1 Introdução

O objetivo desse laboratório foi implementar (em *Python*), uma rede neural para realizar a segmentação de cores de uma certa imagem pré-disponibilizada pelo professor. Para tal, teve-se que implementar os métodos de *forward propagation* e *back propagation* para realizar a descida de gradiente na função custo. Nesse caso, a função custo utilizada foi uma função logística.

Além disso, o código foi implementado com vetorização, por motivos de eficiência e clareza, de forma que a rede neural consegue treinar rapidamente.

2 Código

```
def forward_propagation(self, inputs):
    z = [None] * 3
    a = [None] * 3
    z[0] = inputs
    a[0] = inputs

    self.m=inputs.shape[1]

    para l=1: 3:
        z[l]=self.weights[l]@a[l-1]+self.biases[l]
        a[l]=sigmoid(z[l])

    return z, a

def compute_gradient_back_propagation(self, inputs, expected_outputs):
    weights_gradient = [None] * 3
    biases_gradient = [None] * 3
    z, a = self.forward_propagation(inputs)
    x=inputs
    y=expected_outputs
    dz=[None]*3
    dz[2]=a[2]-y
    dz[1]=self.weights[2].transpose()@dz[2]*sigmoid_derivative(z[1])

    weights_gradient[2]=dz[2]@a[1].transpose()/self.m
    weights_gradient[1]=dz[1]@x.transpose()/self.m

    biases_gradient[2]=np.sum(dz[2], axis=1, keepdims=True)/self.m
    biases_gradient[1]=np.sum(dz[1],axis=1,keepdims=True)/self.m

    return weights_gradient, biases_gradient

def back_propagation(self, inputs, expected_outputs):
    weights_gradient, biases_gradient = self.compute_gradient_back_propagation(inputs,
        expected_outputs)

    self.weights[1]-= self.alpha*(weights_gradient[1])
    self.weights[2]-= self.alpha * (weights_gradient[2])
    self.biases[1]-=self.alpha*(biases_gradient[1])
    self.biases[2]-=self.alpha*(biases_gradient[2])
```

3 Resultados e discussão

Nos casos de teste, duas funções de classificação foram disponibilizadas (*xor()* e *sum_gt_zero()*), as quais tiveram seus resultados apresentados nas figuras 1, 2, 3 e 4. A função *sum_gt_zero* apresentou os resultados esperados, aproximando-se da função $f(x) = -x$, no entanto a função *xor* apresentou um erro aparente, uma vez que deveria separar os 4 quadrantes. Uma possível razão foi o fato da função *xor()* não ter uma convergência tão rápida quanto a *sum_gt_zero()*. Explicar tal fato não é trivial, mas uma possível razão seria os hiperparâmetros estarem mais adequados para resolver um problema do que outro.

No teste final, foi proposto um problema de segmentação de cores, dada uma imagem pelo professor. Pode-se ver comparando as figuras 5 e 6 que a segmentação foi eficiente em identificar as bordas das imagens. A figura 7 mostra que a convergência de tal segmentação não foi tao eficiente, o que pode ser explicado pelo mesmo motivo supracitado, os hiperparâmetros não foram ideais para esse tipo de problema.

