{32} + 1 \cdot b_3$

$v_3 = 0.542 \cdot 0.1 + 0.535 \cdot 0.1 + 0.1$

$v_3 = 0.0542 + 0.0535 + 0.1$

$v_3 = 0.2077 \parallel$

$\varphi(v) = \frac{1}{1+\exp(-0.2077)} = 0.552 \parallel$

$y_3 = 0.552 \parallel$

Cálculo das saídas de N_4:

$v_4 = y_1 \cdot w_{41} + y_2 \cdot w_{42} + 1 \cdot b_4$

$v_4 = 0.542 \cdot 0.1 + 0.535 + 0.1$

$v_4 = 0.2077$

$\varphi(v_4) = 0.552 \Rightarrow y_4 = 0.552 \parallel$

Cálculo do erro na saída da rede para primeira entrada:

$e_1 = d_3 - y_3 = 1 - 0.552 = 0.448$

$e_2 = d_4 - y_4 = 0 - 0.552 = -0.552$

Cálculo do gradiente local no N_3:

$\delta_{o1} = \varphi'(v_3) \cdot (d_3 - y_3)$

$\delta_{o1} = \varphi(v_3)[1 - \varphi(v_3)] \cdot (0.448)$

$\delta_{o1} = \frac{1}{1+\exp(-0.2077)} \left[1 - \frac{1}{1+\exp(-0.2077)}\right] \cdot (0.448)$

$\delta_{o1} = 0.552 [1 - 0.552] \cdot (0.448)$

$\delta_{o1} = 0.111 \parallel$

Cálculo do gradiente local no N_4:

$\delta_{o2} = \varphi'(v_4)(d_4 - y_4)$

$\delta_{o2} = 0.552 [1 - 0.552] \cdot (-0.552)$

$\delta_{o2} = -0.136 \parallel$

Cálculo do gradiente local N_1:

$$\delta_{h_1} = \varphi'(v_1) \cdot \sum_{k=1}^{2} \delta_k \cdot \omega_{kj}$$

$$\delta_{h_1} = \varphi'(v_1) \cdot \left[\delta_{o_1} \cdot \omega_{31} + \delta_{o_2} \cdot \omega_{41}\right]$$

$$\delta_{h_1} = \varphi(v_1)\left[1 - \varphi(v_1)\right] \cdot \left[\delta_{o_1} \cdot \omega_{31}\ \delta_{o_2} \cdot \omega_{41}\right]$$

$$\delta_{h_1} = 0.542\left[1 - 0.542\right] \cdot \left[0.111 \cdot 0.1 + (-0.136 \cdot 0.1\right]$$

$$\delta_{h_1} = 0.248 \cdot \left[0.0111 - 0.0136\right]$$

$$\underline{\delta_{h_1} = -6.2 \cdot 10^{-4}}$$

Cálculo do gradiente local no N_2:

$$\delta_{h_2} = \varphi'(v_2) \cdot \sum \delta_k \omega_{kj}$$

$$\delta_{h_2} = 0.535\left[1 - 0.535\right] \cdot \left[\delta_{o_1} \cdot \omega_{32} + \delta_{o_2} \cdot \omega_{42}\right]$$

$$\delta_{h_2} = 0.249 \cdot \left[0.111 \cdot 0.1 + (-0.136) \cdot 0.1\right]$$

$$\underline{\delta_{h_2} = -6.225 \cdot 10^{-4}}$$

Ajuste dos pesos utilizando a regra de aprendizagem:

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \eta \cdot \delta_k(n) \cdot y_k + \alpha \cdot \omega_{kj}(n-1)$$

