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* Keras facilita o uso do *framework* Tensorflow para problemas de *Deep learning*, transformando a implementação, treino e resultado da rede neural em chamadas de API. O funcionamento dessas chamadas pode ser visto a seguir. Em seguida, será apresentado como isso foi aplicado no contexto do laboratório.

```
from keras import models

# Creates the neural network model in Keras
model = models.Sequential()
```

Sequential() cria uma pilha linear de camadas.

```
from keras import layers, activations,
    regularizers

# Adds the first layer
model.add(layers.Dense(num_neurons,
   activation=activations.some_function,
   input_shape=(input_size,),
   kernel_regularizer=regularizers.l2(lambda)))

# Adds another layer (not first)
model.add(layers.Dense(num_neurons,
   activation=activations.some_function,
   kernel_regularizer=regularizers.l2(lambda)))
```

Para a criação de uma camada na rede neural através do Keras, utiliza-se a função *model.add(layers.Dense())*, sendo o primeiro argumento referente ao número de neurônios nessa camada; "activation"configura a função de ativação; "input_shape"representa o tamanho da entrada; e "kernel_regularizer"configura a regularização para essa camada.

```
from keras import losses, optimizers, metrics

# Configures the model for training
model.compile(optimizer=optimizers.Adam(),
   loss=losses.binary_crossentropy,
   metrics=[metrics.binary_accuracy])

# Trains the model for a given number of
    epochs
history = model.fit(inputs,
   expected_outputs,
   batch_size=size_of_batch,
   epochs=num_epochs)
```

Por fim, configura-se o modelo para o treino, escolhendo o otimizer, a função de custo e as métricas; e se treina o modelo para um determinado conjunto de entrada, tendo as saídas esperadas, o tamanho do batch e quantas épocas serão efetuadas.

II. Implementação

Para a implementação da rede neural conforme os parâmetros requisitados pelo roteiro do laboratório [1] e apresentada na tabela da Figura 1, era necessário utilizar do código de adição de camadas a uma rede, além de configuração e treino do modelo, apresentado na Introdução.

Tendo em vista que em *keras.activations* não há função de ativação Leaky ReLU, utilizou-se a recomendação sugerida pelo roteiro [1] para usar Leaky ReLU no Keras, ou seja, foi adicionado uma camada do tipo LeakyRelu após ter definido uma camada (usando função de ativação linear).

Além disso, para configurar o modelo, setou-se o parâmetro *loss* da função *compile()* para *losses.mean_squared_error*, uma vez que foi utilizado erro quadrático.

Por fim, para treinar, o tamanho do *batch* foi o tamanho total de entradas, para que seja usado todo o *dataset* em cada iteração do treinamento.

Layer	Neurons	Activation Function
Dense	75	Leaky ReLU ($\alpha = 0,01$)
Dense	50	Leaky ReLU ($\alpha = 0,01$)
Dense	20	Linear

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

.

III. Resultados e Conclusões

A. Estudo de implementação de Rede Neural com Keras

O código do arquivo *test_keras.py* foi estudado para se entender a utilização do *framework* Keras na implementação de redes neurais. O que foi aprendido resultou no texto escrito na Introdução.

B. Análise do efeito de Regularização

Esse arquivo *test_keras.py* continha a implentação do aprendizado das funções "soma maior que zero"e "xor"para diferentes valores de regularização.

Após a execução desse arquivo, obteve-se os resultados apresentados nas Figuras de 2 a 11, sendo as Figuras 2 e 7 os *dataset* utilizados para as funções "soma maior que zero"e "xor", respectivamente.

Analisando os resultados, é possível notar que em todas as situações (com e sem regularização, e para as duas funções) a rede obtive uma classificação satisfatória. A classificação com regularização, entretando, apresentou-se muito mais acertiva, conforme se pode observar pelas comparação das Figuras 4 e 6 para "soma maior que zero"e 9 e 11 para "xor".

A questão da convergência da função de custo também pode ser comparada. Para os casos com regularização, a convergência se deu bem antes em número de épocas. Isso pode ser observado nas Figuras 3 e 5, para "soma maior que zero"e nas Figuras 8 e 10 para "xor".

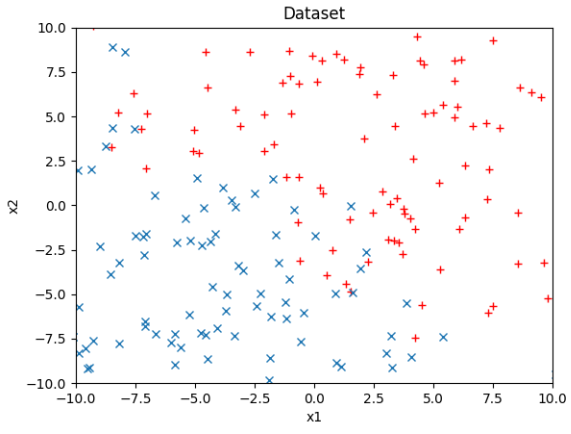

Figura 2. Exemplo de neurônio. Essa imagem de exemplo foi apresentada no roteiro [1]

.

