Relatório do Lab 7 de CT-213

Trabalho 7 – Redes Neurais

Implementando uma Rede Neural para segmentação de cores para futebol de robôs

Aluno:

Bruno Costa Alves Freire

Turma:

T 22.4

Professor:

Marcos Ricardo Omena de Albuquerque Máximo

Data:

04/05/2019

Instituto Tecnológico de Aeronáutica – ITA
Departamento de Computação

1. Implementação da Rede Neural

A rede neural implementada conta com 3 camadas, sendo uma de entrada, uma oculta e uma de saída. A quantidade de entradas e de saídas, bem como a quantidade de neurônios na camada oculta são passadas como parâmetros no construtor da classe `NeuralNetwork`. Além disso, é passada também a taxa de aprendizado do *Stochastic Gradient Descent*, que é o algoritmo de otimização utilizado para realizar o *back propagation*.

A classe `NeuralNetwork` possui em seu funcionamento interno os métodos `compute_cost`, que calcula a qualidade do ajuste inferido pela rede para um conjunto de entradas e de saídas esperadas, e `back_propagation`, que invoca o método `compute_gradient_back_propagation` (que por sua vez invoca o `forward_propagation`) para realizar toda a funcionalidade de treinamento da rede a partir da inferência nas entradas e das saídas esperadas. Enquanto os três últimos métodos realizam o treinamento da rede, o `compute_cost` serve apenas para comunicar o custo da inferência ao *script* externo, para fins estatísticos.

O método `forward_propagation` calcula a ativação de todos os neurônios da rede para um exemplo de treinamento por vez. O cálculo dos vetores `z` e `a` (ativação do neurônio) é feito manualmente para cada camada, sendo utilizada a função sigmoide como função de ativação em todas as camadas.

O método `compute_gradient_back_propagation` toma uma lista de entradas e as respectivas saídas esperadas, e calcula os gradientes das *loss functions* nos pesos e nos vieses para um exemplo de treinamento por vez, e depois tomando a média. O cálculo desses gradientes é dado pelas equações

$$\nabla_w \mathcal{L}^{[2]} = \delta^{[2]} a^{[1]^T}$$

$$\nabla_b \mathcal{L}^{[2]} = \delta^{[2]} = \hat{y}^{(i)} - y^{(i)}$$

$$\delta^{[1]} = (w^{[2]^T} \delta^{[2]}) * \sigma'(z^{[1]})$$

$$\nabla_w \mathcal{L}^{[1]} = \delta^{[1]} a^{[0]^T}$$

$$\nabla_b \mathcal{L}^{[1]} = \delta^{[1]}$$

onde $\nabla_w \mathcal{L}^{[2]}$ é o gradiente da função de custo (relativo ao *i*-ésimo exemplo de treinamento) com relação aos pesos da camada 2. Analogamente, $\nabla_w \mathcal{L}^{[1]}$, $\nabla_g \mathcal{L}^{[2]}$ e $\nabla_g \mathcal{L}^{[1]}$ são gradientes da função de custo com relação aos vieses e pesos das camadas 1 e 2. Uma vez calculados esses gradientes para um exemplo de treinamento, é tomada a média para todos os exemplos.

Por fim, o método `back_propagation` propriamente dito apenas executa um passo da descida de gradiente, atualizando os pesos e vieses das camadas 1 e 2, sempre com a mesma taxa de aprendizado.

Uma vez implementada a rede neural, é hora de testar seu funcionamento com problemas simples.

2. Teste da implementação em funções de classificação simples

Os exemplos escolhidos para testar a implementação da rede neural foram as funções `sum_gt_zero` e `xor`. A primeira apenas classifica um sinal como positivo caso a soma das coordenadas seja positiva, e negativo caso essa soma seja negativa. Foi amostrado um conjunto de pontos no 2D para compor o *dataset* de treinamento dessa função, que consta na figura 1.