$$\omega_{31} = 0.1 + 0.5 \cdot \delta_{o_1} \cdot y_1 + 0.1 \cdot 0$$

$$\omega_{31} = 0.1 + 0.5 \cdot 0.111 \cdot 0.542$$

$$\omega_{31} = 0.1 + 0.03 = \underline{0.130}$$

$$\omega_{32} = 0.1 + 0.5 \cdot \delta_{o_1} \cdot y_2 + 0.1 \cdot 0$$

$$\omega_{32} = 0.1 + 0.5 \cdot (0.111) \cdot 0.535$$

$$\underline{\omega_{32} = 0.130}$$

$$\omega_{41} = 0.1 + 0.5 \cdot \delta_{o_2} \cdot y_1 + 0.1 \cdot 0$$

$$\omega_{41} = 0.1 + 0.5 \cdot (-0.136)\ 0.542$$

$$\underline{\omega_{41} = 0.064}$$

$$\omega_{42} = 0.1 + 0.5 \cdot \delta_{o_2} \cdot y_2 + 0.1 \cdot 0$$

$$\omega_{42} = 0.1 + 0.5\ (-0.136) \cdot 0.535$$

$$\underline{\omega_{42} = 0.064}$$

$$\omega_{11} = 0.1 + 0.5 \cdot \delta_{h_1} \cdot x_1 + 0.1 \cdot 0$$

$$\omega_{11} = 0.1 + 0.5 \cdot (-6.2 \cdot 10^{-4}) \cdot 0.5$$

$$\underline{\omega_{11} = 0.0998}$$

$$\omega_{12} = 0.1 + 0.5 \cdot \delta_{h_2} \cdot x_2 + 0.1 \cdot 0$$

$$\omega_{12} = 0.1 + 0.5 \cdot (-6.225 \cdot 10^{-4}) \cdot 0.2$$

$$\underline{\omega_{12} = 0.0999}$$

$$\omega_{21} = 0.1 + 0.5 \cdot \delta_{h_1} \cdot x_1 + 0.1 \cdot 0$$

$$\underline{\omega_{21} = 0.0998}$$

$$\omega_{22} = 0.1 + 0.5 \cdot \delta_{h_2} \cdot x_2 + 0.1 \cdot 0$$

$$\underline{\omega_{22} = 0.0999}$$

Atualizando os bias pela regra da aprendizagem:

$$b_3 = b_3 + \eta \cdot \delta_{o_1} + \alpha \cdot b_3^{\,0}$$

$$b_3 = 0.1 + 0.5 \cdot 0.111 \cdot 1$$

$$\underline{b_3 = 0.1555}$$

$$b_4 = b_4 + 0.5 \cdot \delta_{o_2}$$

$$b_4 = 0.1 + 0.5 \cdot (-0.136)$$

$$\underline{b_4 = 0.032}$$

$$b_1 = b_1 + 0.5 \cdot \delta_{h_1}$$

$$b_1 = 0.1 + 0.5\ (-6.2 \cdot 10^{-4})$$

$$\underline{b_1 = 0.0997}$$

$$b_2 = b_2 + 0.5 \cdot \delta_{h_2}$$

$$b_2 = 0.1 + 0.5 \cdot \left(-6.225 \cdot 10^{-4}\right)$$

$$\underline{b_2 = 0.0997}$$

Para segunda entrada: $x_1 = 0.8$, $x_2 = 0.6$, $d_3 = 0$, $d_4 = 1$

Cálculo da saída de N_1:

$v_1 = x_1 \cdot w_{11} + x_2 \cdot w_{12} + b_1$

$v_1 = 0.8 \cdot 0.0998 + 0.6 \cdot 0.0999 + 0.0997$

$v_1 = 0.0798 + 0.05994 + 0.0997$

$v_1 = 0.2394$

$\varphi(v_1) = \frac{1}{1+\exp(-0.2394)} = 0.559$

$y_1 = 0.559$

Cálculo da saída de N_2:

$v_2 = x_2 \cdot w_{22} + x_1 \cdot w_{12} + b_2$

$v_2 = 0.6 \cdot 0.0999 + 0.8 \cdot 0.0999 + 0.0997$

$v_2 = 0.05994 + 0.0799 + 0.0997$

$v_2 = 0.2395$

$\varphi(v_2) = \frac{1}{1+\exp(-0.2395)} = 0.559$

$y_2 = 0.559$

Cálculo da saída de N_3:

$v_3 = y_1 \cdot w_{31} + y_2 \cdot w_{32} + b_3$

$v_3 = 0.559 \cdot 0.130 + 0.559 \cdot 0.130 + 0.155$

$v_3 = 0.3$

$\varphi(v_3) = \frac{1}{1+\exp(-0.3)} = 0.574$

$y_3 = 0.574$

Cálculo da saída de N_4:

$v_4 = y_1 \cdot w_{41} + y_2 \cdot w_{42} + b_4$

$v_4 = 0.559 \cdot 0.064 + 0.559 \cdot 0.064 + 0.032$

$v_4 = 0.103$

$\varphi(v_4) = \frac{1}{1+\exp(-0.103)} = 0.526$

$y_4 = 0.526$

Cálculo do gradiente local no N_3:

$\delta_{o1} = \varphi'(v_3) \cdot (d_3 - y_3)$

$\delta_{o1} = \varphi(v_3)[1 - \varphi(v_3)] \cdot (0 - 0.574)$

$\delta_{o1} = 0.574[1 - 0.574] \cdot (-0.574)$

$\delta_{o1} = -0.140$

Cálculo do gradiente local no N_4:

$\delta_{o2} = \varphi'(v_4) \cdot (d_4 - y_4)$

$\delta_{o2} = \varphi(v_4)[1 - \varphi(v_4)] \cdot (1 - 0.526)$

$\delta_{o2} = 0.526[1 - 0.526] \cdot (1 - 0.526)$

$\delta_{o2} = 0.118$

Cálculo do gradiente local no N_1:

$\delta_{h1} = \varphi'(v_1) \cdot [\delta_{o1} \cdot w_{31} + \delta_{o2} \cdot w_{41}]$

$\delta_{h1} = \varphi(v_1)[1 - \varphi(v_1)] \cdot [-0.14 \cdot 0.130 + 0.118 \cdot 0.064]$

$\delta_{h1} = 0.559 \cdot [1 - 0.559] \cdot [-0.0182 + 7.552 \cdot 10^{-3}]$

$\delta_{h1} = -2.625 \cdot 10^{-3}$

Cálculo do gradiente local no N_2:

$\delta_{h2} = \varphi'(v_2)[\delta_{o1} \cdot w_{32} + \delta_{o2} \cdot w_{42}]$

$\delta_{h2} = \varphi(v_2) \cdot [1 - \varphi(v_2)] \cdot [-0.14 \cdot 0.130 + 0.118 \cdot 0.064]$

$\delta_{h2} = -2.625 \cdot 10^{-3}$

Ajuste dos pesos utilizando a regra de aprendizagem:

$w_{31} = 0.130 + 0.5 \cdot \delta_{o1} \cdot y_1 + 0.1 \cdot 0.03$

$w_{31} = 0.130 + 0.5 \cdot (-0.140) \cdot 0.559 + 0.003$

$w_{31} = 0.09387$

$w_{32} = 0.130 + 0.5 \cdot \delta_{o1} \cdot y_2 + 0.1 \cdot 0.03$

$w_{32} = 0.09387$

$w_{41} = 0.064 + 0.5 \cdot \delta_{o2} \cdot y_1 + 0.1 \cdot (-0.036)$

$w_{41} = 0.064 + 0.5 \cdot 0.118 \cdot 0.559 - 0.0036$

$w_{41} = 0.0933$

$w_{42} = 0.064 + 0.5 \cdot 0.118 \cdot 0.559 - 0.0036$

$w_{42} = 0.0933$

$w_{11} = 0.0998 + 0.5 \cdot \delta_{h1} \cdot x_1 + 0.1\ (2 \cdot 10^{-4})$

$w_{11} = 0.0998 + 0.5 \cdot (-2.625 \cdot 10^{-3}) \cdot 0.8 + 2 \cdot 10^{-5}$

$w_{11} = 0.0987$

$w_{12} = 0.0999 + 0.5 \cdot \delta_{h2} \cdot x_2 + 0.1 \cdot (1 \cdot 10^{-4})$

$w_{12} = 0.0999 + 0.5(-2.625 \cdot 10^{-3}) \cdot 0.6 + 1 \cdot 10^{-5}$

$w_{12} = 0.0991$

$w_{21} = 0.0998 + 0.5 \cdot \delta_{h1} \cdot x_1 + 0.1\ (2 \cdot 10^{-4})$

$w_{21} = 0.0987$

$w_{22} = 0.0999 + 0.5 \cdot \delta_{h2} \cdot x_2 + 0.1\ (1 \cdot 10^{-4})$

$w_{22} = 0.0991$

Ajustando os bias pela regra da aprendizagem:

$b_3 = 0.1555 + 0.5 \cdot (-0.140) + 0.1 \cdot 0.0555$

$b_3 = 0.091$

$b_4 = 0.032 + 0.5\ (0.116) + 0.1 \cdot (-0.068)$

$b_4 = 0.0842$

$b_1 = 0.0997 + 0.5\ (-2.625 \cdot 10^{-3}) + (0.1 \cdot 3 \cdot 10^{-4})$

$b_1 = 0.0983$

$b_2 = 0.0983$

Fim da primeira época:

$$RMSE = \sqrt{\frac{\sum_{\mu} \sum_{o} e^2}{N_o N_e}}$$

$$E_1 = \frac{1}{2}\left[(1 - 0.552)^2 + (0 - 0.552)^2\right]$$

$$E_1 = \frac{1}{2}\left[(0.448)^2 + (-0.552)^2\right]$$

$E_1 = 0.2527$

$$E_2 = \frac{1}{2}\left[(0 - 0.574)^2 + (1 - 0.526)^2\right]$$

$$E_2 = \frac{1}{2}\left[(-0.574)^2 + (0.474)^2\right]$$

$E_2 = 0.2771$

Logo, o RMSE:

$$RMSE = \sqrt{\frac{E_1 + E_2}{2}} = \sqrt{\frac{0.2527 + 0.2771}{2}}$$

$RMSE = 0.5147$

Segunda Parte:

→ Primeira época

1ª entrada: $\begin{cases} x_1 = 0.5 & d_3 = 1 \\ x_2 = 0.2 & d_4 = 0 \end{cases}$

Cálculo da saída de N_1:

$v_1 = 0.17$ (aproveitando os cálculos)

$\varphi(v_1) = \max(0.17, 0) = 0.17$

$y_1 = 0.17$

Cálculo da saída de N_2:

$v_2 = 0.17$

$\varphi(v_2) = \max(0.17, 0) = 0.17$

$y_2 = 0.17$

Cálculo da saída de N_3:

$v_3 = 0.2077$

$$\varphi(v_3) = \frac{1}{1+\exp(-0.2077)} = 0.552$$

$y_3 = 0.552$

Cálculo da saída de N_4:

$v_4 = 0.2077$

$$\varphi(v_4) = \frac{1}{1+\exp(-0.2077)} = 0.552$$

$y_4 = 0.552$

Cálculo do gradiente local no N_3:

$\delta_{o1} = \varphi'(v_3)[d_3 - y_3]$

$\delta_{o1} = \varphi(v_3)[1 - \varphi(v_3)] \cdot [1 - 0.552]$

$\delta_{o1} = 0.111$

Cálculo do gradiente local no N_4

$\delta_{o2} = \underbrace{\varphi'(v_4)}_{\text{relu}} \cdot [d_4 - y_4]$

$\delta_{o2} = \varphi(v_4)[1 - \varphi(v_4)] \cdot [0 - 0.552]$

$\delta_{o2} = -0.136$

Cálculo do gradiente local N_1:

$\delta_{h1} = \underbrace{\varphi'(v_1)}_{\text{relu}} \cdot [\delta_{o1} \cdot w_{31} + \delta_{o2} \cdot w_{41}]$

$\delta_{h1} = -2.5 \cdot 10^{-3}$

Cálculo do gradiente local N_2:

$\delta_{h2} = \underbrace{\varphi'(v_2)}_{\text{relu}} \cdot [\delta_{o1} \cdot w_{32} + \delta_{o2} \cdot w_{42}]$

$\delta_{h2} = -2.5 \cdot 10^{-3}$

Ajuste dos pesos utilizando a regra de aprendizagem:

$w_{31} = 0.1 + 0.5 \cdot 0.111 \cdot y_1$

$w_{31} = 0.109$

$w_{32} = 0.1 + 0.5 \cdot 0.111 \cdot y_2$

$w_{32} = 0.109$

$w_{41} = 0.1 + 0.5\,(-0.136) \cdot y_1$

$w_{41} = 0.088$

$w_{42} = 0.1 + 0.5\,(-0.136) \cdot y_2$

$w_{42} = 0.088$

$w_{11} = 0.1 + 0.5\,(-2.5 \cdot 10^{-3}) \cdot x_1$

$w_{11} = 0.099$

$w_{12} = 0.1 + 0.5\,(-2.5 \cdot 10^{-3}) \cdot x_2$

$w_{12} = 0.099$

$w_{21} = 0.1 + 0.5\,(-2.5 \cdot 10^{-3}) \cdot 0.5$

$w_{21} = 0.099$

$w_{22} = 0.1 + 0.5\,(-2.5 \cdot 10^{-3}) \cdot 0.2$

$w_{22} = 0.099$

Atualizando os bias pela regra de aprendizagem:

$b_3 = 0.1 + 0.5 \cdot 0.111$

$b_3 = 0.1555$

$b_4 = 0.1 + 0.5\ (-0.136)$

$b_4 = 0.032$

$b_1 = 0.1 + 0.5\ (-2.5 \cdot 10^{-3})$

$b_1 = 0.09875$

$b_2 = 0.1 + 0.5\ (-2.5 \cdot 10^{-3})$

$b_2 = 0.09875$

→ Segunda entrada: $\begin{cases} x_1 = 0.8 & d_3 = 0 \\ x_2 = 0.6 & d_4 = 1 \end{cases}$

Cálculo da saída de N_1:

$v_1 = 0.8 \cdot 0.099 + 0.6 \cdot 0.099 + 0.09875$

$v_1 = 0.0792 + 0.0594 + 0.09875$

$v_1 = 0.237$

$\varphi(v_1) = \max(0.237, 0) = 0.237$

$y_1 = 0.237$

Cálculo da saída de N_2:

$v_2 = 0.8 \cdot 0.099 + 0.6 \cdot 0.099 + 0.09875$

$v_2 = 0.237$

$\varphi(v_2) = \max(0.237, 0)$

$y_2 = 0.237$

Cálculo da saída de N_3:

$v_3 = 0.237 \cdot 0.109 + 0.237 \cdot 0.109 + 0.1555$

$v_3 = 0.207$

$\varphi(v_3) = \dfrac{1}{1 + \exp(-0.207)} = 0.551$

$y_3 = 0.539$

Cálculo da saída de N_4:

$v_4 = 0.237 \cdot 0.088 + 0.237 \cdot 0.088 + 0.09875$

$v_4 = 0.140$

$\varphi(v_4) = \dfrac{1}{1 + \exp(-0.140)} = 0.535$ $\quad y_4 = 0.535$

Cálculo do gradiente local no N_3:

$\delta_{o1} = \varphi'(v_3) \cdot [0 - 0.539]$

$\delta_{o1} = \varphi(v_3)[1 - \varphi(v_3)] \cdot (-0.539)$

$\delta_{o1} = 0.539 \cdot 0.461\ (-0.539) = -0.134$

Cálculo do gradiente local no N_4:

$\delta_{o2} = \varphi'(v_4) \cdot [1 - 0.535]$

$\delta_{o2} = \varphi(v_4)\ [1 - \varphi(v_4)] \cdot [0.465]$

$\delta_{o2} = 0.535 \cdot (0.465) \cdot (0.465) = 0.116$

Cálculo do gradiente local no N_1:

$\delta_{h1} = \underbrace{\varphi'(v_1)}_{\text{relu}} \cdot [-0.134 \cdot 0.109 + 0.116 \cdot 0.088]$

$\delta_{h1} = 1 \cdot [-4.398 \cdot 10^{-3}] = -4.398 \cdot 10^{-3}$

Cálculo do gradiente local no N_2:

$\delta_{h2} = \varphi'(v_2)\ [-0.134 \cdot 0.109 + 0.116 \cdot 0.088]$

$\delta_{h2} = -4.398 \cdot 10^{-3}$

Ajuste dos pesos utilizando a regra de aprendizagem:

$w_{31} = 0.109 + (-0.134) \cdot 0.5 \cdot 0.237 + 0.1 \cdot (-9 \cdot 10^{-3})$

$w_{31} = 0.092$

$w_{32} = 0.109 \cdot (-0.134) \cdot 0.5 \cdot 0.237 + (-9 \cdot 10^{-4})$

$w_{32} = 0.092$

$w_{41} = 0.088 + 0.5 \cdot (0.116) \cdot 0.237 + 0.1 \cdot (-0.0$

$w_{41} = 0.100$

$w_{42} = 0.088 + 0.5\ (0.116) \cdot 0.237 + (-1.2 \cdot 10^{-3})$

$w_{42} = 0.100$

$w_{11} = 0.099 + 0.5\ (-4.398 \cdot 10^{-3}) \cdot 0.8 + 0.1\ (10^{-3})$

$w_{11} = 0.097$

$w_{12} = 0.099$

$w_{21} = 0.099 + 0.5\ (-4.398 \cdot 10^{-3}) \cdot 0.6 + 0.1\ (10^{-3})$

$w_{21} = 0.0967$

$w_{22} = 0.099 + 0.5 \cdot (-4.398 \cdot 10^{-3}) \cdot 0.6 + 0.1 \cdot 10^{-3}$

$w_{22} = 0.0967$

Ajustando o bias pela regra de aprendizagem:

$b_3 = 0.1555 + 0.5\,(-0.134) + 0.1\,(0.0555)$

$b_3 = 0.09405$

$b_4 = 0.032 + 0.5 \cdot (0.116) + 0.1\,(-0.068)$

$b_4 = 0.0832$

$b_1 = 0.09875 + 0.5\,(-4.398 \cdot 10^{-3}) + 0.1\,(1.25 \cdot 10^{-3})$

$b_1 = 0.096$

$b_2 = 0.096$

Calculando RMSE para primeiro época:

$$RMSE = \sqrt{\frac{(1-0.552)^2 + (0-0.552)^2 + (0-0.539)^2 + (1-0.535)^2}{4}}$$

$$RMSE = \sqrt{\frac{(0.448)^2 + (-0.552)^2 + (-0.539)^2 + (0.465)^2}{4}}$$

$RMSE = 0.503$

Enunciado: Construa um MLP(2, 2, 2) de acordo com a arquitetura mostrada abaixo, para classificar as frutas Maçã e Laranja. As frutas serão identificadas através de duas características (features), tamanho (0,5 = Maçã e 0,8 = Laranja) e textura (lisa = 0,2 = Maçã e áspera = 0,6 = Laranja). Então, como amostras iniciais, considere Maça = [0,5;0,2] e Laranja = [0,8; 0,6].

