Laboratório 7 – Redes Neurais

Inteligência Artificial para Robótica Móvel – CT-213

Aluno: Caio Graça Gomes

Professor: Marcos Ricardo Omena de Albuquerque Maximo

Introdução:

Nesse laboratório, teve-se por objetivo construir uma rede neural de 2 camadas (sendo elas uma de entrada, uma escondida e uma de saída) para segmentar cores da visão de um robô em uma partida de futebol de robôs. Para tal feito, foi implementada a lógica do back e front propagation de modo que a rede seja capaz de realizar classificação multiclasse..

Metodologia e descrição em alto nível do algoritmo utilizado:

Para a construção da rede neural que realiza classificação multiclasse, foi utilizada uma *loss function* de regressão logística multiclasse:

$$L(y^{(i)},\hat{y}^{(i)}) = -\sum_{c=1}^{C}[(1-y_c^{(i)})log(1-\hat{y}_c^{(i)}) + y_c^{(i)}log(\hat{y}_c^{(i)})]$$

Sendo C o número de *outputs* da rede, $y_c^{(i)}$ as saídas esperadas pelo neurônio de número c no i-ésimo treinamento e $\hat{y}_c^{(i)}$ as saídas de fato obtidas. A função custo utilizada foi obtida pela média das *loss functions* calculadas durante o treinamento da rede:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(y^{(i)},\hat{y}^{(i)})$$

Com θ sendo a matriz $[w\ b]^T$, m o número de casos teste para treinamento, w a matriz dos pesos dos neurônios e b o vetor de *biases* dos neurônios. Além disso, considerando a rede com uma função de ativação sigmóide $\sigma(.)$ em todos os neurônios, chega-se às seguintes equações para o algoritmo de Back Propagation nessa rede:

$$\frac{\partial L}{\partial w^{[2]}_{ck}} = \delta^{[2]}_{c} a^{[1]}_{k} \qquad \frac{\partial L}{\partial b^{[2]}_{c}} = \delta^{[2]}_{c} \qquad \frac{\partial L}{\partial w^{[1]}_{kj}} = \delta^{[1]}_{k} a^{[0]}_{j} \qquad \frac{\partial L}{\partial b^{[1]}_{k}} = \delta^{[1]}_{k}$$

em que:

$$\delta^{[2]}_{c} = (\hat{y}^{(i)}_{c} - y^{(i)}_{c}) \qquad \delta^{[1]}_{k} = \sum_{c=1}^{C} w^{[2]}_{ck} \delta^{[2]}_{c} \sigma'(z^{[1]}_{k})$$

Com isso, foi possível implementar a lógica do *back* e *front propagation* fazendo uso de uma descida de gradiente estocástica. Ao final da implementação, testou-se o código em test_neural_network.py (taxa de aprendizado 6.0 e 10 neurônios na camada escondida) usando funções xor(x) e sum_gt_zero(x) provindos de utils.py. Após isso, com 20 neurônios na camada escondida, executou-se o test_color_segmentation.py para a obtenção de uma foto segmentada.

Resultados do test_neural_network.py:

Ao final da implementação da rede neural, obteve-se os seguintes resultados no primeiro teste:

Figura 1: Convergência da função custo utilizando sum_gt_zero.

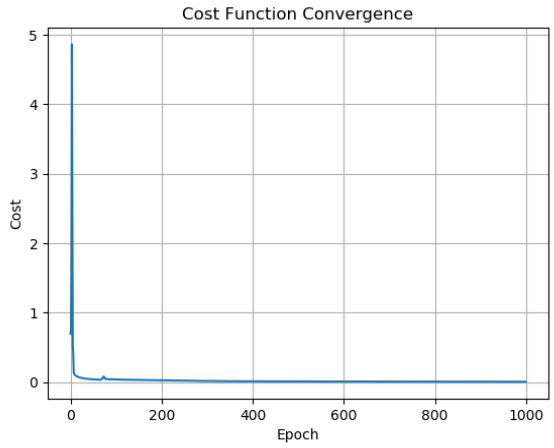


Figura 2: Convergência da função custo utilizando xor.

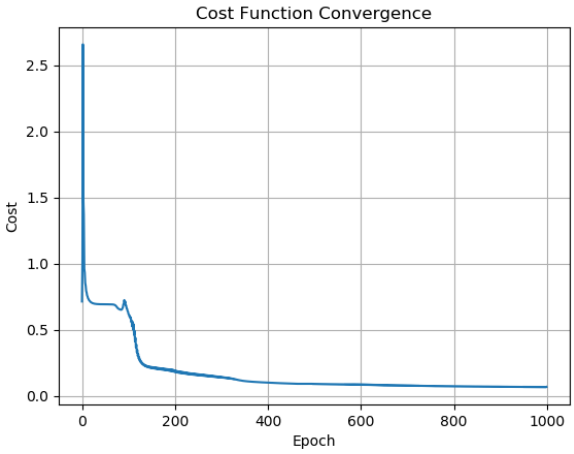


Figura 3: Dataset com devida classificação em sum_gt_zero.