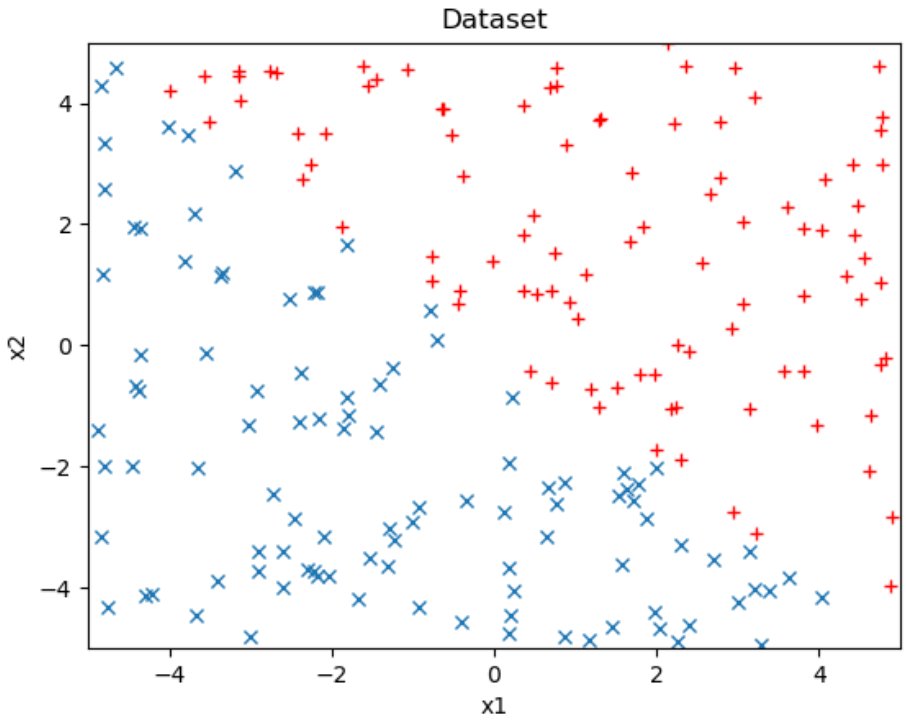

Figura 1: *Dataset* de amostras classificadas pela função `sum_gt_zero`.

A partir desse *dataset*, a rede neural é treinada e em seguida avaliamos o seu desempenho na classificação desse *dataset*, que pode ser observado na figura 2.

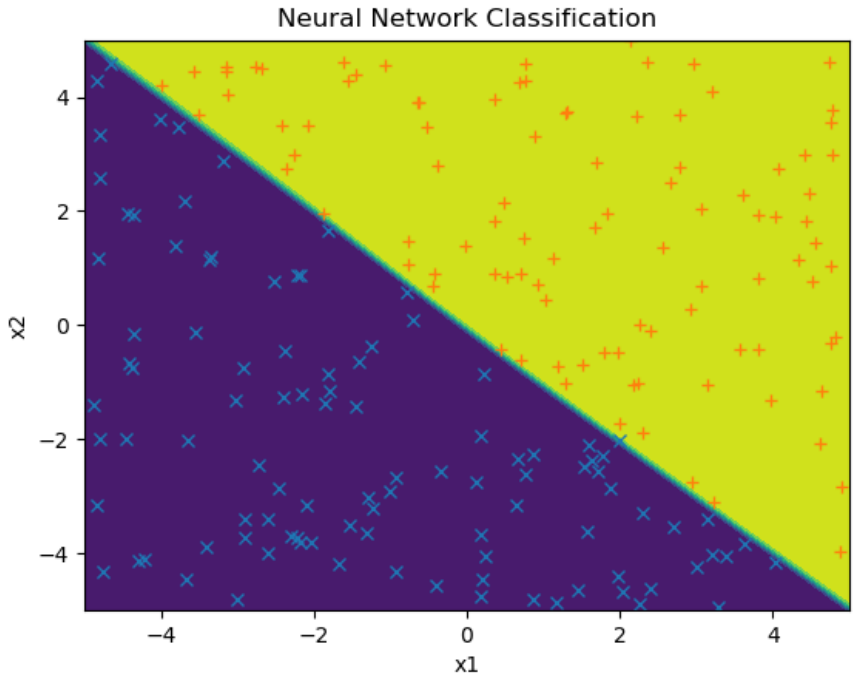

Figura 2: Resultado da classificação do *dataset* da função `sum_gt_zero` pela rede neural implementada.

Pela figura 2 podemos constatar que a rede aprendeu razoavelmente bem o conjunto de dados. Para uma análise mais profunda, vamos estudar como a função de custo evoluiu ao longo do treinamento, com o auxílio do gráfico da figura 3.