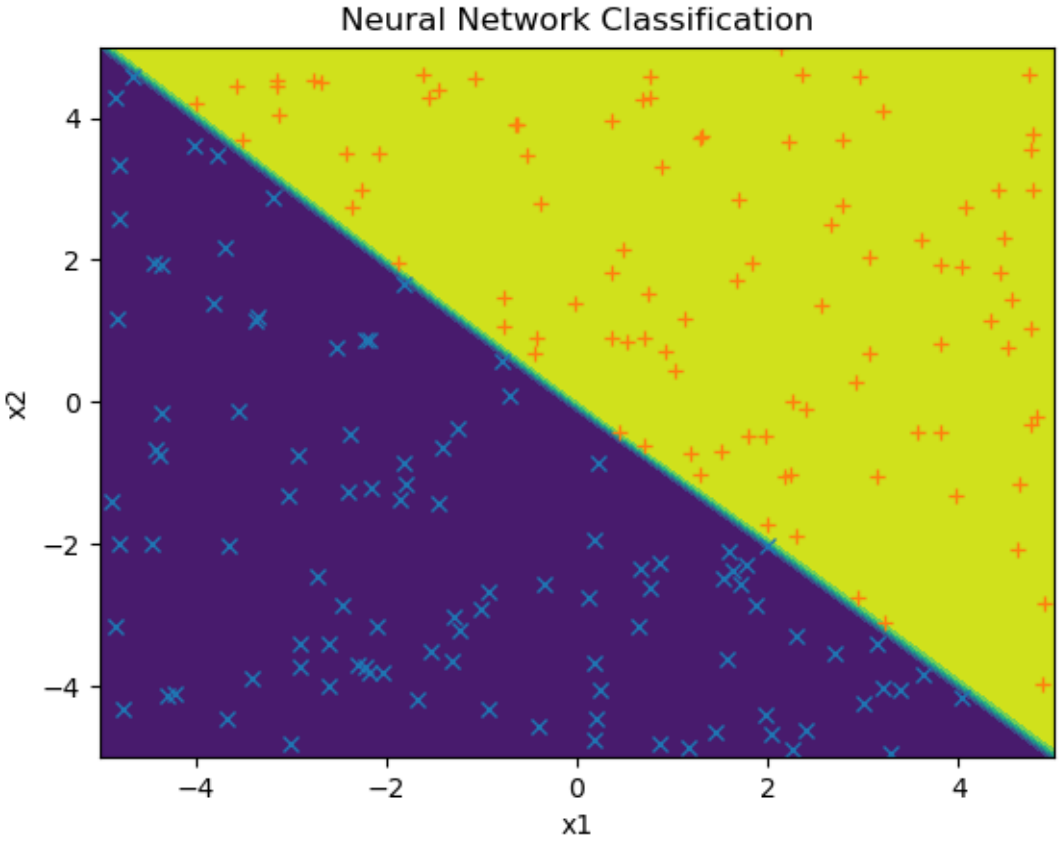

Figura 1: Resultado da classificação da rede neural com a função*sum_gt_zero()*

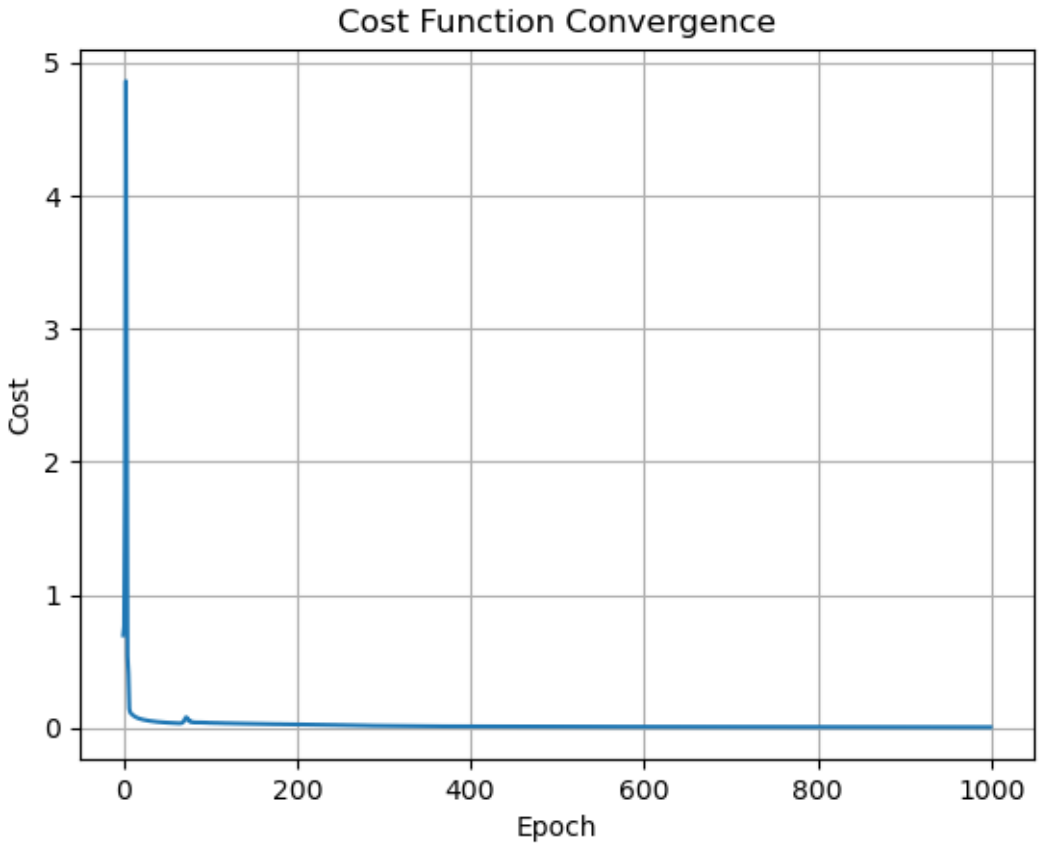

Figura 2: Convergência da função custo para a função *sum_gt_zero()*