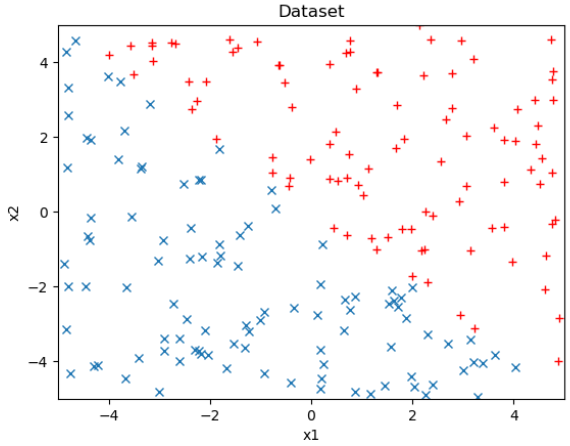


Figura 4: Dataset com devida classificação em sum_gt_zero.

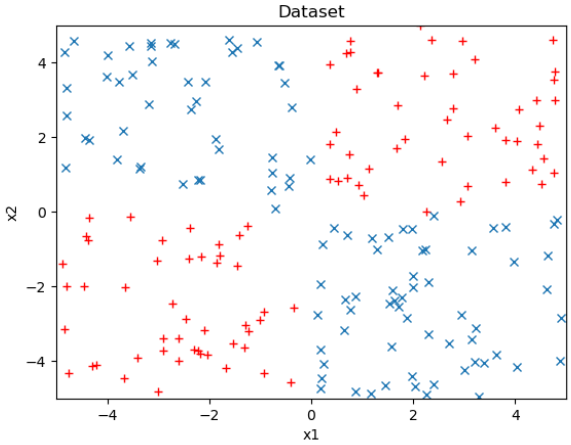
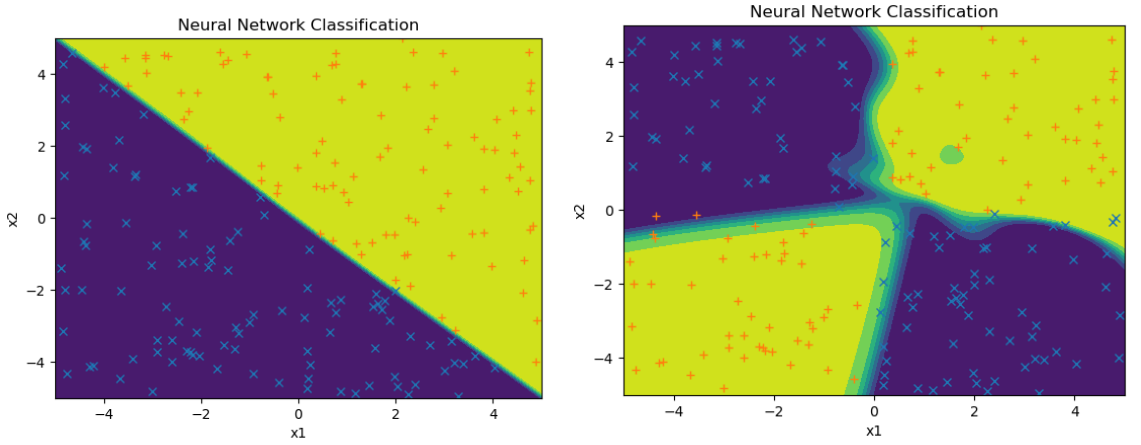


Figura 5: Classificação da função sum_gt_zero pela rede neural.

Figura 6: Classificação da função xor pela rede neural.



A partir desses resultados, é possível dizer que o algoritmo da rede neural foi eficiente para o sum_gt_zero(), mas não tão satisfatório para xor().

Resultados do test_color_segmentation.py:

Aplicando uma rede neural com 3 neurônios de entrada, 2 de saída, 20 na camada escondida e taxa de aprendizado igual a 6 ao problema da segmentação de cores de uma foto, foram obtidos os seguintes resultados:

Figura 7: Gráfico da convergência da função custo da segmetação.