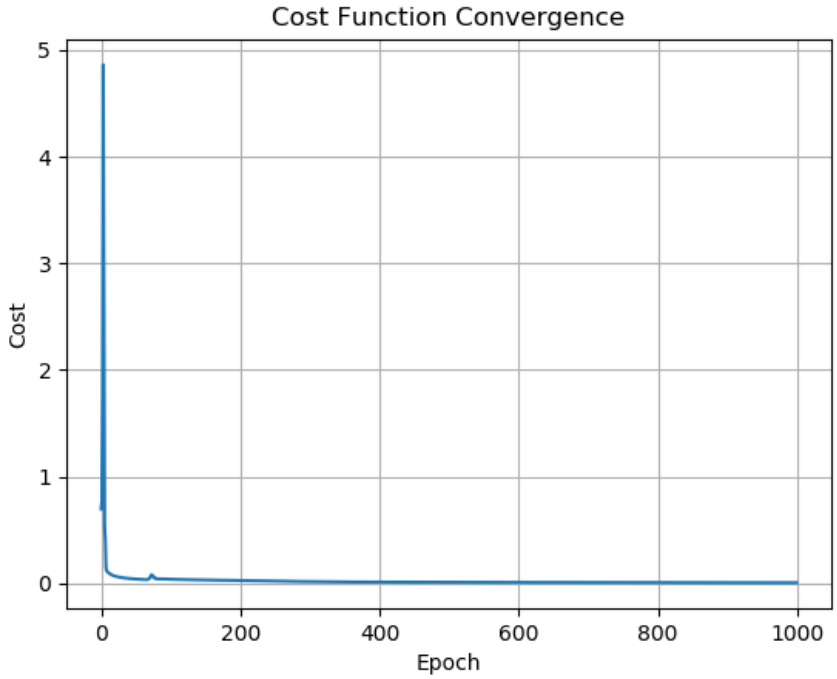

Figura 3: Gráfico da evolução da função de custo ao longo do treinamento da rede neural para a classificação do *dataset* da função `sum_gt_zero`.

Pela figura 3 podemos notar que a função de custo convergiu para zero bem rápida e abruptamente, por se tratar de uma função de classificação relativamente simples. Algo mais interessante será notado no aprendizado da função `xor`.

Para o treinamento da rede neural na função `xor`, foi novamente amostrado um *dataset*, que consta na figura 4.

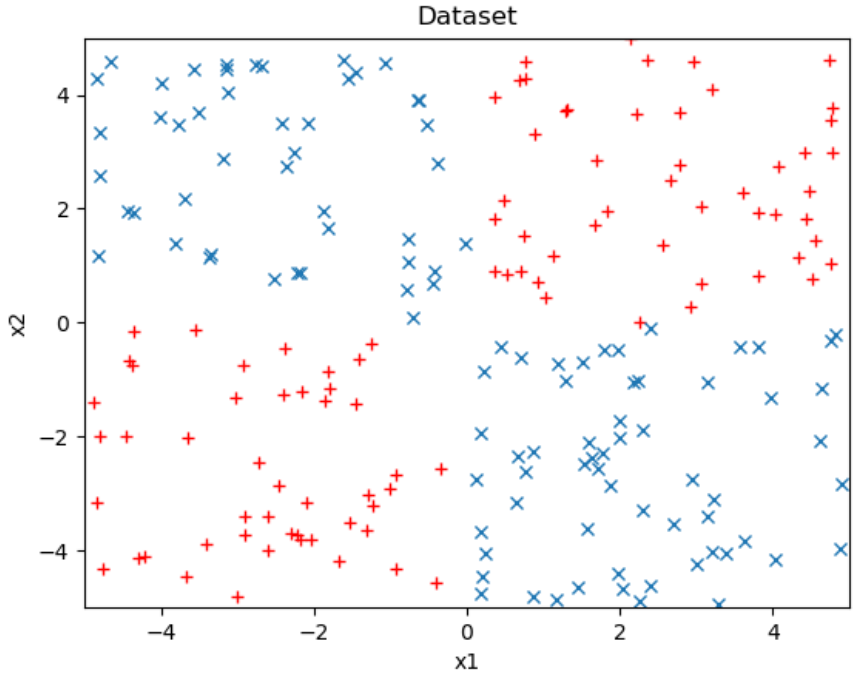

Figura 4: *Dataset* de amostras classificadas pela função `xor`.

A partir desse *dataset* foi realizado o treinamento da rede neural, que após 1000 épocas exibiu o comportamento ilustrado na figura 5. Dessa vez podemos notar que o ajuste da rede não foi tão perfeito, havendo uma notada deformidade nas curvas de nível da função aprendida. No entanto, há uma concordância razoável com a função a ser aprendida.