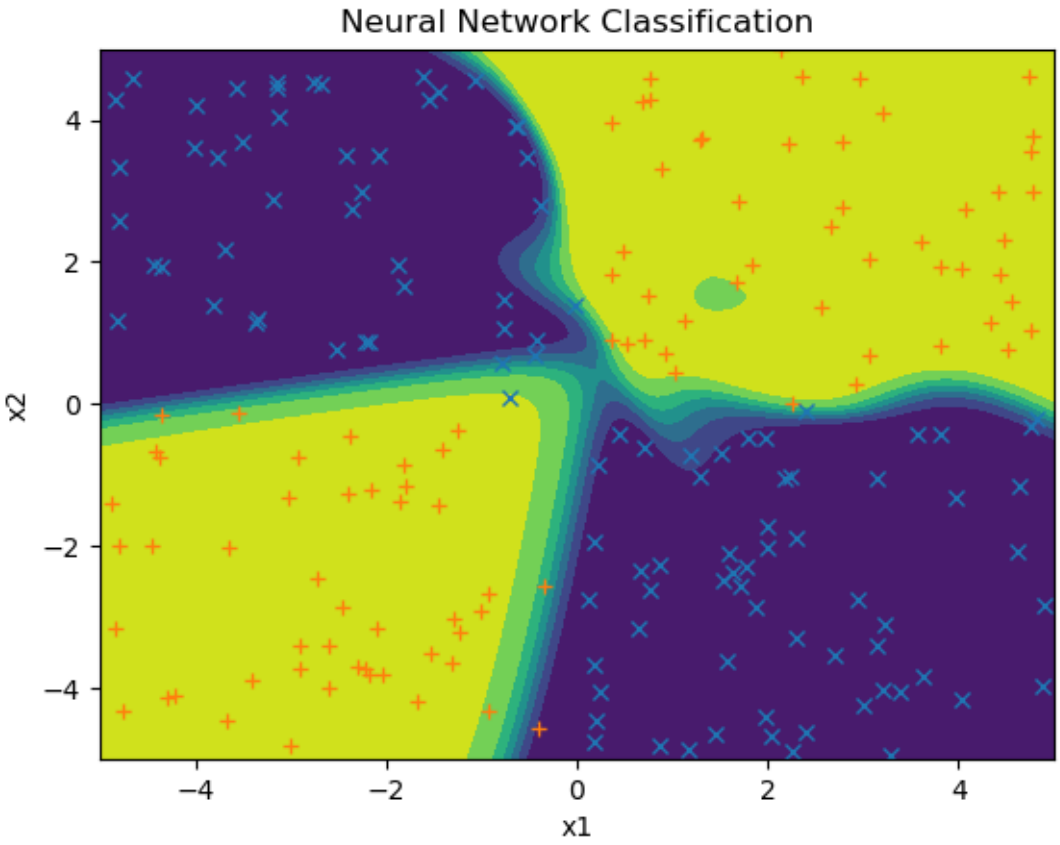

Figura 3: Resultado da classificação da rede neural com a função*xor()*

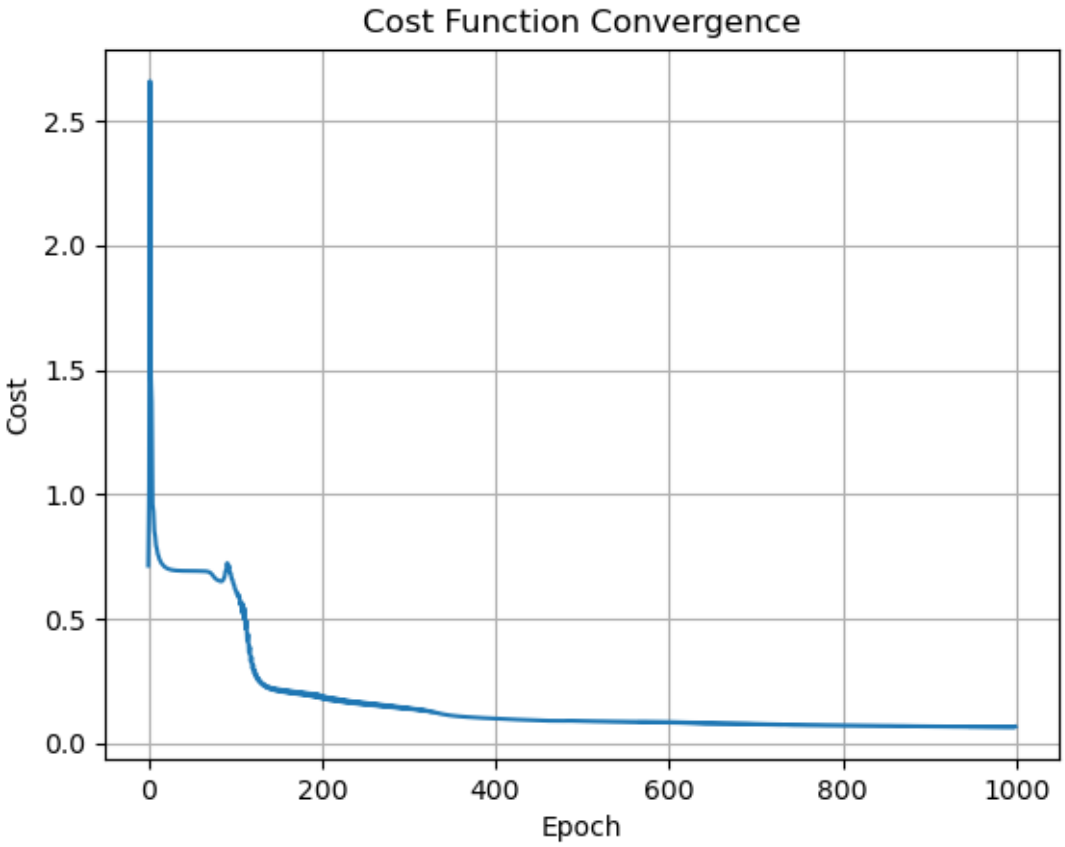