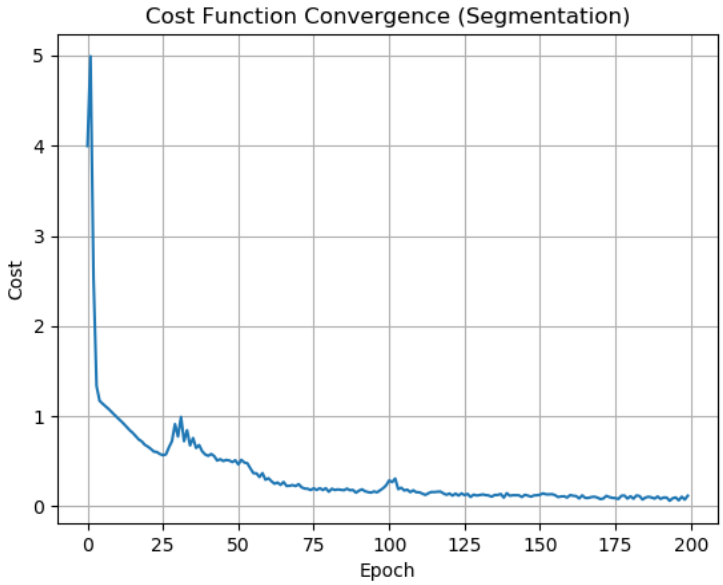


Figura 8: Imagem original a ser segmentada pela rede neural.

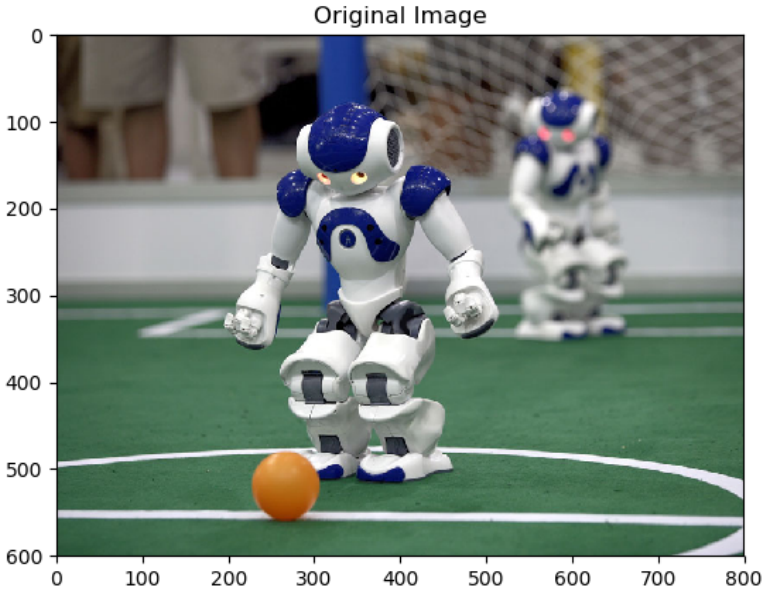
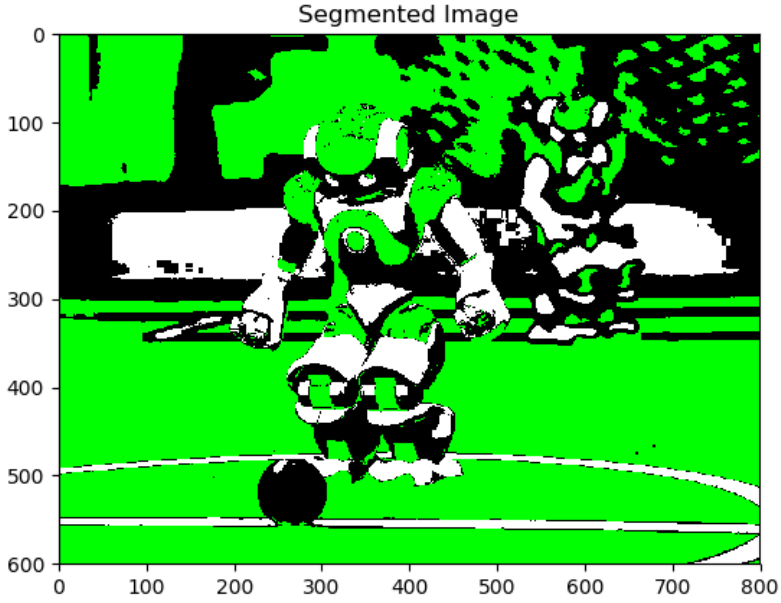


Figura 9: Imagem segmentada pela rede neural.



Assim, com a imagem obtida, é possível concluir que o aprendizado da rede neural foi eficiente, dado que a segmentação foi bem realizada. É possível, ainda, melhorar ainda mais o algoritmo variando parâmetros tais quais o número de neurônios na camada escondida, entrada ou saída e a taxa de aprendizado, mas o resultado obtido já é bem satisfatório para o seu objetivo.