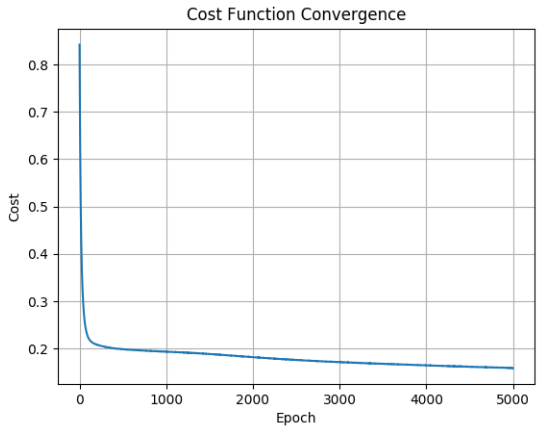

Figura 3. Exemplo de rede neural, com duas camadas (1 camada de entrada, 1 camada escondida e 1 camada de saída), como a trabalhada nesse laboratório. Essa imagem de exemplo foi apresentada no site [2]

.

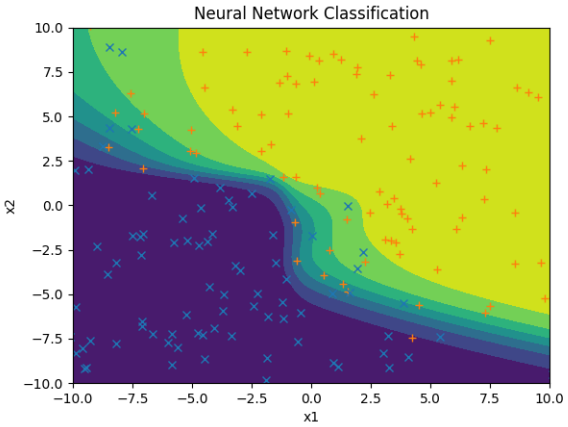

Figura 4. Convergência da função de custo para a função *soma > 0*.

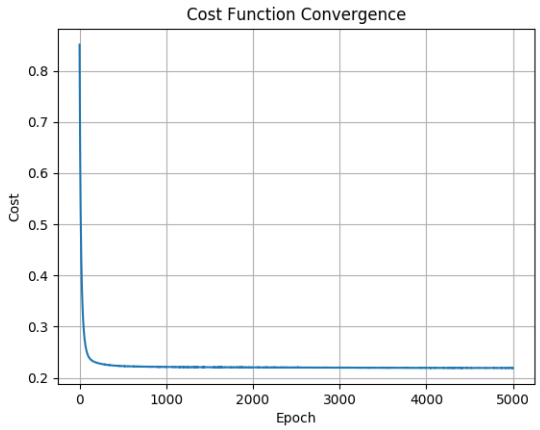

Figura 5. *Dataset* utilizado para o aprendizado da função *xor*.

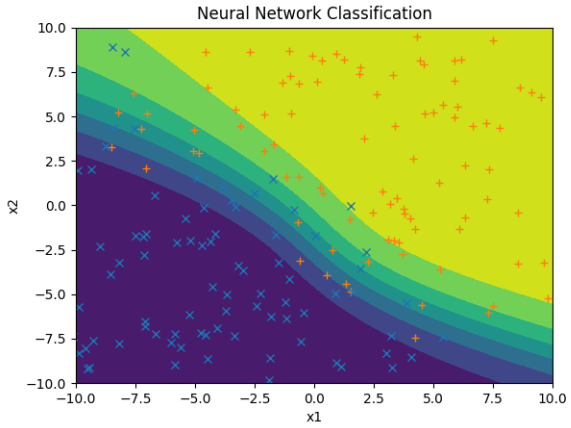

Figura 6. Convergência da função de custo para a segmentação de cores.