Figura 4: Convergência da função custo para a função *xor()*

Figura 5: Figura disponibilizada para posterior segmentação de cores

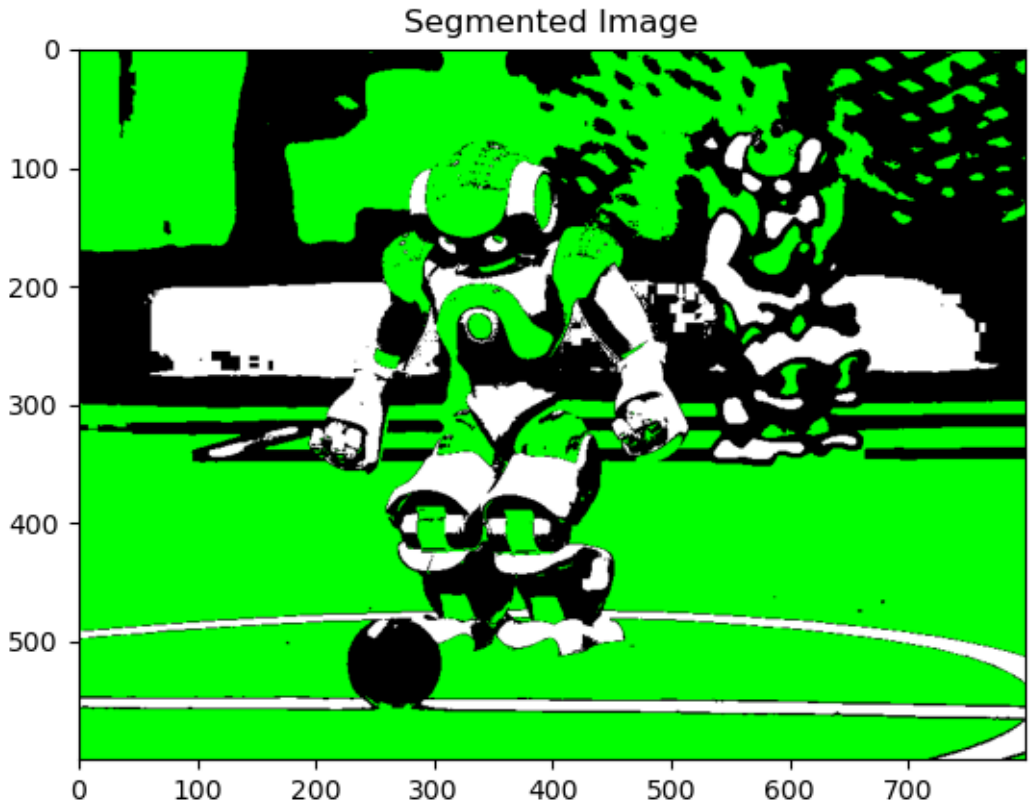

Figura 6: Imagem segmentada após aprendizado pela