

Relatório do Laboratório 8:

Imitation Learning com Keras

Isabelle Ferreira de Oliveira
CT-213 - Engenharia da Computação 2020
Instituto Tecnológico de Aeronáutica (ITA)
São José dos Campos, Brasil
isabelle.ferreira3000@gmail.com

Resumo—Esse relatório documenta a cópia de um movimento de caminhada de um robô humanoide usando a técnica chamada *imitation learning*. Para isso, foi utilizado o *framework* de *Deep Learning* Keras, que facilita o uso do *framework* Tensorflow.

Index Terms—*Imitation learning, deep learning, Keras, Tensorflow*

I. INTRODUÇÃO

Deep learning é um tipo de aprendizado de máquina que treina para aprender através do reconhecimento de padrões em várias camadas de processamento. Entre as tarefas possíveis realizáveis estão o reconhecimento de fala, identificação de imagem e previsões, entre outras tarefas realizadas por seres humanos.

Esse tipo de estratégia possibilita, também, aprender por imitação. Assim, o comportamento desejado (uma política de controle, por exemplo) é copiado usando aprendizado supervisionado.

O *framework* de *Deep learning* Keras facilita o uso do *framework* Tensorflow

Os pseudo-códigos geral dos algoritmos *Forward Propagation*, *Back Propagation* e o cálculo das decidas de gradientes para pesos e *biases* podem ser visto a seguir. Em seguida, será apresentado como esses algoritmos foram implementados no contexto do laboratório.

```
# Forward Propagation
def forward_propagation(input):
    z[1] = weights[1]input + biases[1]
    a[1] = g[1](z[1])
    z[2] = weights[2]a[1] + biases[2]
    a[2] = g[2](z[2])
    return z, a
```

```
# Gradients' Computation
def compute_gradient_back_propagation(inputs,
    expected_outputs):
    for i in range(len(inputs)):
        input = inputs[i]
        expected_output = expected_outputs[i]

        z, a = forward_propagation(input)

        dz[2] = a[2] - expected_output
        weights_gradient[2] += dz[2] a[1].T
        biases_gradient[2] += dz[2]
```

```
dz[1] = weights[2].T dz[2] *
    derivate(g[1](z[1]))
weights_gradient[1] += dz[1] input.T
biases_gradient[1] += dz[1]

return weights_gradient, biases_gradient
```

```
# Back Propagation
def back_propagation(inputs,
    expected_outputs):
    weights_gradient, biases_gradient =
        compute_gradient_back_propagation(inputs,
            expected_outputs)

    weights[1] -= alpha * weights_gradient[1]
    biases[1] -= alpha * biases_gradient[1]
    weights[2] -= alpha * weights_gradient[2]
    biases[2] -= alpha * biases_gradient[2]
```

No pseudocódigo acima, J é a função para medir a qualidade das soluções candidatas; $m0$ e $C0$ são a média e a matriz de covariância iniciais da população que será gerada, respectivamente; m e C são, de forma análoga, a média e a matriz de covariância atuais da população que será gerada, respectivamente; e μ é o tamanho da população considerada como "melhores soluções até então".

II. IMPLEMENTAÇÃO DO ALGORITMO

Para a implementação da rede neural, era necessário preencher a função *forward_propagation()*, *compute_gradient_back_propagation()* e *back_propagation()* da classe *NeuralNetwork*. Essas funções a se completar estavam no código base fornecido [1].

Recebendo os valores de *input*, a função *forward_propagation()* foi implementada conforme apresentado no pseudo-código da Introdução, com o detalhe que a função de ativação g de todos os neurônios utilizada foi a *sigmoid*. Além disso, tanto a função *sigmoid* quanto sua derivada já eram fornecidos pelo código base.

Já na implementação de *back_propagation()*, também foi implementado conforme o sugerido no pseudo-código da Introdução, ou seja, atualizando os valores de pesos e *biases* subtraindo de seu valor um fator de aprendizado α vezes os gradientes de pesos e *biases*, respectiva-

mente. Esses valores de gradientes são aqueles calculados em `compute_gradient_back_propagation()`, supondo que essa função já foi corretamente implementado. Foi utilizado para `alpha` o mesmo que o sugerido pelo código base.