Para entendermos o motivo desse comportamento, precisamos novamente analisar a evolução da função de custo ao longo das épocas, o que será feito com auxílio do gráfico da figura 6.

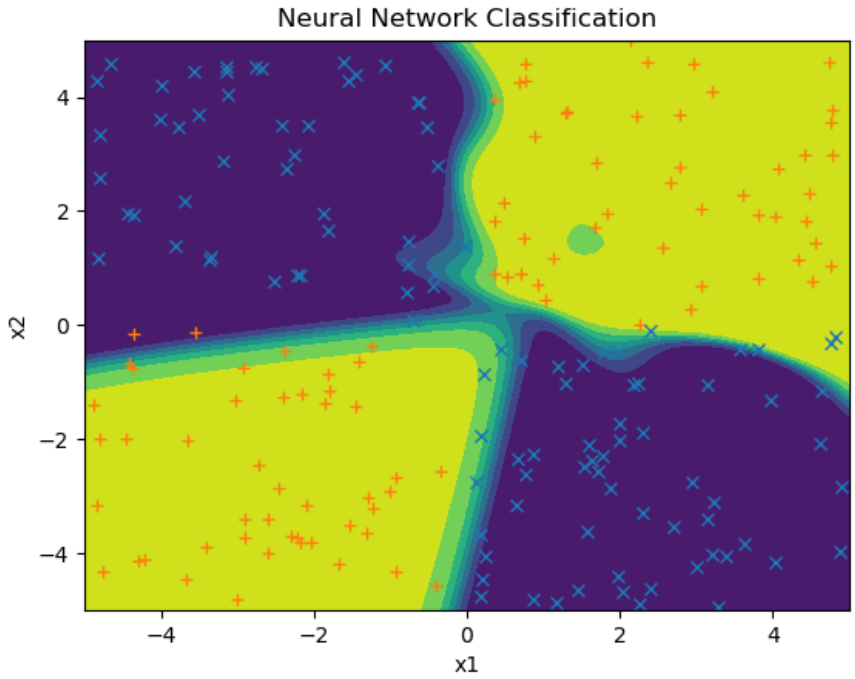

Figura 5: Resultado da classificação do *dataset* da função `xor` pela rede neural implementada.

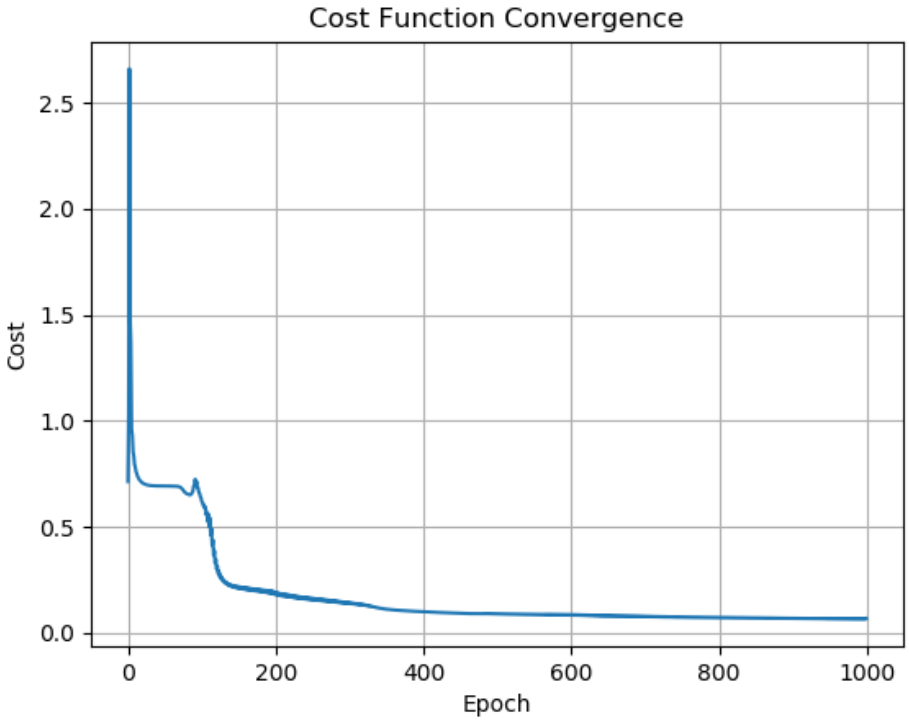

Figura 6: Gráfico da evolução da função de custo ao longo do treinamento da rede neural para a classificação do *dataset* da função `xor`.