rede neural. Observe que o método conseguiu identificar as bordas da imagem de forma eficiente

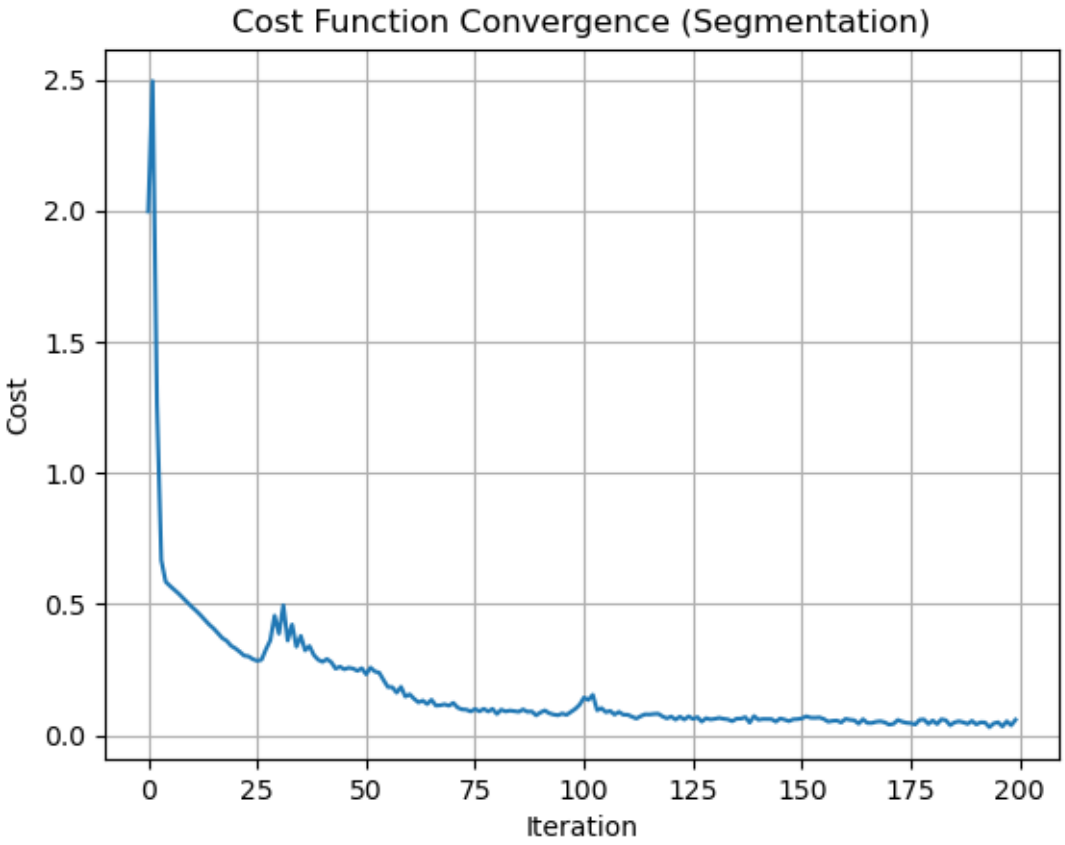

Figura 7: Convergência da função custo com a segmentação de imagens