Finalmente, para implementar a função `compute_gradient_back_propagation()`, também foi feito de forma bastante semelhante ao apresentado na Introdução, com o detalhe de usar `numpy.multiply` para multiplicar termo a termo os vetores referentes aos pesos e a derivada da função de ativação.

Para testar o funcionamento dessas implementações, foi inicialmente alterado o valor da variável `classification_function` (entre "sum_gt_zero" no arquivo `test_neural_network.py` do código base, gerando imagens dos resultados da classificação de cada `dataset` para cada uma dessas funções (função "soma > 0" e função "xor") usando rede neural.

Já tendo ciência da correta implementação dos algoritmos de aprendizado, foi feito por fim um teste da segmentação de cores, executando o código do arquivo `test_color_segmentation.py`, conforme indicado no roteiro [1]. Os gráficos de resultados tanto dessa segmentação de cores, quanto das funções anteriores, foram apresentados nas Figuras ?? a ??.

III. RESULTADOS E CONCLUSÕES

A. Teste das Funções soma > 0 e xor

O aprendizado com a rede neural foi executado para as duas funções já citadas na seção anterior. Os resultados dessas execuções foram satisfatórios e saíram conforme o esperado, comprovando o correto funcionamento da implementação e a validade da utilização de rede neural com aprendizado supervisionado no aprendizado dessas funções. Esses resultados foram apresentados nas Figuras de ?? a ??.

Vale reparar que, nas Figuras ?? e ??, os custos não são sempre decrescentes com o passar das épocas. Isso, entretanto, não configura erro, uma vez que os ruídos nos `dataset` acabam interferindo durante o aprendizado. O resultado continua correto uma vez que a tendência geral é a diminuição desse custo, até a convergência. O mesmo vale, inclusive, para o gráfico da Figura ??, que será melhor discutido na subseção seguinte.

A comparação entre as Figuras ?? e ?? e entre as Figuras ?? e ?? demonstra que a classificação do `dataset` aconteceu de maneira satisfatória.

B. Teste da segmentação de cores

O custo na Figura ?? segue a tendência geral de diminuição e convergência com o passar das épocas, apesar de alguns picos já explicados na subseção anterior. Esse resultado é satisfatório e pode ser melhor observado na comparação entre as Figuras ?? e ??, que demonstra que a rede neural realmente conseguiu aprender a classificar as cores verdes e branco com considerável acerto. Vale notar também as cores não identificadas, como a bola laranja, que foi deixada como preta (cor representativa de impossibilidade de identificar).

Tendo em vista o que foi apresentado, pode-se notar, por fim, que esses algoritmos realmente se demonstraram eficazes

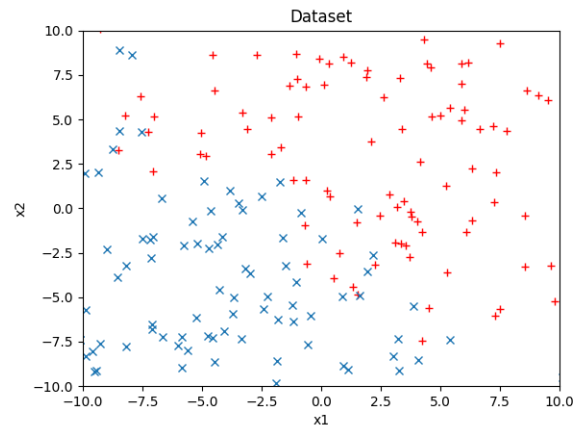


Figura 1. Exemplo de neurônio. Essa imagem de exemplo foi apresentada no roteiro [1]

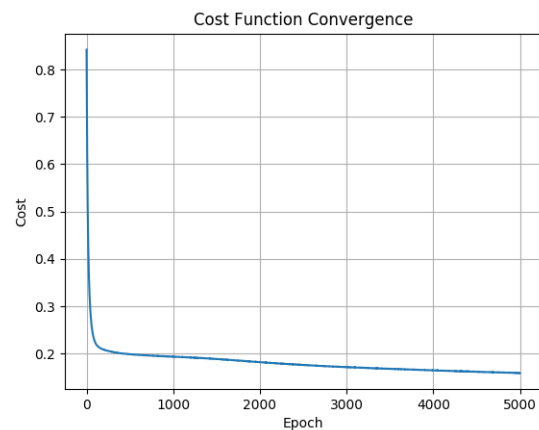


Figura 2. Exemplo de rede neural, com duas camadas (1 camada de entrada, 1 camada escondida e 1 camada de saída), como a trabalhada nesse laboratório. Essa imagem de exemplo foi apresentada no site [2]