Código Python para MLP(2,2,2) - Primeira Parte

```
import numpy as np

def choose_derivate(derivate_function, valor):
  """
  Função que escolhe a derivada
  """
  if derivate_function == "sigmoide":
    return valor * (1 - valor)
  elif derivate_function == "relu":
    return 1.0 if valor >= 0 else 0

def choose_activation_function(activation_function, valor):
  """
  Função que escolhe a função de ativação
  """
  if activation_function == "sigmoide":
    sigmoid = 1 / (1 + np.exp(-valor))
    return sigmoid
  elif activation_function == "relu":
    relu = np.maximum(0, valor)
    return relu

def train_mlp(X:np.ndarray,
              d:np.ndarray,
              initial_weights:float,
              initial_bias:float,
              learning_rate:float,
              alpha:float,
              max_epochs:int,
              activation_function:str):
    """
    Treina um MLP(2,2,2) por um número de épocas e
    retorna o histórico do RMSE.

    Args:
        X (numpy.ndarray): Matriz de entrada (amostras).
        d (numpy.ndarray): Matriz de saídas desejadas.
        initial_weights (float): Pesos iniciais. Nessa implementação
        todos os pesos serão iguais
        initial_bias (float): Vetor de vieses iniciais.  Nessa
        implementação todos os bias serão iguais
```

```
    learning_rate (float): Taxa de aprendizado (eta).
    alpha (float): Taxa de momento.
    max_epochs (int): Número máximo de épocas de treinamento.
    activation_function (str): Função de ativação.

Returns:
    tuple: Uma tupla contendo:
        - numpy.ndarray: Pesos finais.
        - numpy.ndarray: Vieses finais.
        - list: Histórico do RMSE por época.
"""
neurons = [2, 2, 2] # [Entrada, Camadas, Neurônios por Camada]
# Pesos iniciais
matrix_initial_weights = np.full((4, 2), initial_weights)

# Bias iniciais
initial_bias = np.full(4, initial_bias)

# Variáveis auxiliares
nro_total_neurons = neurons[1] * neurons[2]
outputs = np.zeros(nro_total_neurons)
errors = np.zeros(nro_total_neurons)

# Inicializando as variáveis para o momento
first_iter = np.zeros_like(matrix_initial_weights)
first_bias = np.zeros_like(initial_bias)

rmse_values = []

for epoch in range(max_epochs):
    print(f"Epoch [{epoch+1}/{max_epochs}]")
    errors_sum = []

    for sample_index in range(X.shape[0]):
        x_sample = X[sample_index]
        d_sample = d[sample_index]

        ####################################
        # Etapa 1: Forward Propagation
        ####################################
        for neuro_index in range(nro_total_neurons):

          if neuro_index < neurons[1]: # Camada Oculta

            # Cálculo da saída do Neurônio 1,2 (v)
            v = initial_bias[neuro_index]
            + x_sample[0] * matrix_initial_weights[neuro_index][0]
            + x_sample[1] * matrix_initial_weights[neuro_index][1]
```

```
      # Aplicando v na função de ativação
      outputs[neuro_index] =
      choose_activation_function(activation_function, v)

    else: # Camada de Saída
      # Cálculo da saída do Neurônio 3,4 (v)
      v = initial_bias[neuro_index]
      + outputs[0] * matrix_initial_weights[neuro_index][0]
      + outputs[1] * matrix_initial_weights[neuro_index][1]

      # Aplicando v na função de ativação
      outputs[neuro_index] =
      choose_activation_function(activation_function, v)

      # Erro da saída do Neurônio 3 e 4 em comparação à saída esperada
      errors[neuro_index] =
      d_sample[neuro_index - 2] - outputs[neuro_index]
      errors_sum.append(errors[neuro_index])

# print("Saídas y = ",outputs)
#####################################
# Etapa 2: Backpropagation
#####################################
gradient_output = np.zeros(neurons[2])
# Cálculo do gradiente local nos neurônios de saída
for neuro_index in range(nro_total_neurons - 2):
  gradient_output[neuro_index] =
  choose_derivate(activation_function,outputs[neuro_index + 2])
  * errors[neuro_index + 2]

gradient_hidden = np.zeros(neurons[1])
# Cálculo do gradiente local nos neurônios na camada oculta
for neuro_index in range(nro_total_neurons - 2):
  delta = gradient_output[0]
  * matrix_initial_weights[2][neuro_index]
  + gradient_output[1] * matrix_initial_weights[2][neuro_index]
  gradient_hidden[neuro_index] =
  choose_derivate(activation_function, outputs[neuro_index])
  * delta

#####################################
# Etapa 3: Atualização de Pesos e Bias
#####################################

# Ajuste dos pesos utilizados a regra de aprendizagem
gradients_local = np.concatenate([gradient_hidden, gradient_output])
# print("Gradientes locais = ",gradients_local)
```

```
            for linha in range(matrix_initial_weights.shape[0]):
              for coluna in range(matrix_initial_weights.shape[1]):

                momemtum = alpha * first_iter[linha][coluna]

                if linha < 2:
                  learning = learning_rate
                  * gradients_local[coluna]
                  * x_sample[coluna]
                else:

                  learning = learning_rate
                  * gradients_local[linha]
                  * outputs[coluna]

                matrix_initial_weights[linha][coluna] += learning
                + momemtum

            # print("Pesos Atualizados=\n",matrix_initial_weights)

            # Atualizando os bias pela regra da aprendizagem
            for b in range(len(initial_bias)):
              momentum = alpha * first_bias[b]
              initial_bias[b] += (learning_rate * gradients_local[b])
              + momemtum

            first_iteration = matrix_initial_weights.copy()
            first_bias = initial_bias.copy()

        # Calculando RMSE
        rmse = np.sqrt(sum(np.array(errors_sum)**2)) / 2
        rmse_values.append(rmse)
        if rmse < 0.1:
            print(f"Treinamento parou na época {epoch + 1}
            com RMSE de {rmse:.4f}")
            break
    return matrix_initial_weights, initial_bias, rmse_values
```

```
####################################
Para utilizar as funções acima
####################################
X = np.array([[0.5, 0.2],
     [0.8, 0.6]])

d = np.array([[1,0], # Maçã
     [0,1]]) # Laranja

pesos, bias, rmses_sigmoide = train_mlp(X=X,
                                d=d,
                                initial_weights=0.1,
                                initial_bias=0.1,
                                learning_rate=0.5,
                                alpha=0.1,
                                max_epochs=200,
                                activation_function="sigmoide")
```