Observando o gráfico da figura 6, podemos notar que a função de ajuste parece estagnar num valor pouco maior do que zero após algumas centenas de épocas. Possivelmente, o fator impeditivo para um melhor aprendizado da rede reside no fato da taxa de aprendizado ser constante.

Feitos os testes com funções simples, consideramos nossa rede suficientemente boa para tentar cumprir o propósito deste trabalho, a segmentação de cores.

3. Teste da Segmentação de Cores

Para o problema da segmentação de cores, tomamos uma abordagem simplificada, que irá classificar apenas dois tipos de cores, o branco e o verde. As demais cores que não sejam identificadas com nenhuma dessas duas classes serão aprendidas como uma classe neutra, que deverá ficar representada em preto.

A rede nesse caso possui 3 entradas, 20 neurônios na camada oculta, e 2 saídas (as duas classes de cor possíveis). A imagem utilizada para testar a classificação da rede é a da figura 7, que exibe um robô NAO próximo a uma bola da categoria de futebol de robôs humanoide num campo, além de um segundo NAO mais ao fundo.

Os exemplos de treinamento são dados no arquivo de texto `nao1.txt`.

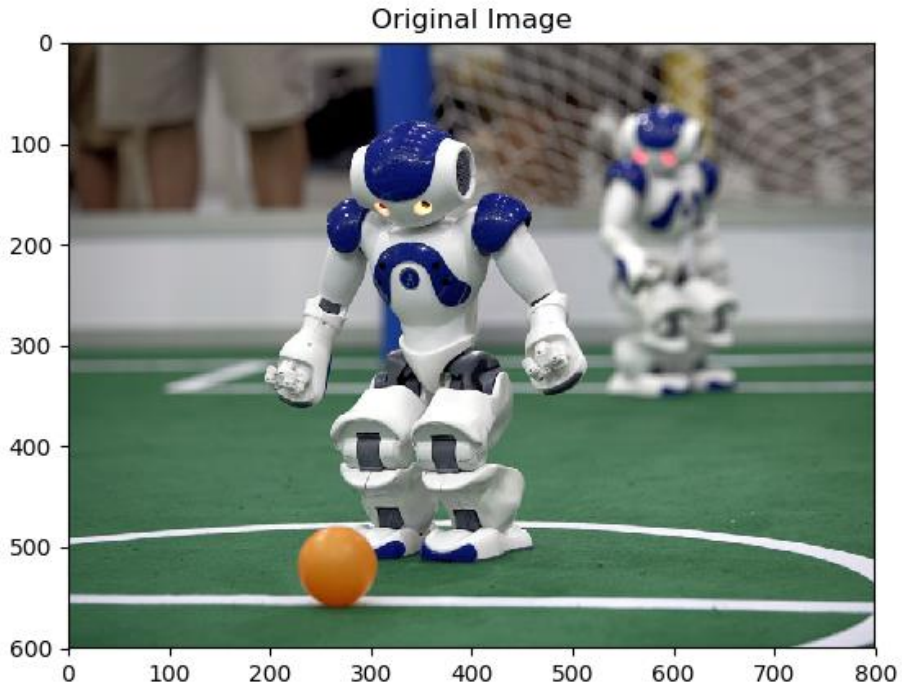

Figura 7: Imagem original sobre a qual será aplicado o algoritmo de segmentação de cores aprendido pela rede neural.

Após completado o treinamento da rede neural, testamos o seu desempenho na tarefa de segmentar as cores da figura 7, produzindo a imagem da figura 8.