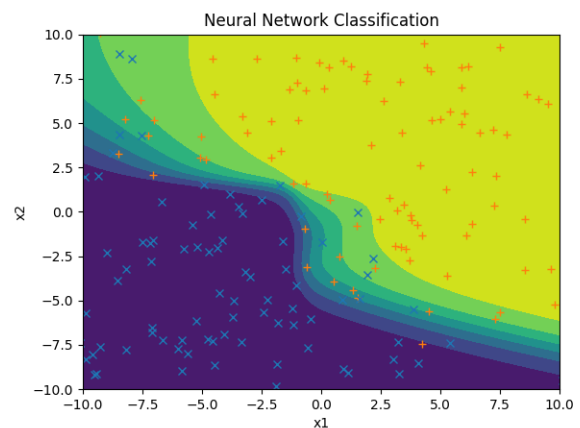


Figura 3. Convergência da função de custo para a função soma > 0.

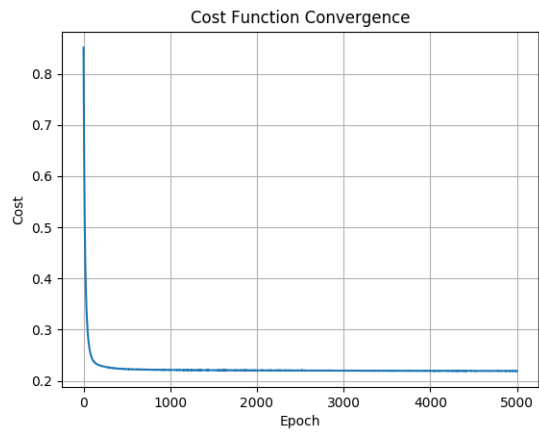


Figura 4. Dataset utilizado para o aprendizado da função *xor*.

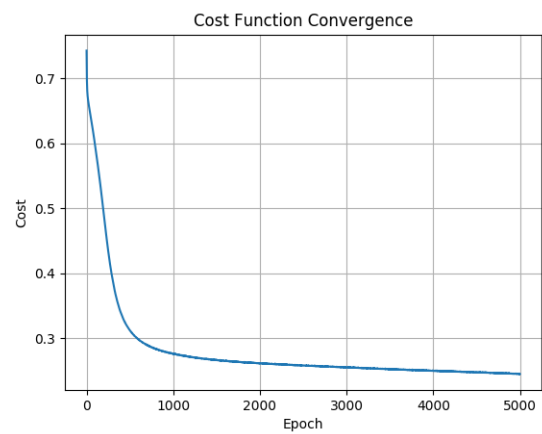


Figura 7. Dataset utilizado para o aprendizado da função *soma > 0*.

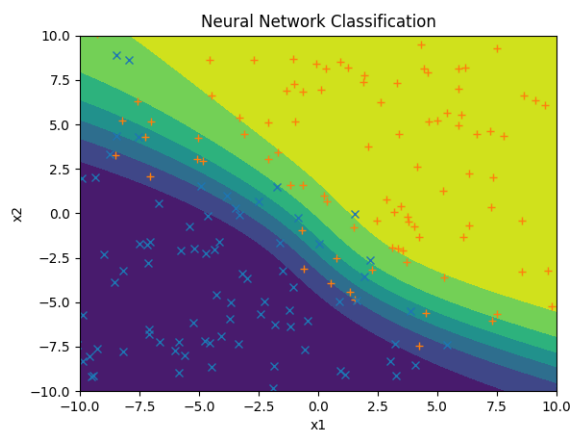


Figura 5. Convergência da função de custo para a segmentação de cores.