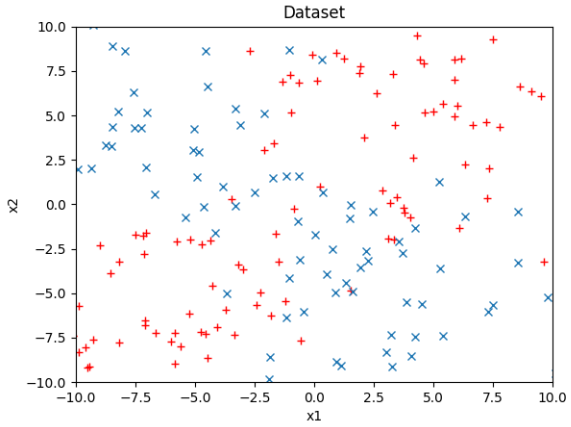

Figura 7. Resultado da classificação por rede neural para a função *soma > 0*.

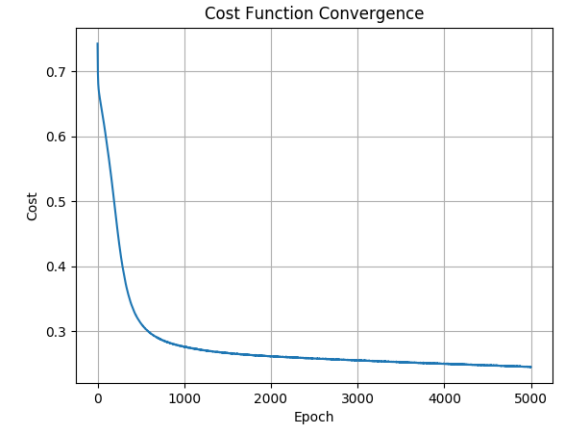

Figura 8. *Dataset* utilizado para o aprendizado da função *soma > 0*.

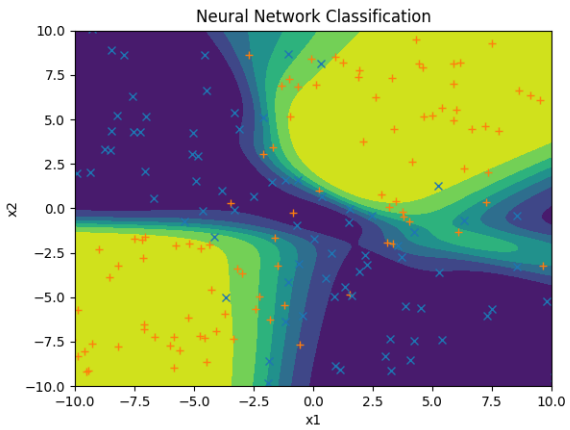

Figura 9. Convergência da função de custo para a função *xor*.

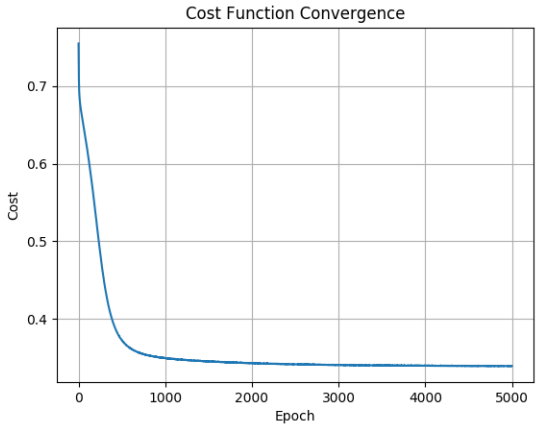

Figura 10. Imagem original a ter cores segmentadas pela rede neural.

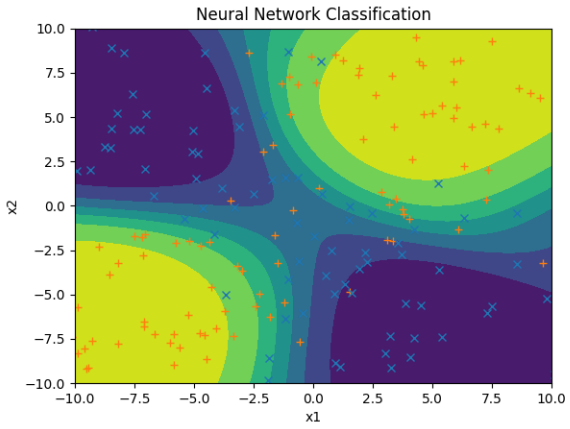

Figura 11. Imagem já com as cores segmentadas pela rede neural.

C. *Imitation Learning*

Após a implementação da rede neural com Keras conforme o explicado na seção Implementação, os resultados obtidos estão apresentados nas Figuras de 12 a 16. A comparação entre os gráficos de azul (curva original) e laranja (função aprendida por rede neural) dessas figuras demonstra que a implementação aconteceu de maneira satisfatória, uma vez que as funções ficaram bastante semelhantes.