A curva de aprendizagem com a métrica *Root Mean Squared Error* (RMSE) utilizando a função sigmóide em todas as camadas é apresentada na Figura 1.

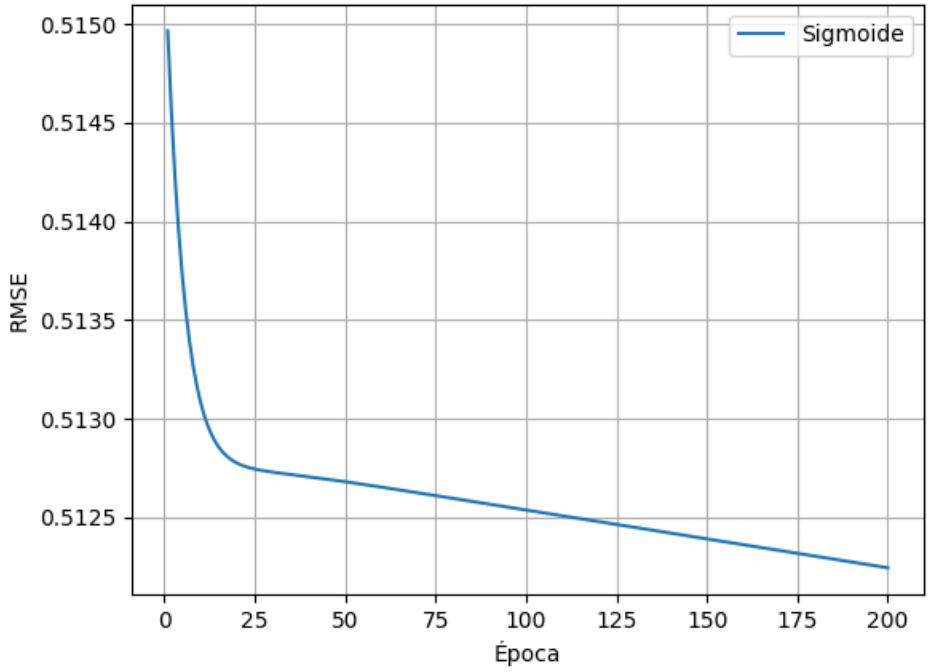


Figure 1: Curva de aprendizagem da rede MLP(2,2,2) com função sigmóide em todas as camadas da rede.