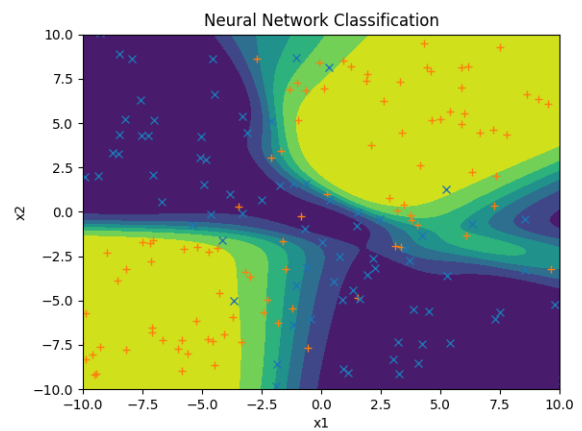


Figura 8. Convergência da função de custo para a função *xor*.

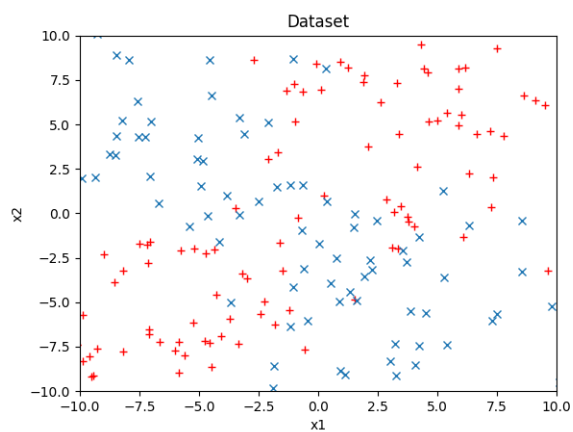


Figura 6. Resultado da classificação por rede neural para a função *soma > 0*.

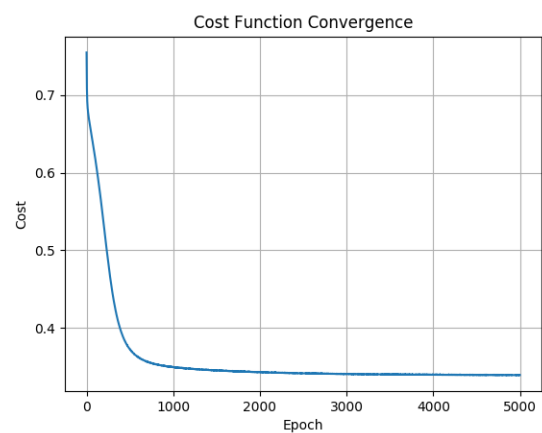


Figura 9. Imagem original a ter cores segmentadas pela rede neural.

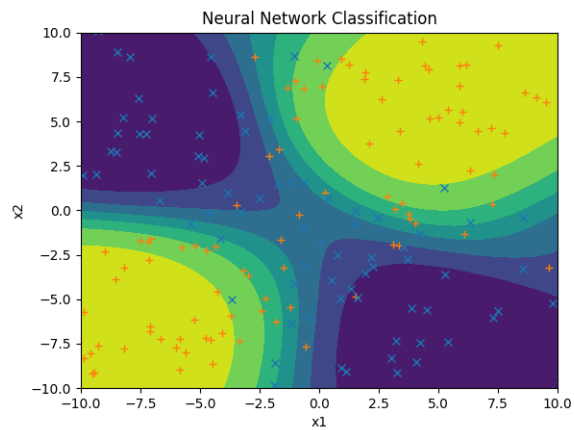


Figura 10. Imagem já com as cores segmentadas pela rede neural.

em encontrar os pesos e *biases* dessa rede neural, além de demonstrar a capacidade de aprendizado de uma rede neural para esses problemas de classificação.

REFERÊNCIAS

- [1] M. Maximo, "Roteiro: Laboratório 5 - Estratégias Evolutivas". Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.
- [2] Towards Data Science, "Neural Net from scratch". Acessado em <https://towardsdatascience.com/neural-net-from-scratch-using-numpy-71a31f6e3675>.
- [3] SAS, "Redes Neurais: O que são e qual a sua importância?". Acessado em https://www.sas.com/pt_br/insights/analytics/neural-networks.html.