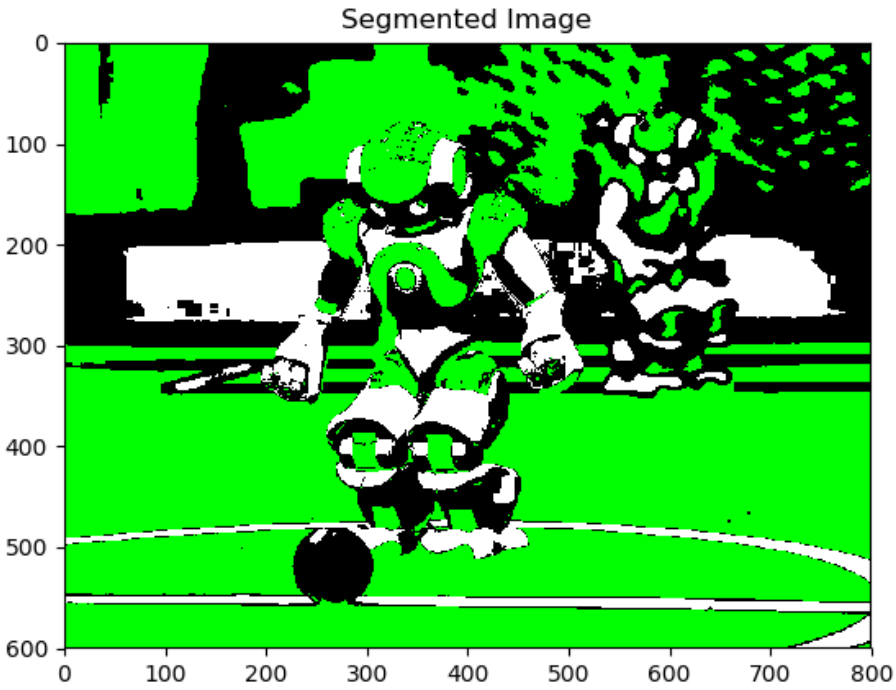

Figura 8: Imagem resultante da segmentação de cores da figura 7 realizada pela rede neural.

A partir da figura 8 vemos que a rede teve um desempenho razoável, tendo alguns problemas com a rede do gol e com as pernas de um transeunte que estava posicionado atrás do campo (as pernas do cidadão foram identificadas com a cor verde).

Novamente, fazemos uma análise mais detalhada olhando para a evolução da função de custo ao longo das épocas, com o gráfico da figura 9. Devido ao elevado custo computacional da segmentação, foram utilizadas apenas 200 épocas no treinamento da rede, uma vez que o tempo necessário para executar 1000 épocas como nos casos das funções de teste de implementação seria impeditivo.

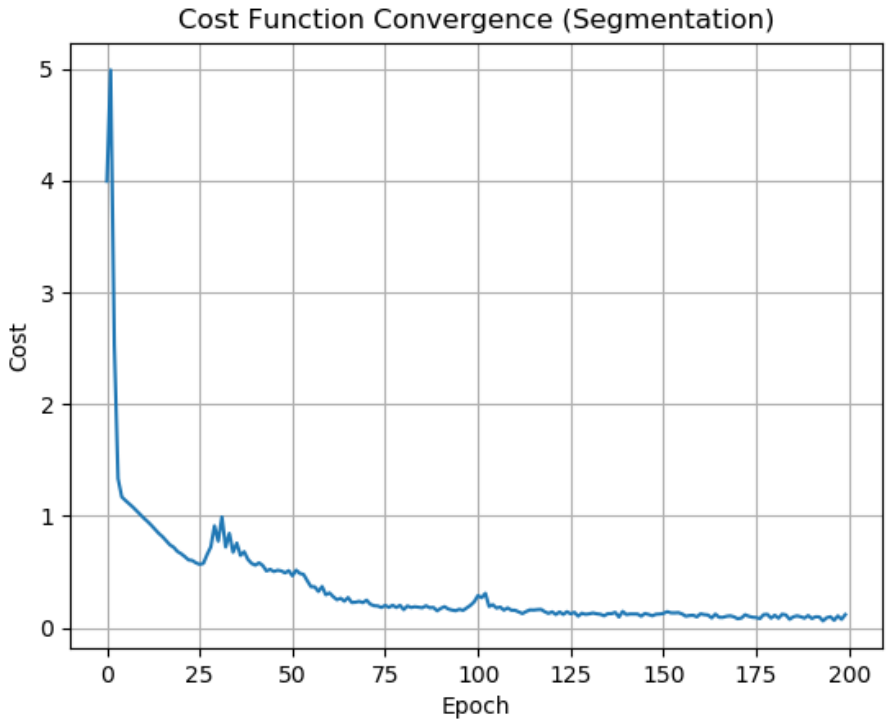

Figura 9: Gráfico da evolução da função de custo ao longo do treinamento da rede neural para a segmentação de cores da figura 7.

Da figura 9 notamos o comportamento ligeiramente ruidoso da função de custo, o que dificulta dizer se a rede tenderia a convergir para o 0 ou se poderia se estagnar em um platô mais alto, como no caso da função `xor`.