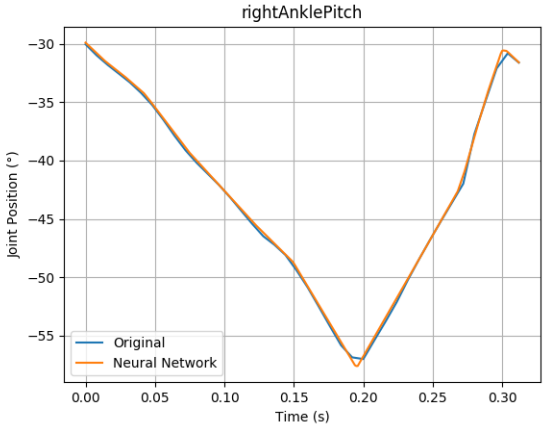

Figura 12. Resultado da classificação por rede neural para a função *soma > 0*.

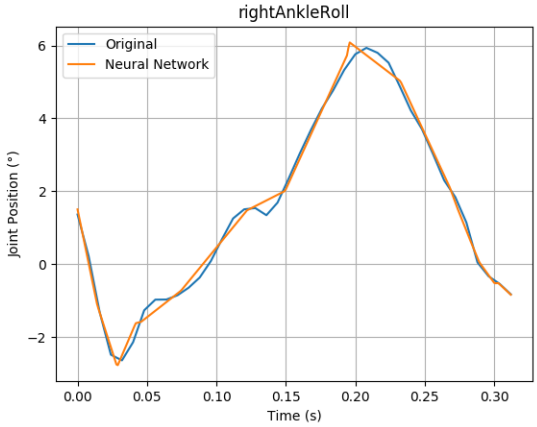

Figura 13. *Dataset* utilizado para o aprendizado da função *soma > 0*.

Tendo em vista o que foi apresentado, pode-se notar, por fim, que algoritmos de *Deep leaning* e o *framework* Keras realmente se demonstraram eficazes em realizar aprendizado por imitação.

Referências

[1] M. Maximo, "Roteiro: Laboratório 8 - Imitation Learning com Keras". Instituto Tecnológico de Aeronáutica, Departamento de Computação. CT-213, 2019.

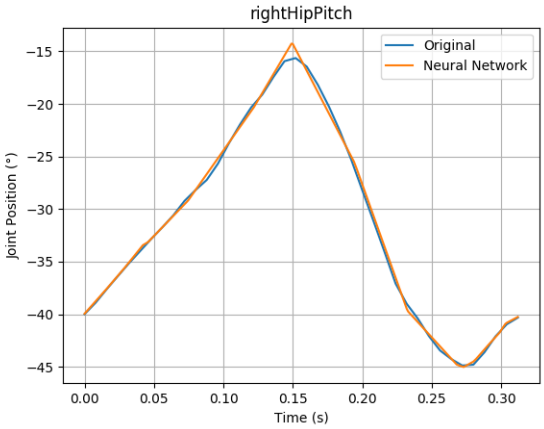

Figura 14. Convergência da função de custo para a função *xor*.

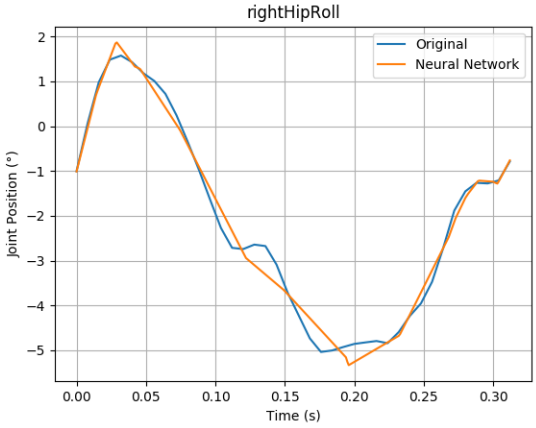

Figura 15. Imagem original a ter cores segmentadas pela rede neural.

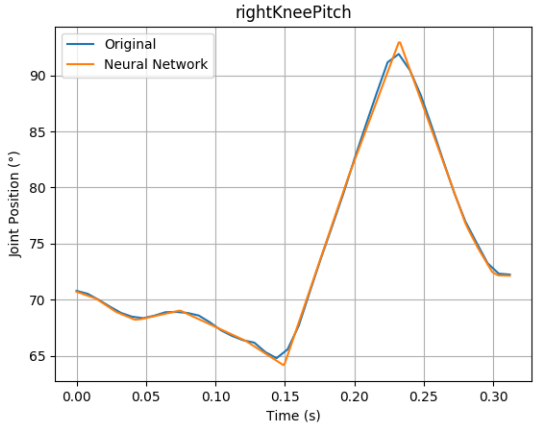

Figura 16. Imagem já com as cores segmentadas pela rede neural.