Código Python para MLP(2,2,2) - Segunda Parte

```
# Entrada
X = np.array([[0.5, 0.2],
              [0.8, 0.6]])

d = np.array([[1,0], # Maçã
             [0,1]])  # Laranja

# Testando a MLP(2,2,2) com a função ReLu em ambas as camadas, obtém-se:
pesos, bias, rmses_sigmoide = train_mlp(X=X,
                                d=d,
                                initial_weights=0.1,
                                initial_bias=0.1,
                                learning_rate=0.5,
                                alpha=0.1,
                                max_epochs=200,
                                activation_function="relu")

# Saída
Treinamento parou na época 73 com RMSE de 0.0957
```

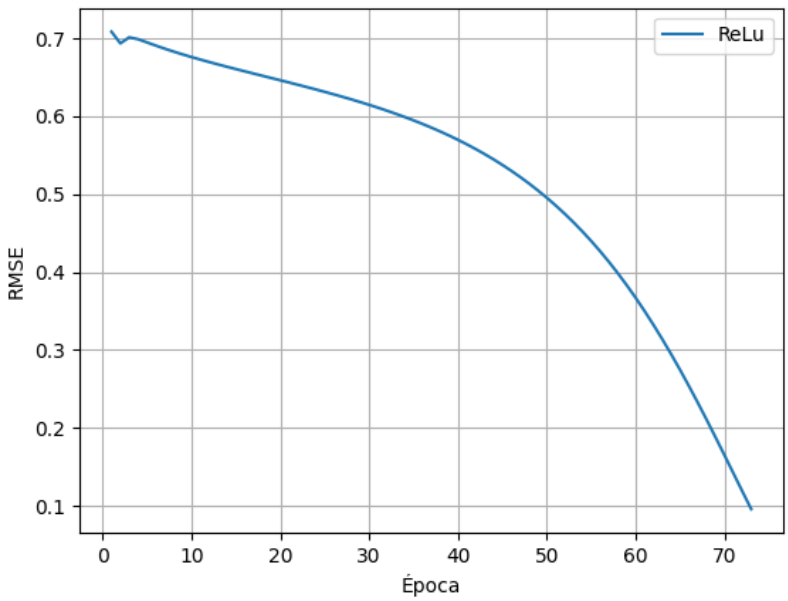


Figure 2: Curva de aprendizagem da rede MLP(2,2,2) com função ReLu em todas as camadas da rede.

Agora, alterando a função de ativação na camada oculta para a função ReLu e mantendo a função sigmóide na camada de saída. Para isso, é preciso alterar a função "train_mlp" apresentada para adequar à essas condições. A curva de aprendizagem é mostrada na Figura 3.

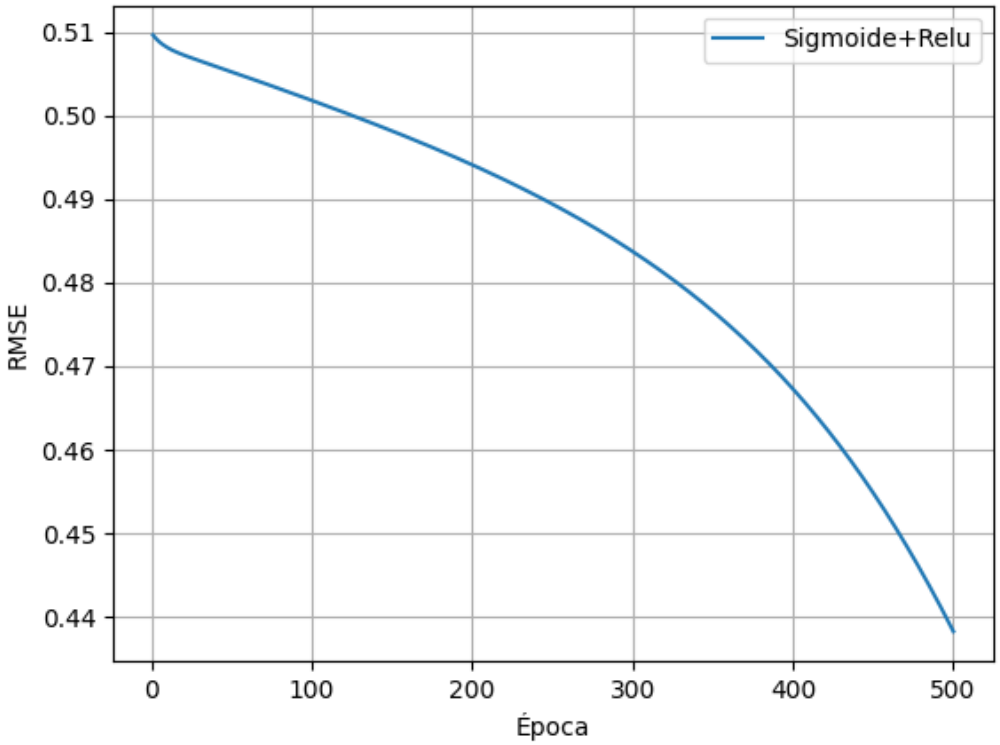


Figure 3: Curva de aprendizagem da rede MLP(2,2,2) com função ReLu na camada oculta e função sigmóide na camada de saída.